Guaranteeing Privacy-Observing Data Exchange

Probst, Christian W.

Published in:
Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation - Foundational Techniques (ISoLA 2016)

Link to article, DOI:
10.1007/978-3-319-47166-2_66

Publication date:
2016

Document Version
Peer reviewed version

Guaranteeing Privacy-observing Data Exchange

Christian W. Probst

Technical University of Denmark
cwpr@dtu.dk

Abstract. Privacy is a major concern in large of parts of the world when exchanging information. Ideally, we would like to be able to have fine-grained control about how information that we deem sensitive can be propagated and used. While privacy policy languages exist, it is not possible to control whether the entity that receives data is living up to its own policy specification. In this work we present our initial work on an approach that empowers data owners to specify their privacy preferences, and data consumers to specify their data needs. Using a static analysis of the two specifications, our approach then finds a communication scheme that complies with these preferences and needs. While applicable to online transactions, the same techniques can be used in development of IT systems dealing with sensitive data. To the best of our knowledge, no existing privacy policy languages supports negotiation of policies, but only yes/no answers. We also discuss how the same approach can be used to identify a qualitative level of sharing, where data may be shared according to, *e.g.*, the level of trust to another entity.

1 Introduction

Privacy is a major concern in large of parts of the world when exchanging information. While we do not have control over, how our data is used in the real world, we would ideally like to be able to have that kind of fine-grained control about how information that we deem sensitive can be propagated and used in the cyber world. Privacy policy languages have been developed to this end, but it is not possible to control whether the entity that receives data is living up to its own policy specification. Techniques such as proof-carrying code enable servers to assure of properties of client code, but not the other way around.

In this work we present an approach that empowers data owners to specify their privacy preferences, and data consumers to specify their data needs. Our approach then finds a communication scheme that complies with these preferences and needs. Instead of relying on the data consumer to obey the data owner's preferences, this approach does only enable interactions that guarantee that the data is treated accordingly.

Using our approach, data owners specify whom they trust, and which items they want to share, with whom, and at which quality. If no sharing is specified for a specific entity, the trust hierarchy is queried, otherwise, sharing is prohibited. Data consumers specify their own trust hierarchy and the data they request

at which quality. The quality of data is an important feature of our approach; it describes not only sharing/not sharing, but also shades in between these extremes. A picture, for example, can be shared in many different shades of quality, making it more or less useful to the recipient.

Based on the specifications, a resolution engine tries to identify a series of interactions that result in the exchange of the requested data, either directly, or at least as "functional" sharing. Functional sharing results in the data consumer obtaining the same information, just not the underlying data. For a credit card, for example, it would be possible to verify payments, but not to obtain the actual card number.

To the best of our knowledge, no existing privacy policy languages (PPLs) support this kind of negotiation of policies. Policy matching usually only checks that the data consumer's intentions of data usage and obligations are compliant with the data owner's preferences. While most PPLs are very powerful, we believe that our approach can be added to most of them to introduce policy negotiations; we are currently working on developing such an extension.

The remainder of this article is structured as follows. After a brief discussion of privacy policy languages in the next section, we present an overview of our framework in Section 3, followed by our approach to specifying data sharing preferences and data needs in (Section 4). These specifications are the input to the generation of joint strategies described in Section 5, followed in Section 5.2 by some considerations about the quality of shared data. Finally, we discuss the application of our approach to software system development in Section 6 and conclude the paper in Section 7 with an outlook on future work.

2 Privacy Policy Languages

Privacy policy languages (PPLs) [1] aim at representing an entity's policies in a computer-readable format, especially to make them available for policy enforcement. These languages exist both for data owners and data consumers, so that they can be employed to check whether a given web site lives up to a users privacy preferences or not. PPLs come in many different formats and with differing features, which makes them hard to compare [2].

The World Wide Web Consortium's (W3C) Platform for Privacy Preferences (P3P) aimed at representing websites' privacy policies in machine-readable format [3, 4] in the P3P Preference Exchange Language (APPEL) [5]. Similar approaches exist for business-to-business communication [6], organisations' privacy concerns [7], and more generally access control languages [8]. Recent developments include the PrimeLife Privacy Language [1, 9] and the Accountability Policy Language (A-PPL) [10].

To the best of our knowledge, neither of these languages supports negotiation of policies, but only yes/no answers. The PrimeLife Privacy Language [1,9], *e.g.*, supports policy matching, but this matching does only check that the data consumer's intentions of data usage and obligations are compliant with the data owner's preferences. While most PPLs are very powerful, we believe that our

approach can be added to most of them; we are currently working on developing such an extension.

3 Guaranteeing Observance of Privacy Specifications

In this section we describe the overall framework for our approach. The individual components will be described in more detail in the next sections.

The goal of the framework (Figure 1) is to ensure that data owner and data consumer agree on a communication protocol that guarantees that the data owner's privacy preferences are observed, and that the data consumer's need for data is fulfilled. The main observation is that in many scenarios data consumers require a functional property of the data, not necessarily the data itself. For credit cards, for example, the essential information is not the credit card number, but the authorization for a payment.

Our framework consists of the data owner, the data consumer, and potentially a number of other components that are used, *e.g.*, for replacing data exchange with authorization exchange as in the example described above. Figure 1 shows the overall structure and the involved phases in our framework:

1. The data owner starts with requesting a service from a data consumer.
2. The data consumer requests in turn some data from the data owner, *e.g.*, credit card data or address.
3. The data owner replies with a privacy policy specification for the data requested. The data consumer combines its data needs with the data owner's privacy policy specification to compute a least upper bound, which observes the privacy specification and the data needs.
4. The data consumer sends back the protocol identified by the policy engine. The data owner checks that the protocol observes the privacy policy specification and initiates the protocol.

In each step any of the two parties can either cancel the communication if the step fails, *e.g.*, if the data owner does not want to share the requested data or if no acceptable protocol can be found, or can shortcut the framework, *e.g.*, if the data owner has no restrictions on sharing the requested data.

The essential component in our framework is the resolution engine, which takes the privacy policy specification from the data owner and the data needs specification from the data consumer, and generates a protocol that guarantees that the data owner's privacy preferences are observed, and that the data consumer's need for data is fulfilled. For credit card data this could result in using a trusted third party such as the credit card issuer.

The resolution engine is obviously part of the trusted code base; both the data owner and the data consumer, as well as other entities participating in the communication, must trust the engine. However, just as with proof-carrying code, each entity can check that the protocol identified by the resolution engine obeys the entity's preferences.

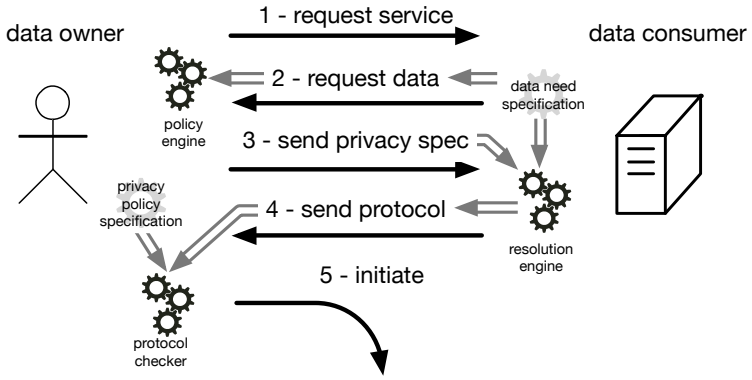

Fig. 1. Framework for guaranteeing observance of privacy specifications. The dark arrows show the flow of control, the gray arrows the flow of information.

Another possibility in our approach is to allow *shaded* sharing of information, where data is shared fully with certain data consumers, not shared with other data consumers, and possibly in shades with others. An example application of shaded sharing are pictures, which a data owner might want to share in high resolution with family members, not at all with unknown entities, and in a low resolution with not so close acquaintances.

Before initiating the protocol from the resolution engine, the all entities can check whether the protocol proposed by the resolution engine actually complies with its privacy policy specification. Similar to proof carrying code [11], checking compliance of a protocol with preferences is easy. This step guarantees that data is only shared with parties approved by the data consumer, in a shading according to the privacy policy.

4 Specifying Preferences and Needs

In this section we describe the specification of privacy preferences and data needs by the data owner and the data consumer, respectively. These specifications are the input to the resolution engine, which generates a joint strategy that guarantees that the data owner's privacy preferences are observed, and that the data consumer's need for data is fulfilled.

In our current work, the specification of the data owner's privacy preferences consists of the user's data items, a hierarchy of the user's trusted entities, and a mapping from entities and items to shade of sharing. The data consumer's data needs are currently a pair of required data item and minimum level of shading, and a hierarchy of trusted entities. In all cases specific items and users can be abstracted by classes, *e.g.*, a specific image could be replaced by the class "jpg file" or "image".

4.1 Data Owners' Privacy Preferences

In its privacy preferences, the data owner can specify, whom to share which data with in which quality. This specification is split up in three parts:

- A definition of the data owner's data items or classes of such items;
- A hierarchy of the data owner's trusted entities; and
- A mapping from entities and items to the level of shade when sharing the item with the entity.

The *items* are represented as elements of a set of strings that represents data and classes of data. As explained above, data can stand for any kind of information at the data owner, and the class names are assumed to come from an ontology that is shared between data users and data consumers.

The *hierarchy of trusted entities* is a directed, acyclic graph with the nodes representing entities and the directed edges representing the trust hierarchy. An edge (a, b) represents that b is more trusted than a, and that everything that will be shared with a also will be shared with b. The set of entities is extended with two special elements $\bot, \top \notin \mathit{Entities}$ that represent the untrusted or unknown entity ($\bot$) or the completely trusted entity ($\top$).

Finally, for each item and entity, the mapping *share* returns the shade of sharing for this item with this entity as a number between 0 and 1, with the

$$\mathit{Items} \quad \subseteq \mathit{Data} \cup \mathit{Classes}$$

$$\mathit{Trust} \quad := (N = \mathit{Entities} \cup \{\bot, \top\}, E = \{(s,t) \in N \cup \{\bot\} \times N \cup \{\top\} | s \neq t\} \wedge$$

$$\nexists 0 \leq k : \{(s_i, t_i) \in E, 1 \leq i \leq k | \forall 1 \leq j \leq k : s_j = t_{j-1} \wedge s_0 = t_k\}$$

$$\mathit{share} : \mathit{Items} \times \mathit{Entities} \rightarrow [0,1]$$

$$\mathit{share}(i,e) := \begin{cases} 1 & \text{, if item } i \text{ is shared with } e \text{ without restrictions} \\ 0 < x < 1 & \text{, if item } i \text{ is shared with } e \text{ at level } x \\ 0 & \text{, if item } i \text{ is not shared with } e \end{cases}$$

Table 1. Specification of data owners' privacy preferences. The trust component is a directed acyclic graph between nodes representing entities or the untrusted ($\bot$) or completely trusted ($\top$) entities. The sharing level is specified as a number between 0 and 1, with the extremes representing no sharing (0) or unconstrained sharing (1), respectively. The semantics of the values in between depends on the kind of item; for images it might represent the quality of the file shared.

extremes represent no sharing or unconstrained sharing, respectively. The semantics of the values in between depends on the kind of item; for images it might represent the quality of the file shared, for credit card data it may be undefined or require rounding, assuming that this data is shared or not.

4.2 Data Consumers' Data Needs

The data consumer specifies in its data needs, which data in which quality it requires to perform an operation. Like the privacy policy specification of the data owner, this specification is split up in three parts:

- A definition of the data consumer's data items or classes of such items;
- A hierarchy of the data consumer's trusted entities; and
- A mapping from items to the required minium level of shade when the item is shared by an entity.

The specifications of *items* and *trust* are identical for the data consumers' and data owners' specifications.

Finally, for each item, the mapping *need* returns the minium required shade of sharing for this item as a number between 0 and 1. The meaning of the extremal values are optional element (0) and mandatory element (1). As discussed for the sharing specification, the semantics of the values in between depends on the kind of item; all values represent a minimum quality of the shared data.

$$Items \subseteq Data \cup Classes$$

$$Trust := (N = Entities \cup \{\bot, \top\}, E = \{(s,t) \in N \cup \{\bot\} \times N \cup \{\top\} | s \neq t\} \wedge$$
$$\nexists 0 \leq k : \{(s_i, t_i) \in E, 1 \leq i \leq k | \forall 1 \leq j \leq k : s_j = t_{j-1} \wedge s_0 = t_k\}$$

$$need : Items \rightarrow [0..1]$$

$$need(i) := \begin{cases} 1 & \text{, if item } i \text{ is required} \\ 0 < x < 1 & \text{, if item } i \text{ is only required at level } x \\ 0 & \text{, if item } i \text{ is optional} \end{cases}$$

Table 2. Specification of data consumers' data needs. The items and trust components are shared with data owners.The *need* component specifies the required level of sharing as a number between 0 and 1, with the extremes representing optional and required elements. As before, the semantics of the values in between depends on the kind of item.

data consumer	data owner
$need = \{(creditcard, 1)\}$	$share(shop, creditcard) = 1$

Table 3. Example specification for sharing of credit card information. Since the data owner is willing to share the information with the data consumer, this will result in direct communication.

data consumer	data owner
$need = \{(creditcard, 1)\}$	$share(shop, creditcard) = 0$
$trust = \{(\bot, bank), (bank, \top)\}$	$share(bank, creditcard) = 1$
	$trust = \{(\bot, bank), (bank, \top)\}$

Table 4. Example specification for sharing of credit card information. In this case, the data owner is not willing to share the information with the data consumer, but with the bank, which is trusted by both parties.

The specification of *need* can easily be extended to take other factors into account, *e.g.*, the entity sharing the data to enable personalized requirements, or a time factor to make the required quality dependent on the time since, *e.g.*, the last authentification of the entity.

5 Generating a Joint Strategy

As mentioned in the previous section, the resolution engine is at the centre of our approach (Figure 1), as it guarantees the consolidation of the data owner's privacy policy specification and the data consumer's data needs. The main goal of the resolution engine is to map direct data sharing that is not permitted by the data owner's privacy policy to indirect data sharing, *e.g.*, with a third party. In general, any entity participating in a data exchange can trigger the protocol negotiation, *e.g.*, this could also be the data consumer.

The resolution engine takes a data consumer's data need specification and a data provider's privacy policy specification, and translates them into a constraint graph that contains entities as nodes and items as data. Table 3 and Table 4 illustrate this in two cases for the credit card example. The data need specification states that the shop requires the credit card data from the customer. In the first specification, the data owner is explicitly mentioning the shop and is willing to share the credit card information. However, the privacy policy in the second case denies this sharing.

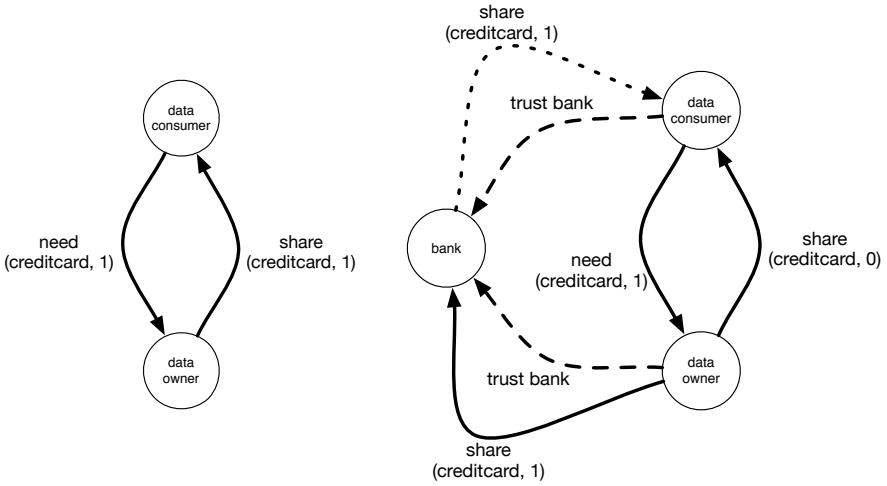

Fig. 2. Resulting graphs for the example specifications from Table 3 and Table 4. In the left hand graph, the data owner shares the credit card data, so the resolution engine finds a circle that fulfils the data consumer's data need. In the right hand graph, the data owner does not share the credit card data. As a result, the resolution engine adds the bank node, since it is trusted by both the data consumer and owner, the edge of sharing the credit card, and eventually the edge back to the data consumer, indicating that here, only "functional" sharing is possible. The labels on the arrows indicate, which part of the privacy preference specification caused this edge to be added.

The resolution engine works lazily from the node representing the data consumer. Based on the data need, the entity from which the data is requested is added together with an edge. The resolution engine then searches for direct or indirect edges from the data owner to the data consumer. In the resulting graph, the resolution engine identifies all cycles, and computes their sharing sum, where edges from the data consumer count positive as obligations for the data owner, and edges from the data owner count negative as fulfillments of obligations.

Figure 2 shows the two graphs resulting from the example specifications in Table 3 and Table 4. The left hand graph represents the example where the data owner is willing to share the credit card information, while the right hand graph represents the result of several iterations. In the first iteration, the sum of the circle is not 0, since the data owner does not want to share the credit card information with the data consumer. As a result, the resolution engine adds the bank node, which is trusted by both parties. In the general case, this step may add several nodes, that lead to a node that transitively is shared by both parties. Since the data owner is willing to share the credit card data with the bank, this edge is added, leading to a situation where an edge is missing from the bank to the data consumer. Since the data consumer trusts the bank, the resolution

engine adds a "functional" sharing edge from the bank to the data consumer, completing the circle and the sharing.

5.1 Properties of the Generated Protocols

The generated protocol is by construction acceptable by the data consumer and data owner (if a protocol is found). This fact is easily established by the translation mechanism from specifications to graph, and easily checked by the two parties once the protocol is shared.

If the privacy preferences of data consumer and data owner are compatible, then a direct communication will be generated and approved by both parties. Compatible preferences are those where either the unshaded exchange is permitted and requested, or where the data owner is willing to share data at a level that is larger or equal to the level requested by the data consumer.

If the two preferences are not compatible, then the resolution engine will attempt to find parties that can provide the functional properties of the data; in the example above, this would, *e.g.*, be the information that there are sufficient funds on the account.

In the generated protocol, direct sharing is translated to communication of the data in question. Functional sharing can be translated in patterns that keep the data in question private, in the case of the bank this could, *e.g.*, be an authorization system, where the customer receives a token from the bank, forwards it to the data consumer, who uses it to check with the bank that the requested amount is available.

5.2 Quality of Data Sharing

An important property of our approach is the ability to specify the quality of the data shared, or the shade of sharing. As discussed above, possible values are from $[0, 1]$, and range from no sharing, represented by 0, to complete or untampered sharing, represented by 1. The values between 0 and 1 do not have a predefined meaning beyond that an increase in sharing is represented by an increased value.

The meaning of values is first defined in relation to the kind of item being shared. For images, it might be the quality or the size of the shared picture, such that high shades result in close to perfect pictures, and low shades result in distorted versions of the picture. For data, often only 0 and 1 may make sense, even though values in between might represent that only part of the data is shared or requested, like for example in the case of credit card numbers, where only the last 4 numbers may be requested. The usefulness of this information depends again on the kind of data shared.

6 Guaranteeing Privacy in Software Systems

While our approach considers items and communicating parties in, *e.g.*, e-commerce systems, the approach fits equally well the development of software systems and guaranteeing privacy in the resulting system. When considering privacy

in software systems, the data items in the discussion above are translated to the data the system is working on. The data consumer and data owner are, *e.g.*, translated to caller and callee, or data store and computation method.

Consider, for example, a robust versus a fragile implementation of a queue [12]. The fragile version would share a direct pointer to the information stored in the queue, giving the consumer of the pointer direct access to the data. The robust version, on the other hand, would select a token from a large token space, and would internally map the token to the real address of the data item in question. In principle, this is exactly the same situation as the credit card scenario described above: the pointer to the information is the credit card, the data owner is the queue algorithm, and the data consumer is a method that creates a queue. Since the pointer gives direct access to the information stored, the data owner should not want to share this information; instead, a hash map can be used to hide the concret information.

The same holds for limitting the risk that parts of an application pose towards properties of data such as availability or consistency. The data designer attaches negated criticality values to the data, and data sinks are annotated with the maximally acceptable level of criticality at each sink. If in the application there is a data flow from a highly critical piece of data, *e.g.*, of .9 translated to .1, to a sink with a lower level, *e.g.*, .5, then flow would trigger an alert.

By specifying, which level of such properties is required by parts of an application,

We are currently working on identifying several such analogies between data privacy and ensuring privacy in software systems. Patterns such as the token that hides the concrete information can be generated automatically just like the functional sharing of credit card data. Also here we do not really require the concrete address of the data, but only a handle to obtain the information stored at the address.

7 Conclusion

In a world that increasingly depends on automated software systems that handle large parts of our sensitive data, we would like to have fine-grained control about how our sensitive information is handled, and where it might end up. Privacy policy languages cannot enforce that data recipients actually use the data in the way they have specified.

We have presented an approach that does not overcome this limitation, but that empowers data consumers and data owners to specify what they need and what they want to share; our framework then finds a protocol for exchange of this information that obeys the data owner's privacy policy specification, and fulfils the data consumer's data needs.

Using our approach, data owners specify whom they trust, and which items they want to share, with whom, and at which quality. If no sharing is specified for a specific entity, the trust hierarchy is queried, otherwise, sharing is prohibited. Data consumers specify their own trust hierarchy and the data they request at

which quality. A resolution engine uses these specifications to compute a protocol that filfils all specified requirements and constraints. Currently, the generated protocol exchanges data either correctly (or in some shade), or as functional equivalent, where the data consumer does not get access to the data but only to its functionality – functional sharing results in the data consumer obtaining the same information, just not the underlying data. For a credit card, for example, it would be possible to verify payments, but not to obtain the actual card number. For email addresses, it would be possible to send the email through a relay server, but not directly.

7.1 Future Work

We have started to explore the quality of shared data. Instead of sharing/not sharing, parties in our approach can specify more fine grained how the want to share data with whom. We are currently working on modelling more systems for shades of sharing, also to develop a hierarchy of different shades to be applied. We also look at extending the specification of needs, for example, to add situational dependencies based on time or actor, and to add functional properties to share, not just the data itself. Last but not least we are currently looking at a re-implementation of our system in JIF [13], to compare our approach with information flow.

Acknowledgment

Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TRE_{S}PASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.

References

1. The PrimeLife Consortium: Policy Languages. (2011) Available at `http://primelife.ercim.eu/images/stories/primer/policylanguage-plb.pdf`. Last visited May 2016.
2. Kumaraguru, P., Cranor, L., Lobo, J., Calo, S.: A survey of privacy policy languages. In: Proceedings of the Workshop on Usable IT Security Management (USM '07) at Symposium On Usable Privacy and Security '07. (2007)
3. Cranor, L.F.: Web Privacy with P3P. O'Reilly (2002)
4. Cranor, L.F., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.: The platform for privacy preferences 1.0. Available at `http://www.w3.org/TR/P3P`. Last visited May 2016. (2002)
5. Cranor, L.F., Langheinrich, M., Marchiori, M.: A p3p preference exchange language 1.0. Available at `http://www.w3.org/TR/P3P-preferences`. Last visited May 2016. (2002)

6. Bohrer, K., Holland, B.: Customer profile exchange (cpexchange) specification, v1.0. Technical report, International Digital Enterprise Alliance, Inc. (2000) Available at http://xml.coverpages.org/cpexchangev1_0F.pdf. Last visited May 2016.
7. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language, v1.2. Technical report, IBM (2003) Available at http://www.zurich.ibm.com/security/enterprise-privacy/epal. Last visited May 2016.
8. Rissanen, E.: eXtensible Access Control Markup Language (XACML), v3.0. Technical report, OASIS standard (2013) Available at http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. Last visited May 2016.
9. Trabelsi, S., Neven, G., Raggett, D.: Primelife privacy policy langauge (ppl) and engine – report on design and implementation. Technical Report D5.3.4, The PrimeLife Consortium (2011) Available at http://primelife.ercim.eu/images/stories/deliverables/d5.3.4-report_on_design_and_implementation-public.pdf. Last visited May 2016.
10. Azraou, M., Elkhiyaoui, K., Önen, M., Bernsmed, K., de Oliveira, A.S., Sendor, J.: A-PPL: An Accountability Policy Language. Technical Report RR-14-294, EURECOM (2014)
11. Necula, G.C.: Proof-carrying code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '97, New York, NY, USA, ACM (1997) 106–119
12. Bishop, M.: Robust programming Available at http://nob.cs.ucdavis.edu/bishop/secprog/robust.html. Last visited May 2016.
13. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: Proceedings of the sixteenth ACM symposium on Operating systems principles. SOSP '97, New York, NY, USA, ACM (1997) 129–142