A Testing Framework Architecture Concept for Automotive Intrusion Detection Systems

Christopher Corbett[1] Tobias Basic[2] Thomas Lukaseder[3] Frank Kargl[3]

Abstract: Vehicles are the target of a rising number of hacking attacks. The integration of in-vehicle intrusion detection systems is a common approach to increase the overall system security. However, testing and evaluating these systems is difficult due to the lack of tools to generate realistic benign and malicious workloads as well as sharing these workloads with other researchers. Currently, testing tools are predominantly intended for Network Intrusion Detection System (NIDS) in company or industrial networks where their usefulness became apparent. Yet, in the automotive domain, development of testing tools is still in the early stages. Existing non-commercial automotive tools only focus on one specific bus technology each. However, in-vehicle communication exceeds bus technology boundaries and a testing tool must cover multiple technologies. We propose a framework architecture concept for in-vehicle NIDS testing and evaluation to enable the creation of realistic network traffic and attacks in consideration of automotive specific challenges. Our concept provides the opportunity to share data without additional anonymization effort therefore improving cooperation and reproducibility of testing results.

Keywords: automotive, network, IDS, evaluation, security, framework

1 Introduction

Automotive networks are essential for both driver assistance and future trends such as autonomous driving. With increasing complexity and rising numbers of network devices, the possible impact of malicious manipulation and malfunction also increases. Additional network bandwidth is mandatory to cover new functional requirements and cannot be met with traditional bus systems such as the Controller Area Network (CAN). Automotive Ethernet is designed to tackle these problems and — in addition to the necessary bandwidth — provides greater flexibility with regard to higher layer protocols.

A rising number of attacks on vehicles (e.g. [MV15],[RM15]) emphasizes the need for more security precautions and extended protection mechanisms in upcoming automobiles. Embedding Intrusion Detection Systems (IDS) into in-vehicle networks is an applicable approach to enhance overall vehicle security complementary to encryption and authentication mechanisms. Evaluating and testing IDS is a difficult task. Realistic datasets that are compliant to the automotive domain specific requirements are necessary for testing but hard to obtain. Furthermore, there is no standardized methodology for the evaluations which in turn leads to a lack of comparability of the results.

[1] Audi AG, 85045 Ingolstadt, christopher.corbett@audi.de
[2] TU Darmstadt, Department of Computer Science, fi59eged@rbg.informatik.tu-darmstadt.de
[3] Ulm University, Institute of Distributed Systems, {firstname}.{lastname}@uni-ulm.de

In this paper, we show that most commonly available tools do not meet the requirements for automotive NIDS evaluations and we introduce our architecture concept to cover those needs. With our approach, we are not only able to test network intrusion detection systems or generate custom network traffic, but — through separation of the evaluation scenario definition from specific network parameters — scenarios can be shared among interest groups without the necessity to anonymize traces or the risk of exposing real network topologies and information.

The remainder of this paper is structured as follows: In Section 2, we provide an overview of automotive domain specific protocols and topologies. We present related work in Section 3 and then give an overview of common in-vehicle network attack scenarios in Section 4. Necessary intrusion detection evaluation steps are described in section 5 and the derived requirements can be found in Section 6. Our framework architecture concept is described in Section 7; followed by our conclusion in Section 8.

2 Background

The automotive industry introduced a variety of bus technologies over the years. Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), Controller Area Network (CAN) and Flexray are well established in in-vehicle networks. With new feature sets, bandwidth requirements increased rapidly and therefore new technologies such as the enhanced CAN — Controller Area Network Flexible Data Rate (CAN-FD) — and the Ethernet (IEEE 802.3) protocol gained attention. As in-vehicle networks are very heterogeneous, data exchange between Electronic Control Units (ECUs) exceeds bus technology boundaries and translations (e.g. transporting CAN frames via Ethernet) are commonly used. To start off with a decent framework feature set to generate testing workload we examined attributes, parameters and characteristics of automotive Ethernet, CAN and CAN-FD.

2.1 Automotive Protocols

With each bus technology and feature set, new automotive protocols were introduced or enhanced over time, from which some are used in industrial networks (e.g. CAN protocols) and for others it is thinkable to be used in company networks (e.g. remote vehicle diagnostics). Figure 1 shows the classification of CAN and Ethernet protocols in the layers of the Open Systems Interconnection Model (OSI).

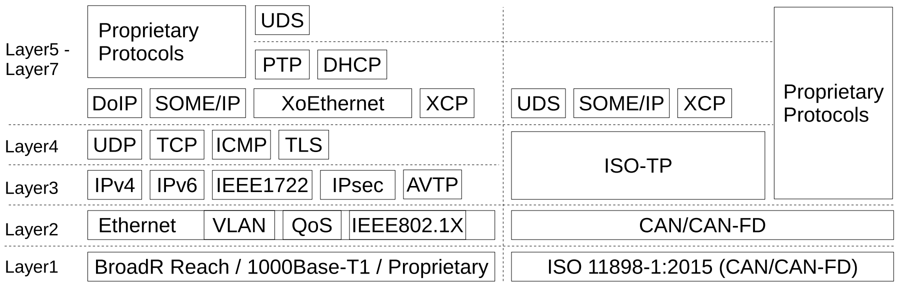

Figure 1: Protocol overview

Zimmermann and Schmidgall [ZS14] give an in depth overview of most of the standardized automotive protocols. In Table 1 we provide a brief summary of protocols we analyzed to derive requirements for the testing framework architecture.

Protocol	**Standard**	**Description**
Unified Diagnostic Services (UDS)	ISO 14229	An application client-server protocol to remotely call diagnose procedures in ECUs and transport information back to the requester.
ISO Transport Protocol (ISO-TP)	ISO 15765-2	Protocol to transport payloads larger than the maximum payload size on CAN/CAN-FD.
Diagnose over Internet Protocol (DoIP)	ISO 13400	An automotive transport layer protocol to transport UDS messages between the client and the server using port 13400 (UDP/TCP).
Universal Measurement and Calibration Protocol (XCP)	ASAM MCD-1 XCP V1.3.0 [AS15]	A bus-independent master-slave communication protocol to exchange information between ECUs and a calibration software on a external device (e.g. PC, Vehicle Tester, Laptop).
Scalable Service-Oriented Middleware over Internet Protocol (SOME/IP)	AUTOSAR [AU15][AU14]	A service oriented protocol that supports remote procedures calls, data serialization and a service discovery and publish/subscribe mechanism.
XoverEthernet	AUTOSAR [AU15][AU14] / Proprietary	Frames of one bus system are transported as payload via Ethernet (e.g. CAN over Ethernet).
Proprietary	Proprietary	Besides standardized and established protocols, car manufacturers make use of own or third party proprietary protocols.

Table 1: List of analyzed protocols

2.2 Communication patterns

Communication patterns in in-vehicle networks are based on design rules which rest upon applied bus technologies as well as application and protocol requirements. Therefore, each Original Equipment Manufacturer (OEM) network acts differently and a general description can not be given. However, several suppliers provide development tools to cover requirements across OEMs and provide a decent overview of CAN bus communication patterns in their documentation (e.g. Vector Informatik GmbH [Ve15]). These patterns are extended by common Ethernet behavior (e.g. fire and forget) as it is not an exclusive replacement for legacy bus systems in the foreseeable future. As a result we derived several factors from CAN and Ethernet communications that result in different patterns. These are:

- time triggering
- events
- fire and forget
- request and response
- state premises (stateful)
- no state premises (stateless)

3 Related Work

3.1 Automotive IDS

The challenges of designing tests of intrusion detection systems are widely understood. Milenkoski et al. [Mi15] provide a very extensive survey of common practices for tests of different kinds of intrusion detection systems. They discuss the three main components of IDS testing: workloads, metrics and measurement methodology. For each of the components, the authors provide a common terminology. For our work, we adopt this terminology and propose an architecture for workload generation in automotive networks.

There has been previous work that deals with designing intrusion detection systems for in-vehicle networks that employ CAN as main bus technology [Ha14][SKK16][KK16][CS16]. However, to the best of our knowledge, there is no work that deals with intrusion detection in modern in-vehicle architectures that also employ Ethernet as a backbone technology for the in-vehicle network. Nonetheless, Herold et al. [He16] have explored anomaly detection for the Scalable Service Oriented Middleware over Internet Protocol (SOME/IP) using complex event processing. To test their anomaly detection regarding performance, they implemented a SOME/IP packet generator, featuring four kinds of simple attacks: 1) malformed packets 2) protocol violations 3) system-specific violations and 4) timing issues. However, they only investigated anomaly detection for SOME/IP, which is an application layer protocol that is employed in upcoming Ethernet-based vehicle networks. The automotive protocol stack for Ethernet-based networks is much more diverse. In our work, we look at all the protocols and designed an architecture that is able to test an NIDS in modern Ethernet-based in-vehicle networks.

Moreover, there has been a lot of works that deal with the generation of workloads for testing IDS [Mi15]. Antonatos et al. [AAM04] have provided an extensible framework for the generation of realistic workloads in their work. Their generator is, however limited to application layer traffic, and focuses on the generation of payloads for these protocols. We do aim for a similar approach for our architecture, but want to provide more flexibility regarding developing and describing different scenarios. Our architecture also supports the generation of traffic down to layer 2. Furthermore, we extend our architecture to fulfill automotive requirements (cf. Section 6).

3.2 State of the Art Tools

There is a variety of tools which can be used to perform specific attacks or scans, such as Nmap, Nessus and Metasploit. However, most of them are limited to very specific use cases such as port scanning in the case of Nmap. While Metasploit can be used to generate traffic, its main purpose is to generate pure malicious traffic with the help of its integrated exploit database.

Manual testing is a very time consuming task. It involves manually generating traffic with a collection of tools, capture the traffic using e.g. Wireshark, and then modifying and replaying the traffic. Furthermore, none of these tools have been adapted for the automotive domain. The manually generated traffic would have to be adjusted to represent realistic traffic in an in-vehicle network for effective testing of automotive NIDS. This adjustment process can also take a substantial amount of time as the traffic model has to be modified for every model and vehicle setup.

Packet generation tools are meant as a solution to the manual generation problem. They facilitate the automated generation of packets, which can be used to test intrusion detection systems. However, they do not provide the functionality required to reliably test automotive NIDS. Most tools do not fulfill our requirements as they do not support traffic generation, modification, and forwarding across multiple interfaces, which e.g. allows man-in-the-middle attacks on layer 2.

We have explored the feature sets of 10 existing packet generation tools and have found that none of them provide support for automotive protocols such as SOME/IP or UDS and DoIP. None of them provide the ability to prioritize traffic when capturing and modifying the response, or when capturing and sending a response to a captured packet. Additionally, a large chunk of the tools did not provide the flexibility to write scripts in order to automate certain tasks and re-use them for further tasks.

Due to the cyclic nature of a lot of messages sent in an in-vehicle network, we also require a packet generation tool to send packets at steady intervals with an insignificant amount of jitter. None of the tools provided a similar feature except Ostinato and packETH. Ostinato only allowed setting an interval of a packet per X seconds; packETH, however, offered millisecond and even nanosecond resolution for interval generation, but lacks other features, such as multiple interface support. Moreover, as the Ethernet layer is more important in in-vehicle networks compared to company networks, we need full flexibility when crafting and modifying Ethernet packets. Only some of the packet generation tools allowed receiving packets on layer 2. A summary of our findings can be found in Table 2.

	PCAP Replay	Multiple Interface Handling	Scripting	Layer 2 Support	IPv6 Support	Automotive Protocols	Generate Mixed Traffic	Priority Handling	Capture Traffic	Packet modification	Packet sending interval	Automation
Tomahawk	✓						✓					✓
Bit-Twist	✓			✓			✓		✓			✓
Hping2							✓					✓
Hping3			✓				✓		✓			✓
Nemesis				✓	✓		✓					✓
Ostinato	✓	(✓)	✓	✓	✓		✓		✓		(✓)	
packETH	✓			✓	✓		✓				✓	
Yersinia				✓			(✓)		✓			
netsniff-ng	✓			✓	✓		✓		✓			
pktgen	✓	(✓)	✓	✓	✓		✓		✓			✓

Table 2: An overview of available packet generation tools and their capabilities.

4 In-Vehicle Network Attack Scenarios

There are different types of attack scenarios for in-vehicle networks. Figure 2 shows how we set up an example network topology with a centralized component (e.g. a gateway or a routing unit) and four network participants. The following scenarios are feasible ways to inject malicious traffic or to modify existing network traffic in vehicular networks.

1. **Man in the middle**: In this scenario a malicious network participant (E) is positioned between the devices d and r to eavesdrop, manipulate or forge network traffic.
2. **Compromised device**: In this scenario, device (a) gets compromised with a piece of malicious software (F) to forge authentic communication or modify communication behavior.
3. **Attached device**: A new malicious network participant (G) is attached to the network and forges network traffic on an existing connection between network participants b and r.

4. **Device replacement**: An existing device (c) gets replaced by a malicious device (H).
5. **Compromised central network device**: Similar to scenario 2) a malicious piece of software is placed into a central network device (I) to modify network behavior or traffic.

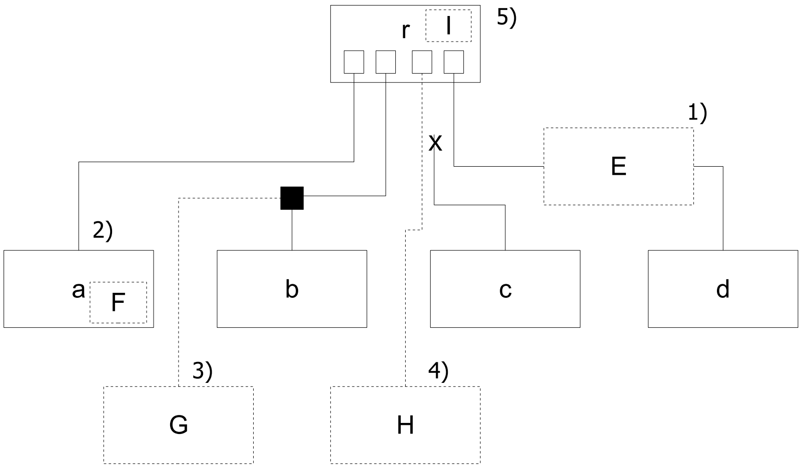

Figure 2: Overview of in-vehicle network attack scenarios.

5 IDS Testing Parameters and Metrics

This section gives a short introduction to different types of intrusion detection systems, general testing metrics and parameters as well as automotive domain specific parameters.

5.1 Categorization

Fallstrand et al. [FL15] state that intrusion detection and prevention systems are commonly categorized based on four properties:

- Scope — What kind of entity or entities does the system protect?
- Location and Distribution — Where and how are the system components deployed?
- Detection method — How does the system identify intrusions?
- Post-detection — How does the system respond to detected intrusions?

In this paper, we focus both on centralized and distributed in-vehicle network intrusion detection systems with a focus on misuse or anomaly based detection algorithms.

5.2 Metrics

IDS testing metrics can be categorized into performance- and security-related metrics which can again be grouped by commonly used basic attributes such as false-negative, true-positive, false-positive, true-negative, positive predictive value and negative predictive value, but also in composite values such as expected cost and intrusion detection capability [MC14]. In addition to the metrics used in prior research, we add detection latency to the list of metrics. Especially for automotive intrusion detection systems, timeliness of the attack detection can be crucial for the applicability in the field.

5.3 Requirements for NIDS Testing

The design of NIDS depends on numerous factors such as the network topology, the protocols, or the detection mechanisms used. These factors also need to be considered when designing a testing architecture. The testing architecture needs to be able to be agnostic to differences in NIDS architectures — i.e. it should not inherently favor one architecture over the other — while still acknowledging specific strengths and weaknesses of NIDS architecture types. For instance, some machine learning based NIDS need a certain time to build their specific neuronal network. A data set with a certain minimal size with labels for both benign traffic and attacks needs to be available both for training and testing the NIDS.

Milenkoski et al. [MC14] showed that workloads are mandatory for intrusion detection system testing and can be divided into three different types: purely benign, purely malicious and mixed workload sets. The acquirement or generation of workloads are either achieved by executables (e.g. manual generation, exploit databases, vulnerability and attack injection or workload drivers) or traces (e.g. through acquisition or generation). Usable training data is scarce and — as these data sets are recorded from real networks — they also show the characteristics of the original network without the possibility to adjust to the network configuration of the NIDS application site. Therefore, a dynamic testing system that can generate traffic on the fly and can generate an unlimited amount of data is advantageous.

Test runs have to be repeatable to ensure scientifically valid results, while the test environment also needs to offer the dynamic of real networks in the form of random changes in the network behavior. For this to work efficiently, automation must be possible.

5.4 Automotive Specific Challenges

An in-vehicle network combines different bus technologies which cannot be strictly separated and influence each other. Therefore, a comprehensive NIDS needs to consist of a combination of bus specific NIDS which adds complexity to the NIDS itself and to the tests of such a system. Currently, research focuses more on NIDS and Network Intrusion

Detection and Prevention Systems (NIDPS) for CAN networks while Ethernet is starting to gain some attention.

The unavailability of automotive specific attacks is another factor that complicates testing of an automotive NIDS. In comparison to attacks on company networks, attacks on vehicles are very vehicle and OEM specific. There is no comprehensive database of known attacks available that could be shared among car manufacturers.

To develop, test, and evaluate automotive NIDS — independent of the chosen detection method — valid and realistic data sets of network traffic must be available. Usually, traces of existing traffic or manually generate traffic are used. However, if no real traffic trace is available, the generated traffic cannot be proven to be realistic.

6 Framework Requirements

Considering attack scenarios, traffic generation, and IDS metrics, we derived a set of essential requirements. The architecture has to meet these requirements to facilitate the generation of realistic automotive workloads.

Protocol support The architecture must provide the ability to parse, manipulate and forge packets sent using protocols used in the automotive protocol stack described in Section 2. This facilitates the communication with other members in the automotive network as a legit as well as a malicious entity, depending on the scenario.

Frame manipulation A lot of network management, such as Virtual Local Area Network (VLAN) segmentation, happens on the Ethernet layer. Therefore, in addition to the previous requirement, the given tool must be able to manipulate packets on layer 2, including the VLAN tag.

Response time In order to deal with real-time applications in automotive networks, the tool has to be able to respond to a packet within the defined deadline for the corresponding vehicle domain. This ranges from 10ms for safety-critical sytems up to 150ms for audio/video streams in the infotainment domain [LP13].

Bandwidth The tool must provide a bandwidth of 100 Mbit/s (better yet 1 Gbit/s). Current automotive applications employ 100 MBit Ethernet, however, in future applications Gbit Ethernet is going to be employed in vehicles.

Time interval support ECUs are very sensitive regarding the interval at which they expect a certain signal or packet to arrive at their interface. Hence, the tool must be able to send and forward packets and frames at steady intervals while keeping the jitter as low as possible.

Fuzzing support Due to the long lifetime of automobiles, they are continuously exposed to new kinds of attacks. The tool must provide a fuzzing functionality to be able to simulate previously unknown scenarios and attacks.

State handling State handling is the ability to establish a certain state in a protocol. For example, messages A,B,C are sent according to specification and then deviate from the specification or modify messages. The state machine is also required to perform e.g. Transmission Control Protocol (TCP) Session Hijacking attacks.

Frame and packet scheduling The prioritization of packets and frames is of higher importance in automotive networks compared to company networks. The architecture therefore both has to be able to deal with incoming packets of different priority, and has to be capable to prioritize their processing accordingly.

Multiple interface support Several interfaces must be usable in parallel. Some devices communicate on several buses such as Ethernet and CAN. The architecture and tool must be able to replicate this behavior.

Scenario and parameter separation The separation of evaluation scenarios and data or value sets is important to enable the exchange and verification of results with and by third parties.

7 Testing Tool Architecture Concept

We propose an architecture for a testing tool that facilitates the proper evaluation of automotive IDS by satisfying all the requirements which we have defined in the previous section. Fulfilling the requirements ensures the generation of realistic automotive workloads. It also overcomes various shortcomings of existing tools. While our goal was to design a tool for the evaluation of automotive IDS, it can also be used to perform functional testing as well as security testing of an (automotive) network.

7.1 Architecture

The tool architecture is divided into three layers: user, developer, and system. This makes the tool's underlying framework easy to extend for those, who have the technical knowledge and easy to use for those, who just want to set up a test quickly using the pre-defined scenarios.

From a user's perspective, either pre-defined testing and attack scenarios or a self-designed scenario description can be used and configured. The configuration file contains various parameters exposed by the scenario, such as interfaces, protocols, layers, and packet values. Furthermore, it includes timing as well as priority information, if needed. Additional parameters can be exposed through the developer layer.

The tool's developer layer offers an extensible framework with which testing and attack scenarios can be developed. It provides three basic modules: **function blocks**, **core** and **network abstraction**. A developer can implement a function block (e.g. SYN scan attack) with custom logic and a defined parameter set. These function blocks can then be used by users to describe scenarios, which resemble malicious, benign, or mixed traffic.

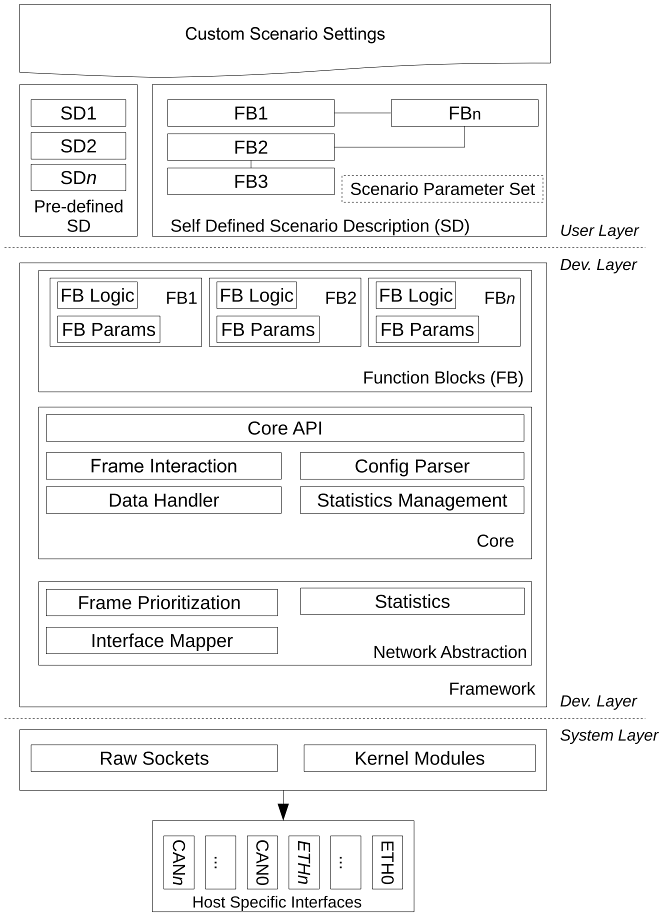

Figure 3: An architecture for an automotive testing tool.

Some simple example scenarios can be:

1. **Pure benign traffic:** We simulate an ECU that sends out a signal at a pre-defined interval.
2. **Pure malicious traffic:** We send out a TCP SYN scan to determine open ports on an ECU.
3. **Mixed traffic:** We combine function blocks 1) and 2) to disguise our attack in regular traffic, and provide a more realistic and challenging scenario.

The core of the framework provides essential functionality such as frame interaction, a configuration parser, a statistics manager, and a data handler. An Application Programming Interface (API) can be used by function blocks to interact with the core's submodules. The data handler submodule provides means to interact with a list of provided payloads for a function block. Moreover, the frame interaction submodule enables parsing, crafting, and manipulation of frames and packets.

The framework's network abstraction module provides functionality to interface with different kinds of networking sockets, such as raw sockets or custom implementations (e.g. ring-buffer based implementations such as PF_RING[PF]) of the system layer. Furthermore, it takes care of the prioritization of frames to meet defined timing constraints. It also maps the physical interfaces of the executing host to logical interfaces defined in the scenario description.

Figure 3 shows the architecture of our proposed testing and workload generation framework for the evaluation of automotive NIDS.

7.2 Discussion

Existing tools shown in 3.2 cover only some features required to create workloads for an automotive NIDS testing. Either several tools must be combined or they lack necessary protocol support. Our architecture concept remedies these shortcomings and provides the ability to generate, modify, and analyze automotive-compliant traffic. Through scenario descriptions, it is possible to generate both benign and malicious traffic, and easily apply these scenarios to different vehicle setups. As our approach supports several interfaces, it is possible to implement more complex scenarios, such as man-in-the-middle attacks on layer 2. Additionally, setups can be shared among other research groups to verify results or to be used in their own research.

8 Conclusion

Evaluating and testing network intrusion detection systems is essential for improving NIDS. Workloads are necessary for testing the NIDS' crucial detection capabilities. For automotive Ethernet, such workloads are not currently available. For our malicious workload model, we consider five attack scenarios as presented in Section 4. We then derived several requirements that are necessary to be able to generate realistic traffic in Ethernet-based in-vehicle networks. In particular, the most important requirements are the ability to handle multiple interfaces and the ability to prioritize the handling of different streams of traffic.

We have analyzed several packet generation tools. We have found that none of the tools we analyzed fulfilled our requirements. The most prominent finding from our analysis showed that none of the tools provided support for multiple interfaces or traffic prioritization. With an extensive amount of features missing in all analyzed tools, we have come to the conclusion, that extending existing tools is not a viable option and that a new architecture has to be designed with the specific challenges of in-vehicular networks in mind.

We have proposed a novel architecture concept for a tool that remedies these shortcomings. Furthermore, we made sure that our proposed architecture is extensible. New scenarios and attacks can be added easily through scenario descriptions. The provided functionality can be extended through additional function blocks, or by extending the framework. Our architecture fulfills the set requirements described in Section 6 by providing the necessary modules in the framework.

A proof of concept implementation of the framework has to be provided to determine whether our architecture proves usable in a realistic scenario. Said implementation then has to be evaluated with regard to our identified requirements by implementing the attack scenarios as described in Section 4. Furthermore, using said implementation to evaluate a given automotive NIDS requires implementation of further scenarios to build a realistic workload model. All these steps are left for future work.

References

[AAM04] Antonatos, Spyros; Anagnostakis, Kostas G; Markatos, Evangelos P: Generating realistic workloads for network intrusion detection systems. In: ACM SIGSOFT Software Engineering Notes. volume 29. ACM, pp. 207–215, 2004.

[AS15] ASAM MCD-1 XCP V1.3.0: Universial Measurement and Calibration Protocol (XCP), 2015.

[AU14] AUTOSAR 4.2 Rev. 1: Example for a Serialization Protocol (SOME/IP), 2014.

[AU15] AUTOSAR 4.2 Rev. 2: Specification of Service Discovery, 2015.

[CS16] Cho, Kyong-Tak; Shin, Kang G: Fingerprinting electronic control units for vehicle intrusion detection. In: 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, pp. 911–927, 2016.

[FL15] Fallstrand, Daniel; Lindström, Viktor: Applicability analysis of intrusion detection and prevention in automotive systems. Master's thesis, Department of Computer Science and Engineering; Chalmers University of Technology;Göteborg Sweden, 2015.

[Ha14] Han, Song; Xie, Miao; Chen, Hsiao-Hwa; Ling, Yun: Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges. In: IEEE SYSTEMS JOURNAL. volume 8. IEEE, 2014.

[He16] Herold, Nadine; Posselt, Stephan-A; Hanka, Oliver; Carle, Georg: Anomaly detection for SOME/IP using complex event processing. In: Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP. IEEE, pp. 1221–1226, 2016.

[KK16] Kang, Min-Joo; Kang, Je-Won: Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. PloS one, 11(6):e0155781, 2016.

[LP13] Lee, Youngwoo; Park, KyoungSoo: Meeting the real-time constraints with standard Ethernet in an in-vehicle network. In: Intelligent Vehicles Symposium (IV), 2013 IEEE. IEEE, pp. 1313–1318, 2013.

[MC14] Mitchell, Robert; Chen, Ing-Ray: A Survey of Intrusion Detection Techniques for Cyber-Physical Systems. volume 46. ACM Computing Surveys, 2014.

[Mi15] Milenkoski, Aleksandar; Vieira, Marco; Kounev, Samuel; Avritzer, Alberto; Payne, Bryan D: Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices. ACM Computing Surveys (CSUR), 48(1):12, 2015.

[MV15] Miller, Charlie; Valasek, Chris: Remote Exploitation of an Unaltered Passenger Vehicle. Black Hat, 2015.

[PF] PF_RING: `http://www.ntop.org/products/packet-capture/pf_ring/`. Last accessed 2016-12-12.

[RM15] Rogers, Marc; Mahaffey, Kevin: How to Hack a Tesla Model S. DEF CON 23, 2015.

[SKK16] Song, Hyun Min; Kim, Ha Rang; Kim, Huy Kang: Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In: 2016 International Conference on Information Networking (ICOIN). IEEE, pp. 63–68, 2016.

[Ve15] Vector Informatik: Vector: CANoe Interaction Layer. 2015. `http://vector.com/portal/medien/cmc/application_notes/SN-IND-1-011_InteractionLayer.pdf`. Last accessed 2016-12-12.

[ZS14] Zimmermann, Werner; Schmidgall, Ralf: Bussysteme in der Fahrzeugtechnik. Springer Vieweg, 2014.