A Statefull Firewall and Intrusion Detection System Enforced with Secure Logging for Controller Area Network

Teri Lenard
Roland Bolboacă
teri.lenard@umfst.ro
roland.bolboaca@umfst.ro
University of Medicine, Pharmacy, Science and Technology
of Târgu Mureş, Romania
Târgu Mureş, Mureş, Romania

ABSTRACT

The Controller Area Network standard represents one of the most commonly used communication protocol present in today's vehicles. While it's main properties facilitate the communication between different control units, several protocol design considerations represent security problems, that in the end makes it trivial for an attacker to gain access and control the system. The current work proposes a Statefull Firewall, together with a signature-based Intrusion Detection System as a response. Beside this, a Secure Logging unit is brought up in addition to support our methods, enforcing them with integrity verifiable logs.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Firewalls**.

KEYWORDS

intrusion detection system, firewall, controller area network

ACM Reference Format:
Teri Lenard and Roland Bolboacă. 2018. A Statefull Firewall and Intrusion Detection System Enforced with Secure Logging for Controller Area Network. In *EICC '21: European Interdisciplinary Cybersecurity Conference, November 10–11, 2021, Targu Mures, RO.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Today's modern vehicle incorporates a mix of interconnected network subsystems, for the purpose of offering an environment which encapsulates a wide variety of features. Although the complexity and heterogeneity of automotive systems grows at a fast pace, in the same manner we find an emerging number of sophisticated threats.

By directly targeting the underlying network communication, most threats intend to control critical system processes by means of replaying legitimate network traffic. The effectiveness of those threats relies on several designs considerations of the Controller Area Network (CAN) [13] protocol. Besides its wide utilization in automotive systems, CAN allows frame exchanges between different Electronic/Sensor Control Units, and between CAN sub-networks. In general, an automotive network incorporates several CAN networks, interconnected via a gateway, that functions alongside other communication protocols (e.g., LIN, MOST, SENT).

In order to attack a CAN system, malicious actors would initiate the attack by gaining physical access to: a in-vehicle CAN network, a On-Board Diagnostic Port, or by compromising an external access points (e.g., Wi-Fi hotspot). Once access is obtained, the attacker may then proceed to reproduce legitimate sequences of CAN traffic by replaying valid frames. Since CAN systems follow a broadcast communication pattern, the actual point from where the attacker conducts its activities may be irrelevant. The communication pattern allows such an attacker to reach any communication unit within the system, situated on the current compromised CAN bus, or on another bus positioned on the other side of the gateway.

The ability for an attacker to communicate with a control unit present in the system is facilitated by the fact that a number of data frames originating from one CAN network, can be further broadcasted by the gateway to other CAN networks. Typically, the CAN gateway is positioned as the delimiting entity of several CAN networks. However, traffic filtering techniques normally found in traditional Information and Communication Technology (ICT) are not present. Subsequently, intrusions within vehicles are not trivial to detect.

To tackle these issues, the current paper proposes a Statefull Firewall (SFW), together with a signature-based Intrusion Detection System (IDS) tailored for the CAN. By leveraging a set of whitelisted and blacklisted rules, the SFW is capable of filtering single, and sequences of related CAN frames in a statefull manner. Likewise, the IDS extends the properties of the SFW with the additional capability of performing deep packet inspection at the CAN frame byte level. By doing so, the IDS can identify known patterns associated with a particular intrusion. To enforce verifiable auditing of generated alerts, the developed approach is enhanced with Secure Logging (S-LOG) capabilities. S-LOG utilizes the Trusted Platform Module (TPM) [26] standard to generate verifiable log messages, resistant to external manipulation.

The rest of the paper is organized as follows. Related studies together with several background concepts are presented in Section 2. An in-depth description of the developed approach is given in

EICC '21, November 10–11, 2021, Targu Mures, RO

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Section 3. Afterwards, Section 4 focuses on the measurements and experiments, and finally, the paper concludes in Section 5.

2 BACKGROUND AND RELATED WORK

2.1 Vehicle Internal Architecture

The internal architecture of an automotive system encapsulates a series of network protocols, together with a significant number of control units and digital/analog sensors. The set of protocols include: the Controller Area Network (CAN) for Electronic/Sensor Control Units, Local Interconnected Network (LIN) or Single Edge Nibble Transmission (SENT) for digital sensors, and Media Oriented System Transport (MOST) for media devices. Typically, these networks are coupled to a more powerful gateway unit, which handles routing tasks and protocol translations. At the same time, the gateway itself may communicate with a Communication Control Unit (CCU) that manages the communication with exterior networks.

Being one of the most widespread protocols in automotive system, the CAN protocol was extended by Robert Bosch Gmbh into the CAN with Flexible Data-rate (CAN-FD) protocol [21], to remove the limitation of the CAN in terms of available data transfer per frame, bandwidth and transfer rates. Apart for the improvements brought up, CAN-FD maintains the general properties of the CAN.

The CAN protocol follows a broadcast communication pattern over a common bus, where control units and digital sensors use a bit-wise arbitration, based on the priority stored in the frame identifier [13] to access the communication bus. A data frame is identified by its unique identifier field; it contains a 64 bit data field, and, among other fields, a Cyclic Redundancy Check field for error correction.

From a security standpoint, the CAN standard does not provide guidelines on how data transfer should be secured. To support this issue, the standardisation unit AUTOSAR developed the Secure On-Board Communication [3] requirements, describing how data should be authenticated in an automotive network. In addition to this, CAN lacks traditional security components (e.g., firewall, intrusion detection). As demonstrated by the literature review, several solutions have been proposed to tackle the present problems, but their integration has shown little progress.

2.2 Trusted Platform Module

The Trusted Platform Module (TPM) [26] standard developed by the Trusted Computing Group (TCG) [9], describes the capabilities that a tamper-resistant, cryptoprocessor should meet in order to execute cryptographic operation in a isolated environment. Besides offering a set of general requirements for Information Communication Systems, TCG recently released a report in [8], describing how the security of a automotive vehicle can he harden with the help of the TPM standard.

Generally, in addition to traditional cryptographic procedures (e.g., encryption, decryption, digital signature), the TPM offers means based on which local applications can be attested during system run-time, together with their assets, using a set of Platform Configurable Registers (PCR).

PCRs represent a key feature offered by the TPM, that provide a hash-based method to aggregate measurements about the state of a

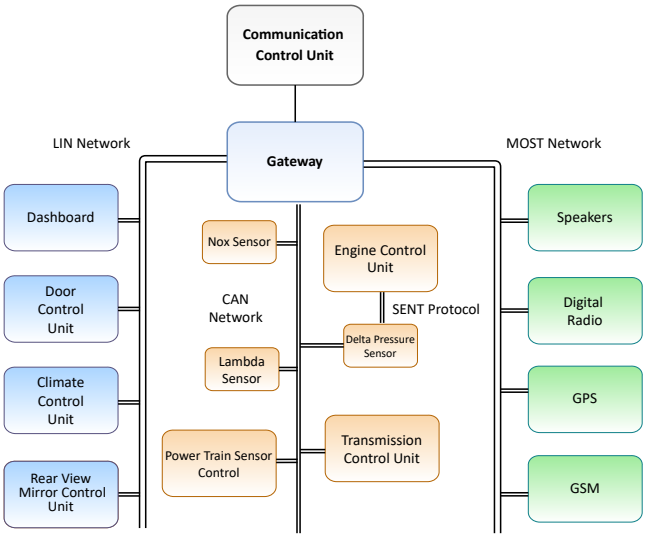

Figure 1: General architecture of the modern car.

software component [2]. A PCR supports a single operation, named *extend*, consisting in a hash function applied on the current PCR value, concatenated with the new data that requires measurement. An important aspect regarding PCRs, is that their state is always initialized during system start-up, or TPM power on, with a default value, denoted in most cases by a set of zeroes or ones. Later during run-time, by performing a series of *extend* operations on a given PCR, an application can attest the state of another application or its configuration file, by comparing the current PCR with a value stored in the past.

From the point of view of data storage, the TPM offers a limited amount of persistent storage inside its Non-Volatile memory. Due to its reduced size, additional security features are offered by the standard to store data outside the TPM. For cryptographic key storage, TPMs possess a set of key hierarchies, where, starting from a root key (or secret seed) available inside the TPM, keys can be deterministically generated during run-time [1]. One such hierarchy is the *storage hierarchy*. This hierarchy defines a Storage Root Key (SRK), representing a randomly generated key by the TPM's Original Equipment Manufacturer, accessible only to the TPM itself, that can't be swapped or recreated. By leveraging the SRK, a distinct type of secure on-disk storage for cryptographic keys, named *sealing* can be accomplished. Once a pair of keys is derived from the SRK, the private part is encrypted with the SRK (sealed), and the public part is kept unencrypted. By doing so, the TPM protects the secret part of the key pair, making it available only to itself.

Later in this paper, the features offered by the TPM will be adopted to implement the secure logging approach.

2.3 Related Studies

In the current scientific literature, we found a significant number of works, aimed at the automotive systems, with the purpose of hardening the system security with firewall and intrusion detection systems. Wolf, *et al.* [27] provided a study where a series of existent

threats and vulnerabilities were identified for automotive systems. Aside from this, they provided recommendations on how firewalls should be integrated at gateway level for protocols such as the CAN, LIN or MOST, to restrict diagnostic traffic during driving.

Luo and Hou [18] proposed an in-vehicle gateway firewall, capable of operating on CAN-FD networks and on infotainment Ethernet-based networks. Their proposed solution for the CAN-FD leverages a time window technique to monitor the changes in the information entropy to detect injection, flood, man-in-middle and replay attacks. Conversely, the Ethernet version, employs a state packet filter using rule tables based on IP address, port, protocol and other IP related parameter information. Similarly, but focused on communication originating from outside the vehicle, Jawahar *et al.* [14] proposed an application-controlled dynamic firewall for managing bi-directional interfaces between ports and applications using IP tables.

Besides firewalls, we find one of the earliest IDS dating back to 2009 [11]. In recent years, the number of proposed in-vehicular IDS increased significantly. Similarly to the IDS found in Information Communication Technologies, the detection methods consider either anomaly-based or signature-based (rule-based) approaches. On a more granular level, based on the algorithms used, the anomaly-based IDS can be frequency-based, machine learning-based, and statistical-based.

Groza and Murvay proposed several frequency-based approaches, such as [19] where they were able to detect illegal frame transmissions, as well as a estimation for position of the attack source, by monitoring the arrival time of the CAN frames. Furthermore, in [10], the same authors proposed an in-vehicle IDS capable of detecting replay and modification attacks by leveraging Bloom filters [4] in order to verify the frame periodicity, based on the CAN identifier and payload fields.

For more advanced, machine learning-based approaches, we find the work of Taylor, *et al.* [25], where one-class support vector machines were used to detect anomalies in the arrival frequencies of CAN frames. Seo, *et al.* [22] described a Generative Adversarial Net-based IDS using a deep learning model. Similarly, Kukkala *et al.* [16] proposed a gated recurrent unit-based recurrent autoencoder anomaly detector for the CAN. On the subject of in-vehicle IDS for the Ethernet network we find the work of Jeong, *et al.* [15], where an intrusion detection model based on feature generation and convolutional neural networks was developed. While neural network-based approaches proved to be efficient in terms of detection accuracy, their integration into real vehicles might prove challenging due to the limited resources and computational power of the in-vehicle systems.

Compared to prior techniques, we believe the advantages brought up by our approach are three-fold: (i) it builds on a rule processing engine that supports the development of a firewall and IDS in a single component; (ii) it provides a lightweight engine with low computational requirements, that can be integrated with ease; and (iii) we enforced our methods with logging unit, which uses the TPM standard to provide verifiable secure logs for generated events.

3 APPROACH

3.1 Overview

The developed approach builds on a Rule Processing Engine (RPE) capable of functioning, first as a Statefull Firewall (SFW), and second, as an Intrusion Detection System (IDS). Its behaviour is configurable and is based on a rule file associated with the RPE. While functioning as a SFW, the RPE processes CAN frames based on the frame identifier field. On the other hand, while operating as an IDS, a byte-level deep packet inspection is performed on the frame data field.

A distinct aspect for the SFW is that is meant to be positioned at the CAN gateway level to filter incoming frames, therefore limiting the amount of traffic further broadcasted by the gateway. To accomplish this with minimum additional latency, the patterns contained by the rule file should not present a complex structure.

On the other hand, the IDS allows a more in-depth analysis of the CAN traffic with more complicated rules. This inspection can introduce a significant delay, therefore the IDS serves as a passive entity, listening on the communication bus, and then generating events if pattern matches are detected.

The rule file associated with the SFW/IDS describes predefined sets of blacklisted or whitelisted rules. Each rule represents a pattern searched within an incoming CAN frame. Accordingly, a rule is bound to a action, which triggers an event if a pattern is found. To *statefully* process frames, rules are further chained together in sequences of rules, allowing the SFW/IDS to make decisions on a current frame, based on the past received traffic.

When the SFW/IDS detects a particular pattern, an action is generated. In the present context, an action defines the operation that must take place as a consequence to a specific event. An action can have one of the four values: `PERMIT`, `DROP`, `PERMIT-LOG` and `DROP-LOG`. The first action specifies that an incoming frame should be allowed to pass. At the opposite end, the `DROP` action indicates that a frame should not be further forwarded. Similar to these, the `PERMIT-LOG` and `DROP-LOG` actions additionally require that the event generated should be logged by the Secure Logging unit to enforce traceable and verifiable auditing of logs. An illustration of this procedure and of the high-level architecture is provided in Figure 2.

3.2 Rule Definition Language

The set of rules contained in the rule file are described using a Extensible Markup Language (XML) [5] based language. A pattern is defined as an *action rule*, which ultimately is applied on every data frame. An *action rule* can be linked together with a sequence of *action rules* creating a *state-chain*. This describes actions that must be taken on a sequence of frames, thus providing contextual detection capabilities. Subsequently, each *action rule* provides the ability to generate a hierarchical expression, allowing definitions for deep packet inspection rules. This step leverages boolean operators, such as `AND` and `OR`, used to link together different expressions.

Next, we provide a formal definition of the developed language. Let `Value(byteIdx, v)` be a predicate that returns `True` if within the CAN frame's data field, the byte at index `byteIdx` has value `v`, and `False`, otherwise. Similarly, let `ValueRange(byteIdx, v1, v2)` be a predicate that returns `True` if the byte at index `byteIdx`

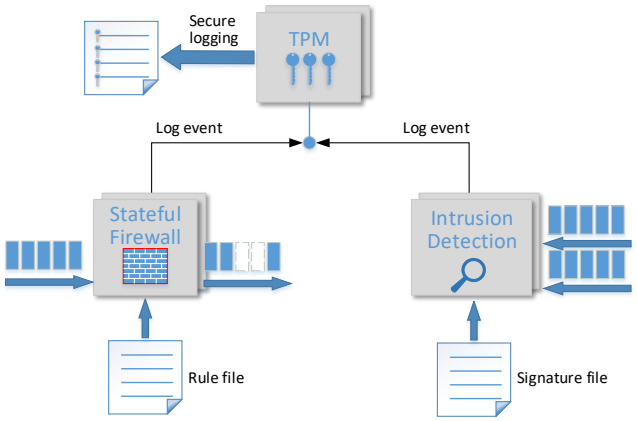

Figure 2: Architecture of the proposed Statefull Firewall and Intrusion Detection System.

is in the range [v1,v2], and False, otherwise. Based on these definitions, an expression that is associated later to an action rule is defined as:

```
actionExpr::= .
              || Value(byteIdx, v)
              || ValueRange(byteIdx, v1, v2)
              || (actionExpr) AND (actionExpr)
              || (actionExpr) OR (actionExpr),    (1)
```

where '.' denotes an empty expression. Building on the present definition, an action rule is defined as follows:

```
Action-rule ::= <Name, CID, actionExpr,
                        Action, Message>          (2)
```

According to the definition above, an *action rule* is characterized by the following fields: a string denoting the rule's name; the CAN identifier (CID); an action expression used in the case of deep packet inspection; and, the message used if the *action rule* requires logging.

Lastly, to enable statefull traffic inspection, a *state chain* is defined as an ordered sequence of action rules:

```
State-chain ::= <Action-rule1, Action-rule2,
                              Action-rule3, ...>   (3)
```

The similarity between SFW and IDS rules resides in the fact that both can support *action rules* and *state chains*. The difference relies in the definition of the *action rules*. For SFW the *action rule* structure does not contain the actionExpr field, which is particular to the IDS.

3.3 Rule Processing Engine

The Rule Processing Engine (RPE) represents the core component of the SFW/IDS. Its objective is to apply the predefined set of rules, as described in the previous section, to filter the incoming CAN frames, and finally, to generate actions. Depending on the set of rules, the RPE can act as a SFW, or as IDS by performing deeper inspection. Figure 3 illustrates the workflow for both cases.

Each incoming frame is processed in two distinct phases: (i) the *action rule* phase (Phase 1); and (ii) the *state chains* phase (Phase 2). In Phase 1, for each incoming frame, every *action rule* is verified independently. If a match is found, Phase 1 finishes, and the RPE continues to Phase 2. In this phase, *state chains* are executed following search patterns identifying series of frames. In a particular chain, every *action rule* is processed according to its definition, meaning that after the execution of each *action rule*, an action always occurs. At the end of the chain, similarly, an action associated with the chain is returned and processed properly. On the contrary, if a match is not found for a frame, implicitly, the PERMIT action is returned.

3.4 Secure Logging

The Secure Logging (S-LOG) unit handles incoming logging requests generated by the RPE in the case of a PERMIT-LOG or DROP-LOG actions. When such an event occurs, a message M is generated by RPE under the following structure (TIME, LEVEL, TEXT), where field TIME denotes the system time, LEVEL the importance of the log, and TEXT, a string containing additional information about the log. S-LOG takes M as input, and generates a cryptographic signature as output. S-LOG takes advantage of the TPMs PCRs in order to aggregate signed messages, and to attest the log files integrity.

To sign messages, S-LOG leverages a pair of asymmetric keys, denoted in the followings as (K_{prv}, K_{pub}). It is assumed that the keys were generated by a trusted authority (e.g., Original Equipment Manufacturer), in relation with the TPM connected to the Control Unit on which S-LOG runs. Using the TPM's *SRK*, the pair is derived as such:

$$\texttt{DERIV(SRK)} = (\{K_{prv}\}, K_{pub})), \tag{4}$$

where DERIV represents the key derivation function supported by the TPM, and {} denotes the *sealing* operation applied on K_{prv}, in which K_{prv} is encrypted with SRK for storage outside the TPM. Last but not least, S-LOG requires the use of a single PCR, denoted as P, to generate a secondary *integrity log*. The terms *signature log* will denote the log file containing signed messages returned by the SFW/IDS, while the term *integrity log* refers to an additional log file, containing a verifiable sequence of hash digests associated with the *signature log* files.

The logging procedure is initiated once a message M is received by the logging unit. S-LOG communicates with the TPM to generate a digital signature using the sealed pair of keys $(\{K_{prv}\}, K_{pub})$ as:

$$SIGN_{\{K_{prv}\},K_{pub}}\ (\texttt{M | P}) = y^M, \tag{5}$$

where y^M is the digital signature, and P represents the current value stored in the respective PCR. This process is repeated for every M until the *signature log* file reaches a predetermined size and a log rotate it's executed. Before rotating to a new file, the current value of P is read, and stored together with it's signature as the tuple:

$$(\texttt{P | } SIGN_{\{K_{prv}\},K_{pub}}\ (\texttt{P})) \tag{6}$$

into the *integrity log*. By doing so, a trace for every *signature file* is maintained, and any modification on a *signature file* will be reflected during auditing.

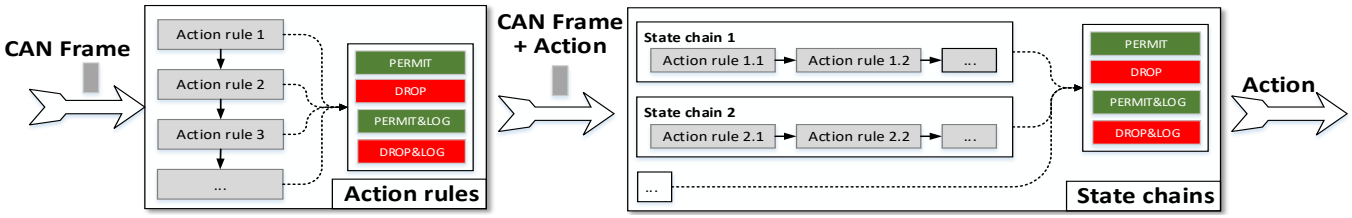

Figure 3: Rule engine workflow.

4 EXPERIMENTS

An important aspect regarding the feasibility of the proposed SFW/IDS relies in their performance, measured in the amount of time required to process a given frame, relative to different sized rule files. This yields the actual delay that the SFW causes to each CAN frame, and, similarly, the amount of time the IDS needs to detect a particular pattern. For the purposes of demonstrating the developed concepts, a physical test-bed replicating a CAN environment was developed.

4.1 Implementation details

The developed test-bed comprises two Raspberry Pi 3 model B+ (RPi) boards, each connected to a common CAN bus, exchanging frames via MCP2515 CAN controllers and a TJA1050 CAN transceiver. To simulate the CAN traffic and bus load of a real CAN environment, a trace of real CAN frames was used, originating from a KIA Soul vehicle. The CAN trace was made available by he Hacking and Countermeasures Research Lab (HCRL) [12, 17] and the University of Seoul, South Korea. The dataset comprises over two million can frames, 45 distinct frames spanning over a period of 17 minutes. From the original dataset, the first 100000 frames from 25 frames were selected.

Each RPi runs a *Automotive Grade Linux* (AGL) [6] distribution system built with *Yocto* [7]. The SFW/IDS is implemented in *C++*, the S-LOG module in *python* with the *TPM2-Tools* [23] provided by the open-source developers of the *Linux TPM2 & TSS2 Software* [24]. Furthermore, a data frame replay and routing module were developed with the help of the *SocketCAN* Linux package [20]. Last but not least, the rule files were defined, firstly for the SFW, and secondly for the IDS. Bellow, an example of a simplified rule is illustrated to showcase the developed approach. A *action rule* follows the below definition:

```
<rule cid="1" id="1-permit" action="PERMIT">
</rule>
```

In the listing above, an *action rule* was given, including its CAN identifier (cid), a unique identifier (id), and an associated action. A series of *action rules* are then linked into a *state chain* as in the following example:

```
<state-chains>
   <chain id="state-chain-1">
     <rule id="1-permit" action="PERMIT"/>
     <rule id="2-permit" action="PERMIT"/>
     <rule id="3-drop" action="DROP"/>
    </chain>
 </state-chains>
```

in the above example a series of *action rules* are chained together, each having its own associated action. If deep packet inspection is required, the following definition of a *action rule* can also be added to the rule definition file:

```
<rule cid="1" id="1-permit" action="PERMIT">
<payload>
  <expression>
    <operator type="AND">
      <byte index="0" value="50"/>
      <byte index="1" value-range="0..127"/>
     </operator>
   </expression>
</payload>
</rule>
```

The expression shown above will allow a frame if and only if the byte at index 0 has the value 50, and the byte a index 1 has a value in the range of $[0, 127]$.

4.2 Experimental Results

To showcase the approach, a scenario was envisioned where the performance, as well as the delays introduced by both the SFW and the IDS components were measured, based on the length of the defined rule file (i.e., the number of frames filtered by both components).

For both the SFW and IDS components 5 sets of rule files were created. The number of rules in each file ranged from 5 to 25 in increasing steps of 5. Both the SFW and the IDS components were installed and set up on a primary Raspberry PI board. For each experiment (i.e., for each component and for each rule file) from a secondary Raspberry PI, connected to the same network, a total number of 1000 CAN frames were sent using a baud rate of 250 kbps over a period of 180 seconds. Furthermore, for each frame that was filtered by the SFW and IDS, the execution time was measured.

The results of the measurements are showcased in Figure 4 and Table 1. Figure 4 presents the average processing time, per frame, for both the SFW and IDS components for each rule file and for the different types of defined actions. Moreover, a detailed description of the minimum, maximum and average processing times is shown in Table 1

5 CONCLUSIONS

The current paper proposed two approaches targeting known threats present in Controller Area Network systems. First, a new Statefull Firewall and a Intrusion Detection Systems were introduced, together with their internal functioning engine, and its rule language.

	Statefull Firewall						Intrusion Detection System					
Defined Actions	PERMIT/DROP			(PERMIT/DROP) & LOG			PERMIT/DROP			(PERMIT/DROP) & LOG		
Nr of rules	**5**	**15**	**25**	**5**	**15**	**25**	**5**	**15**	**25**	**5**	**15**	**25**
Min [ms]	0.035	0.033	0.036	0.038	0.035	0.037	0.035	0.034	0.036	0.037	0.039	0.037
Max [ms]	0.226	0.222	0.226	0.217	0.182	0.185	0.140	0.155	0.205	0.192	0.186	0.152
Avg [ms]	0.043	0.046	0.045	0.051	0.051	0.047	0.045	0.046	0.044	0.052	0.053	0.049

Table 1: Minimum, maximum and average measured frame processing times for the SFW and IDS components

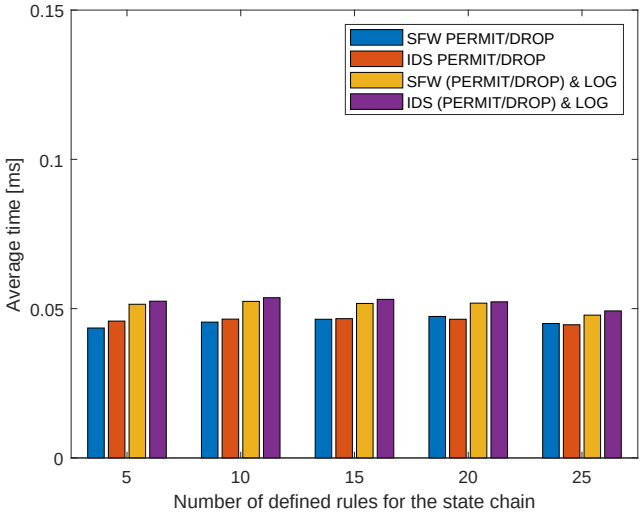

Figure 4: Comparison between the average processing times for the SFW and IDS components

Additionally, a Secure Logging unit was introduced to handle the security of the logs produced. Last but not least, the performance of the Statefull Firewall and Intrusion Detection System was measured on a real reference test-bed.

6 ACKNOWLEDGMENTS

This work was funded by the European Union's Horizon 2020 Research and Innovation Programme through DIAS project (*https://dias-project.com/*) under Grant Agreement No. 814951. This document reflects only the author's view and the Agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Will Arthur, David Challener, and Kenneth Goldman. 2015. *Hierarchies.* Apress, Berkeley, CA, 105–118. https://doi.org/10.1007/978-1-4302-6584-9_9
[2] Will Arthur, David Challener, and Kenneth Goldman. 2015. *Platform Configuration Registers.* Apress, Berkeley, CA, 151–161. https://doi.org/10.1007/978-1-4302-6584-9_12
[3] AUTOSAR. 2017. Specification of Secure Onboard Communication AUTOSAR CP Release 4.3.1. *AUTOSAR* (2017).
[4] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422–426. https://doi.org/10.1145/362686.362692
[5] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation. Available at http://www.w3.org/TR/REC-xml/.
[6] Linux Foundation. 2016. Automotive Grade Linux. https://www.automotivelinux.org/ Last access: June 10th, 2021.
[7] Linux Foundation. 2020. Yocto Project. https://www.yoctoproject.org/ Last access: June 10th, 2021.
[8] Trusted Computing Group. 2020. *TCG TPM 2.0 Automotive Thin Profile For TPM Family 2.0; Level 0.* https://trustedcomputinggroup.org/resource/tcg-tpm-2-0-library-profile-for-automotive-thin/ Last access: June 10th, 2021.
[9] Trusted Computing Group. 2020. Trusted Computing Group. https://trustedcomputinggroup.org/ Last access: June 10th, 2021.
[10] Bogdan Groza and Pal-Stefan Murvay. 2019. Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks. *IEEE Transactions on Information Forensics and Security* 14, 4 (2019), 1037–1051. https://doi.org/10.1109/TIFS.2018.2869351
[11] Tobias Hoppe, S. Kiltz, and J. Dittmann. 2009. Applying intrusion detection to automotive IT-early insights and remaining challenges. *Journal of Information Assurance and Security (JIAS)* 4 (01 2009), 226–235.
[12] Seong Hoon Jeong Hyunsung Lee and Huy Kang Kim. 2018. CAN Dataset for intrusion detection (OTIDS). http://ocslab.hksecurity.net/Dataset/CAN-intrusion-dataset Last access: June 10th, 2021.
[13] ISO. 2003. ISO 11898-1:2003 - Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling. *International Organization for Standardization* (2003).
[14] Anu Jawahar, Anu Gupta, Asadullah Ansari, Rabindra Paikaray, Sabarinathan Ravi, and Muthukumar Alagesan. 2021. Application Controlled Secure Dynamic Firewall for Automotive Digital Cockpit. In *SAE WCX Digital Summit.* SAE International. https://doi.org/10.4271/2021-01-0140
[15] Seonghoon Jeong, Boosun Jeon, Boheung Chung, and Huy Kang Kim. 2021. Convolutional neural network-based intrusion detection system for AVTP streams in automotive Ethernet-based networks. *Vehicular Communications* 29 (2021), 100338. https://doi.org/10.1016/j.vehcom.2021.100338
[16] Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha. 2020. INDRA: Intrusion Detection Using Recurrent Autoencoders in Automotive Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3698–3710. https://doi.org/10.1109/TCAD.2020.3012749
[17] H. Lee, S. H. Jeong, and H. K. Kim. 2017. OTIDS: A Novel Intrusion Detection System for In-vehicle Network by Using Remote Frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, Vol. 00. 57–5709. https://doi.org/10.1109/PST.2017.00017
[18] Feng Luo and Shuo Hou. 2019. Security Mechanisms Design of Automotive Gateway Firewall. In *WCX SAE World Congress Experience.* SAE International. https://doi.org/10.4271/2019-01-0481
[19] Pal-Stefan Murvay and B. Groza. 2020. TIDAL-CAN: Differential Timing Based Intrusion Detection and Localization for Controller Area Network. *IEEE Access* 8 (2020), 68895–68912.
[20] O. Hartkopp, et al. 2020. SocketCAN: Controller Area Network Protocol Family. https://github.com/linux-can/can-utils Last access: June 10th, 2021.
[21] Robert Bosch Gmbh. 2012. CAN with flexible data-rate. *Vector CANtech, Inc., MI, USA, Specification Version 1.0* (2012).
[22] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. 2018. GIDS: GAN based Intrusion Detection System for In-Vehicle Network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST).* 1–6. https://doi.org/10.1109/PST.2018.8514157
[23] Linux TPM2 TSS2 Software. 2020. TPM (Trusted Platform Module) 2.0 tools. https://github.com/tpm2-software/tpm2-tools Last access: June 10th, 2021.
[24] TPM2 software community. 2020. Linux TPM2 & TSS2 Software. https://github.com/tpm2-software Last access: June 10th, 2021.
[25] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. 2015. Frequency-based anomaly detection for the automotive CAN bus. 45–49. https://doi.org/10.1109/WCICSS.2015.7420322
[26] Trusted Computing Group. 2019. Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59 – November 2019.
[27] M. Wolf, A. Weimerskirch, and C. Paar. 2004. Security in Automotive Bus Systems.