CANE: A Controlled Application Environment for Privacy Protection in ITS

Stefan Dietzel*†, Martin Kost‡, Florian Schaub§, and Frank Kargl*†
*Institute of Distributed Systems, Ulm University, Germany
†Distributed and Embedded Security Group, University of Twente, The Netherlands
‡Databases and Information Systems, Humboldt University, Berlin, Germany
§Institute of Media Informatics, Ulm University, Germany

***Abstract*—Many of the applications proposed for intelligent transportation systems (ITS) need to process and communicate detailed personal identifiable information. Examples are detailed location traces or unique identifiers for authentication towards paid services. Existing applications often run as monolithic black boxes inside users' cars. Hence, users cannot verify that applications behave as expected. We propose CANE, an application sandboxing approach that enhances user control over privacy properties while, at the same time, supporting common application requirements. CANE makes privacy-relevant application properties explicit and allows their analysis and enforcement during application runtime. We evaluate CANE using a common ITS use case and demonstrate feasibility with a proof-of-concept implementation.**

I. Introduction

In intelligent transportation systems (ITS), information and communication technologies are added to traffic systems in order to enhance traffic safety, traffic efficiency, and driver comfort. To support novel applications, developers leverage fine-grained sensor data from on-board sensors. Likewise, applications are often marketed in premium segments and require user authentication using unique identifiers. For instance, current commercial car navigation systems leverage fine-grained location and speed reporting of their users for congestion analysis and traffic prediction.

Long-term success of such applications is dependent on how they deal with user privacy. Recently, TomTom's advanced navigation service hit the news when collected data was sold to the Dutch police to optimize speed camera placement.[1] The TomTom example emphasizes that clear contracts, but also technical enforceability of privacy policies are quintessential.

The call for applying a privacy-by-design philosophy during the design of distributed systems [1] is intended to address the privacy problem by making privacy protection a first-class citizen among design goals. However, the application of privacy-by-design requires novel privacy-enhancing technologies (PETs) that help to fulfill privacy requirements. In this paper, we propose such a PET: our *Controlled ApplicatioN Environment* (CANE) allows fine-grained control of an application's behavior to enforce privacy properties.

Our proposal speeds up deployment of complex ITS applications while maintaining privacy protection. CANE provides an application sandbox, which manages installed applications augmented by descriptive properties, instantiates and terminates them, and tightly controls communication between application instances, as well as filesystem, network, and database access.

[1]See http://www.tomtom.com/page/facts for a commentary of Harold Goddijn, TomTom CEO, on the issue.

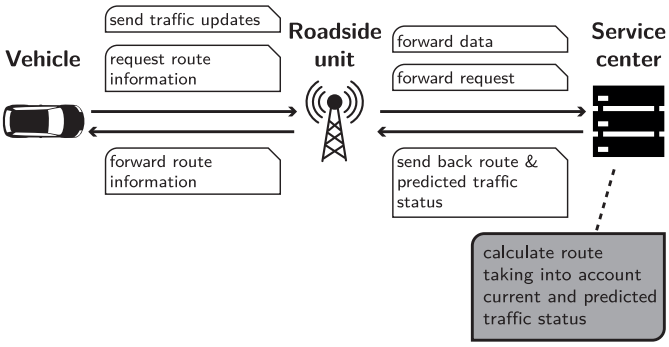

Fig. 1. Online navigation example use case, including floating car data for live traffic prognosis.

A. Example application

The CANE approach is applicable to a wide range of foreseen ITS applications. To foster easier understanding of our concepts, we use an advanced navigation service as an example, which uses live traffic information for routing decisions and is only available to paying customers. Figure 1 illustrates the information flows. The following processes characterize our example application.

Registration. Users need to register in order to use the advanced navigation service. Registration requires users to enter a credit card number, which is billed monthly for service usage. Consequently, each user is uniquely identifiable using her payment details.

Traffic Updates. To offer routes that take into account live traffic information, each user of the system periodically reports current status information to the backend. Status information contains the user id, current position, time, and velocity.

Route request and response. Users send their destination address to the server to receive detailed navigation instructions. Route calculation is performed on the server side, because the server possesses the database with collected traffic updates, which is used to optimize route proposals based on current

12th International Conference on ITS Telecommunications (ITST '12), Taipei, Taiwan, November 2012
Author version.

traffic. Before returning the navigation instructions to the users, the server authenticates the user with the credentials obtained in the registration.

The example application operates on a number of privacy-relevant information items, such as the user's credit card data, detailed location traces obtainable by collecting traffic updates, and destination addresses. At the same time, these information items only pose strong privacy issues if they can be correlated. None of the navigation application components needs access to all collected data items. In the following sections, we will demonstrate step-by-step how CANE can be used to support a privacy-friendly implementation.

In Section II, we derive common requirements of ITS applications and resulting privacy issues that need to be addressed. Then we introduce our controlled application environment (CANE) in Section III. We evaluate our approach in Section IV and conclude our paper with a review of related work (Section V) followed by conclusions and an outlook on future work in Section VI.

II. Requirements Analysis

Many ITS applications, including the example application we described earlier, build on the same functionalities. In this section, we first derive common application requirements. We complement these requirements with privacy requirements derived from common privacy principles [2], [3].

A. Functional ITS Requirements

Most ITS applications periodically acquire dynamic sensor data. Using access network and possibly further infrastructure, such as RSUs, the acquired data is sent to a service center in the backend, for instance, a traffic management server. The backend service may store or process received data together with other information. For applications like the mentioned advanced navigation service, the service provider sends a result back to the vehicle. The following functions are sufficient to reflect the needs of most ITS applications; therefore, an ITS privacy middleware must support such functionality.

Create data. Acquiring sensor data such as location, road condition, or speed to assemble floating car data and gathering user input such as service requests.

Assemble messages. Assembling sensor data together with static vehicle information to create floating car data and assembling data to create service requests.

Send/receive messages Sending floating car data and event information such as detected accidents to other vehicles or service providers and sending requests such as route calculation and traffic updates to service providers.

Store data Storing floating car data to perform traffic analysis, storing user information and location traces to calculate insurance and road usage fees.

Perform custom data processing and transformation. Creating customer-specific invoices; calculating routes, and analyzing traffic information to predict traffic status.

B. ITS Privacy Requirements

CANE aims to protect user privacy by controlling operations on collected information. Specifically, we focus on a number of well-known privacy principles that guide the use of personal information. In the following list, we describe the applied privacy principles and resulting requirements.

Purpose specification. Applications explicitly specify the purpose for which they process data. For instance, a navigation application should not process location data for another purpose.

Requirement. Applications need to declare privacy metadata such as purpose, role, or identifiers and their correctness needs to be enforced.

Limited collection. The amount of collected data must be limited to the minimum necessary for accomplishing the specified purposes. Otherwise, an application could collect additional or too detailed data outside the intended purpose. For instance, it could generate unnecessary unique ids, which may be used to track individuals, or a routing application may create unnecessarily detailed tracks.

Requirement. Provide means to specify the amount of data which may be collected and enforce this limitation. Prevent covert channels and indirect data access.

Limited use. Execute only operations on the data that are consistent with the purposes for which data was collected, stored, and communicated. Otherwise, an application could process data for purposes different from the agreed ones. For instance, individualized advertising and recommendations based on current location or driving behavior.

Requirement. Isolate data flow of different applications and the data flow of different functionalities within one application.

Limited disclosure. Data must not be communicated outside the system for purposes other than those for which the owner gave consent. For instance, data collected for navigation purposes should not be used to issue speeding tickets or to optimize speed camera placement.

Requirement. Guarantee the correctness of specified recipients and limit the list of recipients to those necessary for achieving the specified purpose.

Limited retention. Data will be retained by the system only as long as necessary. For instance, the route request application may not retain the start and destination location data over time for creating long term profiles.

Requirement. Provide means to configure the lifetime of data and application instances and enforce this limitation.

III. Controlled Application Environment

We designed our controlled application environment (CANE) to provide a privacy-aware application sandbox and runtime environment for ITS. As shown in Figure 2, CANE manages a set of applications. Such controlled applications can either work alone, or several controlled applications can act as components of a larger, complex service, such as enhanced navigation services. For each controlled application, CANE starts and terminates its instances. Instances of controlled applications are isolated. No direct communication is allowed

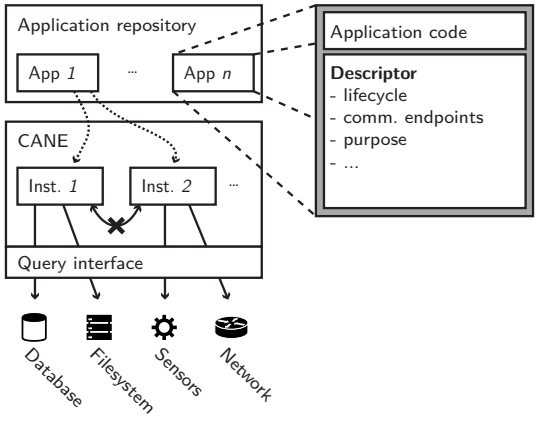

Fig. 2. Overview of CANE's application control.

between instances and they cannot access the file system or network stack directly. Instead, all requests to system resources and stored data are made via CANE.

A. Application modularization support

Existing applications often make use of middleware to ease development. As a result, applications are already modularized into functional components. CANE supports such application modularization, because CANE itself is built on top of an OSGi[2] framework and applications can be composed of OSGi bundles. Moreover, application modularization helps CANE to inspect and control information flow between different components of a complex application. This approach shifts away from classical firewall approaches that only control inter-application communication. In contrast, we make all exchanges of personal information inside an application explicit and controllable.

Example. The vehicle part of the navigation application is composed of the following components, which reflect the processes outlined in Section I-A.

1) *Authentication* – takes care of the authentication towards the backend server and receives an authentication token.
2) *Traffic update* – sends periodic anonymous traffic updates to the server.
3) *Route request* – sends route requests to the server and receives the responses. Uses the authentication token from the authentication component.

We can see that the different components require different types of personal data to work. Most importantly, components 1 and 3 require identifiers, but don't send frequent position updates, whereas component 2 sends frequent updates but can work anonymously. In the following, we will show how CANE controls these components to achieve privacy protection.

B. Application descriptors

We use application descriptors to explicitly describe application properties, which configure application sandboxes. These properties configure and enforce the CANE mechanisms described in the following sections. Namely, CANE supports the following enforced properties.

1) `identifier` — a globally unique identifier for the controlled application.
2) `lifecycle` — the application lifecycle type and parameters; explained in Section III-C.
3) `communication-endpoints` — all external entities that the application communicates with; explained in Section III-D.

During runtime, CANE enforces that applications operate according to their specified properties. When new applications are deployed, installation can be denied depending on doubtful application properties.

Besides such enforced properties, CANE supports certified properties. These properties describe privacy aspects, such as the purpose of an application or its role, which cannot be extracted from runtime behavior. Certified properties can be added to the application descriptor, but need to be signed by a trusted third party. This is required because it is infeasible for CANE to verify that application behavior at runtime matches a certified purpose. However, CANE enforces that an application cannot alter the certified properties during runtime.

C. Lifecycle control

We use lifecycle control to reset an application's state, that is, its caches and collected data. Thereby, we reduce the risk of aggregating information over time. Without lifecycle control, the traffic information component might accumulate detailed location traces over long periods of time. CANE supports three different lifecycle patterns.

1) *Stateless periodic* — stateless periodic applications are terminated each time they fulfilled their purpose once. Different instances cannot directly exchange data.
2) *Stateless event-driven* — event-driven applications are invoked whenever relevant data is received from a remote entity. Again, it is not possible to directly exchange and aggregate information between instances.
3) *Stateful* — stateful applications may run indefinitely and could accumulate a large amount of data. This pattern should only be used if an application needs to perform calculations on data over a longer period of time.

These lifecycle patterns support both separation in time and between data subjects. Periodic applications in the vehicle cannot aggregate personal information, such as location traces. Event-driven applications on the server side cannot correlate data of different data subjects, because each incoming request is handled by a different application instance.

Example. The traffic update component does not need to keep state between invocations, but it should run periodically to keep the traffic information in the backend up to date. Therefore, the component runs with a stateless periodic lifecycle.

D. Communication endpoint control

We require that controlled applications explicitly declare all their external communication endpoints. Instead of specifying a simple list of addresses as possible endpoints, we introduce

[2]http://www.osgi.org/

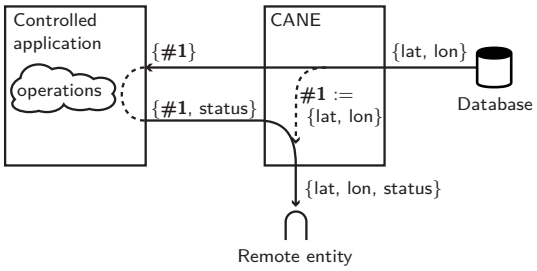

Fig. 3. Information flow with data placeholders.

an abstraction level. Applications specify identifiers for communication endpoints e_i and how they map to IP addresses or URLs as tuples:

$$e_i \ := < ID_i(\leftrightarrow PK_i), IP_i > .$$

Here, ID_i uniquely identifies a communication endpoint and IP_i specifies a network endpoint, such as a TCP/IP address and port. Inside application code, remote connections can only be initiated using the identifiers ID_i. CANE then maps the identifier to the corresponding network endpoint. This prevents malicious application providers from communicating with entities that are not explicitly specified.

To avoid malicious mapping entries, identifiers are bound to corresponding cryptographic public keys PK_i, which are used to encrypt any communication. PK_i is signed by a TTP, attesting the key's identifier. This mechanism prevents covert communication with arbitrary communication partners.

Thus, collected personal information can only be sent to remote entities that are explicitly stated. Moreover, users can inspect the stated remote entities and selectively forbid communication if they suspect that certain remote entities should not receive their information.

Example. The authentication component should only communicate with the navigation server in the backend. To prevent any personal information, possibly including credit card numbers, to a third party, the navigation server is the only configured remote endpoint. All other remote communication is forbidden for the navigation application.

E. Controlled data processing

ITS applications often collect and assemble data to form outgoing messages (see Section II-A). To enable privacy evaluation, we need to infer the contents of the data. Otherwise, information could leak through covert channels. In general, this is hard, because there are endless possibilities to introduce covert channels. The worst case assumption is that *all* previously requested data is potentially contained in *every* outgoing packet. However, working with that assumption would result in most communication being blocked by CANE. Therefore, we introduce an additional mechanism for applications that want to send information but do not need to process it.

Example. In the navigation application, the traffic updates component periodically sends location information to the central server. However, it does so independently of actual data values. Thus, we can use controlled data processing to enhance user privacy. Figure 3 shows the information flow for this example. A controlled application requests latitude (lat) and longitude (lon), but it declares that it does not need to access the values. CANE intercepts the data items $\{\mathsf{lat}, \mathsf{lon}\}$ and replaces them with a placeholder $\#1$. Internally, CANE stores a mapping $\#1 := \{\mathsf{lat}, \mathsf{lon}\}$. When the application wants to send a data item corresponding to a placeholder, it includes the placeholder in the outgoing data packet. CANE then looks up the placeholder and replaces it with the actual data item. As a result, CANE can enforce that the data item in the outgoing packet has not been abused as a covert channel. For the example application, CANE enforces that the status field added by the application does not contain any encoded form of the requested latitude and longitude. Together with a stateless lifecycle (see Section III-C), this mechanism allows to make detailed assumptions about the information contained in outgoing packets.

As another example, consider the route request component. It needs to authenticate towards the backend before using the navigation service. However, the component does not need to know the payment details. Therefore, controlled data processing is used to add the authentication token acquired by the authentication component to outgoing communication. As a result, the route request component does not learn the identity of the user.

In combination, CANE's mechanisms allow to minimize interaction between application components, inspect the data that is exchanged, and guarantee that data is only communicated to known, trusted external components.

IV. Analysis and Implementation

In the following, we discuss how CANE fulfills the privacy requirements identified in Section II and describe the implementation of our CANE prototype.

A. Privacy analysis

The goal of CANE is to guarantee that personal information is processed in a way that is transparent to the user and adheres to the principles listed in Section II-B. Using the advanced navigation application, we outline how CANE mitigates privacy issues and implements the introduced privacy requirements.

If the *purpose* of an application is unknown, we may not decide whether an application performs arbitrary actions on data. CANE implements application descriptors (Section III-B), which are authenticated using digital signatures. Therefore, the purpose of an action performed by an application is always reproducible. The application purpose becomes verifiable for the user and, for instance, location data cannot be abused for unknown purposes.

If the *collection* of data is not *limited*, applications may collect more data than necessary for accomplishing their purpose. CANE prevents such misuse using lifecycle control and controlled data processing (Sections III-C and III-E). Lifecycle control ensures that applications cannot collect large

sets of data. For instance, the route request component's state is reset after each invocation. Therefore, the application cannot accumulate profiles of common user destinations. Likewise, controlled data processing ensures that the application, which needs to authenticate the user as a paying customer, still cannot gain access to the stored credit card information.

If the *use* of data is not *limited*, application providers may use personal information for different services they offer. CANE implements an approach for modularizing applications (Section III-A). Thereby, developers must explicitly separate information flows within one application. CANE isolates these information flows by controlling the involved modules. For instance, if the provider of the advanced navigation service wants to offer targeted advertising, it needs to be implemented as a separate application module. In consequence, location data that is collected within the traffic updates module cannot be reused directly in another module which realizes personalized advertisements.

If the *disclosure* of data is not *limited*, application providers may transfer collected data to external entities without the user's consent. To prevent such arbitrary communication, CANE supports communication endpoint control (Section III-D). In our example, both the traffic updates and route request modules cannot communicate to any servers except those explicitly specified in their application descriptors. In this case, the server of the navigation service provider.

If the *retention* of data is not *limited*, applications may collect large sets of data for longer periods of time. Such data collection is prevented by CANE's lifecycle control (Section III-C). For instance, the route request module is terminated after each route request and its state is reset. Therefore, the application cannot collect sets of different destinations to build user profiles.

In summary, CANE make information flow between different applications and application components in the vehicle explicit due to modularization. CANE minimizes exchange of information between application parts as much as possible. Due to the controlled data processing concept based on placeholders, we enable applications to use information in communication without actually accessing the information themselves. For example, when information is only relevant in the backend. Beyond vehicle borders, we tightly control all communication endpoints of applications to ensure that they only exchange information with authenticated and authorized backends.

B. Prototype implementation

We implemented CANE based on OSGi and the Apache Felix[3] OSGi implementation. CANE, as well as all managed applications, are provided as OSGi application bundles. Using OSGi as a basis enables us to extend and reuse existing functionality. We use permissions to ensure that application bundles can only directly interact with CANE and not with the filesystem, network, or any other installed OSGi bundles. Likewise, we extend Java manifest files to support application descriptors. We integrated CANE with a trusted platform module to ensure the integrity of the CANE framework [4].

We tested our CANE implementation using a number of common ITS application tasks as defined in Section II-B to verify that no substantial delays in data processing are introduced. After 500 runs of a stateless application, an average delay of 463 ± 4 ms per run is introduced by CANE.[4] However, encryption and remote communication make up the largest part of this delay (411 ± 5 ms). Of the remaining 52 ± 3 ms, which are introduced by the CANE core mechanisms, lifecycle control is responsible for the largest share. This is because we use the stock OSGi lifecycle management in our prototype implementation. To improve scalability, the Java reflection API could be used to clear application state (i.e., object instance variables) without completely re-loading the corresponding OSGi bundle.

V. Related Work

Application control environments based on the general concept of application isolation have been developed for different domains. For example, the Java security architecture and sandbox [5] or sandboxes for native code execution by web applications [6]. The Capsicum framework [7] supports modularization of UNIX applications to achieve fine-grained application control based on sandboxing and access capabilities for application components. These approaches are primarily focused on security and do not explicitly consider privacy aspects.

However, there is active research on privacy protection and privacy enhanced execution for mobile applications on smartphones. Android introduced a declarative permission model [8], which informs users about required resource access and has been extended to support selective permissions [9]. However, in contrast to CANE, once an application gains access to a resource, the use of acquired data is not controlled further. Thus privacy consequences are especially unclear for location-based services that require location and Internet access [10]. AppFence [11] supports substitution of sensitive information with falsified data or tracking what data an application requested and subsequently blocking outgoing communication. Our controlled data processing provides more flexibility as applications can also use information without accessing the specific values. The privacy guaranteeing execution container (PGEC) [12] restricts data access locally and limits communication with external entities according to an agreement between user and provider.

In contrast, privacy approaches in the ITS domain have mostly been proposed for specific applications so far. For example, PriPAYD [13], [14] enables privacy-friendly pay-as-you-drive insurance by keeping location traces locally and using a commitment scheme to proof accuracy of calculated fees. PrETP [15] is a similar approach for road charging that uses additional spot checks to validate completeness of

[3] http://felix.apache.org/

[4] Tests were performed on a 2.66 GHz CPU; 1 GB of RAM.

fee calculations. Milo [16] extends PrETP with secure two-party computation to prevent drivers from learning where spot checks are located. VPriv [17] generalizes aggregated reports, commitments and spot check mechanisms for such location-based ITS in a kind of toolbox. However, these approaches do not provide explicit privacy control for drivers and vehicle owners and cannot handle multiple different applications simultaneously. Ginger [18] is a more generic access control framework for telematics applications based on isolated execution, different trust levels for applications, and support for generic privacy policies.

CANE also uses application isolation and builds on ideas from mobile application control. However, CANE extends existing concepts by focusing on controlling application life-cycle and behavior with an emphasis on explicit declaration of communication endpoints and enabling applications to process data without learning specific values.

The PRECIOSA project developed a privacy architecture based on trusted computing for the enforcement of dynamic privacy policies in distributed ITS [19], [4]. Such a system can be used to prevent tampering with the CANE environment and was considered in the CANE implementation.

VI. Conclusion

The proposed controlled application environment (CANE) has been tailored specifically to support privacy in ITS applications. Following a declarative approach, privacy properties of applications can be evaluated without requiring dynamic user interaction during operation. In the application descriptor, we support definition of an application's execution requirements (e.g., statefulness) as well as privacy semantics (e.g., application purpose). Applications must also declare communication endpoints explicitly to prevent data leakage to undesired third parties. CANE further employs controlled data processing with placeholders to enable application components to include certain data in outgoing messages without learning specific values (e.g. location data). In combination with application modularization, CANE and its features can reduce the risk of potential privacy leaks in distributed ITS applications. Therefore, CANE can be used to support, and be integrated with, a privacy-aware design process for ITS applications [20], [21].

We showed how CANE can be utilized in advanced navigation systems to enforce privacy properties while making this enforcement verifiable and demonstrated the effectiveness of our CANE implementation. We are confident that CANE is versatile enough to provide privacy benefits for a wider range of ITS applications and plan to study this in future work. However, CANE is currently limited in its ability to introspect application behavior. In oder to enhance privacy analysis capabilities of CANE, we are working on integrating static analysis of application components at installation time. We are also investigating the potential of expressing certain controlled applications as semantically interpretable statements to facilitate comprehensive privacy analysis.

Acknowledgments

This work has been funded by the European Commission through the FP7 projects PRECIOSA (IST-224201) and PRESERVE (IST-269994).

References

[1] A. Cavoukian, J. Stoddart, A. Dix, I. Nemec, V. Peep, and M. Shroff, "Privacy by design resolution," Issued at 32nd Intl. Conf. of Data Protection Privacy Commissioners, 2010.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases," in *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[3] ISO TC 204/SC/WG 1, "Intelligent transport systems system architecture: Privacy aspects in ITS standards and systems," Tech. Rep., 2008.

[4] M. Kost, B. Wiedersheim, S. Dietzel, F. Schaub, and T. Bachmor, "WiSec 2011 Demo: PRECIOSA PeRA - Practical Enforcement of Privacy Policies in Intelligent Transportation Systems," *Mobile Comp. and Comm. Review (MC2R)*, vol. 15, no. 3, 2011.

[5] L. Gong, "Java 2 platform security architecture 1.2, Java SE documentation," 2002.

[6] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native Client: A Sandbox for Portable, Untrusted x86 Native Code," in *30th IEEE Symposium on Security and Privacy*, 2009.

[7] R. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: practical capabilities for UNIX," in *USENIX Security Symposium*, 2010.

[8] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A Comprehensive Security Assessment," *IEEE Security & Privacy Magazine*, vol. 8, no. 2, 2010.

[9] M. Nauman, S. Khan, and X. Zhang, "Apex: extending Android permission model and enforcement with user-defined runtime constraints," in *5th ACM Symp. on Information, Computer and Comm. Security*, 2010.

[10] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *17th ACM Conf. on Computer and Communications Security*, 2010.

[11] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, ""These Aren't the Droids You're Looking For": Retrofitting Android to Protect Data from Imperious Applications," in *18th ACM Conf. on Computer and Communications Security*, 2011.

[12] M. Maaser and P. Langendörfer, "Privacy from Promises to Protection: Privacy Guaranteeing Execution Container," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 65–81, Nov. 2009.

[13] C. Troncoso, G. Danezis, E. Kosta, and B. Preneel, "PriPAYD: Privacy Friendly Pay-As-You-Drive Insurance," in *ACM Workshop on Privacy in Electronic Society*, 2007.

[14] C. Troncoso, G. Danezis, E. Kosta, J. Balasch, and B. Preneel, "PriPAYD: Privacy-Friendly Pay-As-You-Drive Insurance," *IEEE Trans. on Dependable and Secure Computing*, vol. 8, no. 5, 2011.

[15] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "PrETP: Privacy-preserving electronic toll pricing," in *USENIX Security Symposium*, 2010.

[16] S. Meiklejohn, K. Mowery, and S. Checkoway, "The Phantom Tollbooth: Privacy-Preserving Electronic Toll Collection in the Presence of Driver Collusion," in *USENIX Security Symposium*, 2011.

[17] R. A. Popa, H. Balakrishnan, and A. J. Blumberg, "VPriv: Protecting Privacy in Location-Based Vehicular Services," in *USENIX Security Symposium*, 2009.

[18] D. Herges, N. Asaj, B. Könings, F. Schaub, and M. Weber, "Ginger: An Access Control Framework for Telematics Applications," in *IEEE Conf. on Trust, Security and Privacy in Comp. and Comm. (TrustCom)*, 2012.

[19] F. Kargl, F. Schaub, and S. Dietzel, "Mandatory Enforcement of Privacy Policies using Trusted Computing Principles," in *Intelligent Information Privacy Management Symposium*. AAAI, 2010.

[20] M. Kost, J.-C. Freytag, F. Kargl, and A. Kung, "Privacy Verification using Ontologies," in *First Intl. Workshop on Privacy by Design*, 2011.

[21] M. Kost and J. C. Freytag, "Privacy analysis using ontologies," in *2nd ACM Conf. Data and App. Security and Privacy (CODASPY '12)*, 2012.