A Formal Model to Facilitate Security Testing in Modern Automotive Systems

Eduardo dos Santos Andrew Simpson
Cyber Security Centre for Doctoral Training
Department of Computer Science
University of Oxford
Oxford, OX1 3QD, United Kingdom
eduardo.dossantos@cs.ox.ac.uk andrew.simpson@cs.ox.ac.uk

Dominik Schoop
Esslingen University of Applied Sciences
73732 Esslingen am Neckar, Germany
Dominik.Schoop@hs-esslingen.de

Ensuring a car's internal systems are free from security vulnerabilities is of utmost importance, especially due to the relationship between security and other properties, such as safety and reliability. We provide the starting point for a model-based framework designed to support the security testing of modern cars. We use Communicating Sequential Processes (CSP) to create architectural models of the vehicle bus systems, as well as an initial set of attacks against these systems. While this contribution represents initial steps, we are mindful of the ultimate objective of generating test code to exercise the security of vehicle bus systems. We present the way forward from the models created and consider their potential integration with commercial engineering tools.

1 Introduction

The security of modern cars has attracted a great deal of media attention in recent years (e.g [22]). As the transportation paradigm shifts towards *autonomy*, many concerns have been raised about the in-car computer systems' ability to safeguard the behaviour of cars — and, ultimately, the safety of drivers, passengers and pedestrians.

Several methods are available to test the security of systems against attacks, one of which (and possibly the most well-known) is *penetration testing*. In it, highly-skilled security professionals mimic the actions of an attacker, whose objective is the achievement of unauthorised goals (e.g. data exfiltration or command spoofing) on the target system. Although penetration testing is widely deployed by companies seeking to obtain an understanding of the real security of their systems and networks, it is not a perfect solution. For example, it carries a high cost, it requires a high degree of ability and knowledge, and it is primarily applicable to already deployed systems — it is a *reactive*, rather than a *preventive* security testing approach.

While we recognise penetration testing's role, we believe that there is the potential for other security testing approaches to play a complementary role, with one candidate being the testing of a system's security from models — in short, *model-based security testing*. Thus, the focus of this paper, which builds upon the initial work described in [3], is on answering *how might a model-based testing (MBT) framework support the automatic security testing of a modern car's subsystems?* This work falls into the category of formal and model-driven approaches to engineering safety-critical and security-critical systems.

R. Laleau, D. Méry, S. Nakajima, E. Troubitsyna (Eds): Joint Workshop on Handling IMPlicit and EXplicit knowledge in formal system development and Formal & Model-Driven Techniques for Developing Trustworthy Systems.
EPTCS 271, 2018, pp. 95–104, doi:10.4204/EPTCS.271.7

Our motivations for answering this question are manifold. First, it is consistent with the automotive industry's approach to product development, which is characterised by an emphasis on models, with the actual implementation often delegated to third parties. Second, the distributive nature and complexity of the systems involved require a testing approach reusable across all the different systems. Finally, initiatives like OpenCar2X [7] indicate a momentum towards the creation of in-car apps by various developers (in a similar way to the smart phone eco-system) — these apps will inevitably have to be tested for security.

The remainder of this paper is organised as follows. Section 2 details the motivations behind our research line and provides necessary background information on related work. Section 3 introduces CSP, the formalism backing the work. In the sequence, Section 4 defines the threat model from which security attacks to the system will be derived. Next, Section 5 presents the core of the work: the definition of the in-vehicle architecture and attacks in a formal fashion. Finally, Section 6 concludes the paper, discussing our results in context and paving the way ahead in the research project.

2 Motivation and Background

Cars are very complex entities. This complexity begins with architecture design, which is formed by a large number of inter-connected embedded systems — also known as *electronic control units* (ECUs). An average modern passenger vehicle typically has 40 or more ECUs [9]. Depending on their application, ECUs can have strict networking requirements. As a result, various kinds of networks are available. For instance, a low-latency network (e.g. the Controller Area Network) is more suitable for real-time applications, whereas a high-bandwidth network (e.g. MOST) is more suitable for in-car entertainment applications.

Figure 1 illustrates a simplified version of an automotive architecture, which is based on the topology of bus systems [20]. Realisations of this architecture can vary, with respect to the number of components, as well as, the form of connection between components. Various bus systems are connected to different networks. Gateways manage communication between different networks. Each network type provides its own set of requirements (e.g. latency, transmission speed, etc.), as well as has its own frame structure. The Controller Area Network (CAN) is responsible for safety and non-safety functions of the vehicle. It should be noted that FlexRay intends to replace CAN as standard for safety communication [25].

Sensors and actuators can either be connected to a bus system directly or attached to their own Local Interconnect Network (LIN). Figure 1 shows a centralised architecture with one gateway. Decentralised architectures with two gateways are also possible.

With regards to the security of cars, early academic studies have shown that cars can be exploited by both physical [5, 6] and wireless [2, 4] channels. Physical attacks typically take the form of packet injections directly on the internal network (e.g. the aforementioned CAN) via the OBD-II port, whereas wireless attacks typically exploit vulnerabilities in the implementation of mostly proprietary authentication protocols. Researchers have been able to achieve a range of unauthorised goals, including installation of covert surveillance capabilities within the vehicle, remote data exfiltration and remote control. Although all vehicles share the same basic principles and communication standards, it seems apparent that implementation and architectural differences play a vital role to determine a car's security level [9].

To date, most studies have focussed on finding network issues. Less attention, however, has been given to security issues that are brought about by data [14]. Vehicles nowadays consume a vast amount of data from a variety of sources (e.g. data collected by internal and external sensors, and from online databases). Further, within the vehicle, data needs to be captured, processed and stored [14]. All of these

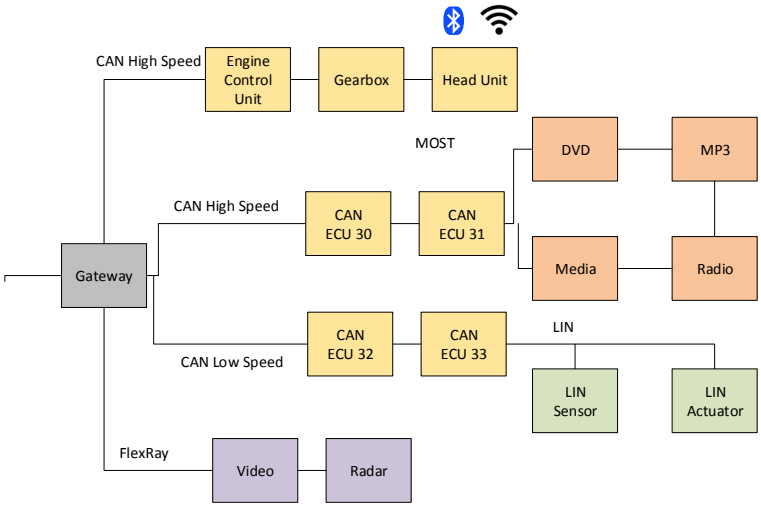

Figure 1: Automotive bus systems in a generic architecture

steps must happen securely to provide the vehicle with higher assurance levels. We consider this data lifecycle in this paper.

With regards to formal methods, there has been little application of them in the automotive domain broadly, and even less for security specifically. Mundhenk *et al.* [12] use continuous time Markov models to facilitate decision making between different automotive architectures with regard to required security properties. Siegl *et al.* [19] formalise automotive systems requirements by test models. However, their approach is concerned primarily with safety, rather than security requirements.

Modern cars inherit many of the more generic verification challenges associated with cyber-physical systems (CPSs). Zheng and Julien [26] outline some of the challenges facing verification of cyber-physical systems, noting that CPSs' real-time requirements and traditional large system size make them difficult to be verified by state-of-the-art formal verification tools.

Seshia *et al.* [17] discuss the need for formal methods for semi-autonomous driving. By requiring human driver intervention at certain moments, semi-autonomous cars introduce many conditional specification and verification problems. For instance, it is not easy to determine which situations require human intervention or whether the driver will be able to resume control of the vehicle on time.

It should be noted that, in the following, we make no distinction between *car* and *vehicle* — their use is synonymous. For simplicity, whenever we refer to *automotive system*, we also refer to the connecting networks associated with the given system.

3 Communication Sequential Processes (CSP)

Our formal models are presented in the language of Communication Sequential Processes (CSP), a process-algebraic formalism that supports the modelling and analysis of concurrent systems. In addition we utilise the refinement checker, FDR (Failures Divergences Refinement) [13].

The choice of CSP can be justified by the systems of systems (SoS) approach to building modern vehicles, which can be modelled in CSP via compositions of processes. In this context, each individual sub-system is modelled as an individual CSP process. Processes, in turn, can exchange messages between each other in a similar way to automotive networks. There are precedents for the use of CSP in this

respect: it has also been in the verification of properties of car protocols (see, for example, [16]).

In the following, we provide a necessarily brief introduction to the language of CSP.

3.1 Syntax

The *alphabet* of a CSP process is the set of events that it is willing to communicate. An *event* requires the agreement of both the environment and the communicating process, occurs instantaneously, and is atomic. The set of all possible events within the context of a particular specification is denoted Σ.

The simplest possible process is the one that offers to participate in no events: *STOP*. Another simple process, *SKIP*, represents successful termination (via the internal event, $\checkmark$). The *prefixing* operator, $\rightarrow$, allows us to prefix an event $e \in \Sigma$ to any process: the process $e \rightarrow STOP$ will refuse to communicate anything other than e, after which it will offer no further events. Recursion allows us to capture infinite behaviour using a finite description. For example, $B = b \rightarrow B$ is the process that will communicate b indefinitely.

External (or *deterministic*) choice is denoted by $\Box$: we write $P \Box Q$ to describe the process that offers the choice between the initial events of both processes, before behaving as one of P or Q. *Internal* (or *nondeterministic*) choice is denoted by $\sqcap$: the process $P \sqcap Q$ may behave as P or as Q, but the environment has no choice over which. Both forms of choice have an associated indexed form: $\Box i : I \bullet A(i)$ and $\sqcap i : I \bullet A(i)$.

A parallel composition of processes tells us not only which processes are to be combined, but also describes the events that are to be synchronised on. The *synchronous parallel* operator, $\|$, requires component processes to agree on all events. As an example, the process $A \| B$, where $A = a \rightarrow b \rightarrow STOP$ and $B = b \rightarrow B$, will deadlock as the combination cannot agree on the first event. *Generalised parallel composition*, of the form $A \underset{X}{\|} B$, requires cooperation on the events of X; all other events can proceed independently. For example, in $A \underset{\{b\}}{\|} B$, the event a occurs without B's cooperation. Some processes do not synchronise on any events; *interleaved* processes are written $A \,|||\, B$.

Channels are the means by which *values* are communicated: $c.x$ is the communication of the value x on the channel c. A natural extension to this is to consider input and output values — of the form $c?x$ and $c!x$ respectively. Input along a channel can also be expressed using external choice: we might write $\Box x : X \bullet c.x \rightarrow P(x)$.

3.2 Semantics

It is often useful to consider a trace of the events which a process can communicate: the set of all such possible finite paths of events which a process P can take is written $traces[\![P]\!]$. However, it is well understood that traces on their own are not enough to fully describe the behaviour of a process. For a broader description of process behaviour, we might consider what a process can refuse to do: the refusals set of a process — the set of events which it can initially choose not to communicate — is given by $refusals[P]$. By comparing the refusals set with the traces of a process, we can see which events the process *may* perform: $refusals[A \Box B] = \{\}$ and $refusals[A \sqcap B] = \{\{\}, \{a\}, \{b\}\}$ (assuming A and B as defined above, and $\Sigma = \{a, b\}$). It follows that the *failures* of a process P — written $failures[\![P]\!]$ — are the pairs of the form (t, X) such that, for all $t \in traces[\![P]\!]$, $X = refusals[P/t]$, where P/t represents the process P after the trace t.

The aforementioned FDR tool compares processes in terms of refinement. We write $P \sqsubseteq_M Q$ when

Q refines P under the model M: Q is 'at least as good as' P. If we were only to consider traces, then

$$P \sqsubseteq_T Q \Leftrightarrow traces\,[\![Q]\!] \subseteq traces\,[\![P]\!]$$

Similarly, we define failures refinement as

$$P \sqsubseteq_F Q \Leftrightarrow traces\,[\![Q]\!] \subseteq traces\,[\![P]\!] \wedge failures\,[\![Q]\!] \subseteq failures\,[\![P]\!]$$

4 Threat Model

We now briefly review the supporting threat model of our work. For a broader review of the topic, we refer the reader to [1] and [11].

1. *Vehicle Owner/Driver.* Car drivers have unlimited access to the vehicle, which, in most cases, is their own property. They do not usually have knowledge about the internal functioning of a car. Some car models allow external apps to interact with the car or to add non-safety related software component to the head unit. Drivers are not limited to the vehicle's legal owner or their relatives, but rather should include everybody allowed to drive the vehicle (e.g. valets, car-sharers, etc.).
2. *Evil Mechanic.* An evil mechanic has unlimited access to many cars at the same time. They also have a deep knowledge of cars' internal functioning as well as specialist diagnostics equipment. As a result, they can easily modify cars' hardware and software components. In addition, they can also manipulate data stored within the vehicle [15]. Commonly, drivers have no way to tell what modifications their car underwent after being collected from the garage.
3. *Thief.* Thieves might be keen to steal the owner's possessions inside a car, car components, or the car as a whole. A thief can attempt to get access to the bus system, attach a malicious ECU and send fabricated messages possibly opening the doors, evading the immobilizer and starting the engine [8]. Since many cars share the same technology and car theft can have a high profit margin, acquiring knowledge and tools is worthwhile for professional thieves. The knowledge and tools might be acquired via reverse engineering or by colluding with evil mechanics.
4. *Remote Attacker.* Some individuals might want to bring havoc to many cars indiscriminately out of various possible motivations, e.g. the seeking of a feeling of power. To have maximum impact, an attacker could try to use any of the openly accessible interfaces of a car such as cellular connections usually used to connect the car via the internet to a back-end system, wireless connections for Vehicle-to-X communication (IEEE 802.11p), or Bluetooth connecting mobile devices to the head unit (cf. [2, 6, 9]).

Our threat model assumes that an attacker has complete access to the wired network of a car in addition to access to public interfaces. An attacker can modify the network, i.e. remove or add any network connection or ECU. With respect to data manipulation, if the attacker manages to modify data (software or configuration data) — as done in the Jeep Cherokee hack [10] — the attacks can cause a change to the built-in functionality

5 Formal Models of the Automotive Architecture

To enable the generation of test cases from models, there is a need to specify the underlying system to be tested in terms of models.

Our formal architecture consists of modelling individual networks and associated bus systems as CSP processes. We use the generic architecture of Figure 1 as a basis. Instead of modelling the generic architecture as a single entity, we decompose it into smaller parts, each of which represent a single network type or system. We are aware that a bus system may be formed by multiple ECUs [20]. However, for the sake of simplicity, we model each bus system as associated with a single ECU only. We use CSP's parallel composition operator ($\|$) to denote communication between bus systems and the corresponding network.

5.1 The system model

Our system model follows the diagram in Figure 1. We model gateways, networks, and bus systems connected to each network.

The first process, *GATEWAY*, manages the communication between the networks directly associated to it: two CAN High-Speed networks (*CANHS*1 and *CANHS*2), one CAN Low-Speed network (*CANLS*), and one FlexRay network (*FLEXRAY*).

Each network has its own CSP process, where associated bus systems are modelled. Events of the form *gateway_*$*$ serve to communicate between the process *GATEWAY* and individual network processes. External choices ($\Box$) split different possible paths (i.e. between gateways or bus systems). After the *gateway_*$*$ event in the network process (e.g. *CANHS*1), there is an external choice between available bus systems within that process. This multi-level choice is repeated many times in our models.

$$
\begin{aligned}
GATEWAY = \;& gateway\_canhs1 \rightarrow CANHS1 \\
& \Box \\
& gateway\_canhs2 \rightarrow CANHS2 \\
& \Box \\
& gateway\_canls \rightarrow CANLS \\
& \Box \\
& gateway\_flexray \rightarrow FLEXRAY
\end{aligned}
$$

$$
\begin{aligned}
CANHS1 = \;& gateway\_canhs1 \rightarrow \\
& (\; engine\_cu \rightarrow STOP \\
& \quad \Box \\
& \quad gearbox \rightarrow STOP \\
& \quad \Box \\
& \quad head\_unit \rightarrow STOP\,)
\end{aligned}
$$

$$
\begin{aligned}
CANHS2 = \;& gateway\_canhs2 \rightarrow \\
& (can\_ecu30 \rightarrow STOP \;\Box\; canhs\_most \rightarrow MOST\,)
\end{aligned}
$$

$$
\begin{aligned}
CANLS = \;& gateway\_canls \rightarrow \\
& (can\_ecu32 \rightarrow STOP \;\Box\; canls\_lin \rightarrow LIN\,)
\end{aligned}
$$

$$
\begin{aligned}
FLEXRAY = \;& gateway\_flexray \rightarrow \\
& (video \rightarrow STOP \;\Box\; radar \rightarrow STOP\,)
\end{aligned}
$$

The MOST and LIN networks do not connect with the main gateway. Their connection to individual ECUs in the CAN High-Speed and CAN Low-Speed networks, respectively, is also represented in our CSP models. The events *canhs_most* and *canls_lin* indicate those connecting ECUs.

$$
\begin{aligned}
MOST = \ & canhs\_most \rightarrow \\
& (\ dvd \rightarrow STOP \\
& \ \Box \\
& \ mp3 \rightarrow STOP \\
& \ \Box \\
& \ radio \rightarrow STOP \\
& \ \Box \\
& \ media \rightarrow STOP\ )
\end{aligned}
$$

$$
\begin{aligned}
LIN = \ & canls\_lin \rightarrow \\
& (lin\_sensor \rightarrow STOP \ \Box \ lin\_actuator \rightarrow STOP\ )
\end{aligned}
$$

5.2 Modelling attacks

The capabilities of the attacker are described as channels and are defined in process *Attacker* below.

The attacker can spoof commands to bus systems, block communication to a bus system, eavesdrop communication between bus systems, and change the functionality of a bus system. These capabilities are expressed by channels *spoofing*, *block*, *eavesdrop*, and *change_functionality*, respectively.

Spoofing, blocking and eavesdropping are associated with command-and-control or monitoring software, which are installed or operated by remote attackers, thieves or, even, the vehicle's owner. Change of functionality, on the other hand, requires a more advanced knowledge of the vehicle's internal systems as well as available time, thus it is more likely to be conducted by evil mechanics or vehicle owners. The Jeep Cherokee vulnerability discovered by Miller and Valasek is an example of spoofing combined with change of functionality [10].

We also define the set *All_Buses*, which include all bus systems in the vehicle. However, any subset of the bus systems can be used, if required. Variable c of channel *spoofing* indicate a command to be spoofed to the bus system.

$$
\begin{aligned}
& Attacker = \\
& \ spoofing?b : All\_Buses?c : \{0..10\} \rightarrow Attacker \\
& \ \Box\ block?b : All\_Buses \rightarrow Attacker \\
& \ \Box\ eavesdrop?b : All\_Buses \rightarrow Attacker \\
& \ \Box\ change\_functionality?b : All\_Buses \rightarrow Attacker
\end{aligned}
$$

Integration between system and attack models is done via alphabetised parallel composition, as defined by process *Attack*1 below. This type of parallel composition demands the definition of the events that we want to execute. In the example below, this is done by the set *AttackPath*, which restricts all executable events to *gateway_canhs*1, *engine_cu*, and *spoofing*.

$$
\begin{aligned}
& Attack1 = Attacker[AttackPath \ || \ AttackPath]GATEWAY \\
& AttackPath = \{gateway\_canhs1, engine\_cu, spoofing.engine\_cu.2\}
\end{aligned}
$$

A benefit of such models is that they allow a fine-grained definition of what systems to test and what attacks to test against. Moreover, the capabilities of each threat actor can be represented by different

test models. Even though process *Attack*1 uses the whole system model (process *GATEWAY*) as input, only a few events of it are actually considered (via the *AttackPath* set). Replacing *GATEWAY* by another subsystem can narrow down the scope of the test models even further.

6 Final Considerations and Future Work

We have created formal models of architectures of the networks and bus systems commonly found in a modern car. We then combine these models with attack models, thus enabling the automatic generation of tests to exercise the behaviour of car systems against attacks.

Our work also addresses the problem of scalability of security testing in distributed environments. Each of the involved systems must have its security tested. Consequently, the use of automatic tools as well as an underpinning formal model to guide the process is mandatory. Moreover, another advantage of our formal model is that it allows network parameters (e.g. latency) to be modelled alongside the network topology. We believe this feature can be useful to test the availability of components — e.g. sensors, actuators and ECUs — while the network they are plugged to is under attack. Remote attackers may selectively shut components off or trigger on-demand malicious commands —, consequently, there is a need to ensure redundancy mechanisms work as expected.

The next step in our work plan is to implement a tool to generate executable test cases from the test models introduced in this paper. This tool will use system and attack models in CSP as input and, with the aid of FDR, will generate traces for each execution of the input process. We anticipate that each trace generated by FDR will correspond to a different test case, which can further vary in terms of process variables. To interface with automotive engineering software, test cases will be generated as XML files or CAPL[1] code.

Furthermore, we also plan to extend the framework to encompass other main sources of threats to the modern car, for instance, attacks from vehicular networks (V2X) and attacks to autonomous vehicles' sensors (e.g. Lidar, cameras, etc). Sensor attacks have been shown in practice (e.g. [18]). A formal model of these attacks can also help to systematise the security testing of modern vehicles. Moreover, we will also pursue the integration of our test case generation method into industry-grade automotive engineering tools (e.g. CANoe [23] and PreScan [21]). Subsequent work will include the incorporation of our method onto a wider automotive testing methodology and its expansion to other kinds of cyber-physical systems with similar requirements (e.g. avionics).

Acknowledgments. Eduardo dos Santos thank CAPES Foundation (grant no. 1029/13-4) and Keble College for financial assistance during the development of this research. The authors thank the anonymous reviewers for their constructive comments.

References

[1] Richard R. Brooks, Seok Bae Yun & Juan Deng (2012): *Chapter 26 - Cyber-Physical Security of Automotive Information Technology*. In Krishna Kant & Nan Zhang, editors: *Handbook on Securing Cyber-Physical Critical Infrastructure*, Morgan Kaufmann, Boston, pp. 655–676, doi:10.1016/B978-0-12-415815-3.00026-1.

[1] Communication Access Programming Language, a language, similar to C, used in the programming of automotive bus systems [24].

[2] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner & Tadayoshi Kohno (2011): *Comprehensive Experimental Analyses of Automotive Attack Surfaces*. In: *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, USENIX Association. Available at `http://static.usenix.org/events/sec11/tech/full_papers/Checkoway.pdf`.

[3] Eduardo dos Santos, Dominik Schoop & Andrew Simpson (2016): *Formal Models for Automotive Systems and Vehicular Networks: Benefits and Challenges*. In: *2016 IEEE Vehicular Networking Conference (VNC)*, Columbus, OH, USA, pp. 1–8, doi:10.1109/VNC.2016.7835940.

[4] Ian D. Foster, Andrew Prudhomme, Karl Koscher & Stefan Savage (2015): *Fast and Vulnerable: A Story of Telematic Failures*. In: *9th USENIX Workshop on Offensive Technologies, WOOT '15, Washington, DC, USA, August 10-11, 2014.*, USENIX Association. Available at `https://www.usenix.org/conference/woot15/workshop-program/presentation/foster`.

[5] Tobias Hoppe, Stefan Kiltz & Jana Dittmann (2011): *Security threats to automotive CAN networks - Practical examples and selected short-term countermeasures*. *Rel. Eng. & Sys. Safety* 96(1), pp. 11–25, doi:10.1016/j.ress.2010.06.026.

[6] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham & others (2010): *Experimental Security Analysis of a Modern Automobile*. In: *Security and Privacy (SP), 2010 IEEE Symposium On*, IEEE, pp. 447–462, doi:10.1109/SP.2010.34.

[7] Sven Laux, Gurjashan Singh Pannu, Stefan Schneider, Jan Tiemann, Florian Klingler, Christoph Sommer & Falko Dressler (2016): *Demo: OpenC2X — An Open Source Experimental and Prototyping Platform Supporting ETSI ITS-G5*. In: *2016 IEEE Vehicular Networking Conference (VNC)*, pp. 1–2, doi:10.1109/VNC.2016.7835955.

[8] Charlie Miller & Chris Valasek (2013): *Adventures in Automotive Networks and Control Units*. *DEF CON* 21, pp. 260–264.

[9] Charlie Miller & Chris Valasek (2014): *A Survey of Remote Automotive Attack Surfaces*. *Black Hat USA* 2014, p. 94.

[10] Charlie Miller & Chris Valasek (2015): *Remote Exploitation of an Unaltered Passenger Vehicle*. *Black Hat USA* 2015, p. 91.

[11] Bassem Mokhtar & Mohamed Azab (2015): *Survey on Security Issues in Vehicular Ad Hoc Networks*. *Alexandria Engineering Journal* 54(4), pp. 1115–1126, doi:10.1016/j.aej.2015.07.011.

[12] Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Suhaib A. Fahmy & Samarjit Chakraborty (2015): *Security Analysis of Automotive Architectures Using Probabilistic Model Checking*. In: *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, ACM, New York, NY, USA, pp. 38:1–38:6, doi:10.1145/2744769.2744906.

[13] Oxford University (2017): *FDR4 - The CSP Refinement Checker*. `https://www.cs.ox.ac.uk/projects/fdr/`.

[14] Jonathan Petit, Michael Feiri & Frank Kargl (2014): *Revisiting Attacker Model for Smart Vehicles*. In: *Wireless Vehicular Communications (WiVeC), 2014 IEEE 6th International Symposium On*, IEEE, pp. 1–5, doi:10.1109/WIVEC.2014.6953258.

[15] Jonathan Petit & Steven E. Shladover (2015): *Potential Cyberattacks on Automated Vehicles*. *IEEE Transactions on Intelligent Transportation Systems* 16(2), pp. 546–556, doi:10.1109/TITS.2014.2342271.

[16] Q. Ran, X. Wu, X. Li, J. Shi, J. Guo & H. Zhu (2014): *Modeling and Verifying the TTCAN Protocol Using Timed CSP*. In: *2014 Theoretical Aspects of Software Engineering Conference*, pp. 90–97, doi:10.1109/TASE.2014.8.

[17] Sanjit A. Seshia, Dorsa Sadigh & S. Shankar Sastry (2015): *Formal Methods for Semi-Autonomous Driving*. 2015-July, doi:10.1145/2744769.2747927.

[18] Hocheol Shin, Dohyun Kim, Yujin Kwon & Yongdae Kim (2017): *Illusion and Dazzle: Adversarial Optical Channel Exploits against Lidars for Automotive Applications*, doi:10.1007/978-3-642-40349-1_4.

[19] Sebastian Siegl, Kai-Steffen Hielscher, Reinhard German & Christian Berger (2011): *Formal Specification and Systematic Model-Driven Testing of Embedded Automotive Systems*. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, IEEE, pp. 1–6, doi:10.1109/DATE.2011.5763028.

[20] Christoph Sommer & Falko Dressler (2015): *Vehicular Networking*. Cambridge : Cambridge University Press, doi:10.1017/CBO9781107110649.

[21] TASS International (2017): *PreScan Version 8.1.0*.

[22] Sam Thielman (2016): *'Someone Is Going to Die': Experts Warn Lawmakers over Self-Driving Cars*. `http://www.theguardian.com/technology/2016/mar/15/self-driving-cars-danger-senate-general-motors-google`.

[23] Vector (11/8/17): *CANoe - ECU Development & Test*. `https://vector.com/vi_canoe_en.html`.

[24] Vector (7/11/17): *CAPL Documentation - Vector :: KnowledgeBase*. `https://kb.vector.com/entry/48/`.

[25] D. Waraus (2009): *Steer-by-Wire System Based on FlexRay Protocol*. In: *2009 Applied Electronics*, pp. 269–272.

[26] X. Zheng & C. Julien (2015): *Verification and Validation in Cyber Physical Systems: Research Challenges and a Way Forward*. pp. 15–18, doi:10.1109/SEsCPS.2015.11.