Ginger: An Access Control Framework for Telematics Applications

David Herges*, Naim Asaj†, Bastian Könings*, Florian Schaub*, Michael Weber*
*Institute of Media Informatics, Ulm University, Ulm, Germany
Email: david.herges@alumni.uni-ulm.de ; { bastian.koenings | florian.schaub | michael.weber }@uni-ulm.de
†Daimler AG, Group Research, Dep. Infotainment & Telematics, Ulm, Germany
Email: naim.asaj@daimler.com

***Abstract*—Telematics applications in automobiles have attracted considerable attention for their promise of value-added services. However, new challenges arise from the integration of telematics applications that require a multitude of vehicular data. The safety-critical context of the automotive domain calls for reliable access control mechanisms, especially when applications access sensitive data. However, those mechanisms must also respect the requirement of minimal driver distraction. To cope with these issues we propose Ginger – an access control framework for telematics applications. Ginger is context aware, provides enhanced privacy protection, and realizes advanced access control paradigms. We demonstrate Ginger's feasibility with an Android-based implementation in a functional evaluation consisting of two representative use cases and in a comparative analysis with related approaches.**

I. Introduction

Enhanced telematics platforms equipped with wireless communication technologies (e.g., UMTS, LTE, 802.11x, etc.) will enable numerous automotive applications in the future. Similar to the ongoing trend of mobile apps for current smartphones, customer expectation of modern vehicles will require customization and integration of applications in automotive platforms. Envisioned telematics applications range from weather forecast, social networks, location-based services up to complex and information-rich applications, such as corporate fleet management and Pay as you drive car insurance. These applications are enabled by tapping into vehicle data, such as vehicle speed and location, or driver's personal information. Similar to smartphones, automotive apps can benefit from vehicle's sensor infrastructure and provide advanced and tailored services to the driver.

However, vehicular sensor data can be sensitive and safety critical. Therefore, access control mechanisms that respect essential security and privacy concerns play a fundamental role in enabling automotive apps. Today's conventional telematics platforms are inadequate and too inflexible to fulfill additional access control requirements such as context awareness and privacy protection for telematics applications. Telematics constraints like regulations on driver distraction further necessitate the development of access control systems that are better tailored to requirements of the automotive domain.

Due to the increased popularity of smartphones, various access control concepts have been introduced for mobile application platforms, such as Android, Blackberry, iOS, and Windows Phone 7. In fact, telematics platforms share similar conditions with smartphone platforms such as mobility and sensor infrastructure. Not surprisingly, some headunit suppliers already started adapting smartphone platforms to the automotive domain, e.g., AutoLinQ[1] based on Android. However, existing access control approaches cannot be directly applied from smartphone platforms to telematics applications. Especially pop-up dialogues are impractical and unsuitable for letting drivers grant or deny permissions to applications during run-time. Moreover, privacy-aware drivers should be able to express their privacy wishes and preferences without having to define and handle complex access rules.

In this work, we propose Ginger – a new approach towards a generic access control framework for telematics applications that comprises different access control paradigms. The framework copes with the requirements of telematics constraints and integrates context awareness as well as privacy protection mechanisms. The Ginger access control framework uses a permission model based on attributes, conditions, obligations, and post-updates, in combination with isolation domains and a simplified trust model.

In Section II, we introduce the concepts of our access control framework in detail, followed by a brief overview of the Android-based implementation in Section III. In Section IV we provide a discussion of related Android frameworks, which serves as a prerequisite for the comparative analysis and functional evaluation of our framework in Section VI based on two representative use cases presented in Section V. Section VII concludes the paper.

II. Ginger Access Control Framework

In this section, we describe the concepts and architectural design of the Ginger framework with a generic, and platform-independent specification of the framework components. We will use the following terms throughout the paper:

- *Entity*: a piece of software, e.g., an application, a library, a software component.
- *Message*: the communication medium that entities use to interact with each other.
- *Subject*: the calling entity that initiates a message pass.
- *Object*: the target entity that handles a message pass.

[1] AutoLinQ project website: http://www.autolinq.de/

11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012), Liverpool, June 2012
Author version. Definitive version at http://dx.doi.org/10.1109/TrustCom.2012.169

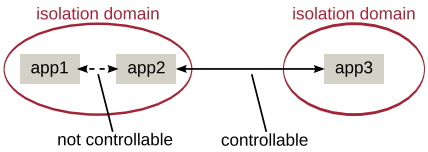

Fig. 1. Control of data flow between apps in different isolation domains.

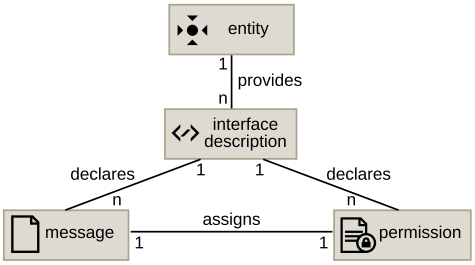

Fig. 2. Interface description and permission assignment to messages.

To clarify these terms, a message is used by entities to interact with each other. The communication process is referred to as *messaging* or a *message pass*. A message pass is initiated by the subject, invoking an operation on the object. The object handles the message pass, e.g., by executing the requested operation and providing some data. Subsequently, the result is returned to the subject. Thus, data is flowing from the object to the subject, being transported through a message.

In the following, we detail the framework's main concepts *isolation domains*, *trust levels* and *permission model*, and explain how they are integrated in the Ginger framework architecture.

A. Isolation domains

A basic design choice is to keep entities in isolation domains. Isolation is a well-known concept realized for example in application sandboxing [1], [2], virtual trusted domains [3], trusted privacy domains [4], or info spaces [5].

In Ginger, isolation domains logically separate entities from each other, thus providing strong security boundaries at runtime. Compared to application sandboxing in mobile operating systems, a prime distinction is that Ginger isolation domains do not grant entities access to private persistent storage.

Operations and data flow within an isolation domain are not controlled by Ginger, whereas operations and data flow between isolation domains, i.e., access that crosses a domain boundary, is controlled (see Fig. 1). In other words, Ginger controls inter-domain messaging but not intra-domain messaging. Ideally, each entity is assigned its very own isolation domain so that Ginger is able to monitor all data flow from and to an entity.

B. Trust levels

Entity is an abstract term used in the framework to refer to active processes or components. Since there is no straightforward distinction between system components, sensor APIs, third-party applications, etc., we assume that each entity provides a self-imposed interface description. This description

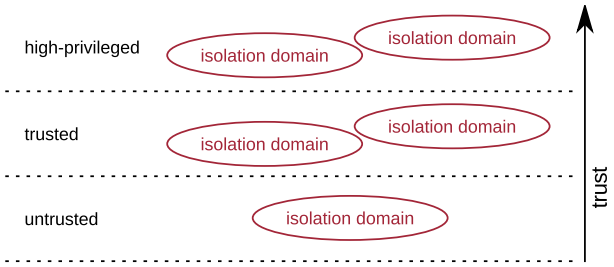

Fig. 3. Isolation domains on different trust levels.

defines the messages this entity can handle and responds to. Furthermore, for each message a *permission* is defined, which is required to access such a message. The relations are clarified in Fig. 2; the permissions concept is described in the following section. Thus, the interface description defines an entity's provided functionality and what data it exports.

To enhance data flow monitoring, we adopt a simple trust model in order to express confidence levels for inter-domain data flow. Various trust definitions and representations can be found in literature, see [6], [7] for an overview. In Ginger, trust is used to measure *Ginger's confidence that an entity genuinely declared the content of its exportable data*. Trust is represented on a discrete, ordered scale.

Thus, a fully trusted entity provides a valid interface description including accurate definition of message data types and does not hide undefined data in messages, i.e., messages are not used as covert channels to export data. An untrusted entity may likely violate these requirements. As a consequence, an untrusted entity must not be the object of a message pass since it may export sensitive data. In addition, we propose that certain entities are granted less restrictive access based on their trust level, as shown in Fig. 3. For instance, system components, such as scheduler or application installer, can be considered high-privileged entities and should be handled less restrictive according to their trust level. If the trust level of an entity does neither allow nor deny sending of a message, that access is mediated according to the permission model.

C. Permission model

A permission is the abstract representation of a specific privilege an application holds. The permission acts as an intermediary between subject and object. This facilitates that authorization rules can be specified on an entity-specific per-permission basis as well as on a device-wide per-permission basis. *Permission assignment* governs when a permission is required for a message pass; *permission authorization* governs under which circumstances the permission is granted.

By assigning a permission to an object (see Fig. 2), Ginger requires the subject to be authorized for that permission; otherwise, the subject cannot access the object. This assignment relation can be modeled as the three-tuple `(o, m, p)`, meaning that message `m` to object `o` requires permission `p`.

The corresponding authorization relation for permission `p` can be expressed by the four-tuple `(s, p, c, u)`. Hereby,

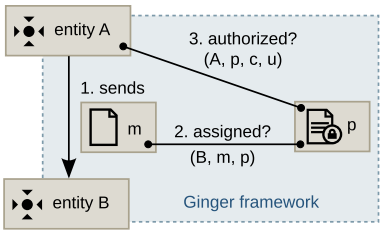

Fig. 4. When entity A sends message m to entity B, Ginger verifies whether a permission is assigned to m and if A is authorized to exercise that permission.

```
<c>     := <x> AND <x> | <x> OR <x> | NOT <x>
<x>     := (<key>, <opr>, <val>)
<key>   := SATTR|OATTR|COND|OBLG:<name>
```

```
<u>     := SATTR|OATTR:<name>, <opr>, <val>
```

Fig. 5. Constraint `<c>` and post-update `<u>` expression language.

subject `s` is granted permission `p` if, and only if, constraint expression `c` holds `true` and post-update `u` is executed. Fig. 4 illustrates Ginger's verification process.

For constraints `c` and post-updates `u`, we adopt the concepts of *attributes*, *conditions*, *obligations*, and *post-updates* from the UCON_{ABC} model [8]. A constraint `c` is expressed as (`key, operator, value`)-triples. The constraint's type and name is identified by `key`, where the type is either *attribute*, *condition*, or *obligation*. Further, attributes are distinguished between subject attributes (`SATTR`) and object attributes (`OATTR`). `Operator` and `value` specify the checks that should be performed, i.e., they determine the circumstance under which constraint `c` evaluates to `true`. Complex constraint expressions can be built by applying `AND`, `OR`, `NOT` operations to the predicates. If the constraint evaluates to `true`, the permission is granted; otherwise, it is denied.

A post-update `u` can be optionally set for an authorization rule. If set, a post-update alters attributes of the subject or object as consequence of the authorization. The complete language for constraint and post-update expression is depicted in Fig. 5. Next, we describe provide a detailed description of *attributes*, *conditions*, *obligations*, and *post-updates* and give examples for intended use:

1) Attributes: Entities are characterized through their attributes. Example attributes are an application's version number, author, or the last time of a certain access. Attributes can be mutable or immutable. *Mutable attributes* may be altered through post-updates as consequence of a granted permission. Moreover, attributes are either volatile or persistent. *Volatile attributes* apply only to an entity's current (run-time) instance, i.e., attribute values are reset when a new entity instance is created, whereas values of *persistent attributes* are stored for all instances of an entity.

2) Conditions: A common definition of context is *any information that can be used to characterise situation*, so that a system becomes context-aware *if it uses context to provide services* [9]. We adopt this definition and provide context-aware access decisions through conditions. A condition refers to device-wide system context and is therefore not under direct control of an entity. For example, a condition may specify a time-window or the state of the vehicle, e.g., speed > 20 km/h.

3) Obligations: In privacy-aware access control models, a common understanding is that *obligations are usually coupled with some action request – that is, subjects commit themselves to complete some obligations to execute a specific action on some objects* [10]. In Ginger, the understanding is that an *obligation is coupled with some action on message data, i.e., the framework completes an obligation on message data in the execution of a message pass*. That definition explicitly means both examination and manipulation of the message data that is being exchanged between entities.

For example, assume that a subject should only be granted access to coarse location data. Such a behavior can be achieved by specifying an obligation `(location-granularity, >, N)`. When the subject is sending a corresponding message to an object, the framework intercepts the message and extracts the returned data. Ginger would then ensure that the object's provided location data adheres to the constraint, i.e. granularity of location data is greater than the desired level N, by applying an obfuscation algorithm. Technically, location obfuscation can be achieved with random noise, spatial cloaking, temporal cloaking, and other approaches. In Ginger, the actual implementation of an obligation action is provided by evulation modules that will be described later.

4) Post-Updates: Setting post-updates is optional. If a post-update is set and a permission is being granted, attributes of the entity are subsequently altered accordingly.

For example if a permission `p1` is granted to a subject, a post-update could be used to append `p1` to the subject's attribute `exercised-permissions`. On subsequent access attempts, an attribute-based constraint could be used to evaluate whether the subject did exercise permission `p1` before, thus enabling features similar to *separation of duties* in RBAC [11].

D. Framework architecture

Ginger's framework architecture consists of four main components that implement the introduced concepts (see Fig. 6): Policy Enforcement Points (PEPs), a Policy Decision Point (PDP), evaluation modules, and Policy Administration Points (PAPs).

A PEP mediates inter-domain messaging between a subject and an object. A PEP has multiple tasks. It handles the message pass, translates the message into a canonical internal data format, and enforces the actual access decision. PEP operation is transparent to the subject, i.e., a PEP manipulates the message in such a way that the calling entity does not recognize the PEP's existence.

The PEP queries the PDP to resolve the access decision for the current message pass. First, the PDP checks trust levels of both subject and object; if the object is an untrusted entity, the access decision is denied; if the subject is a high-privileged

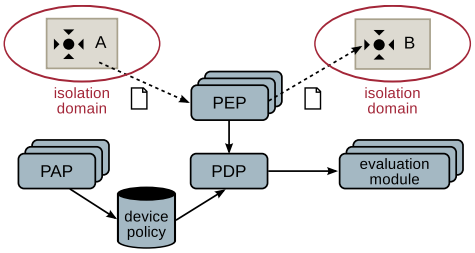

Fig. 6. Framework components and system architecture: an entity's access that crosses the boundary of an isolation domain is mediated by a PEP.

entity, the access decision is granted. If the above is not the case, the PDP determines whether and what permission is required for the message pass according to the permission assignment relation. Then, the PDP evaluates the permission authorization relation and delegates evaluation of constraints and post-updates to registered evaluation modules. In this step, the access decision resolves to either granted or denied and the message data may be modified. Both access decision and message data are provided to the PEP, which enforces the access decision accordingly.

In order to evaluate constraints and post-updates, the Ginger framework uses a flexible approach based on evaluation modules. While conditions and post-updates are generic, descriptive expressions, an evaluation module provides the actual functionality to evaluate a specific constraint or post-update identified by a certain `key`. The evaluation module registers itself for a specific key at the PDP so that the PDP can delegate evaluation of conditions and post-updates to appropriate modules. Evaluation modules take the predicate given by an authorization rule and a key-value list of reference data as its input parameters. For example, an attribute evaluation module would take the predicate describing an attribute expression and a list of entity attributes as input and would check whether the list of entity attributes matches the predicate. A module returns either `true` if the evaluation was successful or `false`, otherwise.

The task of a PAP is administration of a device policy. Similar to PEPs, multiple PAPs exist. For instance, one PAP is consulted upon installation of an entity. It evaluates the entity's interface description, determines its trust level, and updates permission assignment relations, if necessary. Other PAPs handle administration of permission authorization relations. One important question is who should be the policy writer that is authorizing entities to perform privileges? There is no straightforward answer to that question, ranging from user-centered security approaches to remote device administration. Hence, we propose that PAPs offer an interface to edit permission authorization for the stakeholder policy writer and we aim to investigate this issue in future work.

III. Android Implementation

As a proof-of-concept, we implemented Ginger based on the Android Open Source Project (AOSP) v2.3.7r_1 and

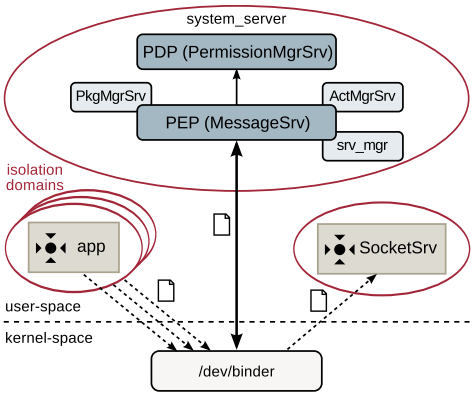

Fig. 7. Ginger Implementation: Framework components are implemented as part of the middleware system_server. Light-blue boxes indicate modified Android components, dark-blue boxes are new components in Ginger.

released it as open source[2]. We argue that Android has all the ingredients to realize the proposed access control framework and implement Ginger on top of the application sandbox and Binder inter-process communication (IPC). Fig. 7 illustrates the Android-based Ginger implementation..

We utilized Android's application sandbox, technically realized by assigning each application a unique and distinct Unix user id [12], [13], to provide an enforcement mechanism for isolation domains.

Interaction between Android applications is built on top of Binder IPC, a synchronous IPC mechanism based on shared memory. We modified the Binder framework in such a way that applications must send messages through a system-level message service; using the message service is mandatory and cannot be bypassed. The message service implements a PEP, transforms message contents into a canonical data format, and queries the permission manager, i.e., the PDP implementation, for an access decision. Furthermore, we provided stand-in replacements for Android API implementations to make calls to the message service transparent.

PEP and PDP implementations are realized according to the framework design presented in Section II. They are running as system services in Android. Android features a permission model of its own which expresses permissions as string-based security labels. We adopted this approach for our implementation and express permissions as plain string labels.

In Android, access to some APIs, such as sockets, is protected by permissions associated with a Unix group id. For such APIs, we set the respective permissions to `systemOrSignature` level to prevent third-party applications from being granted those permissions. Then, we implemented trusted (system) components providing the API functionality to applications and made these components accessible through the message service; as for the other APIs, we provide stand-in replacements of the API implementation classes.

[2]GINGER project website: https://gitorious.org/ginger/

IV. Related Android Frameworks

Recent research focuses on improving security of mobile application platforms. Most notably, several extended security frameworks have been proposed for Android due to the platform's open source code. This section provides a brief overview of available Android modifications and highlights the differences to the Ginger framework (see Sec. II). In Section VI follows a comparative analysis of Ginger and the related approaches introduced here.

The Apex framework [14] allows users to specify run-time constraints for individual permissions. After application installation, a policy configuration tool gives the user the ability to grant, deny, or conditionally allow individual application permissions. Conditional allow specifies run-time constraints such as "let application send a maximum of 5 text messages per day". PrimAndroid [15] augments the Apex framework by implementing a privacy aware RBAC framework on Android. Neither Apex nor PrimAndroid are able to inspect and modify data flow of inter-process communication.

The application-centric SAINT framework [16] facilitates semantically rich policies that are enforced at both install-time and run-time. In addition to permission declaration, the framework enables applications to specify MAC policies that are associated with permissions, thus controlling to whom permissions are granted and how these permissions are used. Furthermore, SAINT's administrative policy gives the user the ability to override policy rules. Like Apex and PrimAndroid, SAINT is not capable of inspecting and modifying data flow of inter-process communication.

ConUCON [17] implements a context-aware usage control framework on Android. It utilizes obligations, states and context for usage decisions based on the family of UCON_{ABC} models for usage control [8]. ConUCON focuses on adapting the existing Android permission model and thus does not extend the concept of obligations to inspect data flow of inter-process communication, hence it does not provide fine-grained privacy controls such as location granularity.

AppFence [18] implements a privacy framework for Android. It is able to block network transmissions that contain data flagged for on-device usage only. AppFence is able to transparently substitute shadow data in place of private data. However, the data labeling relies on taint tracking [19], thus being vulnerable to implicit data flows [18]. The YAASE framework [20] provides a similar approach to AppFence but also relies on taint tracking. Neither of them is capable of respecting contextual constraints in access decisions.

MockDroid [21] provides a similar approach, allowing users to "mock" an application's access to resources so that the resource is subsequently reported as empty or unavailable. However, mocking and data shadowing do not allow fine-grained privacy constraints such as location granularity.

Porting SELinux to the Android kernel provides a kernel-enforced MAC framework that augments the application sandbox and hardens devices [22]. A similar approach is TrustDroid [23], a framework for practical and lightweight domain isolation implemented by TOMOYO Linux. Both frameworks may be used to harden the enforcement mechanism for isolation domains, thus improving the implementation of Ginger.

ACPlib [24] utilizes an approach similar to Ginger by enforcing application-centric permission through trusted services. In addition, the Redexer tool modifies application packages and Dalvik binaries so that applications use the trusted ACPlib services instead of the original APIs. However, ACPlib permissions do not enforce the fine-grained level of privacy controls provided by Ginger.

V. Use Cases

In this section, we outline two use cases, for our evaluation in Section VI, that highlight specific access control requirements of representative telematics applications. A summary of the two use cases is given in Table I.

A. Use case I: Corporate Pay As You Drive Insurance

Recently, car insurance companies started to realize that telematics applications serve as platforms for new business concepts. One of these concepts is the idea of Pay As You Drive (PAYD) insurances, i.e., usage-based or milage-based car insurance. Three common PAYD insurance models exist: (1) coverage based on the odometer, (2) coverage based on time driven and (3) coverage based on vehicle data (e.g., vehicle speed, navigation route and location). In this use case we anticipate the idea of corporate PAYD car insurance where company car fleets are equipped with the PAYD service to save insurance fees. We assume a dynamic insurance rate, which depends on the vehicle data collected by the service. This way companies providing specific vehicle data to the insurance company can benefit by having cars being classified in a lower rate category. In contrast preventing disclosure of vehicle data will change insurance fees to a more expensive basic rate. We assume that the corporate PAYD car insurance application behaves in the following simplified way:

1) The application accesses different vehicle device sensors to read required information, such as *vehicle speed*, *location*, *navigation route* and *odometer*,
2) the application opens a network connection and sends the obtained driving information to a remote endpoint.

We further assume that company cars are also used by employees for personal use. In personal use situations, the continuous collection of sensitive vehicular and personal data (e.g., speed, odometer and location) by the PAYD service could violate privacy expectations of employees.

Here, context-aware access can overcome this issue by distinguishing between the different vehicle states *business* and *personal use* with appropriate access limitations. Thus, preventing the service to collect and use data while a company car is used for personal purpose. In times of personal use, company cars would be charged with the basic insurance rate (no reward). This context-sensitive approach tries to strike a trade-off between drivers' privacy expectations and the company's business strategies, as well as minimizing the inquiries to the driver while the vehicle is moving or in business use.

```
a) (app.corporate.payd, permission.SENSOR, (COND:business-trip, is, true), -)
b) (app.corporate.payd, permission.INTERNET, -, -)
```

Listing 1. Permission authorization rules for use case I.

```
a) (app.info.weather, permission.LOCATION, (OBLG:location-granularity, greater, X), -)
b) (app.info.weather, permission.LOCATION, (OBLG:location-granularity, greater, X) AND
    (SATTR:last-access, was-ago, T), (SATTR:last-access, set-to, NOW))
c) (app.info.weather, permission.LOCATION, -, (SATTR:location-access, set, true))
   (app.info.weather, permission.INTERNET, (SATTR:location-access, is-not, true), -)
```

Listing 2. Permission authorization rules for use case II

TABLE I
SUMMARY OF USE CASE FEATURES.

use case	vehicle data	user wishes	Ginger feature
Corporate PAYD	speed, location, odometer, route	only business trip	context-sensitive
Weather Service	location	exact location protection	privacy-sensitive

B. Use case II: Weather Service Application

A prevalent privacy concern in (auto)mobile platforms is that applications disclose unique device id's (e.g., MAC address, vehicle identification number, etc.) and user location to remote services [18]. Based on such information, long-term tracking is possible and fine-grained movement profiles of users can be reconstructed. Thus, privacy-aware users must be able to control disclosure of their location data, while maintaining an application's functionality.

A typical use case is a weather service application enabled by accessing the vehicle's current location and sending it to a remote server in order to provide weather forecast for the particular location. Thus, the weather application requires Internet access as well as to the vehicle's location. Although the weather service application is not that complex it is one of the most attractive location-based services in the area of telematics application provisioning. We assume that the weather service behaves in the following way:

1) The application accesses the device's location sensors and reads the user's (vehicle's) location data,
2) the application opens a network connection and sends the obtained location data to a remote endpoint.

Now, considering users' privacy wishes, disclosure of user's exact location data to a remote endpoint might violate the user's privacy expectations. On the one side, users want to prevent the weather service from tracking their exact location, but on the other side, they want to receive accurate weather forecasts. By completely blocking access to a vehicle's location data, the weather service would be unusable. Thus, this is neither a desirable solution for users nor for OEMs deploying telematics applications. Ginger provides a solution which respects privacy while maintaining application functionality.

VI. EVALUATION

We evaluate Ginger by showing the effectiveness of proposed access control features for use cases I and II. Furthermore, we provide a comparative analysis on how these use cases are addressable in the related frameworks AppFence, MockDroid, YAASE, ConUCON, and Apex (see Sec. IV). We chose these five frameworks as they provide the most promising features and also have several similarities to the Ginger framework. The results are summarized in Table II.

A. Evaluation Use Case I

In this use case, the entities involved are the PAYD application, which collects data, the networking interface, and a number of sensor interfaces for vehicle speed, location, navigation route, and odometer. For simplicity, all sensor interfaces are assumed to require the same `permission.SENSOR` in order to provide sensor data.

According to the use case, the PAYD application should only collect sensor data when the car is used for business purposes. In Ginger, this can be achieved by means of the condition concept. An adequate condition predicate is `business-trip` and an evaluation module provides the capability to determine whether the car is being used for business or personal purposes. In a simple solution, the evaluation module would check the vehicle's system clock against a fixed business schedule (e.g., 8AM - 6PM on weekdays) to evaluate the predicate. Listing 1 a) depicts a corresponding authorization rule, with the effect that the PAYD application can access sensor data only when the car is used for corporate means. Regarding `permission.INTERNET`, the authorization rule could be more lenient in this case (see Listing 1b); even if the PAYD application would connect to the network outside the business schedule, it would not be able to collect and disclose sensor data during personal use of the car.

In AppFence, such a conditional rule is not expressible. A workaround would be to have a private sensor data source and a corporate sensor data source, i.e., having one data source for each condition. Then, tainting private sensor data as sensitive still leaves the problem of leaking sensitive data through implicit data flows. In Mockdroid and YAASE, a similar workaround would be required to distinguish a private from a corporate sensor data source. By mocking or applying filter actions private sensor data would be made inaccessible to

the PAYD application. However, neither of these frameworks supports modeling of conditional rules. Hence, AppFence, MockDroid, and YAASE do not offer a straight-forward solution for use case I.

In ConUCON and Apex, conditional authorization rules can be set for the permission required to access sensor data, achieving the same effect as Ginger's `business-trip` condition.

B. Evaluation Use Case II

In this use case, the entities involved are the weather service application, the location sensing service of the device, and a networking API that enables the application to connect to the remote weather service. The permission required to access the location sensing capabilities is `permission.LOCATION` and a corresponding entry is set in the permission assignment relation. The required permission for accessing the networking API is `permission.INTERNET`.

In Ginger, there are two possibilities to address this use case. First, an authorization rule for the weather application can be associated with an obligation. The `location-granularity` obligation provides the functionality to obfuscate a user's location data as described earlier in Section II. Setting the authorization rule as shown in Listing 2a ensures that the weather application receives location data that is obfuscated with granularity level `X`. Thus, the application is unable to disclose the user's exact location in subsequent Internet communication. In addition, the frequency of location access can be limited by adding a post-update and an attribute constraint. The final authorization rule is shown in Listing 2b. When accessing location data, the post-update sets the application's attribute `last-access` to the current system time. On subsequent access, the attribute constraint ensures that the attribute value indicates a point in time that is at least `T` timeunits in the past.

Second, an authorization rule can be set to block subsequent Internet access in case the weather application has read location data before. An authorization rule as in Listing 2c results in the attribute `location-access` being set to `true` as consequence of location access. Then, the authorization rule requires the evaluation of the attribute constraint upon accessing the networking API. Since the application did access location data before, the attribute value is set and network access is blocked. Compared to the first approach, while achieving better privacy protection, this stricter approach would prevent the application from offering its weather information functionality. Ginger's privacy approach can be summarized as *giving an application as few data as possible, while giving it as much data as functionally required*; i.e., trading off some privacy with functionality.

In AppFence, tainted location data is labeled for on-device use only. If tainted data is about to be transmitted via the networking API, sending of tainted data is blocked. However, if the application is malicious and employs an implicit data flow trough control flow operations [18] it may leak sensitive data. Since the application obtained the exact position, it would disclose the user's exact location to the remote endpoint.

TABLE II
SUMMARY OF FEATURE EVALUATION.

access control features	Ginger	AppFence	MockDroid	YAASE	ConUCON	Apex
conditional access	●	○	○	○	●	●
data obfuscation	●	○	◐	●	○	○
data flow tracking	○	●	○	●	○	○
data minimization	●	○	○	○	●	●
network blocking	●	●	○	●	●	○

not supported (○), partially supported (◐), supported (●)

In MockDroid, access to location data can be mocked, thus resulting in the location data being reported as *empty* to the application. However, such a coarse-grained data obfuscation would render the weather application's functionality unusable.

In YAASE, a filter action can be utilized to achieve an effect equal to Ginger's `location-granularity` obligation. This approach enables fine-grained data obfuscation, thus effectively preventing the user's exact location from being disclosed. YAASE also supports data flow tracking, but suffers from the problem of implicit data flows.

In ConUCON, the concept of obligations provides some privacy protection but requires a device-wide global action. In this use case, upon accessing location data, a ConUCON obligation could require the networking API to be turned off; i.e., powering down the device's wireless communication. Another possibility would be to use a post-update in combination with an attribute authorization rule, similar to the second possible solution of Ginger. However, ConUCON would render the application's functionality unusable in both cases. ConUCON also enables a data minimization technique similar to Apex and Ginger, as described in the following.

In Apex, it would be possible to set a condition for location access such as *read location data 5-times per hour*. That approach is equal to limiting the access frequency in Ginger. However, it is a data minimization technique only and does not prevent the application from accessing and disclosing the user's exact location.

C. Performance

To show that our approach is feasible, we conducted a performance evaluation comparing the Ginger implementation to the AOSP baseline Gingerbread v2.3.7_r1. We implemented a demo application that mimics the behavior of use case II and uses the policy settings from Listing 2b. The performance evaluation was conducted running an Android emulator on an OS X 10.7.3 machine, 2.26 GHz Intel Core 2 Duo, 4GB 1067 MHz DDR3. We let the demo application repeatedly (n=1,000) receive location data and transmit network data, while measuring time instances for each operation.

Mean timing results are provided in Table III. They show that location access causes an overhead of about 17% for Ginger, due to the evaluation of Ginger policies, including

TABLE III
PERFORMANCE BENCHMARK OF USE CASE II

Location Access

	Android	Ginger	Factor
⌀	29.86ms	34.88ms	1.17
σ	15.47ms	14.12ms	0.91

Network Access

	Android	Ginger	Factor
⌀	2.16ms	9.54ms	4.42
σ	4.44ms	4.56ms	1.03

the location obfuscation algorithm. In contrast, network access causes an overhead of about 442%. This relatively high value might be due to the fact that network access needs an additional IPC operation in Ginger compared to a native library call in Android. This could be further improved by implementing the required functionality in native libraries.

VII. CONCLUSION

As the landscape of mobile platforms has already shown, solutions for flexible integration of third party applications provide the opportunity of driving a dynamic development of new applications with promising features. Similar benefits could stem from automotive application frameworks for future telematics platforms. However, the safety-critical context of automotive applications in particular when accessing vehicle sensors or other in-vehicle systems raises the need for reliable access control mechanisms that respect the requirement of minimal driver distraction. To cope with these requirements we introduced Ginger, a generic access control framework for telematics applications. The core access control model of Ginger is based on attributes, conditions, obligations, and post-updates, combined with a simplified trust model and the concept of isolation domains. The resulting framework provides context-aware access decisions as well as privacy protection. Our framework was implemented on top of Android's application sandbox and Binder IPC. We evaluated the feasibility of the Ginger framework in two use cases highlighting the features of privacy protection through post-updates and of context-aware access control. A comparative analysis with related approaches and the results of our performance analysis demonstrate the effectiveness of Ginger and its underlying concepts for the proposed use cases. We believe that an open framework for telematics applications poses numerous advantages for both users as well as for OEMs. However, reliable and unobtrusive access control plays an essential role in such systems which can be provided by our framework. The current limitations of Ginger are the lack of data flow tracking mechanisms as well as missing PAPs in our implementation. We plan to address these limitations in future work.

REFERENCES

[1] Android Developers, "Security and permissions," March 2012. [Online]. Available: http://developer.android.com/guide/topics/security/

[2] Apple, "Security overview," 2011. [Online]. Available: http://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Introduction/Introduction.html

[3] A. Bussani, J. Griffin, B. Jansen, K. Julisch, G. Karjoth, H. Maruyama, M. Nakamura, R. Perez, M. Schunter, A. Tanner, L. Van Doorn, E. A. V. Herreweghen, M. Waidner, and S. Yoshihama, "Trusted Virtual Domains: Secure foundations for business and IT services," IBM Research Division, Tech. Rep., Nov. 2005.

[4] H. Löhr, A.-R. Sadeghi, C. Vishik, and M. Winandy, "Trusted Privacy Domains - Challenges for Trusted Computing in Privacy-Protecting Information Sharing," in *Information Security Practice and Experience*, ser. LNCS, vol. 5451. Springer, 2009.

[5] J. I. Hong and J. A. Landay, "An architecture for privacy-sensitive ubiquitous computing," in *Proc. 2nd intl. conf. on Mobile systems, applications, and services (MobiSys'04)*. ACM, 2004.

[6] A. Abdul-Rahman, "A framework for decentralised trust reasoning," Ph.D. dissertation, Department of Computer Science, University College London, August 2005.

[7] T. Grandison and M. Sloman, "A survey of trust in internet applications," *Communications Surveys Tutorials, IEEE*, vol. 3, no. 4, 2000.

[8] J. Park and R. Sandhu, "The UCON_{ABC} usage control model," *ACM Trans. on Information and System Security*, vol. 7, no. 1, 2004.

[9] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, 2001.

[10] Q. Ni, E. Bertino, J. Lobo, and S. Calo, "Privacy-aware role-based access control," *Security Privacy, IEEE*, vol. 7, no. 4, pp. 35–43, 2009.

[11] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. on Information and System Security*, vol. 4, pp. 224–274, 2001.

[12] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *Security Privacy, IEEE*, vol. 7, no. 1, pp. 50–57, 2009.

[13] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *Security Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.

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

[15] G. Benats, A. Bandara, Y. Yu, J. Colin, and B. Nuseibeh, "PrimAndroid: privacy policy modelling and analysis for android applications," in *Symp. on Policies for Distrib. Systems and Networks (POLICY)*. IEEE, 2011.

[16] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," in *Proc. IEEE Computer Security Applications Conference*, 2009.

[17] G. Bai, L. Gu, T. Feng, Y. Guo, X. Chen, S. Jajodia, and J. Zhou, "Context-Aware usage control for android," in *Security and Privacy in Communication Networks*. Springer, 2010, vol. 50, pp. 326–343.

[18] 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 *Proc. 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[19] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX Conf. on Operating Systems Design and Implementation*, 2010.

[20] G. Russello, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "YAASE: yet another android security extension," in *Proc. 3rd IEEE Intl. Conf. on Privacy, Security, Risk and Trust (PASSAT)*, 2011.

[21] A. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: trading privacy for application functionality on smartphones," in *Proc. 12th Workshop on Mobile Computing Systems and Applications*, 2011.

[22] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing Android-Powered mobile devices using SELinux," *Security & Privacy, IEEE*, vol. 8, no. 3, 2010.

[23] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in *Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011.

[24] N. Reddy, J. Jeon, J. Vaughan, T. Millstein, and J. Foster, "Application-centric security policies on unmodified android," UCLA Computer Science Department, Los Angeles, TechReport 110017, 2011.