Privacy Assessment using Static Taint Analysis (Tool Paper)

Marcel von Maltitz, Cornelius Diekmann, and Georg Carle

Technische Universität München
{vonmaltitz,diekmann,carle}@net.in.tum.de

Abstract. When developing and maintaining distributed systems, auditing privacy properties gains more and more relevance. Nevertheless, this task is lacking support of automated tools and, hence, is mostly carried out manually. We present a formal approach which enables auditors to model the flow of critical data in order to shed new light on a system and to automatically verify given privacy constraints. The formalization is incorporated into a larger policy analysis and verification framework and overall soundness is proven with Isabelle/HOL. Using this solution, it becomes possible to automatically compute architectures which follow specified privacy conditions or to input an existing architecture for verification. Our tool is evaluated in two real-world case studies, where we uncover and fix previously unknown violations of privacy.

1 Introduction

Privacy enhancing technologies provide measures to improve the privacy properties of systems, when applied correctly. But they are not necessarily sufficient, as privacy must also be incorporated on the level of the system *architectures* and already be considered during the design of a newly developed system [4]. There exist multiple approaches [1–4, 7, 17] which aim for developing a high-level concept of privacy in order to enable privacy assessment and auditing of IT systems and their designs. Nevertheless, detailed, often manual, examination is necessary, making audits a complex and time-consuming task. Driven empirically and by running code, *dynamic taint analysis* has been recently used successfully in the Android world to enhance user privacy [12, 18] by tracking the flow of critical information at runtime. However measures from the formal world still offer unleveraged potential for assessing privacy conformance of architectures. We aim for connecting the best of both worlds by making privacy-relevant aspects more explicit and easier to verify.

This work has been supported by the German Federal Ministry of Education and Research, project DecADe, grant 16KIS0538.

The abstract concept of 'security' has been made more tangible and verifiable by deriving protection goals, in particular *confidentiality*, *integrity*, and *availability*. The same method has been applied to the abstract concept of 'privacy' and another triad of protection goals was derived: *Unlinkability*, *Transparency*, and *Intervenability* [2,17].

In distributed systems, privacy aspects can be examined by focusing on the flow of data between system components. Borrowing ideas from dynamic taint analysis and their success in the Android world, we demonstrate that coarse-grained taint analysis is applicable to auditing of *distributed* architectures regarding the aforementioned privacy goals, can be done completely *static* (preventing runtime failures), while providing strong *formal guarantees*.

We motivate this concept by a simple, fictional example: A house, equipped with a smart meter to measure its energy consumption. In addition, the owner provides location information via her smartphone to allow the system to turn off the lights when she leaves the home. Once every month, the aggregated energy consumption is sent over the internet to the energy provider for billing.

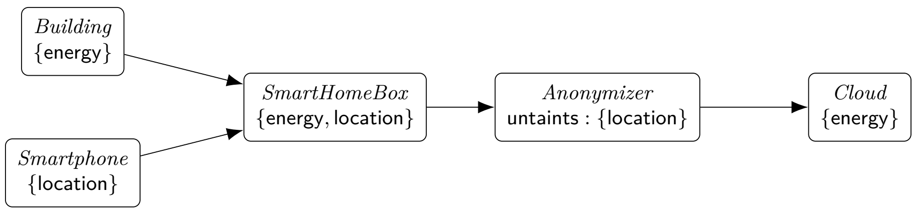

Fig. 1. Example: Privacy Concerns and Information Flow in a Smart Home

We are interested in the privacy implications of this setup and perform a taint tracking analysis. The software architecture is visualized in Fig. 1. The *Building* produces information about its energy consumption, hence we label it as taint source and assign it the energy label. Likewise, the *Smartphone* tracks the location of its owner. Both data is sent to the *SmartHomeBox*. Since the *SmartHomeBox* aggregates all data, it is assigned the set $\{\mathsf{energy}, \mathsf{location}\}$ of taint labels. The user wants to transmit only the energy information, not her location to the energy provider's *Cloud*. Therefore, the *Anonymizer* filters the information and removes all location information. We call this process *untainting*. With the *Anonymizer* operating correctly, only energy-related information ends up in the energy provider's *Cloud*, since $\{\mathsf{energy}, \mathsf{location}\} \setminus \{\mathsf{location}\} = \{\mathsf{energy}\}$.

Even for clearly specified privacy requirements, the confidence in a software evaluation may vary vastly. For example, the Common Criteria [6] define several Evaluation Assurance Levels (EAL). For the highest assurance level, formal verification is required, e.g. using the theorem prover Isabelle/HOL [16]. One remarkable work in the field of formal verification with Isabelle/HOL is the verification of information-flow enforcement for the C implementation of the seL4 microkernel [15]. Similarly, to provide high confidence in our results, we have carried out this research completely in Isabelle/HOL. For brevity, we skip all proofs in this paper. Further details can be found in the full version of this paper and the proofs are provided in the accompanying theory files (cf. Section Availability).

Our proposed solution is a small—yet, fully formal and real-world applicable—step towards modeling (privacy-critical) data flows in distributed systems using taint labels, while being agnostic with respect to the exact notion of privacy chosen by the auditor. Our approach rather opens up a new viewpoint and further enables specifying constraints on data flow which can be automatically verified by our solution. Our case studies show that, even with this restricted toolset, vital insights in real-world systems are already possible. In the first case study, an energy monitoring system similar to Fig. 1, we could make the informal claims of the system's original architects explicit and verify them. In the second case study, a smartphone measurement framework, we demonstrate the complete audit of the real-world implementation in a fully-automated manner, uncovering previously unknown bugs. To the best of our knowledge, this is the first time that such an audit, which bridges the gap from an abstract taint analysis to complex low-level firewall rules, has been performed completely with the assurance level provided by the theorem prover Isabelle/HOL [16].

Our Mission Statement. It is not our goal to formalize the privacy protection goals of unlinkability, transparency, and intervenability. We aim for creating an environment which provides the necessary information for an auditor to start assessing those scenario-specific goals. We aim for statically analyzing distributed systems by considering their architecture specification as we found that this level of abstraction can both be mapped to the real-world implementation of a system, as well as being formally decidable. We intend to lay the groundwork to add *automatic* support for the mentioned privacy protection goals on top; our case study reveals that this is already doable today under certain circumstances.

2 Formalization & Implementation

"The architecture defines the structure of a software system in terms of components and (allowed) dependencies" [13]. We will stick to this high-level, abstract, implementation-agnostic definition for the formalization. As illustrated in Figure 1, a graph can be conveniently used to describe a system architecture. We assume that we have a graph $G = (V, E)$ without taint label annotations which specifies a distributed architecture. Since such a graph specifies the permitted information flows and all allowed accesses, it is sometimes also called a *policy*. To analyze, formalize, and verify policies represented as graphs, we utilize the *topoS* [9,11] framework. It allows specification of predicates over a graph, which are called *security invariants*. They follow special design criteria to ensure the overall soundness of *topoS*. To define a new security invariant, *topoS* imposes strict proof obligations. In return, *topoS* offers arbitrary composability of all security invariants, generic analysis/verification algorithms, and secure auto-completion of user-defined partial attribute assignments [11]. By integrating our formalization into *topoS*, we also obtain a usable and executable tool.

We formalize tainting as a security invariant for *topoS*. To foster intuition, we first present a simplified model which does not support trust or untainting. However, we have aligned this section constructively such that all the results obtained for simple model follow analogously for the full model.

Let t be a total function which returns the taint labels for an entity, for example, $t\ \mathit{SmartHomeBox} = \{\mathsf{energy}, \mathsf{location}\}$. Given an architecture specification $G = (V, E)$, intuitively, information-flow security according to the taint model can be understood as follows: Information leaving a node v is tainted with v's taint labels, hence every receiver r must have the respective taint labels to receive the information. In other words, for every node v in the graph, all nodes r which are reachable from v must have at least v's taint labels. Representing reachability by the transitive closure (i.e. E^+), the invariant can be formalized as follows:

$$\mathsf{tainting}\ (V, E)\ t \equiv \forall v \in V.\ \forall r \in \{r.\ (v, r) \in E^+\}.\ \ t\ v \subseteq t\ r$$

For this formalization, we discharged the proof obligations imposed by *topoS*. This enables us to make use of all generic features of *topoS*, for example, a user may specify a t which is not total.

Analysis: Tainting vs. Bell-LaPadula Model. The Bell-LaPadula model (BLP) is the traditional, de-facto standard model for label-based information-flow security. The question arises whether we can justify our taint model using BLP. *topoS* comes with a pre-defined formalization of the BLP model [11]. The labels in BLP, often called security clearances, are defined as a total order: $\mathsf{unclassified} \leq \mathsf{confidential} \leq \mathsf{secret} \leq \ldots$ Let sc be a total function which assigns a security clearance to each node. Since our policy model does not distinguish read from write actions, the BLP invariant simply states that receivers must have the necessary security clearance for the information they receive:

$$\mathsf{blp}\ (V, E)\ sc \equiv \forall (v_1, v_2) \in E.\ sc\ v_1 \leq sc\ v_2$$

We will now show that one tainting invariant is equal to BLP invariants for every taint label. We define a function $\mathsf{project}\ a\ Ts$, which translates a set of taint labels Ts to a security clearance depending on whether a is in the set of taint labels. Formally, $\mathsf{project}\ a\ Ts \equiv \textbf{if}\ a \in Ts\ \textbf{then}\ \mathsf{confidential}\ \textbf{else}\ \mathsf{unclassified}$. Using function composition, the term *project* $a \circ t$ is a function which first looks up the taint labels of a node and projects them afterwards.

Theorem 1 (Tainting and Bell-LaPadula Equivalence).

$$\mathsf{tainting}\ G\ t \longleftrightarrow \forall a.\ \mathsf{blp}\ G\ (\mathsf{project}\ a\ \circ\ t)$$

The '$\rightarrow$'-direction of our theorem shows that one $\mathsf{tainting}$ invariant guarantees individual privacy according to BLP for each taint label. This implies that every user of a software can obtain her personal privacy guarantees. This fulfills the *transparency* requirement for individual users. The '$\leftarrow$'-direction shows that $\mathsf{tainting}$ is as expressive as BLP. This justifies the theoretic foundations w.r.t. the well-studied BLP model. These findings are in line with Denning's lattice interpretation [8]; however, to the best of our knowledge, we are the first to discover and formally prove this connection in the presented context.

The theorem can be generalized for arbitrary (but finite) sets of taint labels A. The $\mathsf{project}$ function then maps to a numeric value of a security clearance by taking the cardinality of the intersection of A with Ts.

Untainting and Adding Trust. Real-world application requires the need to untaint information, for example, when data is encrypted or properly anonymized. The taint labels now consist of two components: the labels a node taints and the labels it $\mathsf{untaints}$. Let t be a total function t which returns the taints and $\mathsf{untaints}$ for an entity. We extend the simple $\mathsf{tainting}$ invariant to support untainting:

$$\mathsf{tainting}'\ (V, E)\ t \equiv \forall (v_1, v_2) \in E.\ \ \mathsf{taints}\ (t\ v_1) \setminus \mathsf{untaints}\ (t\ v_1) \subseteq \mathsf{taints}\ (t\ v_2)$$

For a taint label a, let $X = \mathsf{taints}\ a$ and let $Y = \mathsf{untaints}\ a$. We impose the type constraint that $Y \subseteq X$. We implemented the datatype such that X is internally extended to $X \cup Y$. For example in Fig. 1, t *Anonymizer* is actually $\mathsf{taints{:}}\ \{\mathsf{energy}, \mathsf{location}\},\ \ \mathsf{untaints{:}}\ \{\mathsf{location}\}$. Which merely appears to be a convenient abbreviation is actually a fundamental requirement for the overall soundness of the invariant. With this type constraint, as indicated earlier, we discharged the proof obligations imposed by *topoS* and all insights obtained for the simple mode now follow analogously for this model, in particular equivalence with a BLP model with trusted entities according to Theorem 1.

3 Conclusion

Several guidelines for verifying and auditing privacy properties of software systems exist. Yet, we found that automated tools for supporting privacy audits are still lacking. We presented a formal model based on static taint analysis which shall contribute to filling this gap. While our model is reduced to the bare minimum to facilitate adding assessment of privacy protection goals on top, the case studies show that improvements of audits are already achievable. We integrated our model into the formal policy framework *topoS* and proved soundness with Isabelle/HOL. From given system specifications or implementations, a model instance can be derived in which flow of critical data becomes explicit and data flow constraints can be verified automatically. We carried out two real-world case studies. They demonstrate the applicability of our approach, exemplifying that insights formally derived from the model are consistent with manual inspections of the architecture. In the second studied system, thanks to our tooling, auditing could be carried out in a completely automated manner.

Availability

Our formalization, case studies, and proofs can be found at `https://www.isa-afp.org/entries/Network_Security_Policy_Verification.shtml`. The full version of this paper is at `https://arxiv.org/abs/1608.04671`.

A Case Studies

The main idea and usage of our model was already motivated by the fictional example of Fig. 1. In this appendix, we present details on two real-world case studies where we evaluate and audit two distributed systems for data collection which are deployed at the Technical University of Munich. For the sake of brevity, we only present the most interesting aspects. We will write node labels as X—Y, where X corresponds to the tainted labels and Y corresponds to the untainted labels. For example, $t\ \mathit{Anonymizer} = \{\mathsf{energy}\}$—$\{\mathsf{location}\}$.

A.1 Energy Monitoring System

Energy monitoring systems (EMS) have severe privacy implications: If installed in an office, such a system for example allows to draw conclusions about the effective working periods and behavior of employees by measuring the devices they use. EMS consist of at least two components: A logging unit which records energy usage locally and a server which stores and processes all recorded data. Considering privacy, storing all collected data in a single, possibly external place without fine-grained access control on the data level is critical.

We examined how to improve privacy of the data before persisting it [14]: Since the logger is an off-the-shelf component which we cannot modify, we suggest to add an additional component, called *P4S*, directly after the logger. This component separates the data by different owners, recipients, or some given predicate and applies further protection measures. The separated data can then be forwarded to (possibly different) cloud services. For the sake of brevity, we do not discuss this service, key management, and how cloud services could collaborate. Our proposed architecture is shown in Fig. 2. We modeled four different kinds of privacy-related data the logger captures by the taint labels A, B, C, and D.

As input to our tool, we provided the set of components including taint labels and system boundaries as architectural constraints. The results are as follows: Our model shows that data flow from the *Logger* to *P4S* (which crosses a system boundary physically over the network) is highly critical. For this taint label specification, *topoS* verified that our architecture is compliant with the security invariants. It also asserts that any attempt to interlink the different data processing pipelines within *P4S* would be a severe privacy violation. These insights generated by *topoS* can be further incorporated into the architecture: The designed pipelines can be separated into individual, isolated, stateless containers within *P4S* that can be instantiated on demand for each different taint label. In summary, our extended *topoS* allowed us to formally assess privacy properties of our proposed architecture before we invested time implementing it.

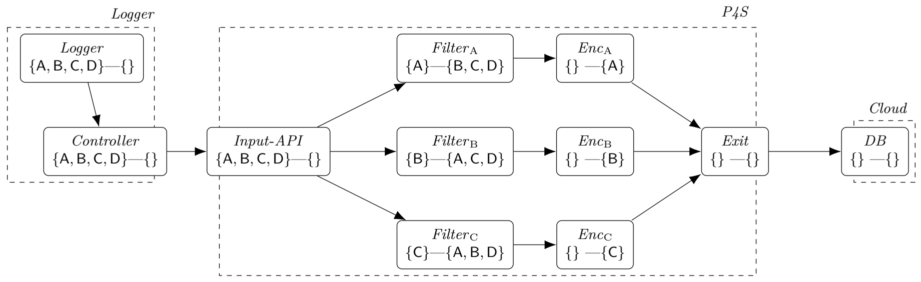

Fig. 2. Architecture of an Energy Monitoring System

A.2 MeasrDroid

MeasrDroid [5] is a system for collecting smartphone sensor data for research purposes. Via an app it may collect location data, information from the smartphone sensors, and networking properties such as signal strength, latency, and reliability. Ultimately, the data is stored and analyzed by a trusted machine, called *CollectDroid*. To decrease the attack surface of this machine, it is not reachable over the Internet. Instead, the smartphones push the data to a server called *UploadDroid*, which is regularly polled by *CollectDroid* for new information. Since *UploadDroid* is particularly exposed, a compromise of this machine must not lead to a privacy violation. Hence, it must be completely uncritical, i.e. not having any taint or untaint labels. This is achieved by having the smartphones encrypt the data for *CollectDroid* as only recipient. Consequently, *UploadDroid* only sees encrypted data. The model of the architecture is shown in Fig. 3. We modeled three users, each producing data with its individual tainting label A, B, or C. To model encryption of some taint label x, we create a pair of related nodes (Enc_x, Dec_x) where the first untaints and the second taints accordingly.

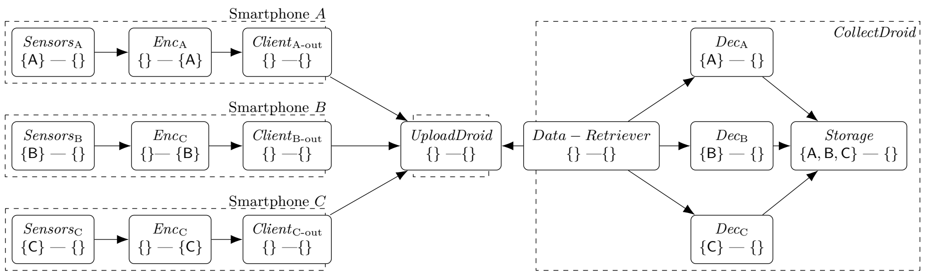

Fig. 3. MeasrDroid Architecture

For this taint label specification, *topoS* verified that our architecture is compliant with the security invariants. In addition, given the taint label specification and adding an additional adversary node, *topoS* automatically computes an alternative architecture which is also compliant with the security invariants. Comparing our manually designed architecture with the *topoS*-generated architecture with adversary, we asserted that we did not overlook subtle information leaks. Our evaluation shows that our architecture is a subset of the *topoS*-generated architecture and only uncritical data can leak to an adversary. It also reveals that our architecture provides no protection against an adversary flooding *UploadDroid* with nonsensical data. We found *topoS* to be a suitable tool to formally support the previous informal privacy claims about the architecture.

Auditing the Real MeasrDroid. The previous paragraphs presents a theoretical evaluation of the architecture of MeasrDroid. The question arises how the real system, which exists since 2013, compares to our theoretical evaluation. Together with the authors of MeasrDroid we evaluate the implementation regarding our previous findings: We collect all machines which are associated with MeasrDroid. We find that they do not have a firewall set up, but instead rely on the central firewall of our lab. With over 5500 rules for IPv4, this firewall may be the largest real-world, publicly available iptables firewall in the world[1] and handles many different use cases. MeasrDroid is only a tiny fragment of it, relying on the protocols http, https, and ssh. For brevity, we focus our audit port 80 (http).

The model of the MeasrDroid architecture (cf. Fig. 3) should be recognizable in the rules of our firewall. In particular, *CollectDroid* should not be reachable from the Internet while *UploadDroid* should, and the former should be able to pull data from the latter. This information may be hidden somewhere in the firewall rule set. We used *fffuu* [10] to extract the access control structure of the firewall. The result is visualized in Fig. 4. This figure reflects the sheer intrinsic complexity of the access control policy enforced by the firewall. We have highlighted three entities. First, the IP range enclosed in a cloud corresponds to the IP range which is not used by our department, i. e. the Internet. The large block on the left corresponds to most internal machines which are not globally accessible. The IP address we marked in bold red belongs to *CollectDroid*. Inspecting the arrows, we have formally verified our first auditing goal: *CollectDroid* is not directly accessible from the Internet. The other large IP block on the right belongs to machines which are globally accessible. The IP address in bold red belongs to *UploadDroid*. Therefore, we have verified our second auditing goal: *UploadDroid* should be reachable from the Internet. In general, it is pleasant to see that the two machines are in different access groups. Finally, we see that the class of IP addresses including *CollectDroid* can access *UploadDroid* which proves our third auditing goal.

For the sake of example, we disregard that most machines at the bottom of Figure 4 could attack *CollectDroid*. Under this assumption, we ignore this part of the graph and extract only the relevant and simplified parts in Fig. 5. So far, we presented only the positive audit finding. Our audit also reveals many problems, visualized with red arrows. They can be clearly recognized in Fig. 5: First, *UploadDroid* can connect to *CollectDroid*. This is a clear violation of the archi-

[1] We make them available at `https://github.com/diekmann/net-network`.

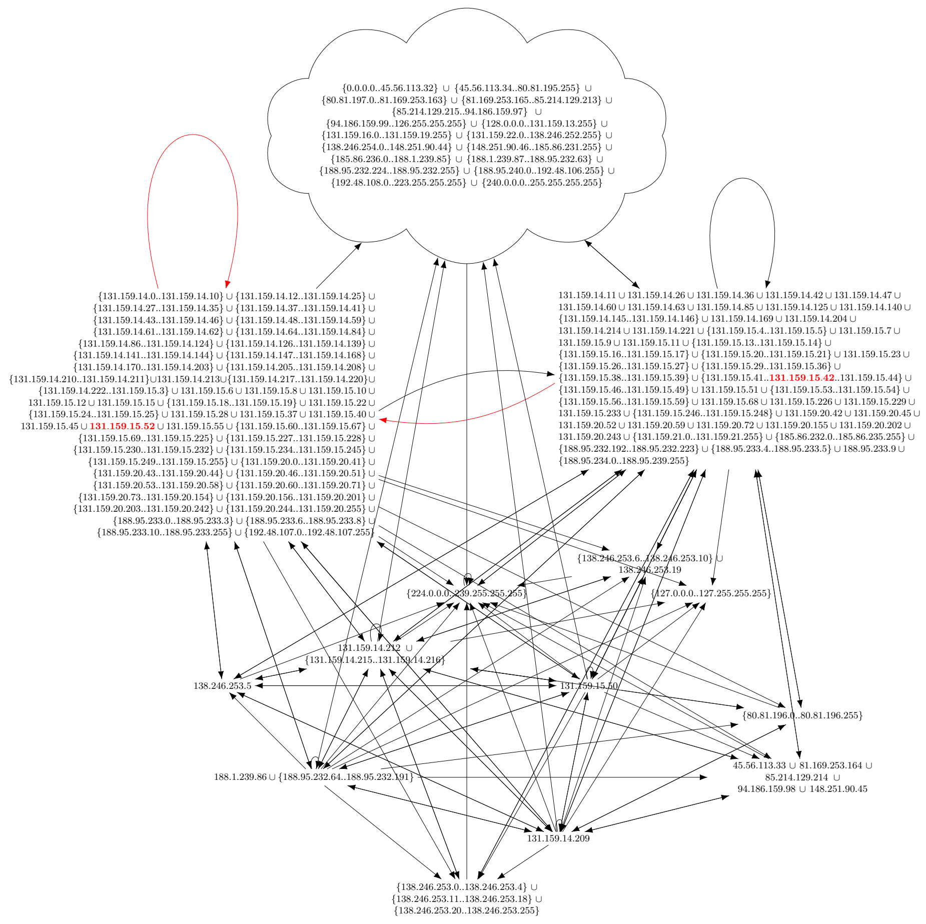

Fig. 4. MeasrDroid: Main firewall – IPv4 http connectivity matrix

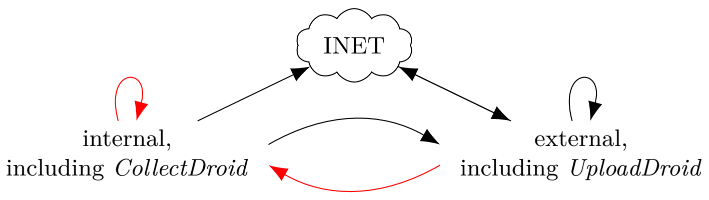

Fig. 5. MeasrDroid: Main firewall – simplified connectivity matrix

tecture. We have empirically verified this highly severe problem by logging into *UploadDroid* and connecting to *CollectDroid*. Second, most internal machines may access *CollectDroid*. Third, there are no restrictions for *UploadDroid* with regard to outgoing connections. In theory, it should only passively retrieve data and never initiate connections by itself (disregarding system updates).

Therefore, our audit could verify some core assertions about the actual implementation. In addition, it could uncover and confirm serious bugs. These bugs were unknown prior to our audit and we could only uncover them with the help of the presented tools. Using the firewall serialization feature of *topoS*, we fixed the problems and reiterated our evaluation to assert that our fix is effective.

References

1. Das Standard-Datenschutzmodell. Tech. rep., Konferenz der unabhängigen Datenschutzbehörden des Bundes und der Länder, Darmstadt (2015), `https://www.datenschutzzentrum.de/uploads/sdm/SDM-Handbuch.pdf`
2. Bock, K., Rost, M.: Privacy By Design und die Neuen Schutzziele. DuD 35(1), 30–35 (2011)
3. Cavoukian, A.: Creation of a Global Privacy Standard (November 2006, Revised October 2009), `https://www.ipc.on.ca/images/resources/gps.pdf`
4. Cavoukian, A.: Privacy by Design. Identity in the Information Society 3(2), 1–12 (2010), `https://www.ipc.on.ca/images/Resources/pbd-implement-7found-principles.pdf`
5. Chair of Network Architectures and Services, TUM: MeasrDroid, `http://www.droid.net.in.tum.de/`
6. Common Criteria: Part 3: Security assurance components. Common Criteria for Information Technology Security Evaluation CCMB-2012-09-003(Version 3.1 Revision 4) (Sep 2012)
7. Danezis, G., Domingo-Ferrer, J., Hansen, M., Hoepman, J.H., Metayer, D.L., Tirtea, R., Schiffner, S.: Privacy and Data Protection by Design - from policy to engineering. Tech. rep., ENISA (2015)
8. Denning, D.: A lattice model of secure information flow. Communications of the ACM 19(5), 236–243 (May 1976)
9. Diekmann, C., Korsten, A., Carle, G.: Demonstrating topoS: Theorem-prover-based synthesis of secure network configurations. In: 11th International Conference on Network and Service Management (CNSM). pp. 366–371 (Nov 2015)
10. Diekmann, C., Michaelis, J., Haslbeck, M., Carle, G.: Verified iptables Firewall Analysis. In: IFIP Networking 2016. Vienna, Austria (May 2016)
11. Diekmann, C., Posselt, S.A., Niedermayer, H., Kinkelin, H., Hanka, O., Carle, G.: Verifying Security Policies using Host Attributes. In: FORTE. pp. 133–148. Springer, Berlin, Germany (Jun 2014)
12. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. ACM TOCS 32(2), 5 (Jun 2014)
13. Feilkas, M., Ratiu, D., Jürgens, E.: The Loss of Architectural Knowledge during System Evolution: An Industrial Case Study. In: ICPC. pp. 188–197 (May 2009)
14. Kinkelin, H., von Maltitz, M., Peter, B., Kappler, C., Niedermayer, H., Carle, G.: Privacy preserving energy management. In: Proceeding of City Labs Workshop, SocInfo 2014. Barcelona, Spain (Oct 2014)

15. Murray, T., Matichuk, D., Brassil, M., Gammie, P., Bourke, T., Seefried, S., Lewis, C., Gao, X., Klein, G.: seL4: From General Purpose to a Proof of Information Flow Enforcement. In: IEEE S&P. pp. 415–429 (May 2013)
16. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002, last updated 2016), `http://isabelle.in.tum.de/`
17. Rost, M., Pfitzmann, A.: Datenschutz-Schutzziele – revisited. Datenschutz und Datensicherheit DuD 33(6), 353–358 (2009)
18. Tromer, E., Schuster, R.: DroidDisintegrator: Intra-Application Information Flow Control in Android Apps (extended version). In: ASIA CCS '16. pp. 401–412. ACM (2016), `http://www.cs.tau.ac.il/~tromer/disintegrator/disintegrator.pdf`