Middleware-Based Security and Privacy for In-car Integration of Third-party Applications

Alexandre Bouard, Maximilian Graf, and Dennis Burgkhardt

BMW Forschung und Technik GmbH, D-80788 Munich, Germany
{alexandre.bouard,maximilian.graf,dennis.burgkhardt}@bmw.de

Abstract. Today's vehicles include up to seventy networked electronic platforms handling simultaneously infotainment and safety functions. Fully connected to the world, the car is now customizable, communicates with several external devices, online services and will be soon hosting third party applications, as our smartphones already do. Such an evolution raises several critical security and privacy issues. While offering numerous advantages, the use of Ethernet, the Internet Protocol (IP) and their associated security protocols as on-board communication standards may not be sufficient. A generic framework focused on information security and on the aforementioned use cases would fill this gap and is still missing. In this paper, we present a combination of car-wide and local security concepts for IP-based middleware securing the integration of unsafe automotive scenarios. We describe the implementation and integration of these mechanisms and show their evaluation.

Keywords: Security & Privacy, IP-based Middleware, Automotive Application, Car-to-X Communication.

1 Introduction

More than just a simple transportation mean, the car evolved into a very complex system, efficiently networking powerful electronic platforms for various purposes. While still serving its primary goal, the car now offers additional mobility services involving road-side units, other cars, smartphones and online services. Like Consumer Electronic (CE) devices, the car will soon host loadable and on-the-fly installable applications allowing better car customization and deeper integration of the aforementioned mobility services [1]. However, current automotive technologies and requirements for high robustness and low latency slow down the process and let little space for security, an essential parameter considering the numerous security issues and their life threatening consequences, that were recently brought up [2,3].

Part of the solution seems to lie in the use of Ethernet and the Internet Protocol (IP) as standard for both on-board and external communications [4]: a larger bandwidth and strong security protocols already designed for the Internet world can secure the communication between two on-board platforms and

C. Fernandez-Gago et al. (Eds.): IFIPTM 2013, IFIP AICT 401, pp. 17–32, 2013.
© IFIP International Federation for Information Processing 2013

with external entities. But security considerations only at the communication layer, without considering information security, may remain insufficient. Future use cases will involve large amounts of data, presenting different levels of confidentiality and integrity, originating from in-car and external sources. In order to integrate such unsafe and uncontrollable scenarios, the on-board architecture needs to be secured accordingly and the driver's privacy needs to be protected.

Our approach proposes an on-board integration of several local and distributed security mechanisms: i) a security communication proxy on the edge of the on-board network, which filters inbound and outbound communications [5]; ii) a dynamic data flow tracking (DFT) tool based on *libdft* [6], which monitors third-party applications and iii) an in-band signaling protocol integrated in the communication middleware *Etch* [7]. Their combination allows a secure tethering of external communication partners with internal functionalities, e.g., original on-board functions, developed by the car manufacturer, or third-party (TP) applications. The resulting framework provides acceptable performances and sufficient flexibility to comply with our security and privacy requirements.

The main contributions presented in this paper are:

- A *security model combining complementary IP-based security concepts* in a car-wide security framework;
- A *simple and efficient taxonomy* for untrustworthy use cases highlighting the security and privacy/trust context of the communication;
- A *prototype implementation* integrating an automotive middleware, its associated communication proxy and a customized DFT tool.

The rest of the paper proceeds as follows: After having given a brief overview about future automotive on-board architectures, Section 2 presents our use cases and related work. Section 3 introduces our concepts for a car-wide security framework. Then, Section 4 describes our taxonomy for "unsafe" use cases. Section 5 presents the implementation and integration of our IP-based security framework. Section 6 provides our evaluation of our framework and Section 7 our conclusion.

2 Background and Related Work

In this section background information on future automotive systems and related work about security and privacy are provided. A threat model and some relevant scenarios are presented as well.

2.1 Current and Future Automotive Architecture

The automotive on-board network comprises up to 70 Electronic Control Units (ECUs) interlinked by different communication buses and organized in several sub-networks around specific domains (e.g. power train, infotainment). On-board applications are divided in elementary function blocks over several ECUs exchanging broadcasted signal-based messages. Due to internal communication in plaintext and a lack of input validation in the ECUs, cars have been shown to be vulnerable to common attacks exploiting local [2] and remote [3] interfaces.

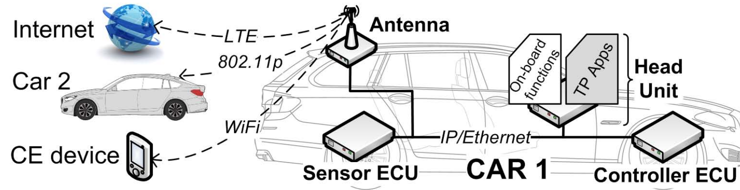

Fig. 1. Automotive scenario and considered communication channels. Solid right-angle lines represent the wired on-board network. The dashed arrows represent external communications over different wireless networks.

The introduction of Ethernet/IP for vehicle on-board network will be beneficial for both functionalities and security. Firstly, a larger bandwidth will allow to exchange bigger objects (e.g. environment models) internally between ECUs and to comply with future requirements of distributed applications for driver assistance and infotainment [8]. Secondly, mature and secure protocols from the Internet world will be instantly applicable. Future automotive applications will certainly require more powerful ECUs and will allow car manufacturers to redesign their software management. On-board applications will remain distributed and the communication management will be simplified thanks to performing engineering-driven middleware infrastructures, abstracting and automating the network addressing and security enforcement [9]. In addition the centralization of most external communication interfaces (e.g., LTE, Wi-Fi) around a multiplatform antenna (MPA) [10] will allow the car makers to design a single security gateway for Car-to-X (C2X) communications.

2.2 Threat Model

Today's cars are facing several challenges. Automotive applications are rarely updated and involve more and more new connected features. Their functional behavior relies on complex software, not free from any security flaw [2] and processing considerable amount of sensitive data. An attacker could therefore take advantage of defects in the logic of an application or in a weak security mechanism. This could result in leaks of private information or industrial secrets, threaten the car integrity and in the worst case the life of its occupants.

Use Cases and Attack Scenarios. The scenario for our use cases is depicted in Figure 1 and features both internal and external communication partners. We take the example of a TP application installed on the Head Unit (HU) of "*Car 1*", which is connected to services of the Internet, another car "*Car 2*", a CE device and to several original on-board functions of "*Car 1*". We mainly focus on attack scenarios trying to elude security policies and leveraging the TP application in order to i) compromise internal resources or ii) leak sensitive data to an unauthorized external entity. We consider a TP application functionally conform to the internal API of the car. However, it may present some flaws that are exploitable by an attacker.

(i) **Integrity attack scenario:** the TP application gets compromised or forwards messages from an external malicious communication partner. As a result the TP application may send bogus packets on the on-board network or access/modify sensitive resources on the HU and may therefore critically disturb the car's functioning.
(ii) **Confidentiality attack scenario:** the TP application gets access to sensitive information, like the driver's home address stored in the navigation system. However even without the authorization to share it, the TP application may still send it to the outside, either directly over the proxy or through an intermediate step, e.g., a forwarding multicast address.

This work aims at improving the information security and addressing the threats related to unfair entities, on which the car manufacturer has no control , while still considering our requirements for high robustness and low latency.

Assumptions. Next-generation ECUs will be equipped with security middleware establishing communication channels over strong security protocols like IPsec [9]. In addition they will soon include a hardware secure extension providing secure key storage and secure boot [11]. As a consequence we assume that the middleware and the hardware platform cannot be compromised. Besides, we trust ECUs to establish secure communications to each other and to enforce the expected security mechanisms. We do not consider denial-of-service attacks here.

2.3 Related Work

During the last decade, some automotive projects investigated the security issues related to external communications. SeVeCom addressed some of them and designed C2X security mechanisms focused on authentication and encryption [12], but did not consider the impact of external inputs on the on-board architecture. The SEIS project proposed a proxy-based architecture for CE device integration [5]. The proxy evaluates the security level of the communication and transmits it to the ECU for an adapted security enforcement. Their choice of use cases and security evaluation are limited, but we propose to extend their concept to our architecture. Corporate network security and automotive on-board security present several similarities, e.g., when integrating mobile devices. The corporate approaches rely mostly on strong authentication mechanisms and device integrity measurements in order to establish network connections or a VPN tunnel [13]. However they only regulate the network access and usually lack specifications for resources, data management and specifically information flows.

As for the TP application monitoring, we chose to focus on DFT-based approaches. They allow to taint and track data of interest within a running application and have been successfully applied for various purposes, e.g., malware monitoring [14] or privacy-aware OS monitoring [15]. These approaches, monitoring the whole OS, rely on a modified runtime environment [15] or on emulators like QEMU [14] and require extensive maintenance. They track every machine instruction performed on the host and as a consequence suffer from a significant performance issues. Considering our requirements for robustness and low latency,

we orient our work toward a lighter approach monitoring only one process [6]. This is more efficient, does not require any OS or source code modification and has been already used for distributed [16] and automotive [17] environments. We propose to limit the DFT monitoring to the TP application only and combine it to the middleware for a car-wide security enforcement.

3 Combining Local Mechanisms for Car-Wide Security

Controlling information flows in distributed systems like cars is essential for holistic security. ECUs internally exchange genuine messages and therefore only necessitate secure communication channels and simple access control mechanisms. But integrating unregulated communication partners or software components requires a more complex security model. In addition, ECUs communicate behind the MPA, i.e. communications with an external partner are decoupled at the MPA level. It allows the car to be able to use a suitable communication protocol for outside while using a unique internal security protocol. It also requires the MPA to help the ECU to determine the right security decision to enforce.

For this purpose, we propose to develop an application independent in-band signaling protocol allowing on-board exchanges of security metadata. Concretely we extend the header of the middleware protocol with a field characterizing the context of security and trust, in which data are exchanged over an external network. Instead of directly qualifying the privacy aspect of an information, we chose to focus on the trust we grant an external receiving peer and to quantify that. The security aspect defines how secure the communication is, while the trust aspect indicates how trustworthy the remote device or service is considered to be. We name this context *Security & Trust Level* (STL) and propose its precise evaluation in Section 4. Two types of STL can be distinguished: the first one, the STL_{status}, describing the actual STL of the received data and the second one, the STL_{req} describing the required STL, necessary to send the data out. The rest of this section presents the three security enforcement points, which make use of the STLs, in more detail: i) the security proxy, ii) the security middleware present on every ECU and iii) the TP application monitoring framework.

3.1 Security Proxy

Implemented on the MPA, the security proxy stands in the middle of every communication between an on-board entity and an external partner. The proxy is in charge of managing the external communication channels and their security. For each external partner, it performs a STL_{status} evaluation and extends the header of every inbound message with it. In addition the proxy enforces a coarse domain-based filtering, for example an online service related to a social network won't be able to access on-board functions of the power train management domain. Inversely the proxy will make sure that every STL_{req} received with an outbound message matches the actual STL of the communication in order to send out the message. Section 4 provides more information about the STL-based policies.

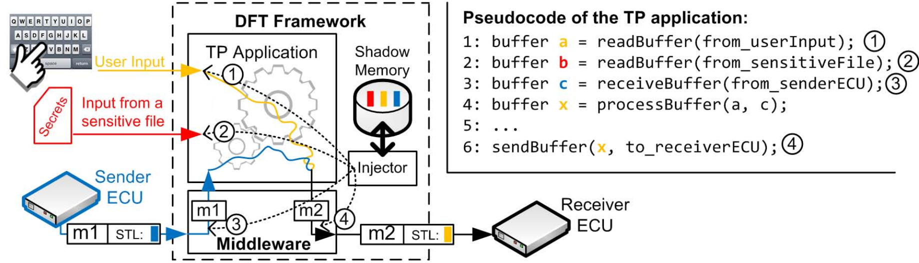

Fig. 2. Overview of the DFT framework in the on-board network. The solid lines show the input and output data of the TP application. The dashed arrows show the instrumentation of the system calls related to the taint sources (1-3) and sinks (4) monitored by the *Injector*. $m1$ and $m2$ represent tainted messages sent respectively to and from the TP application. The circled numbers link the functions of the pseudocode (right side) to their instrumentation (left side).

3.2 Security Middleware

Present on every ECU, the middleware abstracts and automates the communication and security management. The application developers can thereby focus on the functional logic and let the security part to a team of experts. Based on the received STL ($_{\text{status}}$ or $_{\text{req}}$), the middleware decides whether it is safe and allowed to process the payload. Depending on the middleware capacity, the unsafe or sensitive data can be processed in a security parser, in an isolated environment (e.g., an isolated web-browser for unsafe JavaScript) or handled as private data. Inversely, before sending a message, the middleware automatically integrates its STL_{req} in the payload and trusts the proxy or the receiving ECU to enforce the right decision. The STL_{req} reflects the sensitivity of the data and also in which situation such data may leave the car. The STL-based policies, enforced by the middleware, are defined by the car manufacturer at design time.

3.3 TP Application Monitoring Framework

For performance reasons, we limit the DFT-based monitoring only to the TP applications. DFT tools allow to monitor every instruction performed within a running application, i.e. to monitor every system call and to track every data flow between registers and memory. Such tools can raise a warning or stop the runtime in case of a behavior in contradiction to one of its security policies. Besides, they usually rely on dynamic binary instrumentation (DBI) frameworks (named *Injector* in Figure 2), like Intel's Pin [18], in order to inject custom code into the unmodified application binary, e.g., for the enforcement of a policy. The DFT monitoring can be explained by looking at these three instances: i) the taint sources, ii) the taint propagation and iii) the taint sinks. The rest of this section refers to Figure 2 and the pseudocode it presents.

i) Taint sources: Taint sources are the interfaces, through which new data are entering the TP application. If recognized as data of interest, the data are

tainted and the resulting taints are stored in the *shadow memory*, mapped to the actual memory of the application. The number of taints depends on the expected granularity. Originally, DFT was used to detect attacks related to stack pointers overwriting like buffer overflow or string-format and a simple binary tagging was sufficient. Here we consider data with different levels of sensitivity and therefore need more than two taints (taints and taxonomy are specified in Section 4). In our example, we identify as taint sources the input functions shown at lines 1, 2 and 3 of our pseudocode. The DFT tool monitors these functions and taints the buffer a as private data and b based on the sensitivity of the file. For a network input from an ECU like the buffer c, the framework extracts the present STL_{req} and taints the data according to it. In the case of data received from the proxy, where no STL_{req} is specified, we chose to taint the data as being private like a.

ii) Taint propagation: During runtime, tainted data are tracked while being processed in the application. Data resulting from tainted inputs (e.g., line 4) receive the most relevant taint, i.e. the higher trust and security level. In our example we consider data c received from the *sender ECU* as not sensitive, whereas the data a are private or "driver-sensitive", so x receives the taint of a.

iii) Taint sinks: The taint sinks are the functions or memory locations where the presence of a taint is checked in order to enforce a policy. The policies generally decide whether the data can be passed to a specific system call or whether they can be used as program control data, e.g. a return address. In our example it concerns the emission of data over the network (line 6). The DFT tool blocks the emission if the destination address is blacklisted, or otherwise automatically adds the most relevant taint in the middleware header. Even with a DFT tool, no packets coming from a TP application can be directly trusted and processed by an ECU. Like for messages from the outside, ECUs have to evaluate whether they are authorized to process the data; if so, they can trust the STL_{req} provided by the DFT tool and keep track of the data sensitivity.

DFT Security Policies. We do not trust the TP application or its middleware to enforce any policy, but rely on the DFT tool to do it. We distinguish both static and dynamic rules:

a) Static rules: These rules define the taint propagation and the taint management related to user input or file management. They are defined by the car manufacturer and cannot be overridden.

b) Dynamic rules: These rules are loaded with the TP application, in a rule set, similar to the one provided by an Android application. They define which internal and external communications are authorized and specify the trust level of an online service, they will communicate with. This rule set has to be approved and signed by the car manufacture after a testing process. Moreover, a TP application may ask the DFT tool to declassify some data, i.e. taint them with a lower STL in order to send them to an untrusted service. Such cases have to be specified in the rule set as well and concern the driver's data only. For example the declassification of private information may trigger the display of a warning pop-up asking for the driver's approval.

4 Automotive Security and Trust Taxonomy

In Section 3 we defined the STL as describing the security and trust context in which data are ("$_{\text{req}}$") or should be exchanged ("$_{\text{status}}$") between the car and an external communication partner. This section presents its evaluation based on i) security aspects and ii) trust aspects.

i) Security Considerations. We define the *Security Level* (SL) as a qualitative description of the security strength of an external communication. Concretely, we associate to each C2X security protocol a specific SL value. The different levels and the security requirements, they have to comply with, can be characterized as follows:

- ***SL=0:*** Communications providing no security mechanisms or protocols presenting exploitable design flaws;
- ***SL=1:*** Communications providing authentication of the external peer, security integrity for the exchanged message (against unauthorized modifications);
- ***SL=2:*** Communication providing authentication, security integrity and strong confidentiality (i.e. one secret key per user, no shared key between users);
- ***SL=3:*** Communication using protocols of $SL2$ level and assuring the presence of a secure hardware element protecting the cryptographic materials on the external communication partner.

Table 1 shows security protocols and their associated SL.

Table 1. Examples of security protocols ordered within the SL scale

SL=0	Plaintext; WEP encryption; TLS+DES or RC4 with a 56-bits key;
SL=1	WPA2 encryption; Message in plaintext protected by HMAC-SHA1;
SL=2	TLS+AES; IPsec+AES;
SL=3	SL2-protocols + Remote attestation protocol.

ii) Trust Considerations. As explained earlier in Section 2.2, the car contains sensitive data, that have to be controlled when released to the outside. For this purpose, we define the notion of *Trust Level* (TL), an abstract representation of how trustworthy the data emission and the data receiver are. In the literature [19,20,21], the notion of trust is usually defined by three major components: reputation, reliability and security. Since the factor security has already been considered in the previous paragraph, we focus here in the two remaining ones. For an efficient TL management and enforcement, we decide to only make use of criteria clear and easy to assess. We consider that sensitive data can be misused, only if they are i) physically and ii) juridically accessible, i.e. i) if the data leave the car and ii) if the data addressee can and is legally allowed to receive them and to endanger the driver's privacy (e.g., information selling/forwarding, data

Table 2. Method for TL evaluation

	Decision tree			
	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

Table 3. Scenarios and authorized data TL. (LHW: Local Hazard Warning).

	Which criteria are fulfilled?			
Scenarios	***Cr.1*** →	***Cr.2*** →	***Cr.3*** →	TL
Facebook	false	false	false	0
Safebook	false	false	false	0
Banking	false	false	true	1/0
LHW	false	true	-	2/1/0

stored on an unprotected server). The TL should thereby reflect these risks. For this purpose we make use of the following criteria:

– ***Criterion 1 (Cr.1) "Local usage":*** determines whether data have to be used and stored only within the car (e.g., industrial secret).
– ***Criterion 2 (Cr.2) "Anonymization":*** determines whether data, if released, will have to be anonymized, i.e. whether the addressee will be able to trace back the driver or the car based on the received data .
– ***Criterion 3 (Cr.3) "Jurisdiction":*** determines whether data have to be released to an online service storing and using the driver's data in "safe" place of jurisdiction (POJ), i.e. whether the service's server are located in a country imposing a regulation protecting the user's privacy.

In order to determine the TL values, we make use of a simple binary decision tree. Every criterion is iteratively evaluated, a "true" answer stops the process and sets the TL as shown in Table 2. Highly sensitive data, like industrial secrets, are only for a internal usage (Cr.1=true) and are tainted as requiring a very trustworthy usage (TL=3). Very sensitive data, like the car position, can leave the car but have to be untraceable (Cr.2=true), i.e. anonymized by the proxy (TL=2). Data with a low sensitivity, like the driver's name, can be forwarded to services presenting a safe POJ (Cr.3=true, TL=1). While Cr.1 and Cr.2 are easy to assess and enforce by the DFT or the proxy, Cr.3 needs to be specified by privacy experts, for example relying on literature inspecting the data protection laws of different countries [22].

In order to test this taxonomy, we evaluate the authorized TL-tainted data of four realistic TP application scenarios: a TP application linked to the social network Facebook [25]; a TP application for Safebook [26], a privacy-aware peer-to-peer social network allowing the user to locally store its data and having full control about the release thereof; a TP application related to an online banking service having its servers in Germany; and a Local Hazard Warning (LHW) application, broadcasting to other road user safety messages including the car position. Table 3 presents the evaluation results. Because of its servers' unsafe POJ, namely the USA [22], Facebook can only receive non-sensitive data. The Safebook's peers (i.e. "friends") can not be considered as being in a safe POJ and therefore are in the same case as Facebook. The Bank servers in Germany, a safe POJ [22], can receive "TL=1"- and "TL=0"-tainted data. As for the LWH scenario, other cars will receive the "TL=2" tainted data only if the proxy is sure they have been anonymized.

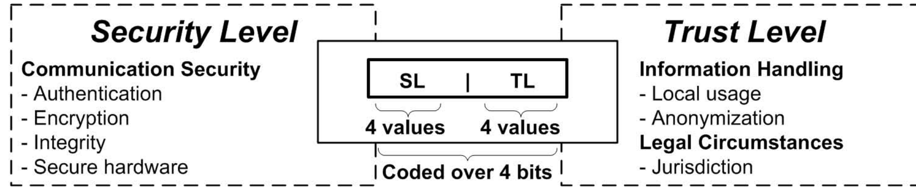

Fig. 3. STL vector and evaluation criteria

Even if the taxonomy seems to provide a suitable way to control the data release to the outside, further tests involving more use cases are required. The evaluation parameters are very coarse, but give to the car manufacturer a simple way to define default configuration settings. For more flexibility the driver should be able to temporary taint a piece of information with a lower TL or to upgrade external peers limited to "TL=0" data as peers allowed to receive "TL=1" data.

Managing and Enforcing STL Rules. We consider security and trust as two independent variables requiring two different types of enforcement. Anonymized data with a TL=2 may be sent with a SL=1 in plaintext (e.g. LHW scenario), while data of TL=1 may be sent with a SL=2 because the driver wants to keep them private. Therefore we define the STL as the concatenation of the SL and the TL, as shown in Figure 3. For an efficient enforcement, we limit ourselves to 4 values for the SL and 4 for the TL and can code the resulting vector on 4 bits.

Concretely, tainted data arriving on the proxy will be allowed to be released to an external peer X: i) if X complies with the conditions of the received TL and ii) if the external communication's SL is higher or equal to the one received. This implies for the DFT engine, that a buffer processed from several pieces of information will be assigned their higher SL and TL. However with such rules data can only flow to a higher level of trust and security and risk to never be able to leave the car. Declassification methods to assign a lower STL have to be possible, but have to be part of cases defined by the car manufacturer and if necessary involving the driver's decision.

As previously said, the STL-based policies of the ECUs are statically defined at design time by the car manufacturer. The STL management does not require any update of the ECUs. Either the ECU generated the data and associates to them a STL_{req} according to its policy, otherwise the ECU received the data and can also label them with the received STL_{req}. In addition, DFT engine and proxy can receive notifications to update the authorized TL values of external entities and the SL of a protocol for example via the security rule set provided by a TP application. As for the CE device case, the proxy authenticates the device as belonging to the driver and attributes an adapted $\text{STL}_{\text{status}}$ to it, i.e. with a SL dependent on the used protocol and a TL=1 since we assume that the driver's smartphone is under her control and therefore safe to handle sensitive data.

5 Implementation

This section describes the extensions, that we implemented in order to combine the DFT tool *libdft*, the middleware *Etch* and its associated communication proxy.

The Middleware. As prototype basis for our implementation, we chose the middleware *Etch*, an open-source software project under the Apache 2.0 license. *Etch* features a modular and extensible architecture providing an efficient serialization and is considered as a serious candidate for the automotive purpose [23]. We made use of the C-binding and extended the header by four bits, in order to add the STL field. Authorized STL can be specified before the code generation thanks to an adapted interface description language (IDL).

Regarding the communication proxy, we extended the *Etch*-proxy developed in Java for [5] and adapted it to the new payload serialization. The proxy provides two communication interfaces and automates the service discovery for both internal and external peers. Internal and external communication partners communicate over a mirror-service, making the communication decoupling totally transparent. Depending on the actual communication features (e.g., network interface, IP address or protocol used), the proxy performs on-the-fly evaluation of the STL and an adapted filtering for both ingress and egress traffic.

The DFT Tool. As for the monitoring tool we chose *libdft* [6], a dynamic DFT framework relying on the Intel Pin [18] for binary instrumentation. *libdft* provides an implementation of the shadow memory allowing an efficient taint propagation and a well-defined API for system call monitoring. Limited to a simple binary tainting (i.e. a bit of the shadow memory tainting a byte of the real memory) we extended the taint propagation mechanisms, in order to have a byte of memory tainted by two values of two bits each, so four bits total.

More than monitoring all inputs, our framework now differentiates a user input (i.e. standard input from the keyboard) from a file input and tags them accordingly. The framework manages the access to files present on the HU thanks to a white-list specifying for each TP application how to tag information read from a file and how data should be tainted in order to be written in a file. The framework monitors system calls related to network inputs and outputs. It allows us to taint ingress traffic depending on the IP address of origin (proxy's case) or on the provided taint present in the payload (ECU's case). For outbound messages, the framework automatically determines the different taints of the payload data and injects the most relevant one directly in the middleware header.

Testing Environment. We performed the implementation and the experiments described in Section 6.1 on several computers interlinked with Gigabit Ethernet and running standard 32-bit Fedora Linux on an Intel Atom N270 (1,6 GHz) with 1GB RAM. While being more resourceful than most of the embedded platforms in cars, they provide performances similar to a HU [24]. Besides we did not perform extensive modifications of the *Etch* middleware mechanisms, providing suitable performances when tested on a microcontroller [23]. Therefore we believe that the addition of this access control layer should not significantly slow down the system. Though this should be verified for a more rigorous validation.

6 Evaluation

In order to evaluate our system, we quantify in this section the performance overhead of our implementation and discuss several of its security aspects.

6.1 Performance Evaluation

Considering the limited space in this paper and our focus on the integration of TP applications, we will limit ourselves to the performance evaluation of our DFT framework. Benchmarks are run on two separated machines running a simple *Etch* service: a client sends a buffer over UDP and waits for an answer from a DFT monitored server. The server taints the received data and copies it in another location. A tainted integer is produced based on the same data and is sent back to the server with its associated taint, which is injected in the midlleware header. We measure the throughput of the client (call/sec) in order to demonstrate the communication overhead of our DFT framework, i.e. tainting a buffer and taking a decision as for sending a tainted information over the network. This experiment does not generate much application processing, but mostly stresses the middleware mechanisms. We performed the measures for different buffer sizes (from 128 to 8192 bytes) and different versions of the DFT engine. First we performed our tests for a native execution without any DFT engine ("null") that we use as reference. Then we did the same with the framework Pin alone, in order to get a lower bound overhead imposed by the instrumentation framework, i.e. without enforcing any security policies or memory tainting. Finally we made use of the original version of *libdft* [6] ("libdft.v1") and of our customized version ("libdft.v2"), in order to compare the impact of the taint number. The results of Figure 4 presents the throughput average for each case, calculated from 10 time measurements of 5000 calls each.

Discussion. The performance results in Table 4 show that the Pin binary instrumentation is responsible for a significant part of the application overhead (∼10%). Even if the framework is not enforcing any policies, Pin gets control of the execution each time a new instruction is invoked in order to provide the new compiled code to run before the next instruction. A second significant overhead (∼10-20%) is resulting from the library *libdft* itself and is caused by the taint propagation mechanisms and the system call monitoring. The induced performance overhead is less consequent for bigger buffers. The instrumentation of the socket connection calls seems to be mostly responsible for this. Finally our tests show an additional overhead when using the customized versions of *libdft*. Increasing the complexity of the taint mechanisms, of the system call monitoring and extending the shadow memory slow down the system performances (∼10%).

As previously mentioned, our evaluation is mostly focused on our middleware and the *libdft* library with a simple application. Tests performed with this DFT engine and bigger applications like a web-browser [6] or a MP3-player [17] have shown more significant latency. The use of a DFT framework adds a significant performance penalty but remains suitable for infotainment applications requiring a limited bandwidth (up to 5,4 Mbit/s). For optimal performances, the TP

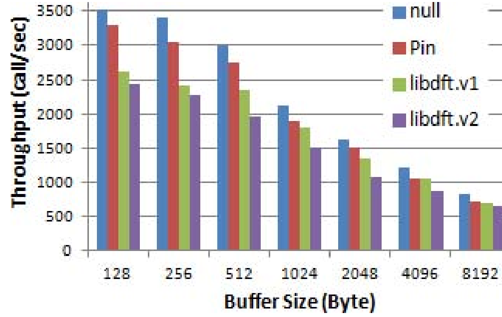

Fig. 4. Middleware throughput average for various buffer sizes and binary instrumentation methods

Table 4. Normalized throughput performance. The case without instrumentation is taken as reference.

Buffer size (kByte)	null	Pin	libdft.v1	libdft.v2
128	1	0,94	0,75	0,69
256	1	0,90	0,71	0,67
512	1	0,92	0,78	0,66
1024	1	0,89	0,85	0,71
2048	1	0,93	0,83	0,66
4096	1	0,87	0,86	0,71
8192	1	0,87	0,85	0,80
average	1	0,90	0,80	0,70

applications should remain small and simple and maximize the use of "trusted", i.e. non monitored, libraries. A second approach, not investigated here, would be to run the application in a fully virtualized environment providing isolation but less control during runtime, e.g., XEN [27]. Then we limit our evaluation to buffers with a size inferior as 8192 kBytes, because the *Etch* version we used has been optimized for relatively small payload. Further investigation are recommended for more realistic and bigger networks, generating more traffic, in order to test the suitability of this system for more demanding use cases, e.g. video streaming and safety applications.

6.2 Security Evaluation

For this section we refer to the attack scenarios presented in Section 2.2 and describe how our system would react to such attacks. Both scenarios feature an attacker getting control of the TP application by launching for example an attack related to the overwriting of a stack pointer. By design, the DFT can detect such exploits and stop the program. As result, the attacker cannot compromise the TP application integrity to perform our attack scenario.

About the Integrity Attack Scenario. This scenario considers an unauthorized access of a HU resource (e.g., file or process) aiming at disturbing the platform functionality. The DFT engine can monitor every system call and function invoked by the TP application. It can therefore blacklist the function and process that the TP application should not get access to and can restrict its file access in writing and reading. This scenario also considered the case of a TP application sending bogus packets to an ECU in order to disturb its functioning. The DFT engine controls the socket management and only allows communication with authorized ECUs. Then based on the received STL, provided by the DFT engine, the ECU is aware of the potential risk and adapts its data processing.

About the Confidentiality Attack Scenario. This scenario mostly considers the release of sensitive information to the outside. The TP application can receive information through multiple ways: by accessing shared memory, via filesystem access, with inter-process communications or from the network. The DFT engine monitors every of these input channels and taints data coming from them according to their sensitivity. On the other hand, the only way to release the information is through the network and then over the proxy. The DFT engine monitors the socket, which tainted information are going through, to which destination and can therefore block an unauthorized flow. If the framework cannot enforce a decision, the addition of the STL_{req} value in the message header allows the proxy to enforce a final decision based on the actual information sensitivity.

Unlike OSes like Android, which control applications with a limited set of coarse permissions, our DFT engine allows a very fine granular security enforcement. It monitors every invoked function, every Input/Output (I/O) channel of the TP application and tracks every byte of the application memory. The taint values, coded over four bits, offer sixteen different values expressing as much sensitivity levels. Such monitoring allows the application to remain functional even when simultaneously handling very sensitive data and communicating with untrusted sources. Let's take the example shown earlier in Figure 2: the TP application takes as input non-sensitive (TL=0) and sensitive (TL=3) data, but is still able to generate outputs tainted as *"not containing any sensitive information"* (TL=0) and those can be sent out. Monitoring the middleware and injecting a taint in its header allows us to export the local DFT benefits all over the on-board network. The in-band middleware protocol makes on-board applications information security aware and contributes to a homogeneous security enforcement in the whole car. However we do not propose any formal evaluation of the STL taxonomy in this paper. Our goal was to describe concrete examples of security and trust levels based on clear security requirements and quantitative parameters.

About Some System Limitations and Countermeasures: Several drivers can drive a single car and may be joined by some passengers. The DFT engine we use only monitors an application and implicitly considers a unique car user. But it would be quite simple to associate the monitored application to one user. The modifications will mostly concern the middleware. The taint field of the middleware header could be extended in order to contain a user ID. Then STL-based policies should be adapted in the whole system in order to take into account the different user requirements.

Then, we assumed in Section 2.2 that the integrity of the OS and the middleware were ensured by a secure boot. But these mechanisms do not protect against runtime attacks, which could be significantly harmful when being performed on critical entities like the proxy or the HU. They may be detected by host-based intrusion detection tools performing scans and recognition of instruction patterns within a running platform [28]. Though these solutions might significantly degrade the system performance and should be used in a carefully selected manner.

7 Conclusion

Upcoming automotive applications and use cases will require higher security standards in order to preserve the car integrity and to protect the information it contains. In this paper we presented a security architecture, leveraging dynamic data flow tainting engine in order to secure an automotive integration of third-party applications and external communication partners. At a local level, the DFT engine monitors the TP application and tracks/taints every byte of information it processes during its execution. The framework locally controls the network I/O and manages security metadata provided by the middleware in-band protocol, in order to enforce adapted security policies, locally and remotely on other ECUs. In addition, we proposed a security and trust taxonomy for external "unsafe" use cases, integrated to our concepts and supporting a distributed middleware-based policy enforcement. While enhancing the security and privacy in cars, such mechanisms have shown some limitations in term of performance and may be not used for any use case, especially the time-critical ones. As following work we will determine the necessary trade-off between granularity and efficiency for an optimal use of such mechanisms. Besides we will investigate alternative solutions making use of full virtualization. Finally we intend to refine our TL concepts and take into account user preferences while still ensuring a secure management of private information.

References

1. Lutz Z. Renault debuts R-Link, engadget and Renault press release at LeWeb 2011 (2011)
2. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental Security Analysis of a Modern Automobile. In: Proc. of the 31st IEEE S&P, pp. 447–462. IEEE (2010)
3. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: Proc. of the 20th USENIX SEC, p. 6. USENIX (2011)
4. Glass, M., Herrscher, D., Meier, H., Piastowski, M., Shoo, P.: SEIS - Security in Embedded IP-based Systems. ATZelektronik Worldwide 2010-01, 36–40 (2010)
5. Bouard, A., Schanda, J., Herrscher, D., Eckert, C.: Automotive Proxy-based Security Architecture for CE Device Integration. In: Borcea, C., Bellavista, P., Gianelli, C., Magedanz, T., Schreiner, F. (eds.) Mobilware 2012. LNICST, vol. 65, pp. 62–76. Springer, Heidelberg (2013)
6. Kemerlis, V., Portokalidis, G., Jee, K., Keromytis, A.: libdft: Practical dynamic data flow tracking for commodity systems. In: Proc. of the 8th ACM SIGPLAN/SIGOPS VEE, pp. 121–132. ACM (2012)
7. Etch homepage, `http://incubator.apache.org/etch/`
8. Maier, A.: Ethernet - The standard for In-car Communication. In: 2nd Ethernet & IP @ Automotive Technology Day (2012)
9. Bouard, A., Glas, B., Jentzsch, A., Kiening, A., Kittel, T., Weyl, B.: Driving Automotive Middleware Towards a Secure IP-based Future. In: 10th Escar (2012)

10. Mecklenbrauker, C.F., Molisch, A.F., Karedal, J., Tufvesson, F., Paier, A., Bernado, L., Zemen, T., Klemp, O., Czink, N.: Vehicular Channel Characterization and Its Implications for Wireless System Design and Performance. Proc. of the IEEE Special Issue on Vehicular Communications 99(7) (2011)
11. Fujitsu Semiconductor Europe, Fujitsu Announces Powerful MCU with Secure Hardware Extension (SHE) for Automotive Instrument Clusters. Fujitsu Press (2012), Release at http://www.fujitsu.com
12. Kargl, F., Papadimitratos, P., Buttyan, L., Muter, M., Schoch, E., Wiedersheim, B., Thong, T.V., Calandriello, G., Held, A., Kung, A., Hubaux, J.P.: Secure vehicular communication systems: implementation, performance, and research challenges. IEEE Communications 46(11), 110–118 (2008)
13. Detken, K.-O., Fhom, H.S., Sethmann, R., Diederich, G.: Leveraging Trusted Network Connected for Secure Connection of Mobile Devices to Corporate Networks. In: Pont, A., Pujolle, G., Raghavan, S.V. (eds.) WCITD/NF 2010. IFIP AICT, vol. 327, pp. 158–169. Springer, Heidelberg (2010)
14. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing Systemwide Information Flow for Malware Detection and Analysis. In: Proc. of the 14th ACM CCS, pp. 116–127. ACM (2007)
15. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: Proc. of the 9th OSDI, pp. 393–407. USENIX (2010)
16. Zavou, A., Portokalidis, G., Keromytis, A.D.: Taint-Exchange: A Generic System for Cross-process and Cross-host Taint Tracking. In: Iwata, T., Nishigaki, M. (eds.) IWSEC 2011. LNCS, vol. 7038, pp. 113–128. Springer, Heidelberg (2011)
17. Schweppe, H., Roudier, Y.: Security and Privacy for In-vehicle Networks. In: Proc. of the 1st IEEE VCSC. IEEE (2010)
18. Pin homepage, A Dynamic Binary Instrumentation Tool, http://www.pintool.org/
19. Shankar, V., Urbam, G., Sultan, F.: Online trust: a stakeholder perspective, concepts, implications, an future directions. Journal of Strategic Information Systems 11(3), 325–344 (2002)
20. Gutowska, A.: Research in Online Trust: Trust Taxonomy as A Multi-Dimensional Model. Technical Report, School of Computing and Information Technology, University of Wolverhampton (2007)
21. Mayer, R.C., Davis, J.H., Schoorman, F.D.: An Integrative Model of Organizational Trust. The Academy of Management Review 20(3), 709–734 (1995)
22. Ling, T.C., et al.: Baker & McKenzie - Global Privacy Handbook. In: IACCM (2012)
23. Weckemann, K., Satzger, F., Stolz, L., Herrscher, D., Linnhoff-Popien, C.: Lessons from a Minimal Middleware for IP-based In-car Communication. In: Proc. of the IEEE IV 2012, pp. 686–691. IEEE (2012)
24. MW AG web site. Navigation system Professional, http://www.bmw.com/com/en/insights/technology/technology_guide/articles/navigation_system.html
25. Facebook homepage, http://www.facebook.com/
26. Safebook homepage, http://www.safebook.us/home.html
27. Xen® hypervisor homepage, http://www.xen.org/
28. Garfinkel, T., Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: Proc. of NDSS Symposium 2003. Internet Society (2003)