# oMAC : Open Model for Automotive Cybersecurity

Vincent Hugot, Adrien Jousse, Christian Toinard, Benjamin Venelle

HAL Id: hal-02497993

https://hal.archives-ouvertes.fr/hal-02497993

Submitted on 6 Mar 2020

# oMAC : Open Model for Automotive Cybersecurity

Vincent Hugot[1], Adrien Jousse[2], Christian Toinard[1], and Benjamin Venelle[2]

[1] INSA Centre Val de Loire, *LIFO*, Bourges, France
{first.last}@insa-cvl.fr
[2] Valeo, *GEEDS*, Créteil, France
{first.last}@valeo.com

**Abstract.** Automotive architectures are constantly evolving to offer new features to end-users. Safety is critical, and cannot, in modern connected architectures, be achieved without strong cybersecurity. This paper proposes an advanced model of architecture supporting efficient cybersecurity properties. The global approach is an open way to include cybersecurity components that dynamically enforce the requested properties in a distributed manner. It supports heterogeneous software and hardware components and allows detection of malicious or unsafe behaviors. The paper illustrates the ability to integrate such a model into a standardized automotive middleware such as SOME/IP.

**Keywords:** Cybersecurity model · Access control · Automotive · Middleware · Heterogeneous architecture.

## Introduction

The automotive industry is moving to an extended computerization of vehicles either through connected cars or the autonomous driving feature. Consequently, modern cars embed their own information system built on top of many heterogeneous Electronic Control Units (ECU) — up to 150 — and field buses (Ethernet-based, CAN, LIN. . . ). This exposes cars to various threats including car theft [1], manslaughter [2], ransomware [3], or coordinated attack through botnet networks [4]. To face these disaster scenarios, car manufacturers and their suppliers have to deploy the appropriate cybersecurity measures [5].

Vehicles are systems capable of causing physical harm (cf. Table 1). Safety measures are therefore critical, and mandatory [6]. As vehicles become increasingly connected, and attackers can remotely take control of the car — *e.g.* the Jeep Cherokee case [7] — cybersecurity measures shall contribute to safety.

More recently, vulnerabilities on BMW cars [9] allowed attackers to disrupt the safety of the car. By exploiting multiple vulnerabilities, intruders were able to gain remote access to the CAN bus and control critical ECU. These attacks showed that the modern car needs a dynamic defense-in-depth to adapt permissions to the vehicle's resources (actuators, sensors, services) according to the

Table 1: Typical ASIL [6] classification according to [8]

| Subsystem | Type of failure | ASIL |
|---|---|---|
| Rear lights | Both side failure | A |
| Brake lights | Both side failure | B |
| Headlights | Both side failure | |
| Instrument cluster | Loss of critical data | |
| Rear view camera | No valid sensor data | |
| Vision ADAS | Incorrect sensor feedback | |
| Active suspension | Suspension oscillates | B - C |
| Radar cruise control | Inadvertent braking | C |
| Engine Management | Unwanted acceleration | C - D |
| Airbag | Inadvertent deploy | D |
| Antilock braking | Unintented full power braking | |
| Electric power steering | Self-steering | |

vehicle's state. Despite the efficiency of mandatory access control [10], the authorization and the enforcement must be dynamic to prevent a false positive or false negative decision.

This paper studies the evolution of automotive architectures *wrt.* cybersecurity aspects. It proposes an open model that improves the cybersecurity of post-2020 vehicle architectures. Additionally, the paper shows how to integrate the model into a standardized automotive middleware: SOME/IP.

Section 1 presents various automotive architectures, their evolution, their flaws and compatible cybersecurity solutions. Section 2 describes our open model for automotive cybersecurity. Section 3 focuses on a mandatory component allowing the integration of our approach into an existing automotive middleware: SOME/IP. Section 4 describes an automotive architecture using our model. The end of the paper concludes and sketches future works.

## 1 State of the art

### 1.1 Requirements for Secured Automotive Architectures

Security aspects of automotive architectures will be evaluated *wrt.* the principle of least privilege [11]. This principle states that an entity should solely have the required privileges to fulfill its tasks. Two corollaries result from this principle:

- Separation of privileges — An entity with a fixed set of privileges shall not obtain further privileges.

– Separation of duties — An entity can legitimately require new privileges. That situation requires a mediating entity to handle the privilege elevation.

The proposed architecture must comply with automotive constraints regarding robustness, cybersecurity and safety. This paper is focused on cybersecurity.

## 1.2 Security of Automotive Architectures

Since the 1970s, the number of ECU has grown significantly, leading to higher network requirements. For instance, Figures 1a and 1b present the shift from a single fieldbus network to several fieldbuses connected through a gateway. This evolution limits the bus occupation for safety considerations. In this type of architecture, the least privilege principle is not reached as each ECU can access to any other ECU. In addition, cybersecurity is limited to perimetric defenses. Such networks are described as "crunchy" [12], and are attractive targets for attackers. For example, the On-Board Diagnostic (OBD) port [13] offers a complete access to the car's fieldbuses. The advanced connectivity of automotive systems (Bluetooth, WiFi, Vehicle-to-Everything) increases the attack surface [7]. Figures 1c and 1d show how to improve the cybersecurity of a connected car. A Cybersecurity Component (CsC) acts as a GW that detects or prevents intrusions. The CsC is the mediating entity providing duties separation.

Devices with similar functional purposes are grouped into functional domains implemented as virtual networks (VLAN) — already in use in current cars — allowing privileges separation. Therefore, this type of architecture can meet the least privilege principle. VLAN also help detaching the logical architecture from the physical one. Future architectures will likely be constituted of at least three



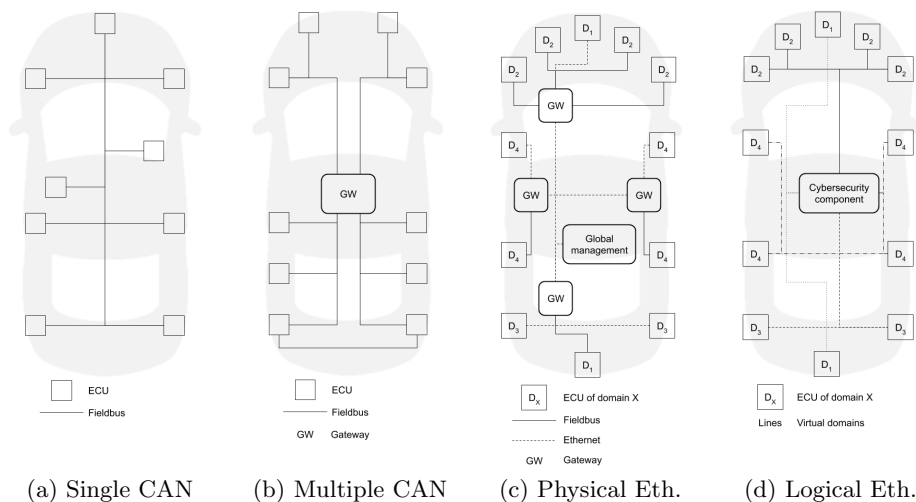(a) Single CAN      (b) Multiple CAN      (c) Physical Eth.      (d) Logical Eth.

Fig. 1: Automotive architectures evolution trend from CAN bus to Ethernet

domains — *Management*, *Infotainment*, and *Core* — isolated from the remainder of the vehicle behind a CsC. The *Core* domain is the only one required to keep the car under control. *Management* and *Infotainment* domains are useful but not safety critical. Exposed assets — Bluetooth, WiFi, 5G — are located at the far end of the architecture within a Telematics and Communication Unit (TCU) implementing cybersecurity measures. These components are logically far from critical components. An additional domain, *Shared*, located between the TCU and the CsC provides features for the entire architecture — Event Data Recorder, OTA Manager... Post-2030 candidate architectures like FACE [14] shift to virtualized ECU controlled by an orchestrator [15]. The proposed model also covers these architectures. A CsC can offer different cybersecurity mechanisms. First, a network firewall can filter malicious communications at the lower layers of the OSI model [16]. Consider for instance an ECU holding multiple functions, such as an Automotive Cruise Control (ACC) and a Line-Keeping Assist (LKA), that need to communicate with a Transmission Control Module (TCM) and a Steering Control Module (SCM), respectively. Using a firewall to block communications between the ACC and the SCM can harm communications between the LKA and the SCM (*c.f.* Figure 7).

Second, NetLabel [17], Next-Gen Firewalls (NGFW) [18] and Filtering Proxys [19], have important drawbacks:

  − NGFW cannot work in real-time and so handle safety messages;
  − NetLabels require labeling capabilities;
  − They are unaware of the vehicle's state;

For instance, regarding the last point, the flow between the ACC and the TCM must be allowed according to the ACC's state (state = on : flow = authorized; state = off : flow = blocked). NetLabel, NGFW, and Filtering Proxys are all unable to adjust permissions dynamically according to the vehicle's state.

Consequently, i) additional controls are required at the upper layers (5 to 7) of the OSI model and ii) automotive middlewares shall control the system interactions; both of them taking into account the state of the vehicle.

### 1.3  Cybersecurity of Distributed Objects

Automotive systems are composed of heterogeneous hardware and software components. This heterogeneity is due to the numerous contractors, tasks and safety constraints at stake. These components need a standardized software interface to communicate: the middleware.

Table 2 briefly sums up the evolution of middlewares of interest. Remote Procedure Calls (RPC) are among the first middlewares [20]. With the development of object oriented programming and a large adoption of networking, the Object Management Group (OMG) standardized the Common Object Request Broker Architecture (CORBA) [21]. This standard specifies the interfaces and various type of services (events, message passing, remote procedure call...), quality of services (time constraints, cybersecurity, transaction, robustness...)

and application domains (process control, health, insurance, bank, transportation...). Web services (WS) are inspired by CORBA. In contrast with CORBA, WS provide limited services and domains of usage. In the automotive industry, WS usage is restricted to end user services such as infotainment or mobile applications. Indeed, they do not support time constraints or real-time applications.

The automotive application domain requires specific services (events, time constraints, real-time...) in order to support safety. Since the CORBA approach fits with these requirements, the automotive Data Distribution Service (DDS) middleware [22] is directly derived from CORBA. In the automotive business, AUTOSAR standardized the Run-Time Environment (RTE) middleware [23] for inter and intra ECU communications. RTE is partially inspired by RPC. In contrast with RPC, RTE has signal-passing and time-constraint services. RTE supports the SOME/IP communication standard that has some connections with CORBA, *e.g.* method calls. In contrast with CORBA, SOME/IP does not include cybersecurity services.

Table 2: Middleware's landscape in various domains

| Middleware | Inspired by | Paradigm | Features |
|---|---|---|---|
| RPC | | Procedure Call | |
| CORBA | RPC | Procedure call Publish Subscribe Fire & Forget | Cybersecurity Time constraints |
| 🚗 DDS | CORBA | Publish Subscribe | Cybersecurity Time constraints |
| 🚗 RTE | RPC | Signal Passing | Time constraints Automotive safety |
| 🚗 SOME/IP | CORBA RTE | Procedure Call Publish Subscribe Fire & Forget | Time constraints |
| WS | CORBA | Fire & Forget Procedure Call | Cybersecurity |

Both CORBA and DDS have built-in and proven approaches for authentication, authorization, and auditing. However, they do not address a dynamical distributed cybersecurity that adapts the enforcement to the vehicle state. Our open Model for Automotive Cybersecurity aims at supporting both the DDS and SOME/IP middleware. Indeed, it is important to be agnostic *wrt.* the underlying communication middleware since the standardization is continuously

evolving. Moreover, keeping the model agnostic is a good approach to address the heterogeneity problem.

The following section describes our model in order to enforce the cybersecurity of automotive architectures.

## 2   A cybersecurity model for automotive systems

This section illustrates our model using a concrete automotive exemple. Our approach takes into account the state of the components in order to provide a dynamic MAC approach. We will illustrate how a MAC approach based on access control automata can enforce stronger cybersecurity properties compared to standard MAC models.

Figure 2 is a simplified view of an automotive Telematics and Communication Unit (TCU). It is composed of a Network Access Device (NAD), in charge of the different network accesses — 4G, Bluetooth, GPS... An Application microprocessor (App $\mu$P) — running Linux — acts as a safety broker between the NAD and the Safety microcontroller ($\mu$C). The Safety $\mu$C — running AUTOSAR — manages communication with the remainder of the architecture.
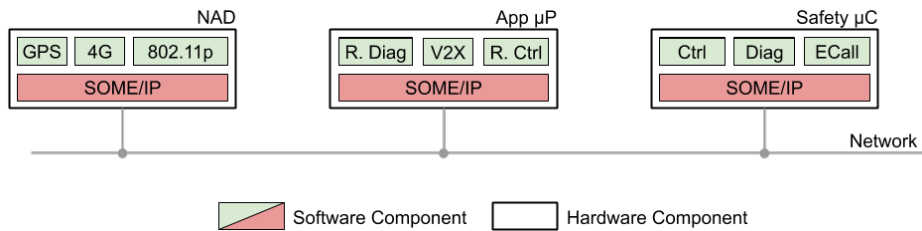


Fig. 2: Simplified view of a TCU's components

Considering the use cases of a remote diagnostic (RD) and a remote control (RC), two annotated functional flows can be defined. The RD flow (1) starts in the 4G software components of the NAD which will send a message to the RD component of the App $\mu$P. The App $\mu$P's RD will in turn send a message to the Diagnostic (Diag) of the Safety $\mu$C. The RC flow (2) works similarly.

$$Allow\ NAD \xrightarrow{4G\ invoke\ RD} App\ \mu P \xrightarrow{RD\ invoke\ Diag} Safety\ \mu C \qquad (1)$$

$$Allow\ NAD \xrightarrow{4G\ invoke\ RC} App\ \mu P \xrightarrow{RC\ invoke\ Ctrl} Safety\ \mu C \qquad (2)$$

These flows describe the sequencing of the network messages in order to realize the desired action. To enforce these behaviors, standard MAC models would define policies of the form : $Allow\ NAD \longrightarrow App\ \mu P$. However, these policies enforce neither duties separation (as $App\ \mu P \longrightarrow Safety\ \mu C$ is possible

before $NAD \longrightarrow App \ \mu P$), nor privileges separation (as $App \ \mu P \xrightarrow{RD \ invoke \ Diag}$ $Safety \ \mu C$ can happen after $NAD \xrightarrow{4G \ invoke \ RC} App \ \mu P$). In addition, standard MAC models suffer from the complexity of their policies as they specify every interaction of the system. An adequate access control policy should solely concern interactions of interest and take into account the sequencing of messages.

Some works have explored solutions using annotated functional flows. [24] proposes to monitor specific relations between objects. If an observed relation is not in the monitored list, it is considered as functional noise and implicitly authorized. This enables having a non-exhaustive policy by specifying only the relations needing to be controlled. A smaller policy being generally faster to process, this solution is faster compared to classical MAC solutions. With fewer relations, update and validation of the policy become easier, allowing dynamicity.

The approach of [24] uses automata to describe functional flows. Figure 3 represents the corresponding automaton of a simplified remote diagnostic scenario. This automaton guarantees privileges separation as it enables the rule $App \ \mu P \longrightarrow Safety \ \mu C$ only after monitoring the rule $NAD \longrightarrow App \ \mu P$. With the sources and destinations software components it also provides privileges separation as the rule $App \ \mu P \xrightarrow{RD \ invoke \ Diag} Safety \ \mu C$ is enabled after monitoring $4G \xrightarrow{4G \ invoke \ RD} App \ \mu P$.
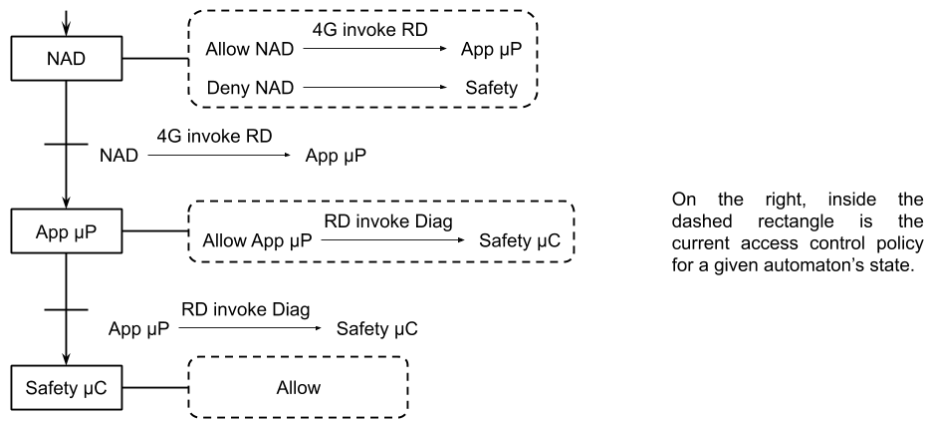


Fig. 3: Access control automaton

Figure 4 sums up our proposition. Several reference monitors [25], distributed on the architecture, jointly enforce the expected global behavior through sub-automata. Each reference monitor can have a partial view of the car's access control policy. A Cybersecurity Component, if present, could add a synchronization mechanism for the different automata.
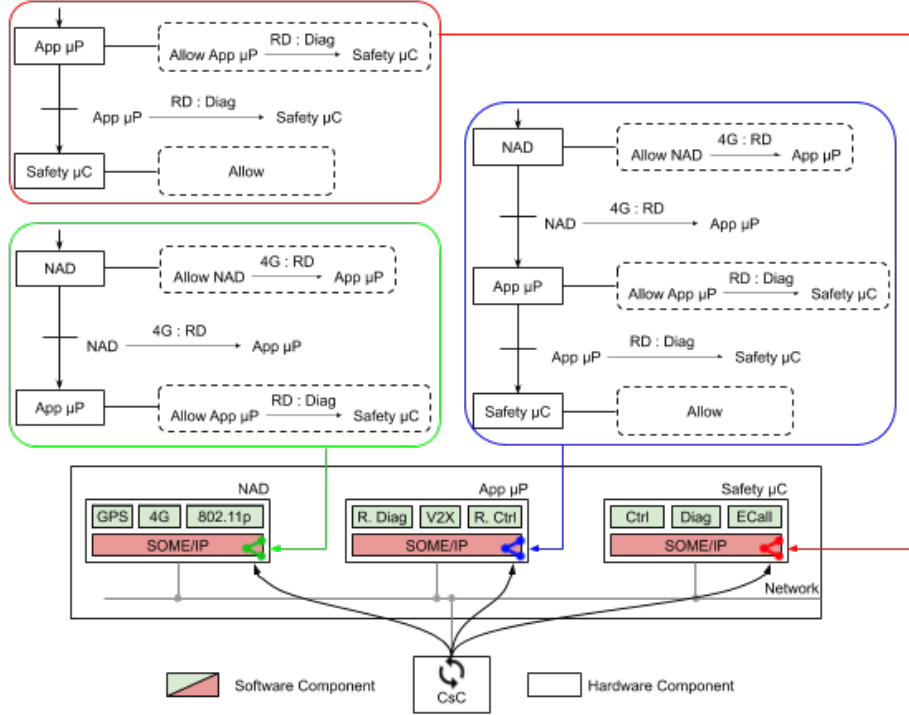
Fig. 4: Implementation of our model in the architecture of Figure 2

As a proof-of-concept, we chose to modify SOME/IP to support our protection model. The following section details the proposed modification of SOME/IP and illustrates a reference monitor that controls an automaton.

## 3 SOME/IP as use case

We now propose an extension of the SOME/IP middleware in order to enforce a policy defined by an automaton.

### 3.1 Test Configuration

A variant of the configuration depicted by Figure 5 was used to test the implementation. For clarity purposes, SOME/IP's hexadecimal identifiers are voluntarily omitted. The policy applied is the one presented in Section 2.

$$Allow\ NAD \xrightarrow{4G\ invoke\ RD} App\ \mu P \xrightarrow{RD\ invoke\ Diag} Safety\ \mu C \qquad (1)$$

$$Allow\ NAD \xrightarrow{4G\ invoke\ RC} App\ \mu P \xrightarrow{RC\ invoke\ Ctrl} Safety\ \mu C \qquad (2)$$
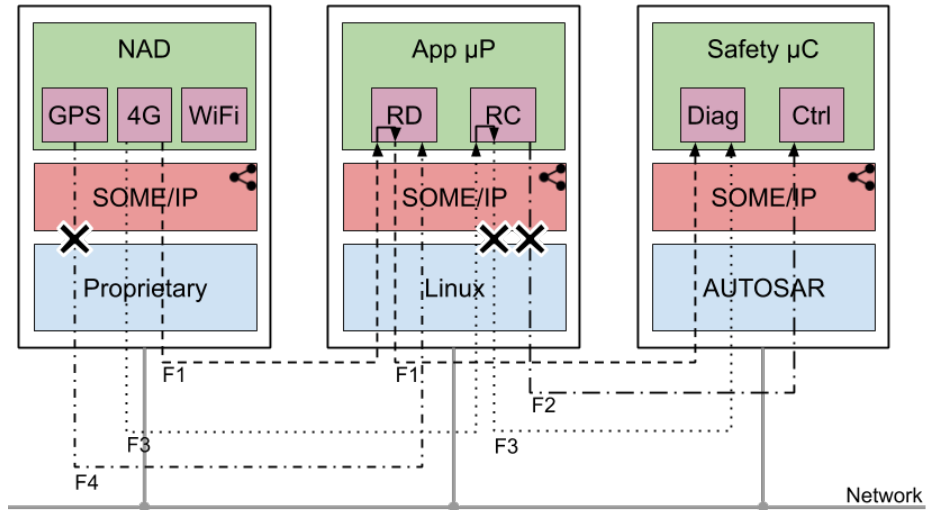
Fig. 5: Test scenario

The functional flow F1 of Figure 5 represents the calls of a Remote Diagnostic (RD) following rule (1) as described in Section 2. The flow F2 shows that duties separation is enforced in our model as the Remote Control (RC) of the App $\mu$P cannot call the Control (Ctrl) of the Safety $\mu$C without receiving a call from the Network Access Device (NAD). With the flow F3, privileges separation is also enforced: once in the functional flow of the RC it is not possible to switch to Diagnostic. The flow F4 shows that local software components are distinguishable as the NAD's GPS cannot call the RD.

Without our access control feature, each call represented in Figure 5 succeeds. With it, the monitor blocks the calls, which is denoted by a cross in Figure 5. Calls can be blocked by the sender or the receiver, depending on the local policy — *e.g.* if the receiver has a local *Deny* rule. With the current implementation, the notion of domain is not controlled. Future works will make it possible to add controls regarding the domains. For example, this feature will be useful if a component of a domain offers a method which should not be freely accessed.

## 3.2 Implementation requirements

To implement our access control mechanism, it was necessary to extend the SOME/IP protocol. The header of SOME/IP [26] already contains enough information for the implementation of a basic access control policy of the form *Client ID* $\longrightarrow$ *Service ID* : *Method ID*. This policy expresses rules of the form *A given client can (or cannot) call a given method of a given service*. If the method is omitted, the interpretation is *A given client can (or cannot) call any of the methods of a given service*. However, the implementation of more general rules requires additional information. As we saw in section 1, ECU tend to be grouped

in functional domains which can be encompassed by a domain policy. Therefore, we enhanced SOME/IP messages with the addition of a domain identifier, specified for each SOME/IP application. Likewise, the header lacks an identifier for the caller method; if different methods of a client call the same method of a service, it is not possible to distinguish them. The addition of a caller method to SOME/IP messages gives us a finer grain. Finally, an authentication field was added, without which the access control feature could be impersonated. The proposed solution is backward-compatible in order to integrate into existing products. The impact on network communication should be kept as low as possible. Real-time constraints will not be addressed for this proof-of-concept.

To add these new fields to SOME/IP messages, different solutions were available. The first one is to encapsulate SOME/IP messages into secured-messages.

### 3.3 Encapsulation

This approach is similar to HTTP with SSL. The HTTP packet is encapsulated in a SSL packet. SOME/IP messages would be encapsulated into secured-messages. Secured-messages carry information presented in section 3.2. This solution has a major drawback in the case considered in this paper: it does not preserves backward compatibility. If the SOME/IP message is encapsulated, the standard library will not be able to find its header because it will be padded by the secured-message's header. Encapsulating the message requires a level of processing before letting the message flow through SOME/IP. If this level is not present on some ECU, they will not receive the messages. Backward compatibility is therefore lost and this solution is inadequate for the considered use case.

### 3.4 Protocol overload

The second option is to change SOME/IP at heart. This can be done in two ways. The first consists of encapsulating the payload with cybersecurity information.

**Payload** Encapsulating the payload is the approach used by Secured Onboard Communication (SecOC). The header of the message — used for routing purposes — remains untouched. The payload is encapsulated in a secured payload, carrying cybersecurity information. This approach mixes application data with unrelated cybersecurity information. It does not overload SOME/IP's own information, but adds another layer of processing on top of it, before the functional processing. Backward compatibility is not preserved because as with the encapsulation presented earlier, another level of treatment is needed.

**Header** The second approach consists of overloading SOME/IP's header. Cybersecurity information would be concatenated with the standard header. In contrast with the last approach, functional data are not mixed with cybersecurity information. However, the change in the size of the header breaks backward

compatibility. The length field of SOME/IP messages indicates the payload's length plus eight bytes (accounting for the five last fields of the header). If the length takes into account the cybersecurity information of the header, a standard library will consider that the payload starts at the first cybersecurity information. $PayloadLength = HeaderLengthField - HeaderStandardLength$. If the header length field does not take into account the cybersecurity information, an offset should be added when deserializing the payload. Without changing the header standard length or the deserilization step, backward compatibility is broken.

As for the encapsulation, overloading SOME/IP breaks backward compatibility. In both cases, it introduces a new level of processing to interpret cybersecurity information. Cybersecurity should be handled by SOME/IP, without changing the header. A footer therefore seems the right approach.

### 3.5 Footer

The third option is to add a footer to SOME/IP messages, using an approach similar to the PRP protocol [27]. Cybersecurity information are packed into a footer — of fixed length —, glued at the end of the payload. The header and payload values remains untouched. This implementation allows us to send secured messages to both secured and unsecured clients. A secured client will look for a special value right after the payload. If this value is found, the message has a secured footer. If not, the message is processed as a standard SOME/IP message. When an unsecured client receives a secured message, it will try to interpret the footer as another SOME/IP message — as SOME/IP allows sending multiple messages in a single UDP packet or TCP segment [26]. By having a different structure than the header, this operation will fail. The footer will therefore be dropped and an error message will be sent to the emitter. However, with this approach, the client still receives and interprets the SOME/IP header and payload. The footer allows flexibility as other fields can be added if need be. As stated earlier, the chosen ones are: a calling method ID, a domain ID and an authentication field as a Message Authentication Code (MACo in this paper in order to distinguish it from Mandatory Access Control).

The footer approach is the only one preserving backward compatibility. The impact on messages' length should be lower than with the encapsulation solution, as no extra header is needed. It keeps application information separated from cybersecurity information. This approach has been the one implemented and used for the tests of section 3.1; the next section reports on the implementation.

### 3.6 Implementation

Adding a footer to SOME/IP's messages has a low impact on the SOME/IP standard library[3] in terms of code overhead. The new messages ($message\_secured\_impl$), which include the footer, inherit the standard messages of SOME/IP (as shown

---

[3] vSOME/IP implementation: https://github.com/GENIVI/vsomeip

in Figure 6). This allows to keep standard messages unaltered, in case they are required for future works. Service-discovery-messages are not modified, as it was not a requirement for this solution. Methods were overrided in order to replace standard messages by secured ones and hooks were added to route messages in the reference monitor before releasing them on the network. By seeing every message, the reference monitor knows the state of each component. If a message does not comply with the policy, it is dropped. The ability to use a control automaton enables safety tasks such as the reset of an ECU in specific situations.
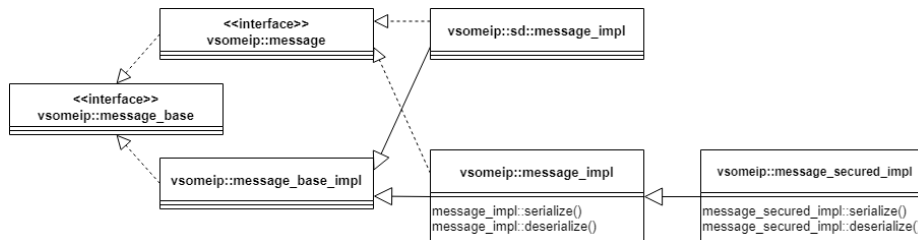


Fig. 6: SOME/IP's messages inheritance UML class diagram

To authenticate the message, a MACo was needed. The hashing function used to compute it is CMAC [28]. Cryptographic tasks such as this are usually handled by a Hardware Security Module (HSM), which is generally faster. The final size of the footer is 24 bytes, 16 of which account for the MACo.

The domain of a SOME/IP client (or service) is passed using an environment variable in the same way as the name of the application is passed. This solution should be replaced by setting the domain in the configuration file, which is a more consistent approach in our opinion.

The calling method field is the only one that can have an impact on developers habits. They can set it in the application code, in accordance with the defined policy. This field is currently used for forwarded calls. In case of a method acting as a broker, the called method ID is copied in the calling method field.

The proposed implementation blocks successfully undesired calls according to the specified automaton. Though an incomplete implementation of our model, our experiment shows that an existing middleware can be extended in order to add advanced cybersecurity. Since the approach is agnostic with regard to the middleware, the cybersecurity does not impact the development chain. The following section describes a state-of-the-art automotive architecture which benefits of our implementation.

## 4    Perspectives

Consider automotive architectures based on automotive grade Ethernet [29] [30]. Figure 7 is a populated version of the architecture of Figure 1d based on the

description made in section 1.2. Here, a *Perception* domain is added as a subdomain of the *Core* domain. Compared to previous architectures — *c.f.* Figures 1a and 1b — domains are interconnected through gateways using Ethernet links in place of CAN buses. This provides the backbone for the system, thanks to the large bandwidth of Ethernet. ECU can either be connected through Ethernet or fieldbuses depending on their connectivity. Far ends actuators and sensors will still use fieldbuses as they are lacking resources to handle Ethernet connectivity.

This architecture provides better cybersecurity properties compared to the architectures presented at the beginning of this paper. The least privilege principle is enforced through the usage of VLAN and the CsC. Domains are clearly separated, restricting an attacker capability to move from one domain to another.
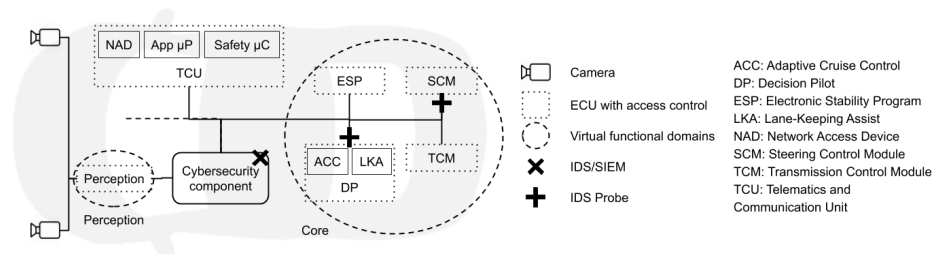


Fig. 7: Post-2020 secured vehicle's architecture

However, if an attacker breaks into a domain (*resp.* an ECU), he gains a complete access to the entire domain (*resp.* ECU). For example, if an attacker corrupts the ACC feature, he can also corrupt the LKA, as there is not necessarily any hardware-based memory protection. He could also send messages to components of the domain, assuming the CsC implements basic filtering capabilities, and disrupt the car's behavior. In this situation, current cryptographic solutions such as SecOC [31] are ineffective, as the attacker has access to the cryptographic material. A compromised ECU implies a cryptographic compromision. Mitigating locally the attack on a ECU could be achieved with a microkernel — such as PikeOS. ECU embedding multiple functions — such as the ECU with the ACC and the LKA from Figure 7 — would also benefit from a microkernel. A microkernel, by its architecture, would isolate functions running on the same microprocessor [32]. Thus, corrupting a function through, for instance, a memory corruption attack, would prevent an attacker from having any impact on other functions hosted on the ECU, assuming these functions are not dependent.

However, a microkernel does not prevent an attacker from sending messages to other ECU. This type of attack can be blocked by our mandatory component integrated to SOME/IP through the use of functional flows (*c.f.* section 2). Without our approach, interactions between the NAD, the App $\mu$P and the Safety $\mu$C cannot be efficiently controlled. In case of compromision of a feature, the reference monitor would constrain the attacker to the defined flow of callable

methods authorized for this feature. In further versions, the reference monitor could react if the ECU's policy is infringed upon. In case of defective policy, the CsC would receive OEM-approved policy updates from the cloud and push them to local components.

Domain-designed architectures offer strong cybersecurity properties. However, without suitable mechanisms to achieve defense-in-depth, cybersecurity stays perimetric. Our mandatory component integrated to SOME/IP allows for a strong and widespread defense-in-depth in automotive architectures.

## Conclusion and future works

The automotive industry is living its computing revolution. Customer needs are evolving, standards follow, and the industry is adapting. In recent years, it has been shown that cybersecurity is necessary to protect safety objectives. The proposed model partly addresses this need. It is agnostic *wrt.* the middleware and could therefore support future automotive middlewares like DDS. As a proof-of-concept, we propose a partial implementation of our model for SOME/IP.

The main objective of the proposed solution has been reached: an access control feature has been added to an automotive middleware without breaking backward-compatibility. This feature allows to block undesired calls according to the specified automaton. The current stage of development does not support the notion of domain. The implementation proposed in this paper shows that adding cybersecurity can be done without disturbing the development process.

Future works will extend the development to fully support the model defined in section 2. They will focus mainly on the way the automata can be specified, synthesized, distributed, and coordinated in order to implement an efficient distributed access control. Additionaly, the CsC can be improved with a Security Intrusion and Event Management (SIEM) system or an Intrusion Detection System (IDS) built on top of our solution. Those security mechanisms can correlate different probes to address broader attacks, thanks to our automaton approach.

## Acknowledgments

## References

1. West Midlans Police. Relay attack solihull. Online, November 2017.
2. Y. Shoukry et al. Non-invasive spoofing attacks for anti-lock braking systems. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 55–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
3. W. Marko et al. Wanna drive? feasible attack paths and effective protection against ransomware in modern vehicles. In *ESCAR 2017*, 2017.

4. M. Turker Garip et al. Congestion attacks to autonomous cars using vehicular botnets. In *NDSS 2015*, January 2015.
5. ISO. ISO/SAE CD 21434 Road Vehicles – Cybersecurity engineering, 2018.
6. ISO. ISO 26262 Road vehicles – Functional safety, 2011.
7. A. Greenberg. Hackers remotely kill a jeep on the highway - with me in it. Online, July 2015.
8. Embitel. Understanding how iso 26262 asil is determined for automotive applications. Online, April 2018.
9. M. Kumar. Chinese hackers find over a dozen vulnerabilities in bmw cars. Online, May 2018.
10. A. M. Harrison et al. Protection in operating systems. *Communications of ACM*, 19(8):461–471, 1976.
11. J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE 63-9*, 1975.
12. McAfee Enterprise. Stealth attacks: Why hackers love networks that are 'crunchy on the outside and chewy on the inside'. Online, January 2014.
13. EPA Office of Transportation and Air Quality. On board diagnostics (obd) basic information. Technical report, U.S. Environmental Protection Agency, 1996.
14. CEA. Ces 2019 : Cea tech to exhibit at ces las vegas on january 8–11 imagine the car of tomorrow. Online, January 2019.
15. NFV White Paper. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1. October 2012.
16. N. Freed. Behavior of and requirements for internet firewalls. *RFC*, 2979:1–7, October 2000.
17. P. Moore. Introduction to labeled networking on linux. Online, 2008.
18. Learn about firewall evolution from packet filter to next generation. Online, January 2015.
19. Vector cyber security solution – vhsm firmware. Online, October 2018.
20. Sun Microsystems. Rpc: Remote procedure call protocol specification. *RFC*, 1050:1–24, April 1988.
21. Object Management Group. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.3*, November 2012.
22. Object Management Group. *Data Distribution Service Version 1.4*, April 2015.
23. Autosar. *Specification of RTE Software v4.3.1*, December 2017.
24. B. Venelle. *Mandatory access control for object systems: defense in depth for Java objects*. PhD thesis, University of Orléans, France, 2015.
25. DoD 5200.28-STD. *Orange Book*. Dod Computer Security Center, December 1985.
26. Autosar consortium. *SOME/IP Protocol Specification*, November 2016.
27. IEC. IEC 62439-3:2016 RLV Redline version Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR), 2016.
28. M. Dworkin. Nist special publication 800-38b recommendation for block cipher modes of operation: The cmac mode for authentication.
29. Ieee 802.3bw-2015 - ieee standard for ethernet amendment 1: Physical layer specifications and management parameters for 100 mb/s operation over a single balanced twisted pair cable (100base-t1). Standard, IEEE, 2015.
30. Ieee p802.3cg 10 mb/s single pair ethernet task force. Standard, IEEE, 2018.
31. Autosar. *Specification of Secure Onboard Communication v4.3.1*, December 2017.
32. A. S. Tanenbaum. *Modern operating systems, 4th Ed.* Pearson Education, 2014.