can-logic: Automotive Intrusion Detection via Temporal Logic

Kidmose, Brooke Elizabeth; Meng, Weizhi

Published in:
Proceedings of the 13th International Conference on the Internet of Things

Link to article, DOI:
10.1145/3627050.3627059

Publication date:
2023

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

can-logic: Automotive Intrusion Detection via Temporal Logic

Brooke Lampe
blam@dtu.dk
Technical University of Denmark
Kongens Lyngby, Denmark

Weizhi Meng
weme@dtu.dk
Technical University of Denmark
Kongens Lyngby, Denmark

ABSTRACT

The controller area network (CAN) protocol is all but synonymous with the terms “automotive network" or “in-vehicle network." Nearly all modern vehicles leverage one or more controller area networks to facilitate intra-vehicle communications. Unfortunately, the CAN bus is inherently insecure and severely resource-constrained. As such, researchers, industry experts, and legislators have been hard-pressed to improve the security of the CAN bus. Automotive intrusion detection has been propounded in the literature as a lightweight CAN bus security solution. Automotive intrusion detection systems (IDSs) can solve many of the resource-related challenges of CAN bus security; unfortunately, they still fall short in terms of practicability. An IDS capable of successfully detecting attacks will inevitably raise some false positives. When it comes to in-vehicle networks, even a relatively low false positive rate, e.g., 0.0001, would still produce approximately 100 false positives every 1,000,000 messages—that is, every five to ten minutes. Needless to say, such an IDS would not be practicable.

In this paper, we highlight several engineering-based indicators of compromise (IoCs) that—for all practical purposes—should never occur in the absence of an attack. We leverage both “normal" and attack CAN traffic captures to rationalize our IoCs. Next, we adapt our chosen IoCs to formal specifications. To formalize our engineering-based IoCs, we examine linear temporal logic (LTL), metric temporal logic (MTL), and signal temporal logic (STL)—to be more precise, a first order extension of STL. We conjecture that many IoCs can be codified as LTL, MTL, and/or STL properties. Finally, we implement our IoCs in Python and conduct a benchmark to assess the practicability (in terms of false positives) of our IoCs. The ultimate goal of our work—beyond the scope of this paper—is to encode each IoC in temporal logic and construct a formal “monitor" capable of detecting anomalies in CAN packet traces. The formal monitor would be an online monitor and would be installed in a real vehicle to protect it from CAN bus attacks.

KEYWORDS

automotive, controller area network, intrusion detection system, temporal logic, metric temporal logic, monitor

ACM Reference Format:
Brooke Lampe and Weizhi Meng. 2023. can-logic: Automotive Intrusion Detection via Temporal Logic. In *The International Conference on the Internet of Things (IoT 2023), November 07–10, 2023, Nagoya, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3627050.3627059

IoT 2023, November 07–10, 2023, Nagoya, Japan

ACM ISBN 979-8-4007-0854-1/23/11.
https://doi.org/10.1145/3627050.3627059

1 INTRODUCTION

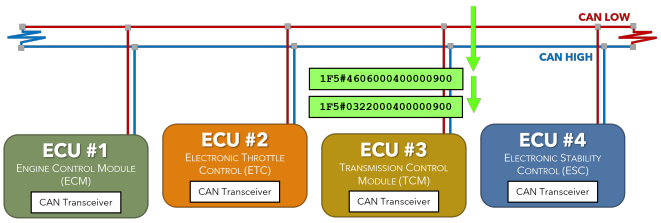


Figure 1: Attack-free scenario

The *controller area network (CAN)* protocol is immanently insecure [6, 22]. It was developed back in the early 1980s [11]—well before the advent of *connected and autonomous vehicles (CAVs).* At the time, automobiles were not *connected*—they did not have built-in Bluetooth, Wi-Fi, and cellular connections—and automobiles were not *autonomous*—they did not have *advanced driver assistance systems (ADASs),* such as parking assist or lane-keeping assist. Therefore, standard security practices (e.g., authentication, encryption, etc.) were not incorporated into the protocol. The CAN protocol is still the de facto standard for in-vehicle networks (IVNs), and it is as insecure today as it was when it was designed. In Figure 1 depicts the CAN bus under attack-free conditions.

A large body of work—from research papers [6, 18] to protocols [13, 16] to prototypes [20, 27] to standards [27] to legislation [9, 10]—has been devoted to improving the security of CAN protocol. Unfortunately, one fundamental problem has inhibited the widespread adoption of these security improvements: *practicability.*

The CAN bus is resource-constrained: it has limited bandwidth, and its endpoints are electronic control units (ECUs) with limited

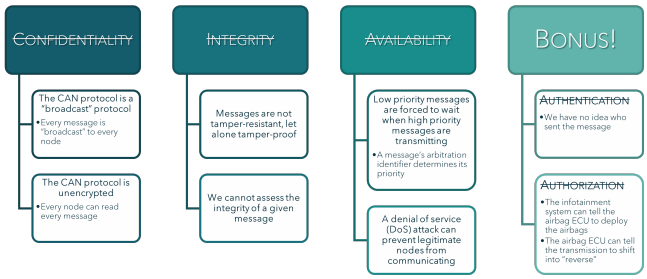


Figure 2: Vulnerabilities of Controller Area Networks

Figure 3: The on-board diagnostic port (left) and a diagnostic device (right).

computing power. As such, ex post facto attempts to integrate either authentication [33, 35] or encryption [8, 32] into the CAN bus generally require significant re-engineering effort as well as new hardware. Similar practicability issues plague new CAN standards, such as [27]. Automotive *intrusion detection systems (IDSs)* have been proposed as a re-engineering-free alternatives to cryptographic security strategies, but they, too, suffer from practicability issues, namely, false positives. When it comes to in-vehicle networks, even a relatively low false positive rate, e.g., 0.0001, would still produce approximately 100 false positives every 1,000,000 messages—that is, every five to ten minutes.

In this paper, we identify several engineering-based *indicators of compromise (IoCs)* that are virtually guaranteed to indicate an attack. Then, we adapt our chosen IoCs to formal specifications. For the purposes of formalizing our engineering-based IoCs, we examine *linear temporal logic (LTL), metric temporal logic (MTL),* and *signal temporal logic (STL)*—specifically, a first order extension of STL. Our conjecture is that many IoCs can be codified as LTL, MTL, and/or STL properties. Ultimately, the goal of this work—beyond the scope of this paper—is to encode each IoC in temporal logic and construct a formal "monitor" capable of detecting anomalies in CAN packet traces. The formal monitor would be an online monitor and would be installed in a real vehicle to protect it from CAN bus attacks. In Figure 3, we can see the on-board diagnostic port of a 2016 Chevrolet Silverado. An IoC-based IDS could easily be deployed on this port, as it is required by law in the United States and the European Union [1].

Contributions: Our contributions are as follows:

(1) We analyze various CAN bus attacks and identify indicators of compromise (IoCs) that should never occur in attack-free conditions. We include snippets of both "normal" CAN traffic and attack CAN traffic to justify our IoCs.
(2) We adapt our engineering-based IoCs to formal—or semi-formal—temporal logic to facilitate the use of a formal monitor for automotive intrusion detection.
(3) We implement our IoCs in Python and conduct a benchmark to determine if our IoCs can reduce false positives to a practicable level.

The remainder of this paper is organized as follows: Section 2 motivates our work, Section 3 covers essential background information, and Section 4 highlights a number of related works. Then, Section 5 details our methodology and Section 6 benchmarks our approach. In Section 7, we analyze the results of our benchmark, and in Section 8, we discuss limitations and opportunities for future work. Lastly, in Section 9, we conclude our work.

2 MOTIVATION

Intrusion detection via deep learning models hinges on the models' ability to differentiate between benign and anomalous traffic. An ideal intrusion detection system would produce neither false positives (benign traffic misclassified as anomalous) nor false negatives (anomalous traffic misclassified as benign). In an automotive environment, a *false positive* would alert the driver that an attack is occurring—and perhaps even attempt mitigation—when, in reality, there was no attack. A *false negative,* then, means that the IDS failed to detect an actual attack. In such a scenario, the driver would not be warned of the ongoing attack, and no mitigation attempt would be made.

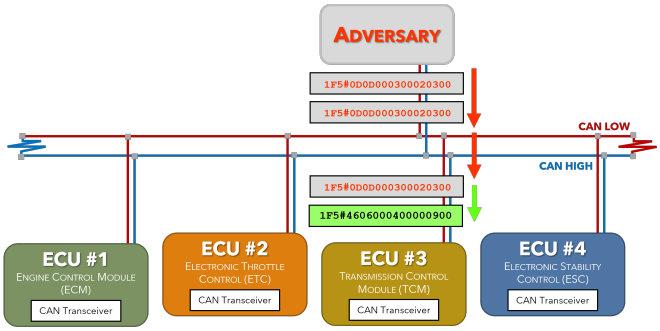
Deep learning models are not infallible; we would expect to see some false positives. In contrast, if we develop a formal model using predicates that we know will *always*—or *never*—be true, then we can reduce false positives. In this context, even a relatively low false positive rate, e.g., 0.0001, will still produce 100 false positives every 1,000,000 messages—or every five to ten minutes (depending on the vehicle). If an automotive intrusion detection system raises false positives every ten minutes, then the user will probably disable it. At best, the user will become desensitized to the warning, and, if an attack does occur, he or she might not realize it is, in fact, an attack—until it is too late.

Deep learning is oft-cited in the literature as a means of reducing false positives, but although deep learning is capable of remarkably low false positive rates (FPRs) of, e.g., 0.005, 0.0001, and even 0.00004 [22], when we receive 1,000,000 messages every five to ten minutes, then even the best-case scenario, 40 false positives, is still unacceptable. Therefore, we need to tighten our detection capability, shifting the bias from false positives to false negatives.

Shifting the bias from false positives to false negatives might be concerning to some. However, as it stands, most—if not all—vehicles face a 100% false negative rate. As it is now, there is no protection, so even if we cannot detect all attacks, we are still significantly advancing automotive security.

3 BACKGROUND

The term "controller area network" (CAN) is nearly synonymous with the term "automotive network." The CAN bus has been the *de jure* standard in the United States since 2008 [1]; as such, the average modern automobile implements controller area networks as one—or more—of its in-vehicle networks (IVNs). IVNs interconnect an automobile's electronic control units (ECUs)—i.e., automotive embedded systems—and its sensors. The CAN bus is amazingly reliable; alas, it is also spectacularly insecure. The CAN bus was developed back in 1983 [11], at a time when automobiles did not have Bluetooth, Wi-Fi, and cellular access. As such, IVN security—i.e., CAN bus security—was deemed unnecessary. In order to compromise the CAN bus, the attacker would have to physically access the vehicle, and, at that point, he or she might as well cut the brakes. As



The adversary sends spoofed CAN frames to the transmission control module (TCM) instructing it to shift to "neutral" mode. The spoofed CAN frames overpower the legitimate CAN frames, and the vehicle shifts to "neutral." If the TCM receives both legitimate and spoofed CAN frames, then it will shift between the mode prescribed by the legitimate CAN frames and the mode prescribed by the adversarial CAN frames.

Figure 4: Attack scenario: Gear spoofing

such, the CAN bus lacks standard security features, e.g., authentication, authorization, and encryption, etc. Figure 2 highlights several major vulnerabilities of the CAN bus.

Modern times have given rise to connected and autonomous vehicles (CAVs), which, as the name implies are connected—e.g., to the Internet, road-side units, and fellow CAVs—and autonomous. Autonomous—and even semi-autonomous—vehicles are especially vulnerable, as they are heavily dependent on electronics. Vehicles that are more autonomous are also more electronic (i.e., less mechanical), meaning that an attacker can seize control of more systems. For example, parking assist and lane-keeping assist are two examples of an automated driver assistance system (ADAS) that necessitates electronic control of the steering column.

In 2014, security researchers Miller and Valasek conducted a remote attack on 2014 Jeep Cherokee. They seized control of the accelerator and the brakes, demonstrating the seriousness of automotive cyberattacks. In 2015, the same researchers implemented a proof-of-concept attack on the Jeep Cherokee's steering column, compelling the vehicle to turn sharply at speed—30 miles per hour [25, 26]. Figure 4 illustrates a gear-spoofing attack: an attacker generates spoofed "neutral" messages, overwhelming the true "drive" messages and prompting the vehicle to shift into "neutral" gear.

In the United States, since 2020, several states have instituted "right to repair" laws that would allow equipment owners and independent repair shops to repair various types of equipment (e.g., laptops, appliances, farm equipment, and even vehicles). "Right to repair" laws are intended to facilitate repair rather than disposal—often in a landfill—and should be better for the environment [30]. However, there are some concerns with regard to safety and security. The National Highway Traffic Safety Administration (NHTSA) instructed automotive manufacturers to disregard Massachusetts's "right to repair" law and instead abide by federal law. Massachusetts's "right to repair" law requires automotive data (e.g., telematics) to be remotely accessible. The NHTSA is concerned that an adversary "could utilize such open access to remotely command vehicles to operate dangerously, including attacking multiple vehicles concurrently" [30, 31]. Josh Siegel, an assistant professor of engineering at Michigan State University who researchers connected and autonomous vehicle (CAV) security, pointed out that the Massachusetts "right to repair" law gave the automotive industry approximately one year to build an open data platform—not enough time to create a safe and secure system [24]. Several more "right to repair" laws have passed even this year—2023—and will be coming into force soon. As such, it is urgent that controller area networks be made secure.

4 RELATED WORK

Listing 1 *A CAN traffic capture.* The capture demonstrates a denial of service (DoS) attack. The red lines indicate attack CAN frames.

```
(1671905222.535656) can0 1E5#5CFE1611030001C0
(1671905222.535661) can0 34C#F9BE0F33000D0FD8
(1671905222.536284) can0 000#0000000000000000
(1671905222.536996) can0 000#0000000000000000
(1671905222.537774) can0 0C7#0074AA47
(1671905222.537779) can0 0F9#025D40092B489C0C
(1671905222.537844) can0 000#0000000000000000
(1671905222.538581) can0 000#0000000000000000
(1671905222.538838) can0 1CE#180007FD
(1671905222.539348) can0 000#0000000000000000
(1671905222.539901) can0 1FE#069E73030000C171
(1671905222.539908) can0 362#00000000
(1671905222.539911) can0 0F1#1C0500400000
(1671905222.539946) can0 000#0000000000000000
(1671905222.540488) can0 000#0000000000000000
(1671905222.540972) can0 0AA#2CC12C0D0253F200
(1671905222.540980) can0 0BE#00030005ABFD
(1671905222.541718) can0 000#0000000000000000
(1671905222.542053) can0 0C9#8017900000001800
(1671905222.542070) can0 0D3#2CC1
(1671905222.542074) can0 18E#00000072372306BC
(1671905222.542077) can0 1ED#614004A3033070DE
(1671905222.542081) can0 135#0200175ECC161607
(1671905222.542457) can0 000#0000000000000000
(1671905222.543123) can0 000#0000000000000000
(1671905222.543699) can0 000#0000000000000000
(1671905222.544204) can0 0C1#01C957DA015E5873
(1671905222.544223) can0 0C5#8071DDFF83BCE1AC
(1671905222.544595) can0 000#0000000000000000
```

Naldurg, Sen, and Thati [28] developed an online monitor—and intrusion detection system—for specifications encoded in temporal logic. They leveraged the `EAGLE` logic language to implement a prototype IDS, `MONID`, based on a temporal logic monitor. Naldurg, Sen, and Thati provide several examples of `EAGLE`-encoded formulas, many of which are linear temporal logic (LTL) properties.

Bonakdarpour, Deshmukh, and Pajic [5] emphasized the use of signal temporal logic (STL) for security and privacy in cyber-physical systems. They designed an STL extension for cyber-physical systems security, dubbed "SA-STL," that provides standard security primitives as first-class predicates, enabling security professionals to compose machine-checkable security properties. Bonakdarpour, Deshmukh, and Pajic suggest that online STL monitoring should be supplemented by predictive metric temporal logic (MTL) monitoring.

Jones, Kong, and Belta [17] leveraged STL to define "normal" system behavior; violations of the STL formulas were then flagged as anomalies. They conducted two case studies: one on a linear system and one on a train braking scenario.

Wu et al. [36] sought to enforce the safety properties of safety-critical CPSs using real-value logical "shields." Boolean shields cannot handle real-value signals; therefore, Wu et al. built real-value shields out of Boolean monitors. Their benchmark included applications such as automotive powertrains, cruise control, and autonomous driving.

Lettnin et al. [23] explored embedded systems verification—specifically, temporal property verification. Computability issues go hand-in-hand with model checking; as such, they investigated simulation-based verification. They extended the `SystemC` temporal checker with new interfaces, equipping it to monitor variables and functions of embedded systems.

Bartocci et al. [4] surveyed specification-based monitoring in the context of cyber-physical systems. The survey includes a section on formal monitoring in automotive CPSs—in particular, runtime monitoring and property falsification.

Beyond formal monitoring, Huang, Lan, and Yu [15] leveraged signal temporal logic to build a formal control framework for autonomous vehicles. Yu and Gligor [37] developed temporal logic-based specifications to address denial of service (DoS), which they regard as both a safety and a liveness problem.

5 METHODOLOGY

To investigate the indicators of compromise that appear in attack traffic but not attack-free traffic, we leveraged the `can-train-and-test` dataset [19, 21]. We exploited both manual (e.g., visual inspection) and programmatic (e.g., filtering, counting, calculating intervals) analysis techniques.

Here, we provide several attack traffic samples, which exhibit the indicators of compromise enumerated in this section:

- Listing 1 illustrates a denial of service (DoS) attack—albeit an unsophisticated one. We can see that the patterns exhibited by a DoS attack are markedly different from the patterns exhibited by a gear spoofing attack.
- Listing 2 provides a CAN traffic snippet from a fuzzing attack. Random arbitration identifiers, paired with randomly generated data fields, are transmitted to the CAN bus.
- Listing 3 showcases a gear spoofing attack, in which the vehicle is tricked into shifting to the "neutral" gear.

5.1 Indicators of Compromise (IoCs)

We have identified four types of attacks—(1) DoS, (2) fuzzing, (3) replay, and (4) spoofing—to be detected via temporal logic predicates. For each attack, we have specified patterns to be transcribed into formal temporal logic predicates.

(1) **Denial of Service (DoS)** (See Listing 1)
   (a) An interval of less than one-half the shortest "normal" interval—i.e., the shortest attack-free interval—is highly suspicious.
   (b) The identifier `000` is highly suspicious (it is not a legitimate identifier in the four vehicles we have evaluated).
   (c) A value of `0000000000000000` for the data field can be valid—e.g., all four wheels are reporting zero speed because the car is stopped at a red light—but it is also associated with unsophisticated DoS attacks.
   (d) If the valid arbitration IDs are `0C1`, `0C5`, `0C9`, `0F1`, `0F9`, `120`, `12A`, `134`, `138`, etc., then `0C1` is the highest-priority arbitration ID. If we see `0C1` more than twice in a row, then it is highly suspicious (we have never seen the same arbitration ID repeated twice—let alone thrice—under attack-free conditions).

(2) **Fuzzing** (See Listing 2)
   (a) An interval of less than one-half the shortest "normal" interval—i.e., the shortest attack-free interval—is highly suspicious.
   (b) If the valid arbitration IDs are If the valid arbitration IDs are `0C1`, `0C5`, `0C9`, `0F1`, `0F9`, `120`, `12A`, `134`, and `138`, then an arbitration ID of `0BB` would be indicative of a fuzzing attack. Note that arbitration IDs are constrained to 11 bits, so `7FF` would be the largest possible arbitration ID (and also the lowest priority). We could check that no arbitration ID exceeds `7FF`, but since the protocol does not allow for larger arbitration IDs, it is doubtful that such a message could even be sent.

(3) **Replay**
   (a) An interval of less than one-half the shortest "normal" interval—i.e., the shortest attack-free interval—is highly suspicious.
   (b) If we see the same arbitration ID more than twice in a row, it is highly suspicious.
   (c) If one arbitration ID suddenly becomes more frequent, it is highly suspicious. Typically, we would expect an attacker to launch a replay attack in order to manipulate the vehicle's behavior—e.g., an attacker might replay a gear message that says "reverse" when the vehicle has since shifted to the "drive" gear. If successful, the replayed message would damage the transmission (i.e., strip the gears). In such cases, the attacker must overwhelm the real message ("drive" gear) with the replayed message ("reverse" gear). Often, overwhelming the real message requires a high volume of replayed messages. As such, we would expect to see the same arbitration ID more frequently.
   (d) Unfortunately, a replay attack replays a legitimate message, so the message itself would not be suspicious. We could check

Listing 2 *A CAN traffic capture.* The capture demonstrates a fuzzing attack. The red lines indicate attack CAN frames.

```
(1671907705.772800) can0 1C6#2E392C8100034DFC
(1671907705.772815) can0 1C7#08D2B72C00003F
(1671907705.772819) can0 1E5#5C002210000000CA
(1671907705.772823) can0 1FC#FF3D0068000081F3
(1671907705.773036) can0 131#9742
(1671907705.773889) can0 0F1#1C0500400000
(1671907705.774207) can0 767#
(1671907705.775279) can0 1EC#5874D575E1498948
(1671907705.776028) can0 17D#222442FF0017
(1671907705.776046) can0 1CE#080007FF
(1671907705.776352) can0 678#70724A2D62477D63
(1671907705.777146) can0 137#1F020FE52E0F5C0A
(1671907705.777431) can0 3C9#B3
(1671907705.778229) can0 0C7#00234882
(1671907705.778240) can0 0F9#00AA40184A0E631E
(1671907705.778505) can0 3AE#
(1671907705.779308) can0 1CC#0000000000
(1671907705.779329) can0 1E1#000000000314E0
(1671907705.779585) can0 10D#1CC1FA2A6F561A70
(1671907705.780410) can0 0AA#2C5F2C6102539E24
(1671907705.780448) can0 0C1#41D11D4C82B2A198
(1671907705.780459) can0 0BE#0001240591FF
(1671907705.780820) can0 376#3FDC
(1671907705.781533) can0 0C5#803E9F0D40682148
(1671907705.781545) can0 0C9#8013190023001800
(1671907705.781553) can0 0D1#C002FFFB00FD00
(1671907705.781562) can0 0D3#2C5F
(1671907705.781566) can0 1E5#5C002230000000EA
(1671907705.781911) can0 694#11B3E209FB65
```

Listing 3 *A CAN traffic capture.* The capture demonstrates a gear spoofing attack (the "neutral" gear is spoofed). The red lines indicate attack CAN frames.

```
(1674131126.787240) can0 1CC#0000000000
(1674131126.787250) can0 0C1#109F068B507C8CC5
(1674131126.787644) can0 1F5#0D0D000300020300
(1674131126.788317) can0 0C5#511F872A10C008BB
(1674131126.788328) can0 0D1#00000000000000
(1674131126.788332) can0 185#1BF2
(1674131126.788336) can0 1C7#096EF68F00003F
(1674131126.788339) can0 1E5#5C00325007000121
(1674131126.789404) can0 0AA#2BBD2BF480540C00
(1674131126.790481) can0 0BE#0C02000591FB
(1674131126.790490) can0 0C9#800F8E2D00081800
(1674131126.790493) can0 0D3#2BC3
(1674131126.790496) can0 1A1#002641414E4000
(1674131126.790498) can0 18E#00001064A63D063D
(1674131126.790500) can0 1A3#80
(1674131126.791111) can0 1F5#0D0D000300020300
(1674131126.791578) can0 1ED#61410000000070DE
(1674131126.791584) can0 1AA#0063D6646A7009
(1674131126.791587) can0 1BA#069569566D695970
(1674131126.791590) can0 1C3#064A069500000000
(1674131126.792649) can0 1C4#6D62C50F060003FD
(1674131126.792654) can0 1C5#2BE92BE92D59
(1674131126.792657) can0 1CE#100007FE
(1674131126.792659) can0 1DF#80000000
(1674131126.792662) can0 1F4#000000000000
(1674131126.792664) can0 287#000000
(1674131126.794197) can0 1F5#0D0D000300020300
(1674131126.795848) can0 0F1#000705400000
(1674131126.796836) can0 1F5#0D0D000300020300
```

for repeated messages, but some messages would normally be repeated—e.g., if we are stopped at a red light, the message `0000000000000000` would be sent repeatedly by the ECU that reports wheel speeds.

(4) **Spoofing** (see Listing 3 and Figure 4)
   (a) An interval of less than one-half the shortest "normal" interval—i.e., the shortest attack-free interval—is highly suspicious.
   (b) If we see the same arbitration ID more than twice in a row, it is highly suspicious.
   (c) If one arbitration ID suddenly becomes more frequent, it is highly suspicious. The justification for this phenomenon is similar to the justification for the same phenomenon during a replay attack.
   (d) Unfortunately, spoofing attacks are intended to spoof a somewhat "normal" value in order to induce a certain behavior—e.g., spoofing "reverse" for the current gear while driving in order to damage the transmission (i.e., strip the gears). As such, spoofed messages will generally appear legitimate.

Several of the above patterns are exhibited by multiple types of attacks. As such, we elected to focus on the following patterns:

(1) An inter-message interval of less than one-half the shortest "normal" interval (i.e., the shortest attack-free interval).
(2) An arbitration identifier that appears thrice in a row (or more).
(3) An arbitration identifier that is invalid (i.e., an arbitration identifier that does not belong to the set of unique arbitration identifiers collected under attack-free conditions).

If an attacker wishes to physically affect the vehicle, then he or she will need to send high-frequency attack messages in order to overwhelm (i.e., drown out) legitimate messages originating from the real ECU. High-frequency messages result in shorter inter-message intervals, which will be detected by the first pattern. The first pattern will also detect attacks in which the attacker varies the arbitration identifier in order to avoid detection. The second pattern is geared toward the detection of replay and spoofing attacks, as messages with the same arbitration ID must be sent repeatedly in order to manipulate the vehicle. The third pattern is intended to detect fuzzing attacks, which generate invalid arbitration IDs. The third pattern would also detect the `000` arbitration IDs common in unsophisticated denial of service (DoS) attacks.

5.2 Logics

Our traces are time-ordered—and timestamped—therefore, to encode our engineering-based IoCs into logical properties, we focus on temporal logic. As the name suggests, temporal logic specifies properties *over time* [14]. Temporal logics include linear temporal logic (LTL) [12], metric temporal logic (MTL) [7], signal temporal logic (STL) [3], real-time temporal logic (RTL) [2], computation tree logic (CTL), and temporal logic of actions (TLA)—to name a few.

Two types of properties are commonly expressed in temporal logic: (1) safety and (2) liveness. A *safety* property specifies that something bad must *not* happen, whereas a *liveness* property asserts that something good *must* happen [14, 37].

In the literature, concurrent programming and deadlock prevention are often cited as applications of temporal logic [4, 29]. In addition, temporal logic can (1) describe the behavior of finite-state systems (e.g., industrial control systems) [14] and (2) monitor cyber-physical systems (CPSs) [4, 5]. The cyber-physical applications of temporal logic are discussed in detail in the *Related Work* section.

In this work, we explore the following logics:

(1) **Linear temporal logic (LTL):** Boolean predicates, discrete time
(2) **Metric temporal logic (MTL):** Boolean predicates, real time
(3) **Signal temporal logic (STL):** Real-value predicates, real time

LTL, MTL, and STL are all expressed over a single computational path or run—whereas CTL (and others) are expressed over a tree of possible executions. Essentially, LTL, MTL, and STL are *linear-time* logics, while CTL (and others) are *branching-time* logics.

Of the aforementioned linear-time logics, LTL is the least expressive; STL is the most expressive. However, expressiveness comes at the cost of computability. Our goal is to select the logic most suited to each IoC. Any finite trace-suitable variant of the above logics would be satisfactory.

For the first IoC, we use STL—or rather, a first-order extension of STL. Specifically, we leverage signal first-order logic (SFO) [3]. To adapt this IoC into SFO, we need to move the time interval into a signal dimension; now, our signal dimensions are (1) timestamp, (2) arbitration ID, (3) data field, and (4) time since the previous message. We also need to establish a baseline for the "normal" time interval; we can determine this baseline from our traces of attack-free CAN traffic.

$$\varphi_1 \equiv \forall\, s, s' : s \neq s' \rightarrow G(|s^t - s'^t| \geq n^t)$$

The s term refers to a distinct sample (i.e., a distinct signal). The t superscript refers to the time since the previous message, and the n^t term refers to the "normal" time interval.

For the second IoC, we can use LTL, as below:

$$\varphi_2 \equiv G\neg(id \wedge Xid \wedge XXid)$$

The id term refers to the arbitration identifier.

For the third IoC, we can again use LTL. We will obtain the set of valid arbitration identifiers from an attack-free CAN traffic trace.

$$\varphi_3 \equiv G(id_1 \vee id_2 \vee id_n)$$

Here, n is the number of valid arbitration identifiers, and each id corresponds to a valid arbitration ID from the set we collected from the attack-free trace.

LTL, MTL, and STL are all temporal logics, which were selected in order to reduce cognitive load. Given that the formulas relevant to detecting certain attacks are quite small, we can—potentially—achieve lightweight monitors. In the case of LTL, for example, our formulas will result in relatively small finite automata, which, when implemented, would yield lightweight monitors, a significant engineering (i.e., computational) advantage. For the resource-constrained environment of an automotive, lightweight monitors are non-negotiable. However, while we can encode MTL formulas as LTL, we elect to use MTL because it provides a more compact representation of the predicate.

6 BENCHMARK

For our benchmark, we initially planned to implement a formal monitor—using the `metric-temporal-logic` [34] Python package. However, we quickly realized that such an implementation would be prohibitively slow: during development and debugging, we used a tiny subset of our data—only 1,000 attack-free messages and 100 attack messages—and the program execution still took several minutes. Therefore, we shifted to an engineering-based implementation for our proof-of-concept benchmark.

Tables 1, 2, 3, and 4 provide the TP, TN, FP, FN metrics for the following combinations of patterns: Patterns #1, #2, and #3; Pattern #1; Pattern #2; and Pattern #3. All patterns were evaluated against all sub-datasets and all testing subsets.

Table 1: TP, TN, FP, FN Metrics - Patterns #1, #2, and #3

Sub-dataset, Testing subset	True Positives	True Negatives	False Positives	False Negatives
#1, #1	34595	5635751	3486	28516
#1, #2	163420	2611190	3672357	577
#1, #3	6473	8613422	917	13772
#1, #4	6742	9484369	3721864	6991
#2, #1	6710	13205750	483	7023
#2, #2	18651	52389	8372869	424
#2, #3	11512	12119170	2044	17222
#2, #4	18002	30753	4741126	0
#3, #1	107031	8411456	1543	52004
#3, #2	115637	1655931	5041439	37425
#3, #3	5024	9427485	2185	12980
#3, #4	142492	1672596	5073424	6694
#4, #1	102585	6742297	3723	46601
#4, #2	51417	86708	17262808	972
#4, #3	56813	6695843	1527	96249
#4, #4	208457	39748	7922459	4441

Table 2: TP, TN, FP, FN Metrics - Pattern #1

Sub-dataset, Testing subset	True Positives	True Negatives	False Positives	False Negatives
#1, #1	0	5638813	424	63111
#1, #2	12	6283072	475	163985
#1, #3	0	8613423	916	20245
#1, #4	2	13205753	480	13731
#2, #1	2	13205753	480	13731
#2, #2	2	8424348	910	19073
#2, #3	1	12120090	1124	28733
#2, #4	0	4771761	118	18002
#3, #1	0	8412999	0	159035
#3, #2	0	6697370	0	153062
#3, #3	0	9429670	0	18004
#3, #4	0	6746020	0	149186
#4, #1	0	6746020	0	149186
#4, #2	0	17349516	0	52389
#4, #3	0	6697370	0	153062
#4, #4	0	7962207	0	212898

Table 3: TP, TN, FP, FN Metrics - Pattern #2

Sub-dataset, Testing subset	True Positives	True Negatives	False Positives	False Negatives
#1, #1	34595	5636175	3062	28516
#1, #2	50898	6278897	4650	113099
#1, #3	80	8614338	1	20165
#1, #4	48	13206230	3	13685
#2, #1	48	13206230	3	13685
#2, #2	788	8425242	16	18287
#2, #3	11511	12120294	920	17223
#2, #4	12122	4771526	353	5880
#3, #1	27000	8411456	1543	132035
#3, #2	17513	6695843	1527	135549
#3, #3	4370	9427485	2185	13634
#3, #4	91789	6742297	3723	57397
#4, #1	91789	6742297	3723	57397
#4, #2	0	17349512	4	52389
#4, #3	17513	6695843	1527	135549
#4, #4	20566	7960889	1318	192332

Table 4: TP, TN, FP, FN Metrics - Pattern #3

Sub-dataset, Testing subset	True Positives	True Negatives	False Positives	False Negatives
#1, #1	0	5639237	0	63111
#1, #2	163420	2611365	3672182	577
#1, #3	6393	8614339	0	13852
#1, #4	6693	9484695	3721538	7040
#2, #1	6661	13206233	0	7072
#2, #2	18651	52396	8372862	424
#2, #3	0	12121214	0	28734
#2, #4	18002	30754	4741125	0
#3, #1	89353	8412999	0	69682
#3, #2	115377	1656043	5041327	37685
#3, #3	654	9429670	0	17350
#3, #4	142462	1672611	5073409	6724
#4, #1	10796	6746020	0	138390
#4, #2	51417	86708	17262808	972
#4, #3	43968	6697370	0	109094
#4, #4	208457	39748	7922459	4441

7 DISCUSSION OF RESULTS

We were surprised to observe a significant number of false positives when evaluating Pattern #1 (i.e., the inter-message interval is too short), Pattern #2 (i.e., the arbitration identifier repeats too often), and—especially—Pattern #3 (i.e., the arbitration identifier is

invalid). For Pattern #2 in particular, the false positives were relatively low compared to the true positives; even so, it was more than we had expected—and more than would be tolerable in a real-world setting. However, upon manual review, we realized that many of the false alarms were occurring because a legitimate message was sandwiched between two or more attack messages. Naturally, the legitimate message would have a very short inter-message interval; therefore, it would be flagged—erroneously—as an attack message by Pattern #1. Similarly, when a legitimate message appears between several attack messages that share its arbitration identifier (i.e., during a spoofing attack or a masquerade attack), then the legitimate message would become part of a pattern of repeated arbitration identifiers, which would be detected by Pattern #2. Essentially, our IoCs are correctly detecting an attack, but they are misidentifying legitimate messages as attack messages. Pattern #3 never generated false positives when the training vehicle and the testing vehicle were the same. However, when the testing vehicle was a different vehicle—especially a vehicle produced by a different manufacturer—then there would be a significant volume of false positives. Naturally, Pattern #3, which captures the valid arbitration identifiers for a specific vehicle (and raises the alarm if it encounters an invalid arbitration identifier for that specific vehicle), would need to be re-trained for each new vehicle. All in all, we believe that our technique is effective. We will continue to explore techniques that can help us to further reduce false positives.

One of the advantages of our IoC-based IDS is that it requires very little “normal" training data—just enough to determine what constitutes the minimum “normal" time interval, and which unique arbitration identifiers are valid. It requires no attack data whatsoever, and it can still detect many types of attacks. Training and testing are rapid; moreover, training can be expedited by reducing the quantity of “normal" traffic that is used in training. Of course, we would still need enough “normal" data to select an appropriate “normal" time interval and ensure that we have accounted for all the unique arbitration IDs. Future work might involve optimizing the volume of “normal" traffic used to “train" each pattern—i.e., determine the appropriate parameters for each pattern.

Compared to previous work based on the `can-train-and-test` dataset [19, 21] dataset, our approach is much faster. We have achieved better metrics—in terms of accuracy, precision, recall, and F1-score—than many of the machine learning-based IDSs. In particular, with Patterns #1 and #2, we have achieved a much lower false positive rate than many of the machine learning-based IDSs actually capable of detecting attacks (we exclude machine learning-based IDSs that are completely biased to the “negative" label and never output the “positive label).

That said, our technique does result in false negatives, and, in a real-world scenario, there is a chance that our technique might fail to detect a particularly clever or sophisticated attack. Nevertheless, our IoC-based IDS adds a lot of security (compared to the minimal—even nonexistent—security of today’s automobiles) without sacrificing practicability. We hope that the practicability of our approach will encourage adoption—both by consumers and by automotive manufacturers.

8 LIMITATIONS & FUTURE WORK

Formal Monitoring. We have encoded our engineering-based IoCs into formal temporal logic. Next, we plan to construct a formal monitor capable of detecting anomalies in CAN packet traces. Our attempt to leverage the `metric-temporal-logic` [34] Python package was unsuccessful, as it was much too slow to be practicable. Therefore, we plan to investigate alternate Python packages as well as alternate software platforms.

Online Monitoring. Once we have successfully developed a formal offline monitor, we plan to adapt the monitor from the *offline, packet trace* scenario to the *online, live vehicle* scenario. Offline monitors are finite, whereas online monitors are infinite. As such, we will need to investigate potential issues with decidability when we transition from finite to infinite. Ultimately, our goal is to develop an online monitor that can be plugged into a vehicle’s on-board diagnostic port, as shown in Figure 3.

Mitigation. When implemented, the monitor should function in an anticipatory fashion; that is, it should detect attacks as early as possible to facilitate mitigation. The longer the attack continues (i.e., the more messages the attacker is able to deliver), the more damage—to the vehicle and others—might occur. For example, one spoofed “reverse" message will not overwhelm all the valid “drive" messages. Therefore, if we can stop the attack before the vehicle’s transmission is tricked into shifting into “reverse," then we can prevent damage.

The mitigation is simple, immediate, and effective, but, to understand it, we have to understand the CAN protocol. The controller area network is a dual-wire differential protocol. In essence, the voltage difference between `CAN HIGH` and `CAN LOW` enables communication. Thus, if we connect `CAN HIGH` either to `CAN LOW` or to the ground (i.e., if we short `CAN HIGH`), then the difference in voltage will disappear, and no further communication can occur. When this happens the electronic control units (ECUs) will fall back to fail-safe defaults, and the attack will be thwarted. However, this is a rather extreme response, so it is absolutely critical to minimize false positives before implementing this mitigation technique.

9 CONCLUSION

In this paper, we investigated indicators of compromise (IoCs) in CAN attack traffic. Specifically, we curated IoCs that should never appear in attack-free traffic; as such, hypothetically, they should never produce false positives. We justified our IoCs with snippets of CAN traffic captures. Next, we adapted our curated IoCs to formal specifications—with the ultimate goal of constructing a formal monitor. To formalize our engineering-based IoCs, we explored linear temporal logic (LTL), metric temporal logic (MTL), and signal temporal logic (STL)—or, more precisely, a first order extension of STL. We made the conjecture that many IoCs could be codified as LTL, MTL, and/or STL properties. Finally, we implemented our IoCs in Python and conducted a benchmark to assess the practicability of our IoCs—particularly in terms of false positives. We were surprised to discover that all of our IoCs produced non-zero false positives when pitted against some sub-datasets and testing subsets. However, upon manual inspection, we realized that many of the “false positives" were legitimate messages that appeared in between attack messages. That is to say, many of our false positives

occurred during attack conditions, even if the messages that were flagged were not actually attack messages. In addition, some false positives occurred because the “normal" behavior of the original training vehicle did not match the “normal" behavior of a different testing vehicle (different manufacturer and/or model). In our future work, we plan to develop a lightweight formal monitor to replace our engineering-based implementation. The formal monitor will ultimately be converted from an offline to an online monitor and will be installed in a real vehicle.

ACKNOWLEDGMENTS

We would like to thank our test drivers, especially Christian Lampe, Julie Lampe, and Wayne & Vickie Olsen.

REFERENCES

[1] Environmental Protection Agency. 1999. Control of Air Pollution From New Motor Vehicles; Compliance Programs for New Light-Duty Vehicles and Light-Duty Trucks. *National Archives* (1999). https://www.federalregister.gov/documents/1999/05/04/99-9062/control-of-air-pollution-from-new-motor-vehicles-compliance-programs-for-new-light-duty-vehicles-and

[2] Rajeev Alur and Thomas A. Henzinger. 1992. Logics and Models of Real Time: A Survey. *Cornell University* (1992). https://ecommons.cornell.edu/handle/1813/7102

[3] Alexey Bakhirkin, Thomas Ferrère, Thomas A. Henzinger, and Deian Ničkovićl. 2018. The First-Order Logic of Signals. *2018 International Conference on Embedded Software (EMSOFT)* (2018). https://ieeexplore.ieee.org/document/8537203

[4] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. 2018. Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications.. In *Lectures on Runtime Verification: Introductory and Advanced Topics* (2018), Ezio Bartocci and Yliès Falcone (Eds.). Springer International Publishing, 135 – 175. https://link.springer.com/chapter/10.1007/978-3-319-75632-5_5

[5] Borzoo Bonakdarpour, Jyotirmoy V. Deshmukh, and Miroslav Pajic. 2018. Opportunities and Challenges in Monitoring Cyber-Physical Systems Security.. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice* (2018-10-30), Tiziana Margaria and Bernhard Steffen (Eds.), Vol. 11247. Springer International Publishing, 9 – 18. https://link.springer.com/chapter/10.1007/978-3-030-03427-6_2

[6] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. 2020. Evaluation of CAN Bus Security Challenges. *Sensors* 20 (2020). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7219335/

[7] Christoph Brzoska. 1995. Temporal logic programming with metric and past operators.. In *Executable Modal and Temporal Logics* (1995), Michael Fisher and Richard Owens (Eds.), Vol. 897. Springer Berlin Heidelberg, 21 – 39. https://link.springer.com/chapter/10.1007/978-3-319-75632-5_5

[8] Tri P. Doan and Subramaniam Ganesan. 2017. CAN Crypto FPGA Chip to Secure Data Transmitted Through CAN FD Bus Using AES-128 and SHA-1 Algorithms with A Symmetric Key. *SAE International* (2017). https://saemobilus.sae.org/content/2017-01-1612/

[9] United Nations Economic Commission for Europe (UNECE). 2021. Three landmark UN vehicle regulations enter into force. *United Nations Economic Commission for Europe (UNECE)* (2021). https://unece.org/sustainable-development/press/three-landmark-un-vehicle-regulations-enter-force

[10] United Nations Economic Commission for Europe (UNECE). 2021. UN Regulation No. 155 - Cyber security and cyber security management system. *United Nations Economic Commission for Europe (UNECE)* (2021). https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security

[11] Ian Foster and Karl Koscher. 2015. Exploring Controller Area Networks. *;login:* 40 (2015), 6 – 10. https://www.usenix.org/system/files/login/articles/login_dec15_02_foster.pdf

[12] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces.. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI '13)*. AAAI Press, 854 – 860. https://www.ijcai.org/Proceedings/13/Papers/132.pdf

[13] Robert Bosch GmbH. 2023. CAN XL: Next step in CAN evolution. *Bosch semiconductors for Automotive* (2023). https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/

[14] Reinhard Gotzhein. 1992. Temporal logic and applications—a tutorial. *Computer Networks and ISDN Systems* 24, 3 (1992), 203–218. https://doi.org/10.1016/0169-7552(92)90109-4

[15] Zhiyuan Huang, Weiyao Lan, and Xiao Yu. 2023. A Formal Control Framework of Autonomous Vehicle for Signal Temporal Logic Tasks and Obstacle Avoidance. *IEEE Transactions on Intelligent Vehicles* (2023), 1 – 10. https://ieeexplore.ieee.org/document/10144389

[16] CAN in Automation (CiA). 2023. CAN FD - The basic idea. *CAN in Automation (CiA)* (2023). https://www.can-cia.org/can-knowledge/can/can-fd/

[17] Austin Jones, Zhaodan Kong, and Calin Belta. 2014. Anomaly detection in cyber-physical systems: A formal methods approach. *53rd IEEE Conference on Decision and Control* (2014). https://ieeexplore.ieee.org/document/7039487

[18] Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno Shwetak Patel, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. 2010. Experimental Security Analysis of a Modern Automobile. *IEEE Symposium on Security and Privacy* (2010). http://www.autosec.org/pubs/cars-oakland2010.pdf

[19] Brooke Lampe. 2023. can-train-and-test. *Bitbucket* (2023). https://bitbucket.org/brooke-lampe/can-train-and-test/src/master/

[20] Brooke Lampe and Weizhi Meng. 2022. IDS for CAN: A Practical Intrusion Detection System for CAN Bus Security. *2022 IEEE Global Communications Conference (GLOBECOM 2022)* (2022). https://ieeexplore.ieee.org/document/10001536

[21] Brooke Lampe and Weizhi Meng. 2023. can-train-and-test: A Curated CAN Dataset for Automotive Intrusion Detection. *arXiv* (2023). arXiv:2308.04972 [cs.CR] https://arxiv.org/pdf/2308.04972.pdf

[22] Brooke Lampe and Weizhi Meng. 2023. A survey of deep learning-based intrusion detection in automotive applications. *Expert Systems with Applications* 221 (2023), 119771. https://doi.org/10.1016/j.eswa.2023.119771

[23] Djones Lettnin, Pradeep K. Nalla, Jurgen Ruf, Thomas Kropf, Wolfgang Rosenstiel, Tobias Kirsten, Volker Schonknecht, and Stephan Reitemeyer. 2008. Verification of Temporal Properties in Automotive Embedded Software. *2008 Design, Automation and Test in Europe* (2008). https://ieeexplore.ieee.org/document/4484680

[24] Aarian Marshall. 2023. A Fight Over the Right to Repair Cars Takes a Wild Turn. *Wired* (2023). https://www.wired.com/story/right-to-repair-cars-hackers/

[25] Charlie Miller and Chris Valasek. 2015. Remote Exploitation of an Unaltered Passenger Vehicle. *IOActive* (2015). https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Car_Hacking-1.pdf

[26] Charlie Miller and Chris Valasek. 2016. CAN Message Injection. *Illmatics* (2016). https://illmatics.com/can%20message%20injection.pdf

[27] Mohomad Mokhadder, Mark Zachos, and John Potter. 2023. Evaluation of Vehicle System Performance of an SAE J1939-91C Network Security Implementation.. In *WCX SAE World Congress Experience* (2023-04-11). SAE International. https://doi.org/10.4271/2023-01-0041

[28] Prasad Naldurg, Koushik Sen, and Prasanna Thati. 2004. A Temporal Logic Based Framework for Intrusion Detection.. In *Formal Techniques for Networked and Distributed Systems – FORTE 2004* (2004), David de Frutos-Escrig and Manuel Núñez (Eds.), Vol. 11247. Springer Berlin Heidelberg, 359 – 376. https://link.springer.com/chapter/10.1007/978-3-030-03427-6_2

[29] Amir Pnueli. 1977. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (1977). https://ieeexplore.ieee.org/document/4567924

[30] Elaine S. Povich. 2023. The latest 'right to repair' law is the broadest one yet. *Omaha World Herald* (2023). https://omaha.com/news/the-latest-right-to-repair-law-is-the-broadest-one-yet/article_bb7204ad-9fca-5685-8568-c5acfd8338fb.html

[31] David Shepardson. 2023. US tells automakers not to comply with Massachusetts vehicle data law. *Reuters* (2023). https://www.reuters.com/business/autos-transportation/us-tells-automakers-not-comply-with-massachusetts-vehicle-data-law-2023-06-13/

[32] Ali Shuja Siddiqui, Yutian Gui, Jim Plusquellic, and Fareena Saqib. 2017. Secure communication over CANBus. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2017). https://ieeexplore.ieee.org/document/8053160

[33] Hiroshi Ueda, Ryo Kurachi, Hiroaki Takada, Tomohiro Mizutani, Masayuki Inoue, and Satoshi Horihata. 2015. Security Authentication System for In-Vehicle Network. *Sei Technical Review* (2015). https://global-sei.com/technology/tr/bn81/pdf/81-01.pdf

[34] Marcell Vazquez-Chanlatte. 2019. mvcisback/py-metric-temporal-logic: v0.1.1. *py-metric-temporal-logic* (2019). https://doi.org/10.5281/zenodo.2548862

[35] Qiyan Wang and Sanjay Sawhney. 2014. VeCure: A practical security framework to protect the CAN bus of vehicles. *2014 International Conference on the Internet of Things (IOT)* (2014). https://ieeexplore.ieee.org/document/7030108

[36] Meng Wu, Jingbo Wang, Jyotirmoy Deshmukh, and Chao Wang. 2019. Shield Synthesis for Real: Enforcing Safety in Cyber-Physical Systems. *2019 Formal Methods in Computer Aided Design (FMCAD)* (2019). https://ieeexplore.ieee.org/document/8894264

[37] C.-F. Yu and V.D. Gligor. 1990. A specification and verification method for preventing denial of service. *IEEE Transactions on Software Engineering* 16 (1990), 581 – 592. https://ieeexplore.ieee.org/document/55087