A Car Hacking Experiment: When Connectivity meets Vulnerability

Sasan Jafarnejad, Lara Codeca, Walter Bronzi, Raphael Frank, Thomas Engel
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg, 2721, Luxembourg
sasan.jafarnejad.001@student.uni.lu,{lara.codeca,walter.bronzi,raphael.frank,thomas.engel}@uni.lu

***Abstract*—Interconnected vehicles are a growing commodity providing remote access to on-board systems for monitoring and controlling the state of the vehicle. Such features are built to facilitate and strengthen the owner's knowledge about its car but at the same time they impact its safety and security. Vehicles are not ready to be fully connected as various attacks are currently possible against their control systems. In this paper, we analyse possible attack scenarios on a recently released all-electric car and investigate their impact on real life driving scenarios. We leverage our findings to change the behaviour of safety critical components of the vehicle in order to achieve autonomous driving using an Open Vehicle Monitoring System. Furthermore, to demonstrate the potential of our setup, we developed a novel mobile application able to control such vehicle systems remotely through the Internet. We challenge the current state-of-the-art technology in today's vehicles and provide a vulnerability analysis on modern embedded systems.**

***Keywords*—Interconnected Vehicles, Car Hacking, Autonomous Driving, Embedded Systems, Mobile Computing, Security**

I. Introduction

Many objects that we use everyday at home and outside are becoming increasingly connected and remotely controllable. Today's high-speed, high-coverage mobile networks enable us to access these objects to influence and automate their behaviour that once required local and manual input. Motor vehicles are the latest addition to the *Internet of Things* (IoT) network of objects and thus an important topic to tackle in order to evaluate the impact and trade-offs that this technology has to offer.

When connectivity meets modern vehicles, we can immediately picture a scenario where we control a car with a smartphone. Moreover, as we present in this paper, autonomous vehicles and self-driving cars represent the next logical step and are now a reality. With the increasing complexity in the electronics of the vehicles, the number of Electronic Control Units (ECUs) and their importance in monitoring the different subsystems of a car has grown steadily over the last decade. In addition, modern vehicles are able to communicate with other devices using wireless interfaces potentially exposing the internal network of the car to vulnerabilities. It is our belief that the current state-of-the-art internal communication systems used in modern cars, are not ready to handle threats from external attackers.

In this paper, we present a platform based on open source hardware and software that is able to gain access to the internal network of a car. We show that the inadequacy of the existing security mechanisms in the internal network architectures is a major concern now that many vehicles are equipped with wireless communication capacities. Given the structure of the internal network, our remote access was able to compromise safety critical systems of the vehicle.

For all our experiments we make use of the *Open Vehicle Monitoring System* (OVMS)[1], which is an open source tool that brings remote access to a number of electric cars in order to retrieve various information such as the battery state, location, and other readings. OVMS supports different vehicles, and amongst them, the *Renault Twizy*, the car that we used for all our experiments.

II. Related Work

Many studies have been carried out for discovering vulnerabilities and proposing solutions to increase the security of embedded car systems. In [1], the authors provide an overview of all the different approaches used to investigate these security issues and list the solution that have been proposed to increase their safety.

Given the complexity of the automotive electronic architecture, works such as [2] and [3] provide a taxonomy of the different attacks such as systematic manipulation and intrusion. In [4], the authors suggest that the internal automotive computer network was not protected from a local attacker able to physically access the car. Here, they present an attack that leverages individual weaknesses in the different subsystems, such as an attack that embeds malicious code in an *Electronic Control Unit* (ECU) and that erases any evidence of its presence after a crash. We further extend this idea by exploring the feasibility of reproducing such attacks at distance by remotely taking over a vehicle's internal control systems. Papers such as [5] and [6] present an overview of the different vulnerabilities, the possible attacks and provide an outlook on security challenges of interconnected vehicles. A notable work that explores possible attacks on vehicle subsystems with a very practical approach is [7] in which authors discuss in detail how to manipulate and control a number of safety critical subsystems of a Ford Escape and a Toyota Prius. One year later same authors in [8] provide an insightful survey of

[1]OVMS website: http://www.openvehicles.com/

remote attacks by evaluating possible penetration scenarios for 20 different vehicles. In [9], the authors provide an overview of the advances in autonomous vehicle technology from the latest US and European research projects and point out the vulnerabilities of current embedded systems. In recent years, many solutions have been proposed to overcome those vulnerabilities. CANAuth [10] and Libracan [11] are two broadcast authentication protocols for the CAN bus based on basic building blocks from cryptography (encryptions, signatures, etc.).

In this work we describe a novel mechanism based on open source hardware and software, that is able to provide remote access to the internal network of the car, and given its structure, we demonstrate how to compromise safety-critical systems such as the engine controller.

III. Background

Before presenting our experimental setup and problem statement, we first provide some background information about the concepts that are used in this work.

A. Electronic Control Unit

An *Electronic Control Unit* (ECU) designates an embedded system that controls one or more subsystems of a vehicle. One type of ECU is the *Engine Control Module* (ECM), which specifies the runtime parameters of the engine (combustion, hybrid or electric). ECMs introduced in the eighties to allow for a fine control of the amount of fuel injected in the motor at each cycle. Modern ECUs are, among other things, able to control air intake, fuel injection, torque and collect various sensor information. They can have many functions, from the door control unit to human computer interfaces and cruise control systems. It is of vital importance that such systems run unperturbed according to factory specifications. A malicious tampering to such components could seriously compromise the car's behaviour on the road and lead to a potential accident.

B. Controller Area Network

In the past, car components, also called nodes, were wired independently to each system they needed to communicate with, resulting in enormous lengths of electric cable used. This led to an increase of the cost and weight of the car and made maintenance more difficult. To address this issue Robert Bosch GmbH developed the *Controller Area Network* (CAN). The CAN is a multi-master serial bus, which allows micro-controllers in a vehicle (generally ECUs) to exchange short messages at a maximum rate of 1 megabits per second. On the CAN bus all the devices are usually connected through a twisted pair of wires and each message is broadcasted to every device on the bus. There is no notion of identification for the source and destination of messages. Each message has an ID, which identifies the purpose of the message and its priority. Priorities are resolved based on an arbitration method. There are four different message types on the CAN bus: *Data Frame*, *Remote Frame*, *Error Frame* and *Overload Frame*. The first two types are respectively used to broadcast a data payload between nodes and to request the transmission

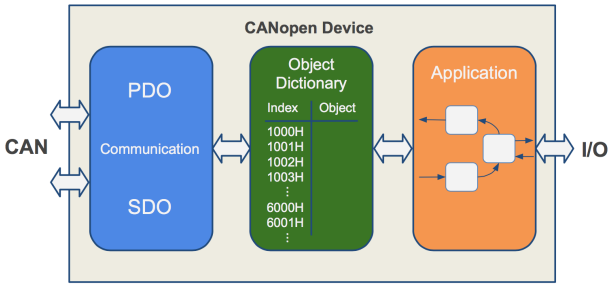

Figure 1: CANopen device structure

of data from a particular node. The error frame is sent by a node that has detected an error in a message to ask the sender to retransmit that message. The overload frame is sent by a busy node to request an extra delay between messages. As there is no central coordination defined by the CAN standard, nodes can be added or removed without impacting the network.

C. CANopen

The CAN standard (ISO11989) defines the OSI layers 1 and 2, but in practice these are already being handled solely by the hardware (CAN controller), which is of great help for engineers. However a flexible and highly configurable embedded network based on CAN, is not possible without employing higher layer protocols, and *CANopen*[2] is one of the existing solutions. CANopen specifies a set of device and application profiles. The device profiles specify the interface of a logical device and the application profiles a set of virtual device interfaces. CANopen devices may implement one or several virtual devices. Devices implementing the same set of profiles are partly or completely exchangeable. This allows a flexible integration of CANopen devices and enables interoperability and interchangeability between multiple devices.

Every CANopen device is described by an *Object Dictionary* (OD). An OD describes a group of objects, each of them addressed by a 16-bit index. To access individual elements of data in each object an 8-bit subindex is used. The OD encloses all the parameters describing the device and its behaviour on the network. Each subindex points to an object that is defined by *Object Name*, *Name*, *Type*, *Attribute*, *M/O*. The Attribute defines object access rights and the M/O specifies whether it is mandatory or optional.

CANopen defines several communication objects to achieve different goals such as network management, timing, synchronization and handling emergency states. But the most vital communication objects are *Service Data Object* (SDO) and *Process Data Object* (PDO). An SDO gives a node read and/or write access to objects on another device's OD knowing its index and subindex. In this case, the first node is the client and the target device is the server. PDO is used to transfer real-time data. In SDO a node requests to read or write an object but in PDO data is being broadcast at certain intervals or is triggered by

[2]CANopen: http://www.can-cia.org/index.php?id=canopen

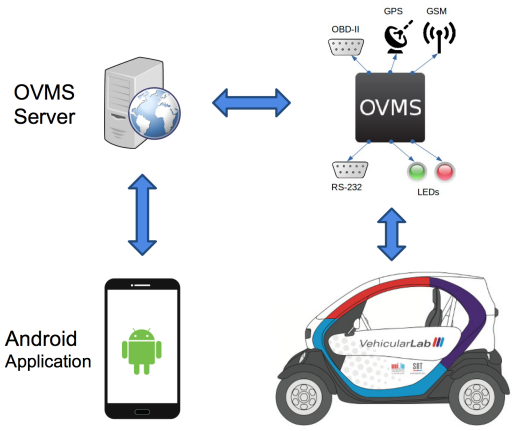

Figure 2: Experimental Setup

an event. The simplified structure of a CANopen device is depicted in Figure 1.

IV. Experimental Setup

For all our experiments we used a factory standard Renault Twizy 80. The Twizy is a compact electric quadricycle for two passengers, including the driver. The motor controller ECU that is mounted in the Twizy is produced by the company Sevcon[3], which manufactures motor controllers and system components for electric vehicles. The actual model installed in the Twizy is a Sevcon Gen4 controller.

The Twizy is equipped with an additional backseat computer that provides direct access to the OVMS in order to make changes to the firmware. Although it is an atypical vehicle, the Twizy can be seen as a simplified version of a modern car as it shares a similar system architecture. For our study the most important requirement is the presence of a standard CAN bus for the communication between the different components.

Access to the internal CAN bus is provided via the *On-board Diagnostics* (OBD) port. Given the increasing number of ECUs in cars over the past few decades, manufacturers started to add self-diagnostic capabilities to their vehicles in order to help technicians retrieving the status of the subsystems and determine faults. The OBD, or more commonly OBD-II (EOBD in Europe) is a standardized interface and it is mandatory by law in both United States and Europe since 1996 and 2001, respectively. In order to interact with the CAN bus and the attached systems, a variety of tools can be purchased without any requirements. For our experiments we choose the OVMS. This tool is open source and open hardware and is equipped with GPRS/GSM module, a GPS and a serial port that allows to configure the device.

This tool was originally developed by and for owners of the Tesla Roadster, a high-end electric vehicle. It allows the user to monitor the status of the car battery, the last known GPS location, and to control some of the car's features such as the lock status of the doors or the heating system through their smartphones. Compared to a USB or Bluetooth adapter, the OVMS allows to extend the range of interaction with the CAN bus by using the cellular network. Moreover, due to open source nature of the OVMS, the source code is openly accessible online [12]. We used it both as a way to acquire knowledge on how the internal systems of the Twizy works, and as an experimentation platform.

Given the presence of a serial port on the OVMS module, we used the backseat computer as a Wi-Fi proxy to speed up our experiments and avoid relying on the GSM signal reception. The connection between the two was made with a USB to RS-232 adapter. In this setup, the module needs to boot into debug mode during, which the output messages are sent through the serial port instead of the cellular network. This allowed us to connect to the backseat computer via Wi-Fi and send messages directly to the OVMS through the serial port. Instead of operating with a serial terminal application we developed a program in C called OVMS Controller that was installed on the backseat computer. This enabled us to automate the communication with the module, and to perform tasks that would otherwise be time consuming and redundant.

In addition to the serial port, one can interact with the OVMS via text messages or through the Internet using the available mobile applications (iOS and Android). When using the mobile applications, the module connects to a public server that acts as an Internet proxy to relay commands between OVMS and the client. A default server is hosted by the *Tesla Motors Club*[4] for convenience. Messages going through the Internet use a Base64 encoding scheme and are encrypted using RC4 stream cipher [13]. To reduce latency and to have a higher degree of flexibility, we deployed our own server locally.

During the project, we developed a number of new features for the OVMS. In order to use these features remotely, we developed a new web interface and a custom version of the OVMS Android application. Using these two interfaces, the module can easily be accessed using smartphones and standard computers through the browser.

V. Exploiting Vulnerabilities

As introduced in the previous section, the main ECU of the Twizy is the Sevcon Gen4 controller. We will now describe its architecture and how we succeeded in accessing the main configuration parameters. The Sevcon Gen4 uses CANopen as higher layer protocol and specifies different level of access rights that prevent manipulating some objects without the correct level of authentication.

Setting up the parameters of a Sevcon Gen4 (see the Sevcon Gen4 manual [14]) is done by reading and writing the values associated to the objects. The tools commonly used for this purpose are either a proprietary software on a computer, or a professional hand-held device. Nevertheless, with a good understanding of the CANopen protocol, the process can be done manually by forging

[3]Sevcon website: http://www.sevcon.com/

[4]Tesla Motors Club: http://www.teslamotorsclub.com/

Table I: Access Levels

Access Level	Usage
1	User (Default)
2	Service Engineer
3	Dealer
4	OEM Engineering
5	Sevcon Engineering

the data frames. There are 5 levels of accessibility, and different login credentials for each level. The access levels are listed in the Table I. In the initial state, the user is not authenticated, and can only access a very limited set of objects. The higher the access level, the more objects can be manipulated (i.e. the configuration changed). The authentication can be done by sending a data frame whose payload contains the ID of the authentication object and the passcode. As a result, anyone with the right passcode could have access to the whole OD of a Sevcon Gen4.

A. Brute-Force Attack

To be able to freely reconfigure the Sevcon Gen4 one needs to gain access to the right passcode preferably for a high access level (e.g. 4 or 5). Like for every other protected system when it comes to finding a passcode, brute forcing is the simplest way of approaching the problem. However, while being easy to perform, it is only practical when applied to short passcodes. Knowing that the passcode for the Sevcon Gen4 is only 2 bytes long, this was the preferred approach. After implementing the required routines on the OVMS to try all 65536 combinations on the ECU, we executed the program. It took our brute forcing routine approximately 11 hours to find the passcodes for all the access levels. Although this was reasonably fast, a delay was introduced due to the the link between the OVMS and our brute forcing routine that has been executed on the backseat computer. Moreover we checked for all access levels in one run, in other words we tested the passcode space 5 times.

B. Proof of concept

After studying the technical documents of the Sevcon Gen4 and having obtained the passcodes for all the access levels, we defined a number of experiments. We only had access to one master OD that was not specifically for the Sevcon Gen4 but was a general format for all Sevcon products. The Master OD is a reference object containing information about all the dictionaries and their constraints. We read the configuration of the controller to determine what were the parameters in use for the Twizy. We then tried to modify those parameters one by one to observe what changes they induced in the behaviour of the vehicle.

There are many parameters to take into account in order to fine tune the motor and change the behaviour of the system. Here, we present two parameters that allowed us to make the car move autonomously.

Throttle Control: By default, the Sevcon Gen4 converts the voltage from the throttle pedal position into a scaled numerical value that controls the power output of the

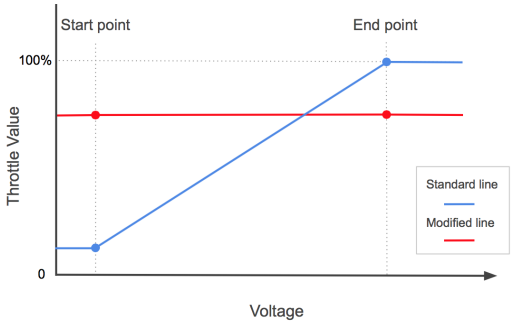

Figure 3: Throttle conversion function

Table II: Gear State

Forward	Reverse	Action
0	0	Neutral (No Torque)
0	1	Negative Torque
1	0	Positive Torque
1	1	Unauthorized

electric motor. By having access to all parameters of the controller, we can modify the conversion function to alter the default behavior of the vehicle. As shown in Figure 3, we modified the parameters of the function in order to interpret any input voltage as the numerical value of our choice. In this example no matter what input voltage (horizontal axis), the output value for the throttle (vertical axis) is about 70 percent. The default conversion line is drawn in blue and the modified line is in red. This modification effectively simulates a push on the throttle pedal and removes the user's control over the acceleration of the vehicle.

Gear State: We identified two Boolean switches that together describe the gear state of the vehicle. One corresponds to the forward direction and the other to the reverse direction. By default, those values are set by physical switches next to the steering wheel. Table II shows the internal logical functions of the Sevcon Gen4 and how they determine the driving mode of the vehicle (e.g. torque positive, negative or neutral).

By programmatically setting those values, we were able to automatically change the gear state and as a consequence the driving direction of the vehicle.

Brake: The previously described technique could also be used as the brake. Only by applying negative torque to the motor, the vehicle will slow down. We used this in our experiments to stop the vehicle while it remained at low speed, under 30 km/h. This was not tested at higher speeds because the amount of negative torque would not be sufficient to stop the car. Another concern is that this method, also called *Plug braking*, generates a lot of heat and endangers the operation of the engine [15].

C. Remote Control

To be able to demonstrate what is practically feasible using this module we decided to implement the aforementioned exploits directly in the OVMS module.

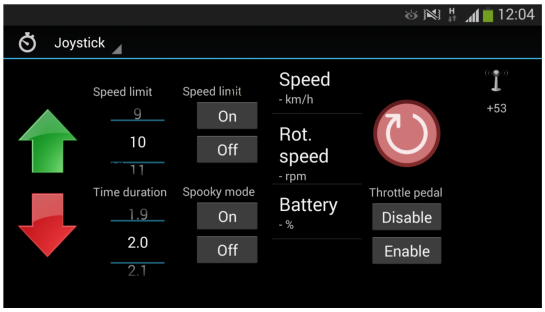

Figure 4: Android Application

We implemented several keyboard short-cuts on the OVMS client to have an easier control over the dynamics of the vehicle. By exploiting the vulnerabilities discovered in the experimentation phase, the module is able to remotely and autonomously drive the vehicle forward and in reverse. Forward and reverse throttle values sent by the client are expressed as percentages of the maximum throttle value acceptable by the Sevcon Gen4. To avoid latency and reliability issues, we implemented the routines directly on the OVMS. The client applications (Android and Web application) are solely used to send basic commands.

Android and Web Application: Two client applications have been implemented to demonstrate our scenario: An Android mobile application and a web application. We modified the already existing mobile application provided by OVMS to include the remote control features that we developed. We also modified the OVMS firmware in order to accept the custom commands of the client applications.

The Android application interface is shown in Figure 4. The vertical arrows represent the forward and reverse throttle, similar to the ones used by game controllers. The application also allows to disable the throttle pedal, limit the maximum speed or enable a demo mode that repetitively moves the car slowly forward and backward.

VI. Discussions

In this section we will present potential attack scenarios, suggest solutions and discuss future applications. First, it is important to note that the Sevcon Gen4 is only accessible while the cars is turned on (i.e. ignition turned on). Therefore an attack is only possible while the car is operated. Given that the Twizy does not have door locks nor windows and that the OBD-II port is freely accessible inside the glove box, it is easy for an attacker to install a system such as the OVMS without the owner noticing. The Sevcon Gen4 centralizes many safety critical functions of the vehicle, some of which are not implemented in the case of the Twizy. For example, the master OD indicates the presence of objects managing brake lights and steering servomotors. However the Twizy does not have assisted steering, therefore these features cannot be exploited. If this was the case, as shown in [7], complete control over the steering wheel would be possible and the attacker would be able to make dangerous turns at high speeds and potentially collide with other vehicles or with road infrastructures. Furthermore, we observed that the passcode is the same for every Sevcon Gen4 controller. As a result, once connected to the CAN bus, there are no security measures to prevent an attacker from taking control over the subsystems of the vehicle. The security of the onboard system only relies on the assumption that an attacker does not have the passcode to access the engine controller.

Possible Attack Scenarios: In order to demonstrate the potential of our system, several attack scenarios are described to show some use case examples of the module. We identified the following attack scenarios:

- Forcing the car to go forward or backward.
- Limiting the speed (e.g. Very low speed on the highway).
- Setting unsafe motor and voltage parameters which, could lead to possible damage to the engine of the vehicle.
- Randomly changing motor direction.
- Interacting with the dashboard to display false data, tricking the driver into making dangerous maneuvers.
- Changing or inverting the conversion function of the throttle input.

The proposed attack scenarios can either be activated remotely by an attacker or triggered automatically by the module upon any arbitrary events such as speed or the location information retrieved from the integrated GPS module (e.g. while being in a predefined area or when the speed is over a certain value).

A. Possible Solutions

There have been many attempts in solving security issues in automotive environments and more specifically on the CAN bus.

One important category of solutions propose use of cryptography for authentication. For instance solutions proposed in [10], [11], [16] are in fact very effective. However, not all the processors used in the ECUs are powerful enough to incorporate them. To solve this issue *Hardware Security Modules* (HSMs) were proposed. The main idea of HSM is to dedicate part of ECU's hardware only for encryption purposes[17]. Although HSM handled the security overhead, they demand a new and more sophisticated ECU design, which leads to much higher costs. Cryptographic solutions are the best candidates as a long term solution but this transition will not happen in near future. Therefore it is important to find adequate solutions that are conveniently integrate into the current vehicles.

What is worth mentioning about Twizy and Gen4, is the lack of a mechanism that prevents brute-force attacks. For example, in [7] it has been shown that the Toyota Prius already uses a challenge-response authentication protocol and when an attacker tries to brute-force the system it fails after 10 attempts.

Another category of solutions are based on detecting anomalies in the network, just like the *Intrusion Detection Systems* (IDSs) for computer networks. For example in [18] authors use time and frequency and in [19] the entropy of messages are used as feature to monitor and detect anomalies. To our knowledge only anomaly based solutions can effectively detect and possibly react upon attacks on the CAN bus without requiring any additional change to the network structure.

We believe that a temporary solution to achieve a reliable security in vehicles lies in implementing more sophisticated IDS that not only detects anomalies and signature based attacks, but also actively inspect current state of the vehicle and ECUs. For example in the Twizy, an observation of write access to the configuration ODs of Gen4 should flag a suspicious event. If this occurs while the vehicle is moving, the IDS could react upon it and trigger an alarm informing the driver of the incident.

B. Other Applications

Based on what we showed in Section V, one can foresee that these security flaws in the Sevcon Gen4 controller can lead to the development of new applications. Using the OVMS bundled with our improvements the Twizy can be transformed into an autonomous vehicle at a reasonable cost. More equipments such as additional sensors (e.g. Lidar) and actuators (e.g. for braking and steering) are needed to build a fully functional autonomous vehicle. In this preliminary work we proved that the power train, which is the most critical system in a vehicle, can be controlled electronically bringing us one step forward towards automated driving.

VII. Conclusion and Future Work

In this paper, we presented an experimental platform able to remotely access and interact with internal systems of a vehicle. This platform is composed of a Renault Twizy 80, an Open Vehicle Monitoring System (OVMS) and an Android mobile application used as communication interface. The goal of this work was to remotely control the safety critical systems of the vehicle. Using the OVMS we accessed and reconfigured the Sevcon Gen4 controller in order to manipulate the behaviour of the vehicle. We showed that with off the shelf hardware it is possible to control vital engine parameters, which allowed us to interact with the operation mode of the vehicle (e.g. slow down or stop the vehicle, reversing the gear while moving). By doing this, we noticed a lack of protection mechanisms, which allowed us to exploit and modify many parameters of the vehicle at runtime (e.g. gear, throttle, speed limitation, etc.). For demonstration purposes, we implemented a web interface and an Android mobile application able to interact remotely with the vehicle. We demonstrated the effects of remotely changing the vehicle's behaviour in a real life situation and pointed out the dangers behind such attacks.

Our future work will consist in adding additional equipment to the vehicle, including sensors and actuators for the braking and steering. Ultimately, our goal is to produce a low cost fully connected and fully automated electric vehicle.

References

[1] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*. IEEE, 2013, pp. 1–12.

[2] M. Wolf, A. Weimerskirch, and T. Wollinger, "State of the art: Embedding security in vehicles," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 074706, 2007.

[3] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 528–533.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, 2011.

[6] R. Moalla, H. Labiod, B. Lonc, and N. Simoni, "Risk analysis study of its communication architecture," in *Network of the Future (NOF), 2012 Third International Conference on the*. IEEE, 2012, pp. 1–5.

[7] C. Miller and C. Valasek, "Adventures in automotive networks and control units," in *DEF CON 21 Hacking Conference. Las Vegas, NV: DEF CON*, 2013.

[8] ——, "A survey of remote automotive attack surfaces," *Black Hat USA*, 2014.

[9] J. Ibañez-Guzman, C. Laugier, J. D. Yoder, and S. Thrun, "Autonomous driving: Context and state-of-the-art," in *Handbook of Intelligent Vehicles*. Springer, 2012, pp. 1271–1310.

[10] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," in *ECRYPT Workshop on Lightweight Cryptography 2011*, 2011.

[11] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "Libra-can: a lightweight broadcast authentication protocol for controller area networks," in *Cryptology and Network Security*. Springer, 2012, pp. 185–200.

[12] Open vehicles monitoring system. [Online]. Available: https://github.com/openvehicles

[13] *OVMS Protocol Guide*, 2013, v2.5.1.

[14] *Sevcon Gen4 Applications Reference Manual*, 2009, rev 3.0. [Online]. Available: http://www.thunderstruck-ev.com/Manuals/Gen4_Product_Manual_V3.0.pdf

[15] C. Richard, *What is Plug/Dynamic Braking for series motors?*, ALLTRAX, jan 2007, technical Note 008.

[16] O. Hartkopp, C. Reuber, and R. Schilling, "Macan-message authenticated can," in *10th Int. Conf. on Embedded Security in Cars (ESCAR 2012)*, 2012.

[17] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *Information Security and Cryptology-ICISC 2011*. Springer, 2012, pp. 302–318.

[18] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 1110–1115.

[19] I. Broster and A. Burns, "An analysable bus-guardian for event-triggered communication," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 410–419.