ICST Transactions Preprint

Middleware-based Security for Hyperconnected Applications in Future In-Car Networks

Alexandre Bouard[1,*], Dennis Burgkhardt[1], Claudia Eckert[2]

[1]BMW Forschung und Technik GmbH, Munich, Germany
[2]Technische Universität München, Garching near Munich, Germany

Abstract

Today's cars take advantage of powerful electronic platforms and provide more and more sophisticated connected services. More than just ensuring the role of a safe transportation mean, they process private information, industrial secrets, communicate with our smartphones, Internet and will soon host third-party applications. Their pervasive computerization makes them vulnerable to common security attacks, against which automotive technologies cannot protect. The transition toward Ethernet/IP-based on-board communication could be a first step to respond to these security and privacy issues. In this paper, we present a security framework leveraging local and distributed information flow techniques in order to secure the on-board network against internal and external untrusted components. We describe the implementation and integration of such a framework within an IP-based automotive middleware and provide its evaluation.

Keywords: Security & Privacy, Access Control, Middleware, CE Device, Third-Party Application, Automotive Application, Car-to-X Communication, Decentralized Information Flow Control, Dynamic Data Flow Tracking

1. Introduction

During the last two decades, vehicles evolved into very complex systems embedding powerful electronic platforms for various purposes, e.g., safety, infotainment. While still fulfilling their primary goal of transportation means, cars are now offering a plethora of new connectivity interfaces and communicate with numerous external communications partners: the Internet, Consumer Electronic (CE) devices, road-side units and other cars [1]. Like smartphones, the car will soon host Third-Party Applications (TPAs) [2]. Such a connectivity and new features will obviously allow a better customization of the car and a stronger tethering between all on-board and external communication partners. On the other hand, this may raise the threat level and increase the attack likeliness through these newly extended communication interfaces.

Recently, cars have been shown to be vulnerable against simple attacks involving packet sniffing/injection and more complex ones, like buffer overflows [3]. These attacks were performed by attackers having physical access to the car and its on-board network, but later work have show the feasibility to compromise the car through most of its external communication interfaces[4, 5]. In addition, today's automotive applications are mostly developed for a specific platform and for a precise car model. The car manufacturer knows the developer and can therefore set contractually certain responsibilities and testing processes. While not providing a complete security, such a strategy allows the car maker to keep the application integration process under its control. Loadable and on-the-fly installable applications have revolutionized the CE world but may shake up the static architecture of the car. While being mostly foreseen for the infotainment purpose, such applications will get access to Internet, several on-board functions and may secretly compromise the integrity and data confidentiality of the car [6].

At a functional level, limited communications technologies (e.g., Controller area network (CAN), Media oriented systems transport (MOST)) and drastic requirements for low latency and high robustness let only very little space to security. Part of the solution seems to lie in the use of Ethernet and the Internet Protocol (IP) as standard for the on-board communications [7]. A larger bandwidth and mature security protocols will allow to secure the communications between two on-board platforms, but may remain insufficient in order to achieve a holistic solution. Future automotive applications will become more and more complex and partly designed by third parties. They will simultaneously trigger critical functionalities of the car and handle large amounts of data presenting different level of sensitivity. Not considering the whole information security problem,

*Corresponding author. Email: alexandre.bouard@bmw.de

i.e., how information travels through the system, may lead to privacy breaches and, even worse, to safety malfunctionings, which could endanger the passengers' life. In order to keep on producing safe and secure vehicles, car manufacturers need to secure the on-board architecture accordingly.

Information flow control (IFC) is about controlling how information spread into a system and has been successfully applied for distributed systems. Our approach proposes to make use of the middleware layer to enforce the security. Through this layer, on-board applications exchange security metadata expressing their security concerns/requirements and can enforce policies relevant for the decentralized information flow control model (DIFC) we developed. Dynamic data flow tracking (DDFT) engines allow to taint data of interest and to follow their propagation within a running application in order to monitor and control its potentially malicious behavior. We chose to use such techniques to secure the integration of TPAs and couple them to our DIFC model via the middleware. For comparison, we design and evaluate a second security solution for integration of TPAs, which makes use of isolation/virtualization techniques and a DIFC-based network input/output monitoring. At the edge of the on-board network, a security communication proxy filters inbound and outbound communications based on pre-defined DDFT and DIFC policies and allows a secure and privacy-aware tethering of online services and external devices.

The main contributions presented in this paper are:

- a DIFC authorization model regulating on-board communications and integration within the middleware logic;
- a customized DDFT environment based on *libdft* [8], which locally monitors TPAs and is coupled to the DIFC framework;
- a security architecture for the communication proxy, extended from our previous work [9] and complying with the enforcement of DIFC and DDFT rules;
- a prototype implementation of our security framework integrated within our automotive adaptation of the IP-based middleware *Etch* [10].

The rest of the paper proceeds as follows. After having given a brief overview of today's automotive security and threats to it, we introduce the main security and privacy goals of our work as well as related work in Section 2. Section 3 describes our architecture for on-board security middleware and communication proxy as well as introducing exemplary *Security & Trust Levels*. Then Section 4 introduces our model for DIFC and DDFT techniques in detail, after which Section 5 describes our implementation. Section 6 proposes an evaluation of our prototype and security concepts and finally Section 7 concludes this article.

2. Background and Related Work

In this section we provide some background information about the automotive on-board architecture and its security shortcomings. We then define the threats and goals we consider in this work, as well as some attack scenarios.

2.1. Current & Future On-board Network

Today, the on-board network of a premium vehicle includes up to 70 interconnected electronic control units (ECUs). The ECU network is organized around specific domains, e.g., infotainment, power train, and is interlinked via several communication bus technologies, e.g., CAN, MOST, which necessitates complex application gateways for interoperability. On-board automotive applications are divided in elementary blocks over diverse ECUs and exchange braodcasted signal-based messages. Recently, some research work highlighted numerous security issues due to a lack of protection on the communication bus and poor ECU implementations. Common attacks have been successfully performed on both local [3, 11] and remote interfaces [4, 5]. Plaintext communications without authentication mechanisms allow an attacker with access to the on-board network to easily sniff and inject packets in order to misuse internal protocols. A lack of input validation of some ECUs allows to bypass authentication mechanisms via buffer overflow techniques in order to reprogram the platform. In some cases, the poor software implementation does not reflect the published standards and allows to directly activate the reprogramming mode of the platform [3]. Remote attackers may also be able to compromise communication interfaces like the Bluetooth interface via a brute force attack or the GSM access gateway via buffer overflow attack in order to gain access to the on-board network [4].

For tomorrow, the use of Ethernet/IP as on-board communication standard has been strongly investigated and could be part of the security answer [7]. First a larger bandwidth will allow to comply with the requirements of future automotive applications [12]. It will allow to exchange large objects like environment model for driving assistance or infotainment content for audio and video purposes. Secondly, mature security protocols, developed specifically for the Internet world, will be instantly applicable and should provide a suitable protection for all bus communications. With Ethernet/IP, automotive applications will remain complex and distributed, but the design of engineering-driven middleware will greatly simplify their management.

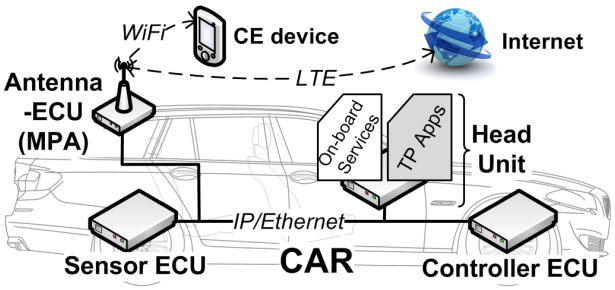

Figure 1. On-board Network Architecture and considered scenarios. Solid right-angle lines represent the wired on-board network. The dashed arrows represent external communications over different wireless networks.

Such middleware will abstract and automate the network addressing and security enforcement [13, 14]. The clear separation between middleware and application logic will allow car manufacturers to separate the security logic from the application part and to significantly decrease the risk of buffer overflows thanks to security programming and code validation techniques [15]. In addition, car manufacturers will centralize most external communication interfaces (e.g., LTE, WiFi) around a multiplatform antenna-ECU (MPA) [16]. The MPA design offers them the opportunity to setup a single security gateway for all Car-2-X (C2X) communications, easy to verify and maintain.

2.2. Threats and Goals

Today's car are facing several challenges. Their functional behavior relies on complex software, which are optimized to run on resource limited-platforms, rarely updated and process a significant amount of data of different sensitivities. Attackers are already taking advantage of defects in the application logic and in weak security mechanisms. This usually results in privacy breaches (vehicle tracking [5]), in endangering the car's integrity (unauthorized reprogramming of an ECU [3]), and even worse in threatening the life of the its occupants (partial brake disabling [3]). If nothing is done, the emphasis of the use of C2X communications, more and more complex on-board applications and the integration of TPAs will only increase the security risk. For the moment these attacks are mostly performed by the research community, but this could quickly change. The use of Ethernet/IP and a security middleware solves some of our issues but do not cover the whole information security problem. These do not address the threats related to unintentional bugs of the on-board applications. Besides, they do not consider unfair authorized parties, internal (e.g., TPA) or external (e.g., CE device), trying to bypass security policies, i.e., by trying to access or leak information they have no authorization for. This work aims at improving the information security in cars and at addressing the threats we just mentioned.

Attacker model: In this paper, we consider an attacker both acting internally and externally. Externally the attacker has access to all communication interfaces, can potentially get authenticated, and have her messages forwarded by the proxy to the inside. In addition, she can get access to the on-board network, sniff the traffic and inject new packets. She may as well write a TPA and have it installed in the car through a legitimate channel, e.g., the application store of the car manufacturer. We assume the attacker to be bounded only polynomially in computational power and storage, so that the current cryptographic primitives, e.g., AES, RSA, can be assumed to be secure, since there are no known algorithms able to break them in polynomial time. Thus she cannot break strong cryptographic protocols or successfully guess random numbers. We restrict the attacker to software-based attacks so that she cannot physically tamper ECUs, e.g., read or flash memory content. This work does not consider denial-of-service attacks.

Use Cases & Attack scenarios: Our use cases are depicted in Figure 1 and consider both internal and external untrusted components, over which the car manufacturer has no control. We define as service a group of on-board application sharing the same middleware layer. We take the example of a TPA running on the Head Unit (HU). It communicates with several on-board services and with the Internet and some CE devices over the MPA. Internet services and CE devices are authorized to get access to some on-board services and can get authenticated. The TPA is considered as conform with the internal application programming interface (API) of the car. However it may present a poor implementation exploitable by an attacker. We here mainly focus on attack scenarios leveraging the TPA to (1) compromise the integrity of the car or (2) leak confidential information.

1. **Integrity Attack Scenario:** A malicious TPA may send bogus packets on the on-board network (e.g., a shellcode) or access/modify locally resources of the HU (e.g., filesystem) in order to disturb the car functioning. Secondly, a malicious external communication partner may send bogus messages to the car, forwarded by the MPA and try to disturb the car as well.
2. **Confidentiality Attack Scenario:** A malicious TPA may get access to sensitive data, stored on the HU (e.g., the home address of the driver in the navigation module) or received from another service (e.g., preference settings of a user from the seat controller). Even without the permission, the TPA may try to send them outside, either directly over the proxy or through an intermediate step, for example a buggy

service communicating with the outside. Otherwise as previously, an external communication partner may leverage a bug of an on-board application and try to retrieve confidential information it should not get access to.

Assumptions: Next-generation ECUs will take advantage of security middleware able to establish communications channels over strong security protocols like IPsec [14]. Each of them will be equipped with a hardware security extension providing key storage and secure boot [17]. Consequently, we assume that after ignition of the car, the middleware, the operating system (OS) and the hardware are not compromised and stay so during the runtime. Thus, we trust the middleware of every service to establish secure communication channels with each others and to enforce suitable security mechanisms when it is required and expected.

2.3. Security Architecture requirements

With regard to our automotive context, we define the following additional security challenges/ requirements. Even when enforcing security, the car should provide performances (for high throughput and large bandwidth) and a robustness at least equivalent as they are currently. Security solutions should be optimally performing on all platforms, even on the resource-limited ones. Our solutions should not require regular updates or financial extra cost. The security should be easy to manage and should not increase the application complexity for all application developers and end-users.

2.4. Related Work

Cars already communicate with our smartphones via USB, 3G or Bluetooth. Depending on the standard, traditional challenge/response schemes ensure the access to basic web browsing, car information, phone- and audio-functionalities. More critical features like remote door opening/locking are preformed via GSM or 3G through a server of the car manufacturer. The server acts like a firewall. However this solution is expensive, not scalable on the long term and may not be secure [18].

Industry projects: Until recently, automotive security has focused on anti-theft devices such as immobilizer and secure RFID transponder for car key. But the newly highlighted security issues and an increasing use of C2X communications reoriented the academic and industry research toward automotive holistic security solutions. The EVITA [19] project and its follow-up SEIS [7] aimed at securing the on-board network. They both proposed a modular framework establishing internal secure communication channels and leveraging secure hardware platforms. On the other hand, a project like SeVeCom [20] addressed the security issues of future vehicle communication networks. They designed C2X protocols using encryption and authentication mechanisms, which got implemented on the V2X platforms of the sim^{TD} project [21]. But none of these projects really formalize the transition of data between outside and inside or consider the damages that external data could cause on the inside. They all rely on strong security components on the edge of the on-board network performing the enforcement of static access control lists (ACLs).

Securing a corporate network presents some similarities with the automotive context, e.g., when integrating mobile devices. Their approaches make use of strong authentication mechanisms and device integrity measurements in order to establish network connections and VPN tunnels [22]. However they usually lack specifications for a secure resource- and data-management. In the context of SEIS, [9] proposes a proxy-based architecture for a secure CE-device integration. The proxy evaluates the security level of the device and communication and shares it with the ECUs. We chose to extend these concepts to our architecture and to complete it with a more formal security model.

About IFC: IFC is a form of mandatory access control. Resources (e.g., documents) and principals (e.g., persons) having access to them are given a label, i.e., a clearance level. A label-based partial order defines whether the access is authorized or not [23]. DIFC extends the IFC concepts [24]. A resource (e.g., an application) can be allowed to divide its access rights and create new labels to manage with more flexibility its access control. DIFC was adapted at the granularity of a process: processes are separated between trusted and untrusted during runtime. Label-based rules are enforced locally by a customized OS [25, 26]. But for distributed applications, OSs can exchange their labels through the network as well [27]. However these approaches are too fine granular and suffer from a too significant performance overhead. For a lighter approach, we chose to enforce DIFC labels only on on-board communications between services. Exchanging labels to enforce IFC is not new. For Pedigree, a central server and customized network switches distribute and enforce IFC policies on every network communication [28]. However in-car applications are distributed over ECUs with different OSs/hardwares. In order to reduce the risk of errors, latency and maintenance complexity, the DIFC cannot rely on any central entity and cannot be enforced in the OS or its hardware. We therefore chose to enforce DIFC at the application level, especially in the middleware layer.

About DDFT: For more security, smartphone applications are tested before their release on an online application store and are usually isolated from each other thanks to the sandboxing mechanisms of the

mobile OS. However it is not flawless [6, 29]. We therefore consider 2 other runtime options. Either (1) we isolate the TPA from other on-board services and monitor its inputs/outputs, like in Section 4.1. Otherwise (2) we monitor the TPA itself and what happens during runtime, for example by using DDFT techniques, like in in Section 4.2. DDFT allows to taint and track data of interest within a running application/system and have been successfully applied for various purposes, e.g., malware monitoring [30] or privacy-aware smartphone monitoring [31]. DDFT offers two approaches: monitoring the whole host [31] or just one process/application [8]. Considering our requirements for robustness and low latency, we orient our work toward a lighter approach, the second one. This solution causes some overhead but does not require any OS or source code modification and has been already used for distributed [32] and automotive [33] environments. But monitoring or isolating TPAs will not be sufficient. Thus, we decide to combine the isolation/monitoring techniques to our DIFC model via the middleware.

3. On-board Security Architecture: Secure Middleware and Communication Proxy

As mentioned in the introduction, Ethernet/IP will be intensively used by car manufacturers as standard for the on-board communications. The rest of this section provides an overview about our secure middleware layer (Section 3.1), the architecture of our security proxy (Section 3.2), and) a taxonomy for external untrusted communication partners (Section 3.3). Finally, Section 3.4 discusses the benefits of such a security architecture and pinpoints its shortcomings.

3.1. A Security Middleware Extension

By definition, the middleware abstracts the communication interfaces and hides the network complexity from the application logic. It may as well automate the security enforcement and therefore allow the application developer to be completely security-unaware. We present here an architecture for a security middleware extension (SME) [14], that can be easily coupled to any middleware layer.

Figure 2 presents the three-layer SME architecture we are considering here. Such a modularization offers enough adaptivity and combination possibilities to comply with all the different security levels required by our use cases. For example, a simple temperature sensor, which only sends information to the engine controller, will not provide as much security features as the HU which deals with multiple communication partners and very untrusted data. The security layers are organized as follows:

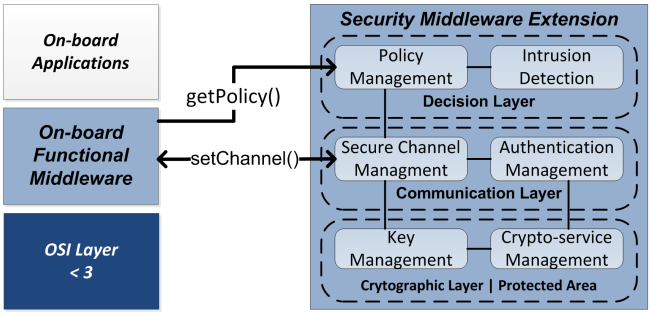

Figure 2. Architecture of the security middeware extension and its association to a functional middleware.

- *The decision layer* provides security decisions by means of static policies. These policies mostly regulate the establishment of a communication channel between 2 on-board platforms and the access to all resources present on an ECU. It provides a direct API *getPolicy()*, which is available from the functional middleware level and returns the policy decision. In addition it may include functionalities for network monitoring and intrusion detection.
- *The communication layer* provides the security plug-ins for the communication protocol implementation and associated filtering mechanisms. It is accessible from the middleware through the interface *setChannel()* which opens a network socket and allows to specify the chosen security protocol, authentication scheme and encryption strength.
- *The cryptographic layer* is in charge of the key management and the cryptographic processing, i.e., the data encryption/decryption and signature generation/verification. For more security, this layer may be included in a hardware security module for protecting the key material from an attacker and providing integrity mechanisms such as remote attestation.

Due to the risk of latency and errors, the configuration of the middleware and its SME is statically set up during the vehicle assembly or during periodic system updates. It mostly concerns the definition of security associations for IPsec channels between two ECUs and their associated preshared keys.

3.2. A Security Communication Proxy

Controlling information flows in distributed systems like cars is essential for a holistic security solution. ECUs internally exchange genuine packets and therefore only require secure communication channels and simple access control mechanisms. But the integration of untrusted devices and online services, over which the car manufacturer has no control, necessitates a more complex authorization model. In order to get the best

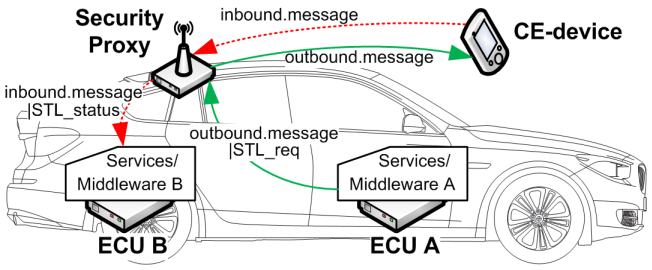

Figure 3. STL life cycle.

user experience, the car manufacturer cannot ban any type of CE device or trendy online community service, only because they do not comply with certain security prerequisites. It can only restrict their access to the car and adapt the security mechanisms on a case by case basis. In addition, ECUs communicate with the outside behind the MPA. Being more than just a super-antenna, the MPA is a complete ECU and decouples the communication between on-board and outside like a NAT router. Such decoupling allows the car to use a unique optimal security protocol for the on-board network, while letting the choice of the outside protocol to CE- and web-application developers. It also requires the MPA to be able to support the "blind" ECU, e.g., by providing it information about the external device or service, the identity of the concerned user, the used wireless protocols or the likeliness of a security threat, so that the ECU can enforce the right security decision.

For this purpose, we developed an application-independent in-band middleware protocol, allowing internal exchanges of security metadata. Concretely, the middleware header is extended with a new field, specifying the security and trust context in which data are or may be exchanged with an external peer. Instead of directly considering the privacy aspect of any single piece of information, we focus on the trust we grant the peer and quantify that. The security aspect characterizes how secure the external communication protocols are. On the other hand, the trust aspect characterizes how trustworthy the remote device or online service is considered to be. We call this context *Security & Trust Level (STL)* and provide its precise evaluation in Section 3.3. In order to distinguish whether the STL defines the current communication situation or whether it defines a required situation to send a message out, we define the STL_{status} and the STL_{req}, respectively. The life cycle of the STL as represented in Figure 3 is explained in the following:

The communication security proxy: The proxy is implemented on the MPA and stands in the middle of every communication happening between an internal entity and the outside. Unlike traditional NAT routers simply forwarding IP packets, the proxy really decouples these communications and acts like a translation interface between external and internal middleware-based protocols. It dynamically manages all external communication channels and their security features, i.e., encryption process, key management and authentication schemes. Internally, the proxy communicates with ECUs over static IPsec channels. Future C2X use cases foresee exchanges of big objects at a high frequency. The proxy will not be able to perform deep packet inspection and is as a consequence application-unaware. The proxy evaluates for each external entity a STL_{status} and adds it in the middleware header of every inbound packets. In addition, the proxy enforces on them a first coarse domain-based filtering, for example an online social network service will not get access to any service of the power train management domain. Inversely, for every outbound message, the proxy, before forwarding it, makes sure that every received STL_{req} is conform to the actual STL of the communication situation. Section 3.3 provides more information about STL-based policies.

Middleware & STL: The middleware of every on-board service intensively uses the STL concept. All STL-based policies are managed in its *Decision Layer* and are invoked when receiving or emitting a packet. Based on the received STL_{status}, the security middleware decides whether it is safe and authorized to process such a packet. Depending on the ECU capacity and the STL, the middleware can adapt its security processing and pass the data through a security parser, e.g., access request to a SQL database, or run the data in an isolated environment, e.g., JavaScript code in an isolated web-browser. A received STL_{req} determines the data sensitivity and requires the middleware to decide whether its applications are allowed to receive such data. Inversely, when sending a packet to the on-board network, the middleware automatically extends the message header with a STL_{req} reflecting the sensitivity of the payload information, i.e, industrial secret or private information. Any communication is concerned, since a multicast address could inadvertently forward a packet with private data to the proxy, i.e., to the outside. The applications on top of the middleware are totally STL-transparent, the STL enforcement happens in the middleware. Like most policies on the ECUs, the STL-based policies are defined by the car manufacturer at design time and are statically set up in the SME.

3.3. The STL Taxonomy

Section 3.2 introduced the concept of STL. It defined it as the security and trust context in which data are ("$_{status}$") or should be ("$_{req}$") exchanged with the outside. The rest of the section proposes an evaluation of (1) its security aspects, (2) its trust aspect and (3) its enforcement.

1) SL definition: We define the SL as a qualitative characterization of the security strength of an external communication protocol. The SL is characterized as follows:

- ***SL=0*** Communication providing no security or presenting exploitable design flaws.
  Example: Plaintext; WEP encryption; TLS+DES or RC4 with a 56-bits key;
- ***SL=1*** Communication providing strong authentication of the external peers and data integrity (i.e., against unauthorized modifications).
  Example: WPA2 encryption; Message in plaintext protected by HMAC-SHA1;
- ***SL=2*** Communication as secure as SL=1 and, in addition, providing strong confidentiality (i.e., one secret key per user, no shared key between users).
  Example: TLS+AES; IPsec+AES;
- ***SL=3*** Communication as secure as SL=2 and assuring the presence of a secure hardware element protecting the cryptographic materials of the external peer.
  Example: SL2-protocol + remote attestation.

2) TL definition: We define the TL as an abstract representation of how trustworthy the data sender and receiver are. The notion of trust is usually defined as a mix between 3 components: reputation, reliability and security [34]. The security has already been considered, thus the TL focuses on the 2 remaining ones. The evaluation criteria of the TL should be clear and easy to assess. We consider that data may only be misused, if they are (1) physically and (2) juridically accessible, i.e., (1) if the data leave the car and (2) if the receiver is legally allowed to endanger the user's privacy (e.g., data selling/forwarding, data stored on an unprotected server). The TL should reflect such risks and is evaluated based on the following criteria:

- **Criterion 1 (Cr. 1) "Local Usage":** determines whether the data are limited to an on-board usage only.
- **Criterion 2 (Cr. 2) "Anonymization":** determines whether data have to be anonymized, when released out, i.e., whether an external receiver may be able to trace back the identity of the car or of the user.
- **Criterion 3 (Cr. 3) "Jurisdiction":** determines whether the external receiver is considered as a safe "place of jurisdiction" (POJ), i.e., whether the servers hosting the online service are located in a country imposing a regulation protecting the user's privacy.

In order to determine the TL of an external peer, we use the simple binary decision tree of Table 1. Every criterion is evaluated iteratively, a "true" answer stops the process and sets the TL value. Highly sensitive data, like industrial secrets, should never be released

Table 1. Binary decision tree used for TL evaluation.

	Cr. 1 →	*Cr. 2* →	*Cr. 3* →	TL
Case 1	true	-	-	3
Case 2	false	true	-	2
Case 3	false	false	true	1
Case 4	false	false	false	0

(Cr. 1=true) and thus are assigned TL=3. Very sensitive data, like the car position, should be able to leave the car but not endanger the driver's privacy (Cr. 2=true) and therefore should be anonymized (TL=1). An application for local hazard warning, broadcasting the position of an accident (and also the position of the car) should be able to do so, only if the emitted packets do not include traceable information about the user's or car identity. Data with a low sensitivity, like the driver's username and settings, can be released without anonymization but only to services presenting a safe POJ (Cr.3=true, TL=1), for example a banking service, whose servers are in Germany. A service presenting an unsafe POJ, like Facebook in USA, should only be able to receive nonsensitive data with TL=0. While Cr. 1 and Cr. 2 are easy to assess and enforce, Cr. 3 needs to be determined based on recommendations from international privacy experts [35].

The TL taxonomy is quite simple and provides an efficient way to control the data release with the outside. But further tests with more use cases should be performed. The TL criteria are very coarse, but give to the car manufacturer a simple way to configure a "by default" privacy/trust-aware behavior. For a more flexible usage, the user should be able to change the assigned TL of an online service, like a social network of her choice, and allows it to receive some data with TL=1 as well.

3) STL enforcement: We consider security and trust as two independent variables necessitating separated enforcements and evaluations. Indeed, anonymized data with a TL=2 may be sent with a SL=1 in plaintext (e.g., local hazard warning scenario), while data with a TL=1 may be sent with a SL=2 (e.g., banking scenario), because the user does not want such information to be eavesdropped. As a consequence, we define the STL as the concatenation of the SL and the TL, i.e., STL=(SL, TL). For an easy management, we limit ourselves to 4 SL values and 4 TL values, coded over 4 bits in the middleware header.

Concretely, data with a STL_{req}, which arrives on the proxy will be allowed to be released to an external service or device X: 1) if X complies with the conditions of the received TL and is authorized to receive data with such a TL and 2) if the communication with X provides a SL greater than or equal to the received SL. However, such conditions may be too constraining

and may never allow certain data to leave the car. Declassification methods allowing to assign a lower STL to some data or to just add an exception on the proxy should be possible. But those methods should only be part of use cases predefined by the car manufacturer and if necessary should involve the driver's decision, e.g., if it is her private data. Further considerations about declassification methods are not provided in this work.

STL-based polices are statically implemented in the ECUs and do not require any update. Either the on-board service generates the data to be sent and associates its own STL_{req} depending on the appropriate policy, or the ECU received the data from another ECU and before forwarding them, labels them with the received STL_{req}. The proxy should regularly receive notifications to update the TL of new external services and the SL of new or flawed communication protocols. The CE device case is bit particular, as it gets authenticated by the proxy and is assigned a STL_{status}. This STL depends on the used connection protocol for the SL and is assigned a TL=1, since we assume that the user's device is under her control and is therefore safely handling her private data.

3.4. Intermediary Discussion

Enforcing the security in the middleware provides a clear separation between security/networking management and application logic. Such an approach abstracts the security model and makes it more efficient to enforce and easier to verify. Security programming methods can be easier to apply and can solve many security flaws related to stack pointers overwriting attacks, e.g., buffer overflows. However our STL model has shown its limits: it implicitly considers a unique user in the car and cannot handle the information of more than one simultaneous passenger or driver, respectively. Adding the unique ID of a user to the STL label may not be sufficient to secure on-board information flows. For the moment our approach do not consider any TPA or any solution for constraining it to respect authorized information flows and to not act maliciously. The following Section 4 provides a more formal security model completing our architecture and STL-based enforcement.

4. Controlling Information Flows in Cars

IFC is about monitoring the in-car propagation of data defined as data of interest. Such data may be interesting to track within the car either because they are sensitive data and their release should be controlled, or because their integrity is essential to preserve, e.g., when processed by a safety-critical application. We propose to monitor the information flow at two different levels: at the network level between on-board applications in Section 4.1 and within the TPA in Section 4.2. Like for our previous approach, we chose to enforce security and IFC via the middleware.

4.1. Decentralized Information Flow Control

We defined the term services as a group of on-board applications running on top of a same middleware layer. The applications belonging to a same service share the same security concerns for confidentiality (e.g., because they share data of same sensitivity) and integrity (e.g., because they trigger the same critical mechanisms). For this reason, a security label, characterizing such concerns, is assigned to every service perpetrating network exchanges. The middleware, independent from the on-board applications, is in charge of monitoring and labeling the on-board network communications. Comparisons between the labels of 2 services allow to protect the integrity and the confidentiality of the information they process, e.g., by isolating corrupted data from a critical application or preventing an unauthorized information disclosure. The rest of the section presents our formal DIFC model, inspired from [27].

Security Labels. One label is assigned to each service. A label includes two subcomponents: a secrecy label S and an integrity label I. S and I are two sets of tags. A tag is defined as a security concern of an individual about the secrecy (in S) or the integrity (in I) of the information they process. For the ECUs, a tag is a unique value implemented as a bit-string. We refer to it with a symbolic name like b_s or b_i, where the subscripts s and i designate the concerns for respectively the secrecy and the integrity and b the principal (e.g., the service b or the CE device b), whose concern is characterized. We call service tag and user tag a tag which designates the security concerns of respectively an on-board service or a user and her CE device. The secrecy tags are "sticky", i.e., information from a service labeled with b_s cannot flow to a service lacking it. The integrity tags are "fragile", i.e., information from a service labeled with b_i can flow to a service lacking but will then lose its label.

The labels establish a lattice enforcing a form of mandatory access control, as shown in Figure 4. Formally, information from a service A labeled with S_A and I_A can flow to a service B labeled with S_B and I_B if and only if the tags of S_A are included in S_B and I_A contains the tag of I_B. The partial order "$\prec$" (pronounced "can flow to") is defined as follows:

$$L_A \prec L_B \text{ iff } S_A \subseteq S_B \text{ and } I_A \supseteq I_B,$$

$$\text{where } L_A = (S_A, I_A) \text{ and } L_B = (S_B, I_B)$$

However the on-board services are distributed over several ECUs and do not know each other's label. We therefore chose to label the messages as well. When service A with label L_A sends a message M to service

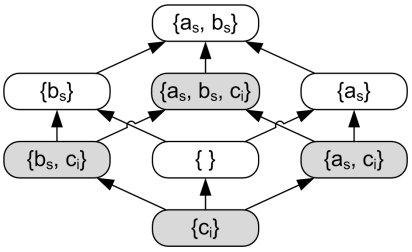

Figure 4. Label-based lattice. This example includes 2 secrecy tags a_s and b_s and one integrity tag c_i. Boxes represent service labels. Grey boxes show labels including c_i. Arrows link labels between which information can flow.

B with label L_B, A assigns to M the label L_M such as $L_A \prec L_M \prec L_B$. Assigning such a label to M allows A to disclose the information first to M ($L_A \prec L_M$) and to make sure that B will receive it if it fulfills the condition $L_M \prec L_B$, i.e., $L_A \prec L_B$ by transitivity of the partial order $\prec$.

Concretely in our scenario, the HU stores data belonging to different users with different secrecy tags, so that only the appropriate TPA and CE device can receive the data of a particular user.

Tag Ownership. If an information could only follow the partial order $\prec$, labeled messages could only travel to services with a greater or equal secrecy level and some information would never be able to leave the car. Our DIFC model decentralizes the exception management. In addition to its label, each service is assigned a set of tags called ownership O. A tag t included in the ownership O of a service S allows S to derogate to the restriction imposed by t. We say that S owns t. Obviously, no service should own all the tags of the system. Each service should rather own a minimal set of tags in order to remain functional.

We note $\prec_O$ (pronounced "can flow to, given O") the new partial order including the concept of ownership. So that an information can flow from A labeled with L_A to B labeled with L_B given the ownership O, given (1) the secrecy tags of S_A should be included in S_B except for the secrecy tags of O; and (2) the integrity tags of I_B should be included in I_A except for the integrity tags of O. We formally define $\prec_O$ as:

$$L_A \prec_O L_B \text{ iff } S_A - O \subseteq S_B - O \text{ and } I_A - O \supseteq I_B - O,$$

$$\text{where } L_A = (S_A, I_A) \text{ and } L_B = (S_B, I_B)$$

Practically, a service A with ownership O_A can send a message to a service B with ownership O_B if and only if $L_A \prec_{O_A} L_M \prec_{O_B} L_B$. For example, a TPA will not be given any ownership and as a consequence will not be able to omit restrictions imposed by its label, e.g., it will not be able to bypass the secrecy tag of a user in order to leak her private data. On contrary, the proxy will be given ownership of the secrecy user tag in order to be able to communicate simultaneously with several CE devices requesting data labeled with different secrecy tags. We consider the proxy as providing a high security level and therefore we trust it to use its ownership in a secure manner.

In order to express a new security concern, during runtime a service can create and own a new tag. At its discretion, it can grant the ownership to other services. For each new user U, the proxy generates a new secrecy tag u_s and grants it to the HU, so that the HU can label and protect the private data of U.

Dynamic Label Assignment (DLA): A DLA is an explicit request from a service A to another service B to increase the label of B with a new tag. As mentioned in the Section 2.4, we consider here a TPA enclosed in an isolated cell on the HU. Like for a "black box", the HU can only monitor the inputs and outputs of the isolated cell and therefore of the TPA. At first the TPA is empty labeled without any ownership and thus cannot receive any sensitive, i.e., secrecy-labeled, information or contact integrity-critical functions. For example, in order to exchange private data of the driver d, the HU, which owns d_s, imposes per DLA the TPA to extend its label with d_s. The TPA cannot take the tag d_s out of its label later and is therefore only able to send messages to services including d_s in their label or ownership. The TPA is label unaware and does not manage its own label. Instead, a trusted dedicated service of the HU is in charge of it and filters all inputs/outputs of that TPA.

Automotive DIFC Architecture. We chose to monitor network communications and not every process of the ECU in order to not suffer from a too big overhead and to limit the risk of errors. Services are isolated from each other in their own address space or are on different ECUs. Applications of a same service share the same security concerns and the same middleware layer. We therefore chose to label them together and to rely on the middleware layer to enforce the label-based conditions. As shown in Figure 5, applications in different services communicate through their respective middleware. The Secure Channel Manager and the labeler are part of the *Communication Layer* of the SME. The Secure Connection Manager provides the logic and protocol implementation to establish secure communication channels. The labeler extends the header of each message with a field for the label and enforces the partial order $\prec_O$. The applications of a service are totally label-transparent and are not involved in the DIFC security process. The rest of this subsection provides more information about label assignment and policies.

Label assignment: At first, the label of each service S includes the integrity and secrecy tag (s_i and s_s) characterizing its own security concerns. The assignment of additional tags in the label or ownership

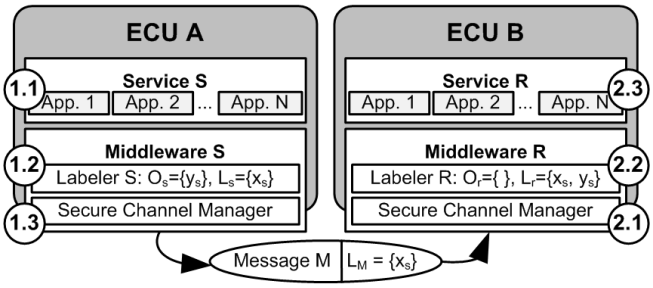

Figure 5. DIFC-based communication between 2 services R and S. L and O are the label and ownership of a service. x_s, y_s, and x_i are secrecy and integrity tags.
1.1 - App.1 of service S sends the message M.
1.2 - Middleware S labels M such as $L_S \prec L_M$.
1.3 - Middleware S sends the labeled message M.
2.1 - Middleware R receives the labeled message M.
2.2 - Middleware R checks whether $L_M \prec L_R$.
2.3 - App. N of service R receives the message M.

	Shadow memory, taint of		
	x	**y**	**z**
0: init();	(0,0)	(0,0)	(0,0)
1: buffer **x** = receiveBuffer(from_a_ECU);	(1,1)	(0,0)	(0,0)
2: buffer **y** = readBuffer(from_sensitive_file);	(1,1)	(2,2)	(0,0)
3: buffer **z** = proccessBuffers(**x**, **y**);	(1,1)	(2,2)	(1,1)or(2,2)
4:	(1,1)	(2,2)	(1,1)or(2,2)
5: write(**z**, in_output_file);	(1,1)	(2,2)	(1,1)or(2,2)
6: sendBuffer(**z**, to_another ECU);	(1,1)	(2,2)	(1,1)or(2,2)

Figure 6. Example of code with data dependencies (left side – in bold, the data to taint) and its taint propagation (right side).

are then specified during design phase by the car manufacturer and depend on the use cases the service is involved in. An example of assignment is provided for evaluation in Section 6.1. During runtime, the proxy is the only service able to generate new tags related to new users and to grant them to the relevant ECUs. We do not consider the addition of any new on-board service, after the car left the assembly line.

DIFC label-based policies: These are managed by the *Decision Layer* of the SME. They provide the decisions to enforce in the labeler, e.g., whether to pass a received labeled message to the applications or which label to add to the output messages. Like for the label assignment, these policies are defined by the car manufacturer during the design phase.

4.2. Dynamic Data Flow Tracking

Our second approach for TPA integration makes use of DDFT techniques. Like for a "grey box", this allows us to get an insight of what happens during the execution. These techniques provides an efficient way of detecting data of interest and to track their propagation within a running application.

Tracking and Controlling the Execution. DDFT tools monitor each machine instruction performed by the TPA and detect every system call and every data flow between registers and memory. They usually make use of dynamic binary instrumentation (DBI) frameworks like Intel's Pin [36] in order to inject custom code within the execution for a policy enforcement. They can therefore raise a warning or stop the execution in case of an application behaving in contradiction with one of their policies. When they are configured properly, they allow to eliminate numerous attacks related to stack pointers overwriting, e.g., buffer overflow [37], string format [38] and return-oriented programming [39] attacks. In order to explain how DDFT tools proceed, we focus on the following 3 points: taint sources, intra-taint propagation and taint sinks. For the rest of this section we consider the pseudocode provided in the left part of Figure 6.

Taint sources are programs or memory location, through which data enter the application. If recognized as data of interest, they are tainted in the shadow memory. The shadow memory is a mapping between the actual memory of the application and its taints. In our scenario, we identify as sources every traditional input/output channel potentially used by the TPA, e.g., inter-process communications (like pipes), filesytem and network socket. Concretely, the DDFT tool monitors the functions "receiveBuffer()" (line 1) and "readBuffer()" (line 2) and taints their returned buffers "x" and "y" accordingly.

Taint propagation: All along the application execution, tainted data are tracked, while they are altered and processed. An example of taint propagation is given in the right part of Figure 6. In our example, the function "processBuffers()" (line 3) produces out of two tainted buffers the buffer "z" that should be tainted as well. Originally, DDFT tools were about detecting security attacks and a simple binary tainting (i.e., one bit of the shadow memory tainting a byte of the real application memory) was enough to detect whether untrusted data were overwriting critical parts of the stack. However, our scenarios consider data of different sensitivity that require several taint values. We therefore propose to taint the data according to the STL taxonomy introduced earlier. For reminder, the STL does not consider data integrity, but only considers the security of the communication and the trust of the external communicating entity. Since the car manufacturer does not control the development of the TPA, every output of the TPA has to be considered as potentially dangerous to process.

Taint sinks are programs and memory locations, where the presence of a taint is checked and where a policy may be enforced. The policies mostly concern the decision about passing the data to a function or using the data as program control data, e.g., return address.

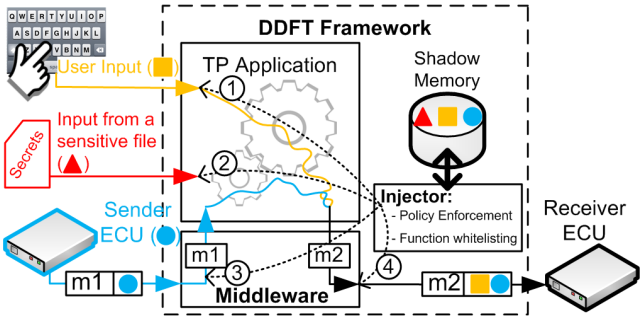

Figure 7. Overview of the DDFT framework in the on-board network. The colored shapes represent different levels of sensitivity, that are expressed by the taint values (i.e., yellow square (1), red triangle (2), blue round(3)). These taints are injected using binary instrumentation (*Injector*). The *Injector* monitors the execution, especially system calls (dotted lines) and the taint propagation between memory and registers. $m1$ and $m2$ are tainted messages sent respectively to and from the TPA. The TPA output $m2$ shows a combination of the sources "round" and "square" but not "triangle" and is therefore tainted accordingly.

For our example, it concerns the fact of writing "z" in a file (line 5) or sending it (line 6).

Car-wide Taint Propagation. DDFT locally ensures security and can detect a security attack or a privacy leakage. For its integration in our car-wide security framework, our DDFT tool instruments the middleware layer as well and takes an active part in the in-band middleware protocol.

The Middleware-based taint propagation is presented in Figure 7. System calls related to the network socket management are monitored and intercepted by the Injector. The header of the incoming message is scanned, the STL field is extracted and its value is used as taint value for the data payload in the shadow memory (bullet 3). On the other hand, before being sent out, the outgoing message gets injected a new STL value relative to the sensitivity of the whole payload (bullet 4). The STL value is determined from the shadow memory by the Injector and carries the most relevant STL found, i.e., the highest SL and TL found. In addition the DDFT tool checks if the TPA is authorized to communicate with the remote service over a dedicated secure channel.

Middleware enforcement: After receiving a TPA message, the middleware layer of the on-board service extracts the STL field and enforces the appropriate policy. The middleware implementation of the service is static and cannot enforce a different policy for each new TPA. Instead, the service middleware relies on the DDFT framework and trusts it to have authorized the communication and to have provided an accurate STL value. Then based on DDFT/DIFC generic policies, applicable to all TPAs, the middleware decides to pass the payload to the aimed application. More details about the DDFT/DIFC policies, linking the STL of the DDFT engine to the DIFC model, are provided in Section 4.3.

DDFT Policies: We identify 2 types of DDFT policies, both of them enforced in the DDFT framework:

a) *Static polices:* Such rules are embedded in the DDFT framework and enforced for all TPAs. They list the authorized services, with which a TPA can communicate. They provide the taint propagation rules (intra- and middleware-based) and the rules related to the taint sinks and taint sources (e.g., how to taint keyboard inputs, which tainted files can be read/overwritten). These rules are static and defined by the car manufacturer at design time.

b) *Dynamic policies:* These rules allow a better customization of the permission given to a TPA and specify additional services or files that can be accessed. They are specific for one application and loaded in the DDFT framework like a Android rule set during the TPA installation. The rule set is evaluated against the existing static rules and needs to be approved and signed by the car manufacturer. Other privacy-relevant policies and exceptions may be directly specified by the driver thanks to on-board configuration interfaces and displays of pop-up windows.

4.3. Coupling DIFC and DDFT

The DDFT/DIFC interface concerns the middleware layers of the services having direct communications with the TPA. It gave them a way to interpret the received STL taint from the TPA based on their DIFC label. Like the DIFC approach, the applications of a service are unaware of this interface.

The TPA and the DDFT framework are not part of the DIFC model and therefore not assigned any label. It allows the TPA to receive information from the whole car without any constraint. The DDFT engine provides accurate STL taints, which gives a precise idea of the output sensitivity. For example in Figure 7, even if the TPA gets as inputs highly sensitive data from a file, the output STL indicates that the payload was processed only from data of lower sensitivity and should be processed accordingly. In comparison to the DIFC approach, DDFT allows the TPA to be more functional, even when handling critical data.

For this DIFC/DDFT approach, the DDFT is DIFC label-unaware and only receives messages with STL taints. It allows us to keep the DDFT tool simple, efficient and generic and not to worry about the different specificities of all car models. Then, the TPAs are most likely to receive information from on-board services and the outside, process them and directly communicate with the outside. The amount of traffic from the TPA to on-board services will remain minor.

As a consequence, the security should be based on a taxonomy oriented towards a secure information release with the outside, like the STL.

Due to the limited number of taints and the privacy risk, the data of only one user should reach the TPA. Therefore all monitored TPA are assigned one user identity (ID). Like for the user tags, these IDs are defined and distributed by the proxy. For each message exchanged between a service and a TPA, a STL and a user ID are added to its header. The DDFT tool filters inbound messages based on the provided ID. The middleware of the concerned service can easily characterize whose privacy is concerned.

Proxy interface: Most sensitive outputs of the TPA will aim toward the outside. In a same manner as explained in Section 3.3, the proxy enforces a STL-based filtering. In addition to the STL considerations, the proxy makes sure that the ID joint to the STL is appropriate for the communication, e.g., the ID of a user U communicating with U's CE device. The proxy provides therefore 2 types of filtering: (1) STL-based for communications with the TPA and (2) DIFC-based for communications with on-board services.

Service interface: All on-board services can send data to the TPA. Their middleware just provide the right user ID (if these data are private) and a suitable STL value. The DIFC labels are enforced to create information flows respecting their integrity and confidentiality. Since the integrity of the TPA outputs cannot be assessed, the transition between DIFC label and STL taint only focus on the information confidentiality.

For messages flowing from an on-board service to a TPA, the sent STL value depends on the secrecy labels of the service:

- for a label involving tags expressing a high secrecy for the car manufacturer, the STL gets a TL=3. The SL is not relevant since the data will not leave the car. The list of high-secrecy tags is defined by the car manufacturer and available in each service interface.
- for a label involving tags expressing the secrecy of a user, but not expressing a high secrecy for the car manufacturer, the STL getss a TL=2 or 1 depending on their privacy level. The SL depends on the user's settings since it is her own data, but as a default value a SL=2 is advised.
- in any other case, the STL gets a TL=0 and a SL=0.

For the opposite case, i.e., when the service receives a message from the TPA, the service middleware decides whether to pass the data to its applications based on the received STL and its own label:

- a STL=(*,3) forces the middleware to pass the data to an application handling highly confidential data that cannot leave the car. Thus an application having a user secrecy tag in its label is not able to receive such data. An authorized service should also include high secrecy tag of the car manufacturer in its label.
- a STL=(*,1) or (*,2) forces the middleware to pass the data to an application handling the private data of a user. Therefore a service with a user's secrecy tag corresponding to the received ID field should be able to get the data.
- a STL=(*,0) indicates that the data are not sensitive and can be passed to all kind of applications.

These last rules do not really consider the SL part of the STL. If sent to the outside, a communication from a service has to go through a DIFC-based enforcement at the proxy level, which is already statically configured. The SL is more relevant for unknown and dynamic cases where security has to be evaluated and configured on-the-fly, like with the TPA.

5. Implementation

This section describes our prototypical implementation combining a middleware-based DIFC enforcement to a DDFT engine.

5.1. The Middleware

As basis of our implementation, we chose to make use of the C-version of the middleware *Etch* [10]. *Etch* is an open-source software project under the Apache 2.0 licence and is considered as a serious candidate for the automotive purpose [13]. Our middleware copes with two types of enforcement: the first one related to the DIFC model and the second one to the DDFT monitoring. We therefore developed two middleware versions. The DIFC version extends the serialization of the middleware header with two fields of 15 bytes, one for the secrecy label, the other for the integrity label. For the DDFT version we extended the header with a 2 integer fields including the 2 values of the STL , i.e., the SL and the TL. The DIFC version is used between all on-board services, while the DDFT one is only used for communication involving the TPA. As a consequence, services communicating with the TPA are aware of the 2 types of header serialization. Like any traditional middleware, the payload serialization and security enforcement are separated from the application logic. *Etch* allows to precise the label of a service and the authorized taints through an adapted interface description language (IDL) and provides an automatic code generation for the enforcement of DIFC and DDFT policies.

We developed a communication proxy similar to the one presented in our previous work [9]. The proxy provides two secure communication interfaces: an external one using the TLS protocol with CE devices and

online services and an internal one for on-board services establishing IPsec communication channels. Internal and external communication partners communicate over a mirror-service of the proxy, which makes the communication decoupling totally transparent for both of them. The proxy is application unaware. It enforces a message filtering based on the labels or taints present in the middleware header of all outbound messages. For inbound message, the proxy adds in the middleware header the corresponding labels or taints, depending on the communication target, i.e., an on-board service or a TPA. The proxy determines the user identity for the establishment of the related user tags and ID values based on the TLS certificate provided by the user's CE device or based on other security credentials that an online service can provide.

5.2. Isolation Cell

Regarding our first approach in Section 4.1 for integration of TPAs, we make use of the XEN® hypervisor 4.2 [40], that we set up on the HU. We run the trusted HU middleware and applications, which are developed by the car manufacturer, in the most privileged domain, called Dom0. The untrusted TPA runs in an unprivileged cell, called DomU. Communications between Dom0 and the DomU occur over a virtualized bridge. The XEN environment enforces a complete isolation of the DomU otherwise. The TPA runs on top of a label-unaware *Etch* middleware. The middleware of the HU service in Dom0 is therefore able to receive both labeled and unlabeled middleware header. The HU service acts like a forwarder and enforces DIFC policies for all traffic going to and coming from the TPA.

5.3. The DDFT Engine

Regarding our second approach in Section 4.2, we make use of the DDFT framework *libdft* [8]. *Libdft* relies on the Intel's Pin for DBI. This tool provides relatively good performance in comparison to other DDFT engines [33] and a well-defined API for a customizable security enforcement. More than just using this framework, we extended its expressiveness and its taint propagation mechanisms in order to deal with the 16 values of the STL. Originally, one byte of memory was tagged with one bit in the shadow memory, it is now one byte tagged by 4 bits. We limited our choice to 16 values in order to keep the size of the shadow memory reasonable and the taint propagation mechanisms efficient. We extended the *libdft* framework with the possibility to differentiate user inputs, i.e., from the keyboard, from file inputs. For the file management, we implemented in *libdft* a system of whitelist, which specifies which file can be accessed by the TPA in reading or writing and for which taint values. We developed a network system call monitoring able to scan every incoming message, extract the taints from the middleware header and taint the data accordingly. In a same manner, the framework can now detect the function calls which send network messages and automatically inject them with the suitable taint values of their payload.

5.4. Testing Environment

We performed this prototypical implementation and the experiments presented in Section 6.2 on three computers: the CE device, the proxy and the HU. They are interlinked with a Gigabit Ethernet and are running a standard 32-bit Fedora Linux on an Intel Atom N270 (1.6 GHz) with 1 GB or RAM. The DomU runs a Debian 6.0 Linux with 256 MB of allocated RAM. While being more resourceful than most embedded platforms of the car, our platforms provide a performance similar to a current HU [41]. Our *Etch* middleware presents a suitable performance, when tested on resource limited microcontroller [13]. Our implementation does not perform extensive modifications of the middleware and therefore should not significantly impact the middleware performance. However this last point should be verified for a more rigorous validation.

6. Evaluation

In order to evaluate our system, we first discuss the security of our concepts and how our system would react during the attack scenarios presented in Section 2.2. Then in the second part, we quantify the overhead of our implementation and discuss the functional requirements presented in Section 2.3.

6.1. Security Evaluation

For this section, we refer to the two attack scenarios defined in Section 2.2. We describe for our two security approaches – isolation and DDFC – how our system would react and which threats can be stopped.

First Approach - DIFC & Isolation. Figure 8 presents an example of label distribution and helps us to understand how DIFC can secure on-board communications and the integration of TPAs and CE devices. The CE device connects to the proxy and gets a direct access to on-board services and an access to the TPA through the intermediary of a HU service. The TPA, always through the intermediary of a HU service, can access the driver's data stored on the HU, receives data from ECU A and triggers mechanisms of ECU B.

Our DIFC model does not propose any service hierarchy, instead all services are distrustful with each other. As a consequence, a successful attack or bug will have a limited impact and will compromise only the tags of the affected services. In our scenario, the CE device of the driver d is authenticated by the proxy,

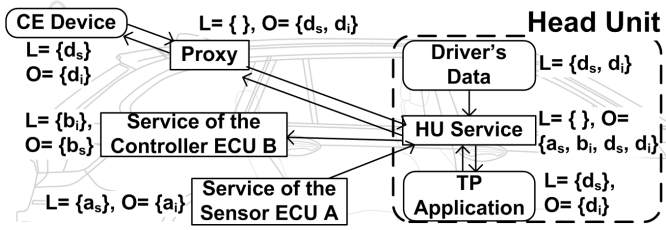

Figure 8. Security scenario. Rectangular boxes represent services running on an independent IFC aware middleware. Round boxes represent IFC unaware applications, devices and files. Solid arrows represent middleware-based communications.

which binds the device to the tags d_s and d_i. The proxy afterwards grants the HU with the ownership of the new tags. The HU performs a DLA of these tags with the TPA, so that the latter can access the driver's data. CE device and TPA cannot be trusted to enforce DIFC rules, that's why their middleware is DIFC unaware and we rely on the HU service and proxy to enforce security.

The TPA is confined to the two driver's tags d_s and d_i. The presence of d_s in its label allows the TPA to get access to the driver d's data on the HU. The ownership of d_i allows the TPA to write on the d's data. The presence of d_i in its label would constrain it to receive data labeled with d_i and would prevent it from accessing for example non-sensitive configuration files. A label with d_s but without d_i would limit it to a "read-only" access. The d_s labeling forces a malicious TPA to only communicate with the d_s-labeled CE device (i.e., belonging to the driver d) and prevents it from communicating with other peers. Because the CE device is bound to one person and labeled with the tags of one user, we do not allow the TPA to be labeled with several user tags.

In a same manner, a CE device is limited to one user and gets access to the data of one user and to non sensitive data. Sensitive data, i.e., from a service labeled with a secrecy service tag, like a_s for ECU A, are blocked by the proxy and are not forwarded to the CE device. Like the TPA, the CE device cannot trigger mechanisms of services with an integrity service tag in their label, like b_i for ECU B. The proxy is the only on-board entity which generates new user tags and grants their ownerships. The proxy is empty labeled in order to always be able to communicate with several CE devices and other online services.

The labels provide also an efficient way to constrain information flows between on-board services. The HU service can receive information from ECU A if and only if it has the tag a_s in its label or ownership. As a consequence, even if a message from ECU A, labeled with a_s, is forwarded by a multicast address. The proxy will never send such message out and no unauthorized service, i.e., without the tag a_s, will process it. On the other hand, since the HU service owns the tags a_s, it may occasionally forward a_s-labeled data to the TPA. The (safety) mechanisms of ECU B will only be triggered by authorized services, i.e., services having b_i in their label or ownership.

Approach conclusion: No service is fully trusted and owns all the tags of the system. Label, ownership and DLA allow services to express their security concerns for integrity and secrecy The on-board services rely on their remote on-board communication partners to enforce the right DIFC policy. If we consider our integrity attack scenarios, the TPA is isolated in a cell and cannot disturb the HU functioning. The label enforcement of the TPA and of the CE device are performed by the HU service and the proxy, respectively. The exchange of labeled messages allows the communicating services to determine if the message/its source have a sufficient integrity level in order to be processed or trigger a mechanism. Regarding the confidentiality scenario, the labeling of the CE device and the TPA prevents them from accessing highly sensitive data. In case of a bug causing a privacy issue, the labeling of the messages allows the proxy to detect an unauthorized communication and to block it.

Second Approach - DIFC & DDFT. This second approach differs only in the way we monitor the TPA. Communications between services and with CE devices are secured by the same methods as previously, except that the communications with the TPA benefit from a new security enforcement. Both our attack scenarios feature a TPA presenting exploitable vulnerabilities. A well configured DDFT can detect such attacks and stop the application before it harms the car or a user's privacy. The rest of this section reasons directly with our attack scenarios.

Integrity attacks: The DDFT framework is configured to detect every system call which involves exchanges of information with the outside. It detects in particular inter-process communications, e.g., with critical HU processes, shared memory and filesystem access and can block all of them. The DDFT framework only authorizes certain network communications and file accesses. Authorized and non critical files are whitelisted by the DDFT and their access is monitored. Therefore the HU functioning cannot locally be disturbed, the DDFT framework can even enforce policies limiting its resource consumption, e.g., against denial of service attacks. At a remote level, a communication with another service is only possible if it is authorized by the rule set of the TPA and if the service has a policy authorizing the communications with a TPA. Therefore a TPA will not be able to reach critical functionalities, e.g., from the brake controller, and disturb the car functioning.

Confidentiality attacks: These attacks mostly concern the release of sensitive information to the outside and have to go through the proxy. The DDFT framework whitelist the file that the TPA can access and makes sure that it does not access files containing the private information of a user it is not assigned to. The services which communicates with the TPA specify in the middleware header the STL of the data and the ID of the user, whose privacy is concerned. A TPA can be sent all information, the DDFT is trusted to propagate the STL taint and to check that the ID suits the user it was assigned. If the TPA tries to directly send some data through the proxy, the DDFT engine injects a STL and ID field in the payload. The proxy can then ensure that the addressee is the one specified in the header and that the STL condition is fulfilled. The situation is a bit more complex if the TPA attempts to release the data through another service communicating with the outside. That is why a service dealing with the private data of a user, i.e., having a user's tag in its label or ownership, will refuse a message with a TL=3. Only services not communicating with the outside may receive data with TL=3, i.e., services with secrecy service tags reflecting a high secrecy for the car manufacturer. The processing of data tainted with a TL=1 or 2 will depends whether the service can handle private information, i.e., whether the service has the user's secrecy tag in its label or ownership. The decisions about whether to process and which SL to use later are defined by the car manufacturer and statically set up in the middleware of the service. As a consequence, sensitive data coming from the TPA are processed by services respecting the concerns expressed by the STL and allowing a privacy-aware release with the outside.

Approach conclusion: This second approach combines the advantages of DIFC for on-board and CE device-based communications and provides more granularity as for the integration of TPA. The TPA accesses more data, even the confidential ones and is still able to communicate with untrusted partners if the message does not involve sensitive information.

Limitations and potential solutions. In Section 2.2, we assumed the integrity of the middleware/OS of every ECU and the proxy. However secure boot and remote attestation do not protect or detect runtime attacks, which can be very harmful if they compromise critical ECUs like the proxy or the HU. Other runtime intrusion detection systems have to be considered as well [42]. Such security tools perform scans of critical data structures and recognition of instruction patterns within a running platform. They generally significantly degrade the system performance and should be used in a carefully selected manner. Successful attacks on the proxy could be mitigated by its compartmentalization thanks to hypervisor or microkernel techniques. Even if an isolated cell of the proxy gets compromised, the proxy detects it and shuts down the cell without impacting the other genuine cells.

Regarding the integration of a TPA in an isolated cell, we mentioned that they can only be used by a unique user. Several users, willing to use simultaneously a TPA, may require the car to assign a virtual machine per user in order to preserve their privacy. This solution may be too resource costly. The DDFT approach, on the other hand is less heavy. It can monitor several applications for different users and isolate them from each other.

Partial security conclusion. Unlike OSs like Android, which control their applications with a set of coarse permissions, DDFT allows a fine security granularity. More than just isolating, DDFT stops the TPA, before the attacker takes control of it. Unlike the isolation solution, TPAs monitored by DDFT are more functional and several applications can be used by several users. DIFC/DDFT is therefore a better solution from a pure security point of view.

6.2. Functional Evaluation

Section 6.1 justifies that the DDFT approach is providing a deeper monitoring and a better flexibility than the isolation approach. This section assesses our implementation overhead and determines whether this choice can be corroborated from a functional point of view.

We measure the middleware throughput (in call/sec) between an application of the CE device and an on-board TPA for a scenario similar to the one presented in Section 6.1. We performed our tests with different security features and in various situations in order to demonstrate the overhead caused by our two approaches for TPA integration. Benchmarks are run on three separated machines as described in Section 5: the HU, the proxy and the CE device. The CE device application sends a simple *Etch* message including one integer to the TPA. Before arriving to the TPA, the message goes through the intermediary of the proxy and then a service of the HU, where DIFC can be enforced. After reception, the TPA retrieves a series of integers from a file on the HU. Based on the received and read integers, the TPA generates a buffer and sends it to the CE device through the same inverse path, i.e., through the HU service and then the proxy. We vary the size of the buffer in order to stress the middleware and the taint propagation mechanisms. In this scenario the TPA plays the role of an infotainment server, e.g., for music or picture. Our first set of measurements is performed without any security features enabled (1) and we use this case as reference. We then did the same while encrypting the communication channel (2), i.e., by using TLS for the link CE device–proxy and IPsec for the proxy–HU. After which, in addition, we enforced

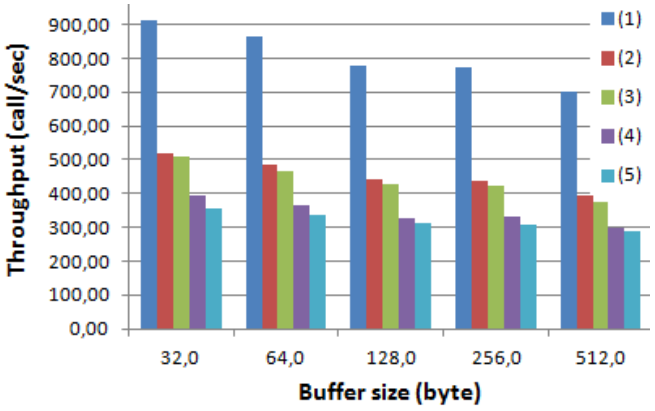

Figure 9. Middleware throughput average for various buffer sizes and security modes enabled.
Mode Description:
(1) No security feature is enabled
(2) The encryption between CE device, proxy and HU is enabled.
(3) DIFC is enforced between proxy and HU + (2).
(4) The TPA runs in a Virtualization cell + (3)
(5) The TPA is DDFT monitored + (3)

Table 2. Normalized middleware throughput performance of the scenario presented in Section 6.2. These results are computed from the throughput averages of the different modes. The mode of this table are similar to the used for Figure 9. Factor (i) presents the normalized performance with (1) as reference, while factor (ii) takes (3) as reference.

Enabled Mode	(1)	(2)	(3)	(4)	(5)
Factor (i)	1	0.57	0.55	0.43	0,40
Factor (ii)	-	-	1	0.78	0.73

DIFC between proxy and HU (3). The measurements of (3) gives a lower bound overhead caused by the communication security without any TPA consideration. The case (4) uses the same security features as (3) and monitors the TPA with our DDFT framework and the HU service, which enforces the DDFT/DIFC coupling rules. The measurements of (4) allow to determine the overhead imposed by a DIFC/DDFT framework and our custom taint propagation mechanisms. Finally the case (5), like (4), uses the security of (3), but instead of DDFT, the TPA runs in an isolated XEN cell and the HU service enforces DIFC with the TPA. The measurements of (5) allow to evaluate the impact caused by DIFC and the isolation cell. The throughput results of Figure 9 present the average of ten measurements of 5000 calls each.

Discussion: After normalization of the results of Figure 9, we realized that the performances for different enabled security features and regardless of the buffer size are proportionally similar. The normalized average throughput can be found in Table 2. First this table shows that the use of security protocols like TLS and IPsec is responsible for the biggest impact (~43%). The processes for encryption/decryption and generation/verification of security fields like HMAC are extremely costly, when used with a high throughput system. A second less consequent performance decrease (~4%) is due to the enforcement of DIFC between proxy and HU. For each packet, proxy and HU service checks whether the integrity and secrecy labels are valid for the invoked function. The last performance impact is caused by the security feature securing the TPA. For the isolation case, the performance decreases by 27% and for the DDFT by 22%. In the first case, it is due by the virtualization of the network bridge and the OS, in the second case by the function call monitoring and the taint propagation mechanisms. As a conclusion, for our scenario isolation and DDFT presents a similar performance and could be used for infotainment use cases involving a CE device and requiring a moderate bandwidth (1,18 Mbit/sec for the DDFT and 1,23 Mbit/sec for the isolation). Our experimentation involves a communication link between HU service and proxy, this link is used here to show the impact of DIFC on our scenario and because the isolation approach requires an intermediary step on a HU service. However with our DDFT approach, a TPA can directly communicate with the proxy and reaches a bandwidth of 2,14 Mbit/sec when using DDFT and encryption.

However, our evaluation is only focused on the middleware level in a small 3-node network and for a specific scenario involving simple TPAs and CE-based applications. Tests performed with *libdft* for bigger applications such as a web-browser [8] or a MP3-player [33] have shown a more significant latency (up to 28 times slower), whereas the isolation cell tends to provide a constant performance, independently from the application complexity. As a consequence, DDFT should only be applied for simple application handling private data and requiring a lot of interaction with external partners. DDFT should besides optimize its use of trusted libraries, i.e., libraries, which are not monitored. The use of the isolation approach can be used for a whole system and its TPAs, e.g., an Android partition. Additional investigations involving larger networks generating more traffic are strongly recommended for real-world validation.

About some functional requirements: This paragraph offers a brief evaluation of our system based on the requirements defined in Section 2.3. Our implementation partly fulfills our performance requirement, it does better than low-speed communication like CAN but is far from the high-speed one like MOST. For bandwidth demanding use case, like for audio/video streaming, other protocols like IEEE1722 [43], which

skips the use of UDP and IP, are more efficient. Our security framework has been implemented with an efficient automotive middleware, but the testing with resource limited platform should be performed. Unlike the DDFT engine, the security solution involving isolation and virtualization is resource-costly and therefore may not be suitable for resource-limited platforms. Then security was designed in an engineering-driven way. Enforcing security in the middleware allows the security engineering team to not know in detail the application logic and to focus on abstracted flows of information. Finally with our proxy-based architecture, developers of CE device application are totally security-unaware and are not imposed any (security) protocol, as far as the proxy provides a suitable interface and can compute the STL.

Before giving our conclusion, Table 3 proposes a short summary of the evaluation of our two approaches.

7. Conclusion

In this paper, we presented a security architecture for next-generation automotive on-board networks, which combines different information flow control techniques. Our system proposes two levels of security. Locally very untrusted components, like TPAs, are monitored thanks to a custom DDFT engine. DDFT allows to fully control the TPA behavior without modifying the binary executable: At the network level, our DIFC model allows every on-board service to independently express its security concerns and to trust the remote on-board communication partner to respect it. For this purpose, the security is ensured by the middleware. This layer is in charge of the network communications, is strictly separated from the application logic and allows the on-board exchange of security metadata. The middleware is also used as a glue between DIFC and DDFT and provides the translation interfaces between the two models. A Proxy-based architecture, on the edge of the embedded network, filters inbound and outbound messages and allows an on-the-fly evaluation of the security and trust level of all external communications. The taxonomy used to evaluate this security and trust level is also directly used by the DDFT engine and by the service middleware. More than just providing the monitoring of one application, our security framework proposes a DIFC-based monitoring approach of a whole system thanks to isolation and virtualization techniques. With respect to the integrity and confidentiality attack scenarios in Section 2.2, our proposed framework successfully prevents those attacks and offers a light-weight and easy-to manage middleware-based security and privacy framework by using the aforementioned techniques. It also meets almost all architecture requirements from Section 2.3. More than giving a recommendation about which solution to use, we see our 2 approaches for TPA integration as complementary; the choice for an approach should be carefully evaluated and dependent on the application complexity and the requirements for performance and security flexibility. However, while enhancing the in-car security and privacy, these solutions have shown some performance limitations. More rigorous validations therefore require additional investigations in order to determine a suitable tradeoff between acceptable performance and security enforcements on resource-limited hardware.

References

[1] Elliott, M.-A. (2011) The Future of Connected Cars. http://mashable.com/2011/02/26/connected-car/ (accessed on 14 march 2013).

[2] Lutz, Z. (2011) Renault debuts R-Link, an in-dash Android system with app market http://www.engadget.com/2011/12/09/renault-debuts-r-link-an-in-dash-android-system-with-app-market/ (accessed on 14 march 2013).

[3] Koscher, K., Czeskis, A.,Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H. and Savage S. (2010) Experimental Security Analysis of a Modern Automobile. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (Washington DC: IEEE Computer Society), 447–462. doi:10.1109/SP.2010.34

[4] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F.and Kohno, T. (2011) Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX conference on Security* (Berkeley: USENIX Association), 6–6.

[5] Rouf, I., Miller, R., Mustafa, H., Taylor, T., Oh, S., Xu, W., Gruteser, M., Trappe, W. and Seskar, I. (2010) Security and Privacy Vulnerabilities of In-car Wireless Networks: a Tire Pressure Monitoring System Case Study. In *Proceedings of the 19th USENIX conference on Security* (Berkeley: USENIX Association), 21-21.

[6] Slivka, E. (2012) Apple Pulls Russian SMS Spam App from App Store. http://www.macrumors.com/2012/07/05/apple-pulls-russian-sms-spam-app-from-app-store/ (accessed on 15 March 2013).

[7] Glass, M., Herrscher, D., Meier, H., Piastowski, M. and Shoo, P. (2010) SEIS - Security in Embedded IP-based Systems. In *ATZelektronik worldwide, 2010-01*, 36–40.

[8] Kemerlis, V., Portokalidis, G., Jee, K. and Keromytis, A. (2012) libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments* (New York: ACM), 121–132. doi:10.1145/2151024.2151042

[9] Bouard, A., Schanda, J., Herrscher, D., Eckert, E. (2012) Automotive Proxy-based Security Architecture for CE Device Integration. In *Procceedings of 5th International Conference Mobilware on Mobile Wireless Middleware, Operating Systems, and Applications*

Table 3. Summary of our system evaluation with respect to the attack scenarios of Section 2.2 and the requirements of Section 2.3 (More details are provided in Section 6).

	1 - Approach DIFC/Isolation	**2 - Approach DIFC/DDFT**
Monitoring paradigm	system oriented	single-process oriented
Integrity scenario		
• Protection of the HU from a TPA	Yes, by isolation	Yes, by monitoring
• Protection of other ECUs from a TPA	Yes	Yes
• Protection of the ECUs from external attackers	Yes, same DIFC-based mechanisms for both approaches	
Secrecy scenario		
• Protection of the user's data from a TPA	Yes	Yes, more fine-granular
• Privacy-aware data release, with a TPA	Yes	Yes, more dynamic/flexible
• Protection of the user's data from external attackers	Yes, same DIFC-based mechanisms for both approaches	
Requirements		
• Performance	Partially, independent of the application complexity	Partially, dependent on the application complexity
• Solutions for all platforms	No	Yes
• No regular updates necessary	Yes	Yes
• No financial extra cost	Yes, software-based solution	Yes, software-based solution
• No complexity increase for developers and users	Yes, the communication decoupling of the proxy makes users and developers security-unaware	

(Heidelberg:Springer-Verlag), 62–76. doi:10.1007/978-3-642-36660-4_5

[10] Homepage of *Etch*. http://incubator.apache.org/etch/ (accessed on 15 March 2013).

[11] Hoppe, T., Kiltz, S., Dittmann, J. (2008) Security Threats to Automotive CAN Networks & Practical Examples and Selected Short-term Countermeasures. In *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security* (Heidelberg: Springer-Verlag), 235–248. doi:10.1007/978-3-540-87698-4_21

[12] Maier. A. (2012) Ethernet - The Standard for In-car Communication. In *2nd Ethernet & IP @ Automotive Technology Day*. http://www.ethernettechnologyday.com/downloads/18 _Alexander_Maier_-_BMW.pdf (accessed on 15 March 2013).

[13] Weckemann, K., Satzger, F., Stolz, L., Herrscher, D., and Linnhoff-Popien, C. (2012) Lessons from a Minimal Middleware for IP-based In-car Communication. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2012* (Washington DC: IEEE Computer Society), 686–691.

[14] Bouard, A., Glas, B., Jentzsch, A., Kiening, A., Kittel, T., tadler, F., and Weyl, B. (2012) Driving Automotive Middleware Towards a Secure IP-based Future. In *Proceedings of the 10th ESCAR Embedded Security in Cars Conference*.

[15] Clarke, E. M., Grumberg, O. and Peled, D. (1999) Model Checking. MIT Press.

[16] Mecklenbrauker, C. F., Molisch, A. F., Karedal, J., Tufvesson, F., Paier, A., Bernado, L., Zemen, T., Klemp, O., Czink, N. (2011) Vehicular Channel Characterization and Its Implications for Wireless System Design and Performance. In *Proceedings of The IEEE Special Issue on Vehicular Communications* (Washington DC: IEEE Computer Society), 99(7): 3646–3657.

[17] Fujitsu Semiconductor Europe (2012) Fujitsu Announces Powerful MCU with Secure Hardware Extension (SHE) for Automotive Instrument Clusters. In Fujitsu Press Release. http://www.fujitsu.com/emea/news/pr/fseu-en_20121129-1044-fujitsu-mcu-secure-hardware-extension-atlas-l.html (accessed on 15 March 2013).

[18] McMillan, R. (2011) 'War Texting' Lets Hackers Unlock Car Doors via SMS. http://www.pcworld.com/article/236678/War_Texting_Lets_Hackers_Unlock_Car_Doors_via_SMS.html (accessed on 15 March 2013).

[19] Homepage of the EVITA project. http://evita-project.org/ (accessed on 15 March 2013).

[20] Homepage of the SeVeCom project. http://www.sevecom.org/ (accessed on 15 March 2013).

[21] Homepage of the sim^{TD} project. http://www.simtd.org/ (accessed on 15 March 2013).

[22] Detken, K.-O., Fhom, H. S., Stehman, R., Dietrich, G. (2010) Leveraging Trusted Network Connected for Secure Connection of Mobile Devices to Corporate Networks. In Pont, A., Pujolle, G. and Raghavan, S.V. [eds] *Communications: Wireless in Developing Countries and Networks of the Future* (Heidelberg:Springer-Verlag). doi:10.1007/978-3-642-15476-8_16

[23] Department of Defense (1983) Trusted Computer System Evaluation Criteria In *Orange Book*

[24] Myers, A. C., Liskov, B. (2000) Protecting Privacy Using the Decentralized Label Model. In *ACM Transactions on Software Engineering and Methodology* (New York:ACM), 9:410–442.

[25] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., Morris R. (2005) Labels and Event Processes in the Asbestos Operating System. In *Proceedings of the 20th ACM symposium on Operating systems principles* (New York:ACM), 17–30. doi:10.1145/1095810.1095813

[26] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., Mazières, D. (2006) Making Information Flow Explicit in Histar. In *Proceedings of the 7th symposium on Operating systems design and implementation* (Berkeley:USENIX Association), 263–278.

[27] Zeldovich, N., Boyd-Wickizer, S. and Mazières, D. (2008) Securing Distributed Systems with Information Flow Control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley:USENIX Association), 293–308.

[28] Ramachandran, A., Mundada, Y., Tariq, M., and Feamster, N. (2008) Securing Enterprise Networks Using Traffic Tainting. In *Special Interest Group on Data Communication*.

[29] Zdziarski, J. (2012) Hacking and Securing iOS Applicaitions. (O'Reilly Media, Inc.), chap 13.1 Sandbox Integrity Check. http://my.safaribooksonline.com/book/-/9781449325213/jailbreak-detection/sandbox_integrity_check (accessed 15 March 2013).

[30] Yin, H., Song, D., Egele, M., Kruegel, C. and Kirda E. (2007) Panorama: Capturing Systemwide Information Flow for Malware Detection and Analysis. In *Proceedings of the 14th ACM conference on Computer and communications security* (New York:ACM), 116–127. doi:10.1145/1315245.1315261

[31] Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P. and Sheth, A. (2010) Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (Berkeley:USENIX Association), 393–407.

[32] Zavou, A., Portokalidis, G., Keromyitis, A. (2011) aint-Exchange: A Generic System for Cross-Process and Cross-Host Taint Tracking. In *Proceedings of the 6th International conference on Advances in information and computer security* (Heidelberg:Springer-Verlag), 113–128.

[33] Schweppe, H. and Roudier, Y. (2012) Security and Privacy for In-vehicle Networks. In *1st IEEE International Workshop on Vehicular Communications, Sensing, and Computing* (Washington DC: IEEE Computer Society).

[34] Shankar, V., Urbam, G., Sultan, F. (2002) Online trust: a stakeholder perspective, concepts, implications, an future directions. In *Journal of Strategic Information Systems*, 11(3):325–344.

[35] Ling T. C. et al: Baker & McKenzie - Global Privacy Handbook. IACCM (2012)

[36] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J. and Hazelwood, K. (2005) Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation* (New York:ACM), 190–200. doi:10.1145/1065010.1065034

[37] Levy, E. (Aleph One). (1996) Smashing the Stack for Fun and Profit. In *the Phrack Magazine*, 7(49), chap 14.

[38] Scut, team teso (2001) Exploiting Format String Vulnerabilities. Technical Report, http://julianor.tripod.com/bc/formatstring-1.2.pdf.

[39] Shacham, H., Page, M. and Pfaff, B. (2004) On the Effectiveness of Address-space Randomization. In *Proceedings of the 11th ACM conference on Computer and communications security* (New York:ACM), 298–307. doi:10.1145/1030083.1030124

[40] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A. (2003) Xen and the art of virtualization. In *Proceedings of the 19th ACM symposium on Operating systems principles* (New York:ACM), 164–177. doi:10.1145/945445.945462

[41] BMW AG. Navigation System Professional, http://www.bmw.com/com/en/insights/technology/technology_guide/articles/navigation_system.html (accessed 15 March 2013).

[42] Garfinkel, T., Rosenblum, M. (2003) A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of NDSS Symposium 2003*, Internet Society

[43] IEEE Standards Association (2011) IEEE 1722 - Layer 2 Transport Protocol Working Group for Time-sensitive streams http://grouper.ieee.org/groups/1722/ (accessed 15 March 2013).