Security Improvements in Connected Cars

Case Study: CEVT Connected Cars

Master's Thesis in Computer Science and Engineering

XINYU JI

HANWEI CHENG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Security Improvements in Connected Cars

Case Study: CEVT Connected Cars

XINYU JI
HANWEI CHENG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Security Improvements in Connected Cars
Case Study: CEVT Connected Cars

XINYU JI,
HANWEI CHENG

Supervisor: Robin Adams, Department of Computer Science and Engineering
Advisor: Reza Jalalinia, China Euro Vehicle Technology AB
Examiner: David Sands, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The cover picture is the infotainment system of Lynk & Co [1], which is our test target in this master thesis.

Typeset in LaTeX
Gothenburg, Sweden 2019

Security Improvements in Connected Cars
Case Study: CEVT Connected Cars

XINYU JI,
HANWEI CHENG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The Connected Car brings a fast development of intelligent and better driving experiences by realizing more functions and providing more services. Connected Cars usually contain a number of electronic control units (ECUs), which are small computer modules providing different functions. ECUs are connected by different automotive buses, which forms the vehicle system. The rapid development of the technologies brings a lot of advantages such as safety, convenience, sustainability and a better user experience. However, lots of security problems with the vehicle system also occur. The modern vehicle systems is no longer only formed by the hardware and wires. Instead, software is installed in each ECU, which gives chances to attackers to hack the car. For the importance of car security, we need to find a way to evaluate the potential vulnerabilities.

This master thesis is a case study on the connectivity vehicle system of CEVT, which is focused on the on-board connectivity system of the in-vehicle network. We have studied and compared some fuzzing tools. Among these tools, a suitable fuzzing tool has been selected and modified to be implemented on the current vehicle system in order to investigate how well fuzzing can be used to find the potential vulnerabilities on the vehicle system. In this work, we choose an interesting private protocol of CEVT as the test target. In this thesis, we call the protocol the Internal Communication Protocol (ICP). The results of the test are discussed.

Keywords: Fuzzing, Connected cars, In-Vehicle networks, Vehicle Security, Security Testing, Sulley.

Acknowledgements

First we would like to express our sincere gratitude to our supervisor in Chalmers, Robin Adams, for his kindly help and guidance with detailed comments and feedback throughout the thesis work.

We would like to thank Sarah Wills Carlsson, Team Leader at CEVT Infotainment, who gave us this opportunity to study and work with CEVT vehicle system. We are thankful to our CEVT supervisor Reza Jalalinia and Stefan Oscarsson who shared cyber security knowledge with us and provided advice on our work. We would like to warmly thank other engineers in CEVT, who have kindly support us and share their knowledge about the testing rig and tools, especially we would like to thank Rui Liang.

Xinyu Ji, Hanwei Cheng, Gothenburg, July 2019

Contents

List of Figures

1
Introduction

With the development of the IoT (Internet of things), devices used in our daily life become much "smarter". Instead of working alone, devices can interconnect with each other now. They can not only communicate or interact with each other, but also can be monitored or controlled remotely by just using your personal smart phone in any place in the world [2]. Turning on the air-conditioner and starting the rice cooker on your way home is no longer impossible. IoT technology really helps people to make their life more convenient and intelligent. It introduces a lot of new concepts such as the smart home, smart grids and now even "smart" cars, which are usually called connected cars.

As said by Volkswagen [3], *"The car will soon be our second home."* To accomplish this, the connected car is one of the most important steps. A connected car is a car that allows devices to be connected to other devices inside the vehicle, or connect to the devices or network outside the vehicle, such as the home and office [4]. The connections are realized by Internet access, which is usually by a local area network. Several systems or devices could be connected in the vehicle, which creates convenience and better user experience for the customers. For example, a smart phone connected via Bluetooth, voice control for the Infotainment system; intelligent driving assistance in ADAS (Advanced driver-assistance systems); real time traffic information in Navigation system. In the future, it will even realize Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure communication, which enables the vehicles to “talk” to each other and infrastructure directly. This can manage and control the traffic in a better and safer way, which is an important progress for autonomous driving.

Modern vehicles consist of not only mechanical components and electronic components, but also software are indispensable parts in them. As a successful product, the Tesla Model 3 has various software and connection points to the Internet, which realizes the Autopilot function, the use of a smart phone as key and its intelligent infotainment system [5]. However, cyber-security issues occur in the automotive industry and become a big challenge. With connection to the Internet and smart phones, vehicles open a digital door for the hackers and malicious attacks. Several successful attacks happened on different car brands, such as the Tesla Model X hacked by Chinese hackers [6], Ford Fusion and Toyota Prius losing control of the brakes [7] and Jeep Cherokee recalled 1.4 million cars that were hacked [8]. Hackers are able to remotely open the doors and trunk, blink the car lights, play the music from the car’s radio, control the cruise control, steering, parking assist and get ac-

cess to the remote keyless entry system, which means the hackers can enter the car without the key [7]. What's more, they could even control the brakes to brake suddenly while the car was driving. These, no doubt, are huge security threats. Some of these controls are realized when the car was physically near to and connected to a malicious Wi-Fi hotspot. Some are realized by getting access through the infotainment system through a smart phone. It raises the alarm to many car manufacturers to strengthen their car systems and fix possible security vulnerabilities to guarantee car safety.

There are three important systems that are relevant for a connected car, which are the vehicle, the portal at the vehicle company and the communication link between the vehicle and the portal. Electronic Control Units (ECUs) and the internal vehicle networks constitute the vehicle. Each ECU is responsible for one or several different vehicle functions. They are connected to each other through different bus system technologies in the vehicle system, such as Ethernet, Controller Area Network (CAN), FlexRay and Local Interconnect Network (LIN), which form a internal car network together. This is usually called the in-vehicle network [9].

This master thesis is focused on the internal vehicle network. The test method used to find potential vulnerabilities on the in-vehicle system is fuzzing. Fuzzing is an automated security testing method. It generates random semi-valid data as input to crash the program or system [10]. The goal of this master thesis is to find as much as possible potential vulnerabilities by using certain fuzzing tool. Knowledge of the connectivity system, internal vehicle network and protocols in the internal network is needed in order to better understand the test target system. In particular, we study and compare different types of fuzzing methods and tools. To accomplish the goal, a suitable fuzzing tool is chosen to be modified and implemented on the target system.

1.1 Purpose of This Thesis

The purpose of this thesis is to study and compare different fuzzing tools and choose a suitable one which could be implemented on the current connectivity system of CEVT. In addition, the chosen fuzzing tool is investigated for how well it could be used to learn the vulnerabilities in the CEVT's vehicle system. As fuzzing could damage the vehicle system, the challenge is that chosen fuzzing tool may need to be modified to meet the requirements of CEVT's system and formulate the testing strategy more carefully to avoid unexpected results.

1.2 Thesis Partner

This thesis is based on the vehicle connectivity system of CEVT, China Euro Vehicle Technology, AB. *CEVT is a development center for future cars of the Geely Group and describes itself as 'focusing on finding smarter ways to build cars – through modular development, by pushing the boundaries of virtual engineering, and by taking*

on new customer desires to create all-new cars [11]. This master thesis will focus on studying and comparing different fuzzing tools, and applying a suitable one to the vehicle connectivity system of CEVT, to test, for example, some private protocols in order to investigate how well fuzzing can be used to learn vulnerabilities in the CEVT's vehicle system.

1.3 Limitations

Fuzzing finds bugs by just sending invalid or semi-valid packets to the target system, so it could only find simple bugs. The limitation of fuzzing is that fuzzing could not analyze the precise location of the bugs and detect all the bugs of the target. We need to use other tools or manually analyze the bugs. In other word, fuzzing is a good attacking tool, but it is not a good analysis tool.

In this master thesis, we focus on the in-vehicle network security. We do not research any security vulnerabilities for the external networks. The fuzzing tool Sulley can be used to fuzz any protocol, but we only focus on Internal Communication Protocol (ICP), which is a private protocol currently used in the On-board vehicle system of CEVT. We have modified the monitor of Sulley in order to run it on the local computer. However, the modified monitor cannot give the exact message caused the problems, when network delay happens. We need to find the correct semi-valid packets manually from the list.

1.4 CEVT Limitations

This project was carried out in CEVT AB. As the work is focused on the security testing of the vehicle system, it relates to some sensitive and confidential information, such as the detailed design information for the architecture of vehicle system, detailed information for ECUs and security vulnerabilities founded during security testing. Since this master thesis is available to the public, the sensitive information is removed or hidden in this report due to the Non-disclosure agreement with CEVT AB.

1.5 Thesis Outline

This master thesis is written in the following structure:

- **Chapter 1:** This chapter gives a brief overview of this thesis, the goal and the motivation, the thesis partner CEVT AB, the limitation of the work, non-disclosure agreement (NDA) with CEVT AB and a timeline of how this thesis work was divided into different steps.

- **Chapter 2:** This chapter provides all the relevant background information that readers need in order to understand the whole thesis work. Chapter 2 is divided into two parts. The first part is an introduction to fuzzing. It gives an overview of what is fuzzing, the limitations of fuzzing, the classification of fuzzing and how a fuzzing system works. The second part gives background knowledge about connected cars. It briefly introduces the connected cars, functions that connected cars have, automotive networks in connected cars, the connectivity system and attack methodologies on a vehicle system.

- **Chapter 3:** Chapter 3 is a study of different fuzzing methods. It describes and compares the capacities, prices, if it is easy to modify and if it is suitable for the target system. Finally, the most suitable fuzzing method is chosen among all these fuzzing methods. This method is used to test the target system in the following chapters.

- **Chapter 4:** This chapter introduces closed system fuzzing. It describes how the test targets are selected. The targets are the target protocol and the target ECUs in the system. It also describes how the fuzzing is implemented on the target system, what has been modified during the implementation and how the data generator and monitor work during the whole fuzzing process.

- **Chapter 5:** This chapter is a case study of CEVT connected cars. It introduces the fuzzing topology and test strategy. It also shows the final results.

- **Chapter 6:** The results of fuzzing and the future work are discussed in this chapter.

- **Chapter 7:** This chapter is the conclusion of this thesis.

2
Background

2.1 Fuzzing

2.1.1 Overview

Fuzzing or fuzz testing is an automated software testing technique which enables testing of various boundary test cases. It was first proposed by Miller in 1989 [15]. The main concept of it is to generate random malformed data automatically as input, send the data to a target program and monitor the process with the intention to find bugs. The inputs are semi-valid data which can be generated randomly or based on the internal knowledge of the target program automatically or semi-automatically [10]. Monitors are set up to monitor the exceptions thrown by the program, such as crashes or failing built-in code assertions, in order to find potential software bugs and vulnerabilities. If a failure is found, the malformed data will be saved by the tool for further analysis and the next piece of data will be sent to the target. If there is no failure, the data is deleted by the tool and the testing continues by sending the next piece of data [10].

Developers need to consider how the program behaves when they define a program. However, unexpected behaviors may happen in certain areas which are often undefined and beyond the expectation of the developers. Exploring these areas becomes one of the challenges. Fuzzing can help with this. The main purpose of fuzzing is not to test whether the functions of the program are correct or not, but to explore and to test those undefined areas and undefined behaviors [13].

Fuzzing can be used to find a variety of issues, for example, errors in the programs, security vulnerabilities, buffer overflow and Denial of Service. As mentioned above, it is used to test how well program can deal with unexpected results. Fuzzing can find bugs like memory errors, buffer overflows and heap overflows. It becomes a test oracle (a mechanism that testers use to judge whether the test is passed or not [14]) when fuzzing is used to trigger when the bug has occurred [17]. Fuzzing is always used to test programs which have structured inputs. The structure defines the difference between valid and invalid inputs. However, fuzzing generates semi-valid inputs, which means the inputs are correct enough to be accepted by the parser, but they are still invalid, which may cause unexpected results in the program.

2.1.1.1 Fuzzing Limitations

Fuzzing can only find the simple bugs in the target program effectively. When attacking a black-box system, it has the limitation that it is harder to evaluate the impact of the vulnerability. Deeper code investigation is needed to analyze the crashing program. It could not provide a complete picture of the bugs by only using fuzzing. Generating random inputs is effective to find bugs. However, it cannot detect all the bugs in the program, and it does lag while finding the boundary values (Testing are usually based on a range of values. Boundary values are the values near the boundary of the range. For example, if the testing range is between 2 to 14, then the boundary values can be 2.1 or 13.9).

2.1.2 Classification

Many classifications exist for fuzzing techniques according to the target, attack vectors, fuzzing method and perspectives. Depending on the knowledge of the structure of the targets, fuzzing can be divided into black-box, white-box and grey-box fuzzing. Depending on the presence or absence of feedback, fuzzing can be divided into feedback and no-feedback fuzzing. Depending on how the input data is generated, fuzzing can be divided into generation-based fuzzing and mutation-based fuzzing.

2.1.2.1 Black-box, White-box and Grey-box Fuzzing

- **Black-box fuzzing**

  Black-box fuzzing is a form of black-box random testing, which randomly or semi-randomly fuzzes a well-formed inputs and send the inputs to the target to check how the target responses [18]. It does not need to consider the internal structure or the code of the target program. It only provides the input data to the target program and analyses the output result.

  Black-box fuzzing is generally used by the tester while they have limited knowledge of the target system or they don't have access to the code of the target. It is conceptually simple and effective. This type of fuzzing is heavily used in security testing, for example, July 2006 'the Month of Browser Bugs (MoBB)' run by security researcher HD Moore [30]. The MoBB project effectively helped to find out 100s of bugs. Fuzzing is a quick and cost-effective way to find security vulnerabilities.

- **White-box fuzzing**

  The main idea of white-box fuzzing [20] is to mix fuzzing with dynamic test generation. The whole process begins with a fixed input data. Then the algorithm symbolically executes the program and the input constraints are

collected from conditional statements encountered the execution, which are then systematically negated and solved with a solver during constraint to generate new inputs. The whole process is repeated by using a novel algorithm in order to find vulnerabilities as fast as possible [21] and increase the code coverage. The process is also called systematic dynamic test generation using both DART (Directed Automated Random Testing) and Fuzzing [22].

White-box fuzzing requires access to the source code, design specifications, detailed structure, process and run-time information of the target program. In order to better perform white-box fuzz testing, the tester needs to be a skilled programmer so that he can understand the source code. Different from the tester for black-box fuzzing, the tester for white-box fuzzing must be a developer, not just a user.

- **Grey-box fuzzing**

  Grey-box fuzzing [23] is between black-box fuzzing and white-box fuzzing. The term grey-box fuzzing is mentioned for the first time 2007 by Demott, Enbody and Punch [24]. Grey-box fuzzing is a combination solution of black-box fuzzing with other functions which depend on how the developers design it under different situations. It requires some knowledge of the target program, but not, for example, runtime coverage information. Grey-box fuzzing does not require access to the source code and requires no program analysis.

  Compared to black-box fuzzing, grey-box fuzzing can be used in a more sophisticated matter to test automatically, since grey-box fuzzing can automatically learn to create valid inputs without relying on the seed inputs (Seed input is a value used to generate random data [25]). This enables the execution of the program to get away from the branches which causes exceptions in the parser code.

  For analyzing the program, grey-box fuzzing is more generic than white-box fuzzing, since white-box fuzzing requires knowledge to understand the source code and languages syntax in order to analyze the internal of the program. When there is a combination of different languages, it is difficult to test the program automatically by using the white-box fuzzing. It takes time to understand different languages syntax. What's more, if the language is not known or recognized by the fuzzing tool, white-box fuzzing may stop.

2.1.2.2 Feedback and No-feedback Fuzzing

Feedback fuzzing gets the runtime information which is used to guide the generation of the test cases in the next loop. No-feedback fuzzing means that the fuzzing does not get any runtime information from the last run. Coverage-guided fuzzers [26] such as AFL [26], honggfuzz [27] and Syzkaller [28], rely on a corpus of sample

inputs, which contain both valid and invalid inputs for the program to be tested. For example, when testing a document library, the corpus could contain a variety of .txt, .mobi and .doc files. These fuzzers could get the information generated by the instrumentation tools to maximize the path coverage.

2.1.2.3 Generation-based and Mutation-based Fuzzing

Fuzzing can be divided into two types according to how the test cases are generated, which are generation-based fuzzing and mutation-based fuzzing [10]. The inputs of mutation-based fuzzing are modified from existing inputs, while generation-based fuzzing generates inputs from scratch [16].

Mutation-based fuzzing is also known as dumb fuzzing since it just modifies the existing input values blindly. The user does not need to understand the structure or the format of the input data. For example, they just do some changes like data addition, bit flipping which is flipping the bits randomly or in some order, data removal etc. So mutation-based fuzzing can be considered as a black-box testing. In contrast, generated-based fuzzing is also known as intelligent fuzzing. It needs to understand the file format or protocol to generate the inputs from scratch. Generation-based fuzzing can be considered as grey-box testing.

The inputs of the mutation-based fuzzing may be refused early during the fuzzing process, since the format of the mutated data may deviate too much from the original one which is expected by the target program. Generation-based fuzzing could avoid this because it uses the input data specification. Compared to generation-based fuzzing, mutation-based fuzzing always has lower code coverage. This means that mutation-based fuzzing always takes longer time to create the input data specification which may not include all the possible input formats.

2.1.3 The Fuzzing System

2.1.3.1 Basic Model

The following is a common model for the fuzzing methods. The last two modules, Monitor and Logger and Automation Module, can be considered as optional modules.

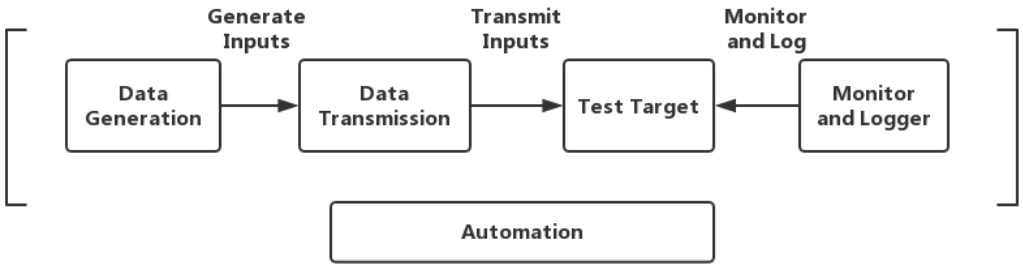

Figure 2.1: A Basic Model for Fuzzing Methods.

- **Data Generation Module** where the inputs are generated according to some algorithm or randomly. The inputs will then be sent to the target in the next step.
- **Data Transmission Module** where the inputs are transmitted from Data Generation Module to the test target.
- **Monitor and Logger** where the behavior of the target is monitored and recorded when different inputs are sent to the target.
- **Automation Module** where the testing process is automated as much as possible.

2.1.3.2 Fuzzing Phases

Most fuzzing methods follow the same phases to execute the testing. Sutton [30] has listed six common phases for fuzzing methods, which are 1) Identify target, 2) Identify inputs, 3) Generate fuzzed data, 4) Execute fuzzed data, 5) Monitor for exceptions, 6) Determine exploitability.

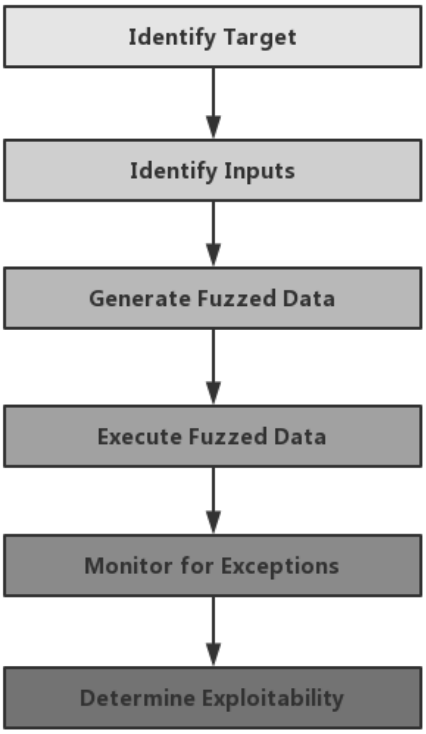

Figure 2.2: Fuzzing Phases.

- **Identify Target**

  As the first step, the target needs to be identified. The choice or design of the fuzzing method is based on the type of the target. We can identify the target using either the tester's experience or using an application. From the tester's experience, the previous history log of similar systems for the vulnerabilities or the sites of vulnerabilities aggregation can be looked up. If some developed application are used, it will take care of the choice of the target. The target can be a system, a specific file or a program, where the inputs are going to be sent.

  The choice of target is based on the risk, impact and the user base. There will be several risks which may happen in different targets. First of all, when the target is an application which receives inputs through the Internet, remote attacks, such as Man-in-the-middle attack and Morris worm, may happen. Secondly, when the target is a system which contains and deals with private information, the attacker may get access to the private information to modify and copy the data. This will violate the integrity, availability and confidentiality of the data. Last but not least, when the target is an application, which has higher privilege level than the user, this may cause privilege escalation attack.

- **Identify Inputs**

  The input is the key point of fuzzing. If the attack surface space(where the attackers can enter malicious data or perform the attacks [33]) can not be enumerated exhaustively, the inputs could not cover all the attack surface. Anything can be considered as the inputs which are sent from the data generator to the target. The inputs can be some environment variables, files, registry keys, headers in the protocol and so on.

  There are several input classes, which are described by Sutton [31]. They are command line arguments, environment variables, file formats, network protocols, memory, COM objects and Inter Process Communication. They can be divided into two groups, remote and local. Network protocols and COM objects are relevant for finding remote vulnerabilities, while the rest are relevant for finding local vulnerabilities [30].

- **Generate Fuzzed Data**

  This step is automated. Once the inputs have been identified, fuzzed data is generated. The generation of the data depends on the target and data format.

  The main idea of fuzzing is to use malicious inputs to test for the existence of vulnerabilities. This is achieved by using various kinds of inputs. The inputs should be accepted by the target, but be malicious enough to cause failure. The generation of the inputs can be prior to the testing process, or the inputs can be generated iteratively at the beginning of each test of the testing series. Usually, there are several ways to generate the inputs, which can be divided into two classes, zero knowledge testing and analysis-based testing. Generating random inputs is one example of zero knowledge testing and protocol implementation testing is an example of analysis-based testing.

- **Execute Fuzzed Data**

  After the fuzzed data is generated, this phase occurs. This phase is different for different fuzzing methods. This phase is closely connected to the input generator and the target system. The process of execution can be delivering the fuzzed data to the target system. This phase is automated.

- **Monitoring for Exceptions**

  Monitoring for exceptions is an important additional part of the procedure. Sometimes, it is useless to crash the target system by sending hundreds or thousands of malicious packets, but without recording the crashing logs. Monitoring is overlooked. However, it is a critical phase of the whole fuzzing process. The type of monitor directly determines which type of exceptions or failures can be captured during the testing. The setup of the monitor usually depends on the type of the fuzzed input and the target system. The monitor module detects the errors, reports the failure or exceptions and helps to analyze the root cause of the failure.

  A test oracle [32] is usually used as a monitor, which is a separate system from the target. The test oracle is a mechanism which can decide whether a test case passes or fails and reports the failures or exceptions when it has detected those errors. It can run as a debugger on the target system to monitor and intercept the errors and log the detailed information. It can also just do a simple liveness check by pinging the target system continuously between certain time gaps to make sure that the target system is running properly without crashing.

- **Determine Exploitability**

  Once the defects are identified from the previous phases, the form of all the defects will be sent to the tester for further analysis. This phase is usually carried out manually. Normally, it depends on the testers' security testing experience and specialized security knowledge. It could be better that the tester who is going to analyze the results of the defect, is not the tester who designed or executed the fuzzing method. From the result, the tester will determine the cause of the bugs and whether those bugs are exploitable or not. They will also judge whether the uncovered bugs can be further explored. Furthermore, together with the developers, they will decide how to fix these bugs and improve the target system.

2.2 Connected Cars

2.2.1 Overview

A connected car is a car equipped with internet access, which enables the car to communicate and share the internet access and data both inside the vehicle and outside the vehicle [36]. The inside connection can be the devices connecting to other devices, for example, mobile phones connecting to the car via Bluetooth. The outside connection could be the car connecting to other devices, networks and services, which could be other cars, OTA(over-the-air), Wi-Fi outside the vehicle, a

home or infrastructure.

The first connected car OnStar was brought to market by General Motors in 1996 in Cadillac DeVille, Seville and Eldorado [37]. OnStar was made by General Motors together with Motorola Automotive. At that time, the cellular voice connections were not reliable. The purpose of the creation of OnStar was to increase safety by a voice call to a call center which connected to the emergency responder directly, when a car accident happens. The sooner the rescue team comes, the more likely people will survive the car accident. The introduction of the first connected car led to the development of car safety systems and telematics systems.

With the development of Artificial Intelligence, communication technologies and other new techniques, people not only expect cars for transportation, reliability and safety, but also for intelligent functions and new user experience. The connected car has become one of the popular topics among automotive industry and tech companies. A Business Insider report [38] forecasts that 380 million connected cars will be driving on the road by 2021. The appearance of large numbers of connected cars with connection to different outside services from tech companies among different industries will highly increase the driving experience and potential business opportunities.

2.2.2 Types of Connectivity

Connectivity is changing the automotive industry. The technology's development is changing the way that drivers interact with the cars, how people live with their cars and improving safety during driving. There are five types of communication system, which are V2I, V2V, V2C, V2P and V2X [39].

- **V2I**

  V2I refers to Vehicle to Infrastructure, which exchanges the data wirelessly between the vehicle and road infrastructure [40]. The main propose of V2I is to capture the traffic data generated by the vehicle and provide traffic information from the road infrastructure to the vehicle. The road infrastructure includes lane markings, road signs, traffic lights and so on, which can inform the driver of the traffic information, safety, mobility and environment-related information. This is one of the stages for realizing safe autonomous driving [41].

- **V2V**

  V2V refers to Vehicle to Vehicle, which provides the vehicle with information from surrounding vehicles such as speed and position wirelessly. V2V enables cars to broadcast their information to all surrounding cars and enables drivers to receive omni-directional messages. Drivers will be notified of all the potential threats from the other cars, such as potential crashes. This

is realized by radar, ladar and cameras. The main goal of V2V is to ease traffic congestion, avoid car accidents and increase safety while driving the car [42].

- **V2C**

  V2C refers to Vehicle to Cloud, which enables the communication between the vehicles and the cloud [43]. This enables the vehicle to have access to more services which are on the cloud, such as services from telematics companies, transportation and smart home grids, which makes people's life more convenient and easier.

- **V2P**

  V2P refers to Vehicle to Pedestrian, which establishes connection between the vehicles and pedestrians. As the killing of pedestrians, bicyclists and other non-vehicles is a large percentage of car accidents, a technology which can provide safety for them becomes very important. V2P can sense and collect the information from the vehicle's surroundings [44]. Then the information will be sent to other vehicles, infrastructures, or private phones. This improves both safety and mobility on the road.

- **V2X**

  V2X refers to Vehicle to Everything, which enables vehicles to connect with everything, such as infrastructure, other vehicles, the cloud, devices, pedestrians and so on. This is a future technology. The final goal is to connect all transportation together. Not only the vehicles on the road, but also the highways, ships, trains and even airplanes [45]. This may become the next breakthrough in the transportation industry. V2X can help the driver to choose the best route to the destination, so that it ensures less traffic congestion on the road. V2X helps to enhance traffic flow, which helps to control the use of gas in a efficient way. This realizes environmentally friendly driving. V2X can control the traffic signals and broadcast all the information to the surrounding infrastructure and vehicles. This enables an efficient way to allocate resources.

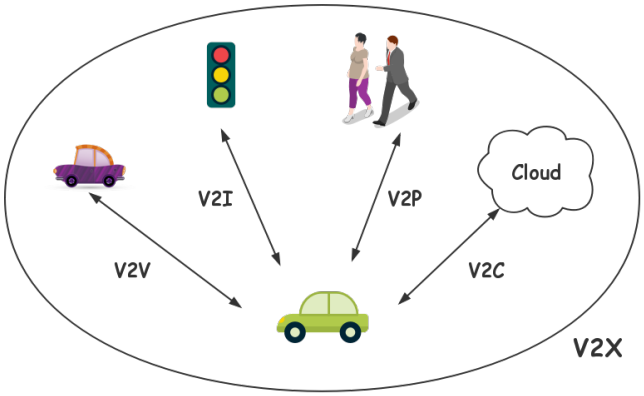

Figure 2.3: A summary of V2V, V2I, V2P, V2C and V2X.

2.2.3 Connected Car Functions

Connected cars are usually equipped with a head unit or an in-vehicle infotainment system for information and entertainment and a dashboard for providing driving information directly in front of the driver [46].

The Head Unit is the central system for sound and information, which is located in the center of dashboard. It provides a unified hardware interface, which has traditional buttons to change certain settings or play the entertainment system and a screen to show the information. It contains entertainment media such as AM/FM radio, CD and MP3 etc.

Interior buttons controlling the car settings, such as playing the music or climate control, are now gradually being removed and replaced by an in-vehicle infotainment system. Instead of a united hardware interface, it provides audio or video entertainment with both hardware and software [47]. It supports much more powerful functions than the head unit, e.g. Bluetooth connection, in-car internet and Wi-Fi. The infotainment system includes a screen, which is usually called CSD (Center Stack Display), to realize the interaction between cars and humans. This not only makes the design of the interior simpler and increases the sense of modernity, but also provide a more friendly user experience for the customers. By touching the screen, customers can find the settings more clearly and easily from the menu, compared to the traditional buttons. No more guidance books are needed before finding the correct button to set.

A dashboard, which is also called a dash instrument panel (IP), is placed directly in front of the driver. It is a control panel, which displays the vehicle operation information and instrumentation, such as the indicator lights, speedometer, odometer, instrument panel and so on [48].

- **Infotainment**

Figure 2.4: Infotainment system of Lynk & Co [1].

Infotainment, also known as in-car entertainment, is a system which consists of the audio system, navigation systems, video players, USB, Bluetooth, in-car internet and Wi-Fi [47]. The audio system consist of apps such as podcasts, internet radio and music apps like Spotify. The driver or passengers can connect their smartphone via Bluetooth to the car to make or receive phone calls and play music. The infotainment system usually supports both iOS and Android phone systems. Speech function enables the driver to free their hands during driving by just using voice commands to control; for example, the driver can say "make phone call to someone" during his driving without pressing anything. 4G Wi-Fi hotspots enable passengers to use Wi-Fi in the car.

- **Navigation**

There are two ways to use the navigation function in the car. One way is to use the navigation app on a smartphone. The other way is to use the navigation app in a built-in GPS navigation system. While using the navigation, the driver can get real-time data from the outside during driving. Real-time traffic provides the driver with the traffic situation at present. This enables the driver to choose the best way which uses the shortest time to reach his or her destination. This helps the driver avoid waiting in a long queue and to some extent relieves the traffic. There are some alerts which may also be useful during driving, for example, leave alerts which can notify drivers the best time to leave, and warning alerts which can notify the driver to slow down when the driver drives over the speed limit.

- **ADAS**

  As the purpose of the first connected car was to increase car safety, the Advanced Driver Assistance System (ADAS) is implemented and continuously improved in connected cars which target car safety and road safety [49]. Connected cars can connect to city infrastructure, navigation or phone by V2V, V2I, V2P or V2X techniques. One of the functions of the ADAS system is warning alerts. Notifications can come out automatically to notify the drivers, such as traffic jams, car collisions, wrong-way driving and speed cameras on the road. Intelligent features like automatic parking, driving monitoring systems and Emergency Driver Assistant also help the driver to avoid human error and bring better driving.

2.3 Connectivity System

The Vehicle Connectivity System is the system covering all the functionality involved in realizing connected functions. The system includes different vehicle functions, supporting services, systems and processes outside the vehicle. The system includes all the functionality of the Electrical System (On-board) which is involved in the connected functions. The Connectivity system consists of subsystems, defined domains and ECUs [59].

Vehicle connectivity system can be divided into two parts, On-board and Off-board. The On-board connectivity architecture (Figure 2.5) consists of ECUs and automotive buses [59]. The Off-board connectivity architecture usually consists of the cloud (which provides external services), infrastructure, other vehicles and so on. The detailed information for this part is listed in 2.2.2. The On-board functionality is wirelessly communicating with the Off-board functionality. This master thesis will focus on the study and implementation of the On-board connectivity system.

The interaction points of the On-board connectivity system can be classified into two parts, wired interaction points for the vehicle and access points for wireless connectivity. Wired interaction points include the OBD (On-Board Diagnostics) port and automotive buses such as Ethernet and CAN (see section 2.3.1). Wireless connectivity access points include the Cloud, Wi-Fi and Bluetooth.

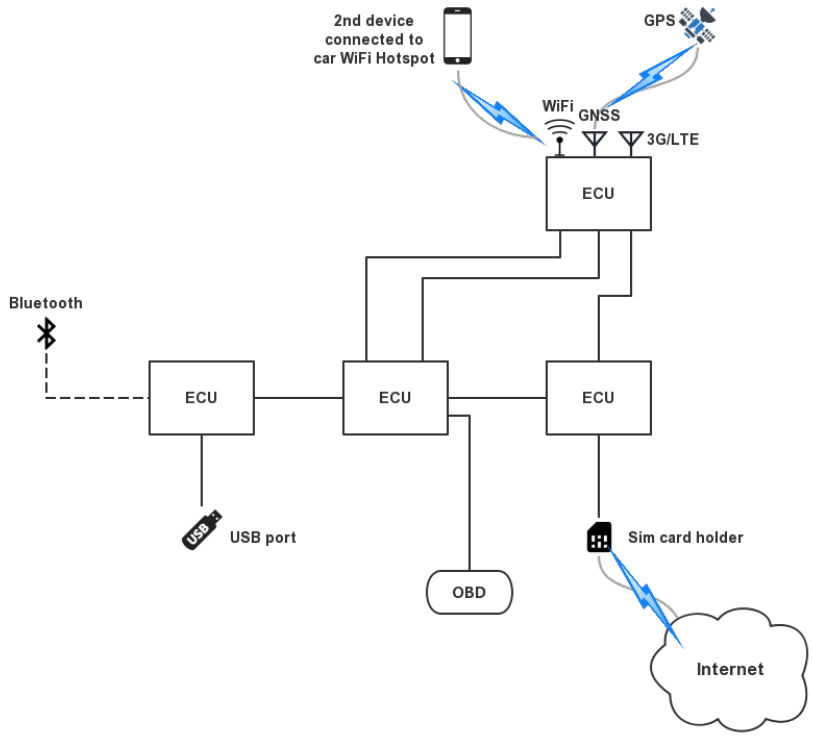

Figure 2.5: On-board Connectivity System Architecture [59].

2.3.1 Automotive Networks

Connected Cars bring fast development of intelligent and better driving experiences by realizing more functions and providing more services. A connected car can be considered as a system, which includes around 100 ECUs (Electronic control unit) and buses connecting between these ECUs [50]. ECUs are small computer modules which give control and provide different functions and services. These components build up the in-vehicle network. The entire network enables the transmission of sensor data and ECU messages smoothly and fast. There are four main types of automotive networks. In this thesis, Flex-Ray and CAN buses are used in the case study.

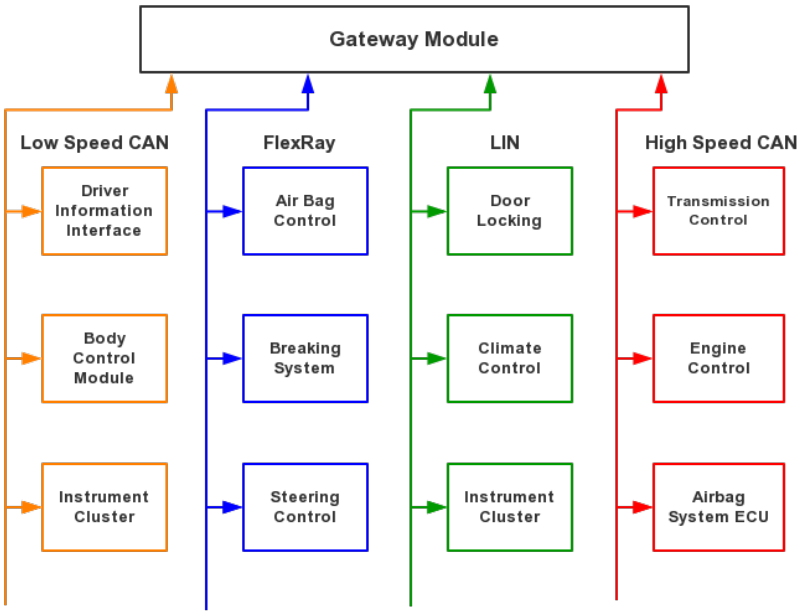

Figure 2.6: In-vehicle Network Architecture [59].

- **CAN (Controller Area Network)**

  CAN is a message-based protocol and a serial communication protocol. The architecture of CAN is Multi-Master, which typically contains 10 to 30 nodes in total [51]. By using a serial bus, messages which communicate through CAN are between individual nodes in the network. The nodes here are also called processors. If one of the nodes fails, the other nodes will continue working well without being influenced. Messages contain identifiers and are broadcast to all the connected nodes in the network, which means the messages are available to every other node. In order to make sure all the messages are transmitted correctly, there is an error checking mechanism for CAN. The frame of each message will be checked by a Cyclic Redundancy Code (CRC) mechanism [52]. The incorrect ones will be ignored by the nodes. CAN uses CSMA/CD (Carrier Sense Multiple Access/ Collision Detection) for access control. The data transmission type for CAN is asynchronous. CAN is designed for transport of short messages, which are no longer than 8 bytes. This enables CAN to be used for trigger events like temperature measurement and pressure measurement. CAN is also used for soft real-time sub-systems such as engine control, airbag, electronic gear box, driving assistants and Anti-lock Break Systems (ABS).

- **FlexRay**

  FlexRay is more reliable and transports data much faster than CAN with rates up to 10 Mbit/sec. The disadvantage is that FlexRay is much more expensive than CAN. Similar to CAN, FlexRay is also a multi-master architecture, but with up to 64 nodes. The data transmission type for FlexRay could be either synchronous or asynchronous. FlexRay uses TDMA (Time Division Multiple Access) as access control [56]. Due to the reliability and fast speed, FlexRay is usually used for the real-time sub-systems which need very fast data transport, such as brakes, steering, shift-by-wire systems and emergency systems.

- **LIN (Local Interconnect Network)**

  LIN bus has the lowest speed among all these four buses, and can only reach at 20 Kbit/sec. LIN is a cheap serial network protocol and it becomes the substitute for CAN since it is too expensive to implement CAN to connect all the ECUs in the vehicle system. Different from the other three buses, LIN is a Single-Master Architecture, which contains one master and 2 to 10 slaves [53]. LIN is a broadcast serial network. The master broadcasts the message to all the slave nodes. At most one node could reply to that message by a given message identifier. The node could be either one of the slave nodes or the master nodes, as all the messages are initiated from the master node, there is no need to check for collision [55]. LIN is usually connected with sensors, which constitute subsystems. LIN has a sleep and wake up mechanism, which save power. Due to the advantage of easy extension and lower cost, LIN can support remote and non-critical applications in a car's network. LIN network is often used for roof, seat, engine, climate, steering wheel, door and illumination modules [54].

- **MOST (Media Orient Systems Transport)**

  MOST has the highest transport speed among all these four buses and can reach 24 Mbit/sec [57], and has the largest bandwidth. It is also a multi-master architecture. MOST can be used both inside or outside the vehicle. It is often used in the multimedia area to transport the audio, video, voice, stream, navigation and data signals. For the real-time audio and video transmission applications, MOST is always chosen to be used. The access control is provided through TDM (Time Division Multiplex).

The following table shows a short summary of the features of CAN, FlexRay, LIN and MOST.

	CAN	FlexRay	LIN	MOST
Architecture	Multi-master	Multi-master	Single master	Multi-master
	10 - 30 nodes	up to 64 nodes	2 to 10 slaves	up to 64 nodes
Data Rate	1 Mbps	10 Mbps	20 Kbps	24 Mbps
Wire	Dual wire	Dual wire	Single wire	Dual wire
Message Identification	Identifier	Time slot	Identifier	
Message Transmission Type	Asynchronous	Synchronous and Asynchronous	Synchronous	Synchronous and Asynchronous
Usage	soft real time	Hard real time	Subnets	Multimedia
Latency	Load dependent	Constant	Constant	Data stream
Error Protection	CRC Parity bits	CRC Bus Guardian	Checksum Parity bits	CRC System Service
Access Control	CSMA/CA	TDMA	Polling	TDM CSMA/CA

Table 2.1: Comparison between Automotive buses [58].

2.3.2 Attack points

Attack points for the vehicle can be divided into two classifications, internal attacks and remote attacks.

2.3.2.1 Internal Attacks

As mentioned above, internal buses such as CAN, FlexRay, LIN and MOST and ECUs comprise the inside system of a modern vehicle. If the attacker can access any ECU or any bus in the internal network, he can access any part of the car, since all the ECUs are connected with each other by the buses.

An internal attack can be a software attack on an ECU. Software attacks are usually caused by software errors and flaws during the software design or implementation phase. Examples of software attacks are incorrect input or output data, buffer overflow, resource exhaustion and buffer overruns. By exploiting any vulnerabilities or weak point in the system, attackers can easily cause unexpected results in the system such as crashing the whole system. One way to accomplish an software attack on ECU is to inject malicious code to the ECU during the software downloading or updating time [60]. This will result in a malicious or invalid message being generated by the ECU. When the message is sent to the other ECUs on the internal buses, it may disturb the communication between the ECUs. Since ECUs are independent modules and can be physically disconnected from the vehicle systems, this enables the attacker to accomplish the attack more easily. This means attackers have enough time to find the vulnerabilities and perform attacks on the ECUs if they have the physical module. They can crash the whole vehicle system by just replacing it with a malicious ECU.

2.3.2.2 External Attacks

Interfaces on vehicles provide access to the internal network of the vehicles either by wired connection or by wireless access. External attacks can be divided into three types, which are indirect physical access, short-range wireless access and long-range wireless access [61].

- **Indirect Physical Access**

  There is one important point which is used as the door to the outside system by wired connection, the OBD-II port. The OBD-II port, which is also called the On-Board Diagnostic II port, provides direct access to the CAN bus and provides both diagnostics and ECU programming for the full range of the vehicle system [62]. We can connect our laptop with a "PassThru" device, which is usually through USB or WiFi. The other side of this device is plugged into the OBD-II port on the vehicle [61]. Software for different ECUs can be downloaded directly from the laptop through this cable. Diagnostics requests can also be sent and received from the laptop in this way. However, if the laptop is malicious or infected by some virus or maybe even controlled by an attacker, the whole vehicle system is under threat. From the laptop, malicious data can be sent to the vehicle system and cause abnormal behavior of the system. In our study, this is how we connected our laptop to the vehicle system to perform the whole test.

  There are other physical interfaces on the infotainment systems, such as USB, iPod and Disc, which allows users to control and play media personally by using external multimedia methods, audio players or phones. The potential vulnerabilities could be a malicious CD which includes malicious code and malicious software which is installed on the USB or phone. These will all cause attacks to happen when they are connected to the vehicle system.

- **Short-range Wireless Access**

  We define the range for short-range wireless access to be between 5 and 300 meters depending on the channel. Short-range wirelss access includes Bluetooth, Remote Keyless Entry, RFIDs, Tire Pressure Monitoring Systems, WiFi and Dedicated Short-Range Communications [61]. The vulnerabilities for these channels can be the software in the ECUs which analyse the channel message. Attackers can attack the vehicle system by just sending some malicious message through any of these channels.

- **Long-range Wireless Access**

  The range for long-range wireless access is defined to be more than 1 km. It can be divided into two categories, which are broadcast channels and addressable channels [61].

Broadcast channels: As broadcast channels are used to broadcast messages to multiple receivers at the same time, they can be controlled by the attackers to send a malicious message. Examples for broadcast channels are the Global Positioning System (GPS), 3 Satellite Radio, Digital Radio and the Radio Data System (RDS) and Traffic Message Channel (TMC) signals transmitted as digital subcarriers on existing FM-bands [61]. They are all implemented in the infotainment system of the vehicle. Malicious data can get access to the important part of the vehicle system through the internal network buses of the vehicle.

Addressable channels: Exposing the vehicle connectivity system by the remote Telematics system which provides continuous connectivity via cellular voice and data networks is the most important vulnerability for a long-range wireless attack. As these systems provide a large range of services such as diagnostics, anti-theft, crashing reports and hands-free data access, these cellular channels provides lots of opportunities for attackers. Since cellular data infrastructure has a wide range coverage, attackers can easily get access to the vehicle system from an arbitrary distance in an anonymous way. The attacks usually happen in a two-way channel which means it supports interactive control and information leakage. The attacks are individually addressable [61].

2.3.3 Attack methodologies in vehicle systems

Kosher et al.[63] introduce three attack methodologies for the vehicle system, which are packet sniffing and targeted probing, fuzzing and reverse-engineering, to perform attacks in order to test the security of an in-vehicle network. Among all the three methodologies, fuzzing is simple but very effective. It can cause significant attacks by just iteratively sending random packets and users do not need a good understanding of the hardware. Since fuzzing is more powerful than the other two methodologies, fuzzing can be an important complement to the security test method currently used at CEVT.

3
Fuzzing Tools

In this chapter, we will discuss and compare various fuzzing tools. According to the features of those tools, a suitable one will be used in this research to meet the requirement of CEVT's in-vehicle system. The last part will give the overall structure of the chosen fuzzing tool.

3.1 Open Source and Closed Source Fuzzing Tools

Between open source fuzzing tools and closed source fuzzing tools, we chose to use open source fuzzing tools. One of the reasons is that open source fuzzing tools are free. Since we need to integrate the fuzzing tool to the vehicle testing environment of the CEVT, we have to modify the source code of the fuzzing tool freely. This is what closed source tools cannot provide.

3.2 Generation-based and Mutation-based Fuzzing

In this thesis, specific protocols in the vehicle connectivity system are considered for fuzzing. Simple mutation-based fuzzing will not meet the requirements. There is a checksum in the head of each packet which takes charge of the validation and integrity. Simply modifying the head of the packet without understanding of the format of the packets and the rules of the protocols does not work. The program will refuse to accept the simply modified packets. As mentioned above, generation-based fuzzing is usually used to test specific protocols or file formats. By using generation-based fuzzing with an understanding of the protocol and packets, generating semi-valid packets (which can be accepted by the program, but can cause unexpected results) becomes a better choice. Thus, generation-based fuzzing tools were chosen to be studied further and compared.

3.3 Generation-based Fuzzing

There are three commonly used open source generation-based fuzzing tools, SPIKE, Peach and Sulley. Because Sulley is a successor of SPIKE [64], Peach and Sulley are the two fuzzing tools chosen to be compared. The better one will be chosen to be used in the following works.

3.3.1 Peach

The Peach Fuzzer Platform can perform both generation-based fuzzing and mutation-based fuzzing. Peach has several components, which are Data Modelling, State Modelling, Publisher, Agents, Monitors and Loggers etc. [66]. To run a fuzzing session, Peach Fuzzer requires a test definition called a Peach Pit [65]. The Peach Pit files are XML files and consist of data structure, type information and the relationship of the data. The Peach Pit is a configuration file written before starting fuzzing. It defines the data modelling, the process of data exchange during fuzzing, the monitoring of the fuzzing process and a publisher element which is used for connecting the data during the fuzzing. After fuzzing, the user can get a log file which records the crash information. There are several reasons for using Peach. Firstly, Peach can capture system flaws which other methods cannot. Another reason is that Peach can discover hidden vulnerabilities before the attackers, which avoid zero-day attacks [65].

There are two versions of Peach, one is Peach Fuzzer Community Edition and the other is Peach Premier [67]. Peach Fuzzer Community Edition is open source and changes largely consist of bug fixes. Peach Premier is the latest version of the Peach Fuzzer, which consists of Peach Professional and Peach Enterprise solutions. Peach Premier provides more powerful functions and a GUI interface, but it is not free.

3.3.2 Sulley

Sulley Fuzzer is a fuzzer development and fuzz testing framework, which is named after a cartoon figure in Monsters Inc. [68]. The goal of Sulley is to simplify data representation, data transmission and data monitoring. Sulley only provides generation-based fuzzing and is written in Python. Sulley can not only focus on the data generation. It also can monitor the network and maintain records, monitor the health of the target and get back to a good state by using various methods. Sulley can fuzz in parallel which can increase the testing speed.

Sulley has two monitors. One is called the network monitor, which is used to monitor the attacker part. It will capture all the network traffic between the attacker and the targets. It will also record a log of all the traffic. The other one is called the process monitor, which is used to monitor the health of the target. When a fault is found, the process monitor gets back to the recent good state and then restarts to find fault again.

3.3.3 Comparison

Peach only has one open source version, Peach Fuzzer Community Edition. However, it does not provide many functions. Although the other version of Peach Fuzzer provides strong functionality, it is expensive to buy that version. Sulley is totally open source. It is important to consider whether it is worth buying the Peach version to carry out the testing and risk that the paid fuzzing tool may not be fully

accepted in the testing case.

For Peach, all the configuration and protocols are written in XML. For Sulley, the program is written in Python and all the configurations and protocols are written in Python. It works out much better than Peach, since the configuration files used by Sulley (written in Python) are much shorter and easier to read and modify than the configuration files used by Peach (written in XML). So Sulley is more usable. However, Sulley is less mature and still has some bugs. Compared to Sulley, Peach Fuzzer is much more mature.

Based on the comparison above, Sulley was chosen as the final fuzzing method in this project. However, many bugs in Sulley needed to be fixed and code needed to be changed to fit the target before testing.

3.4 The Overall Structure of Sulley

Sulley consists of four major components: Data Generation or Data Generator, Session Management/Driver, Agents and Utilities. Figure 3.1 is from Sulley's official introduction document and gives the overall structure of Sulley:

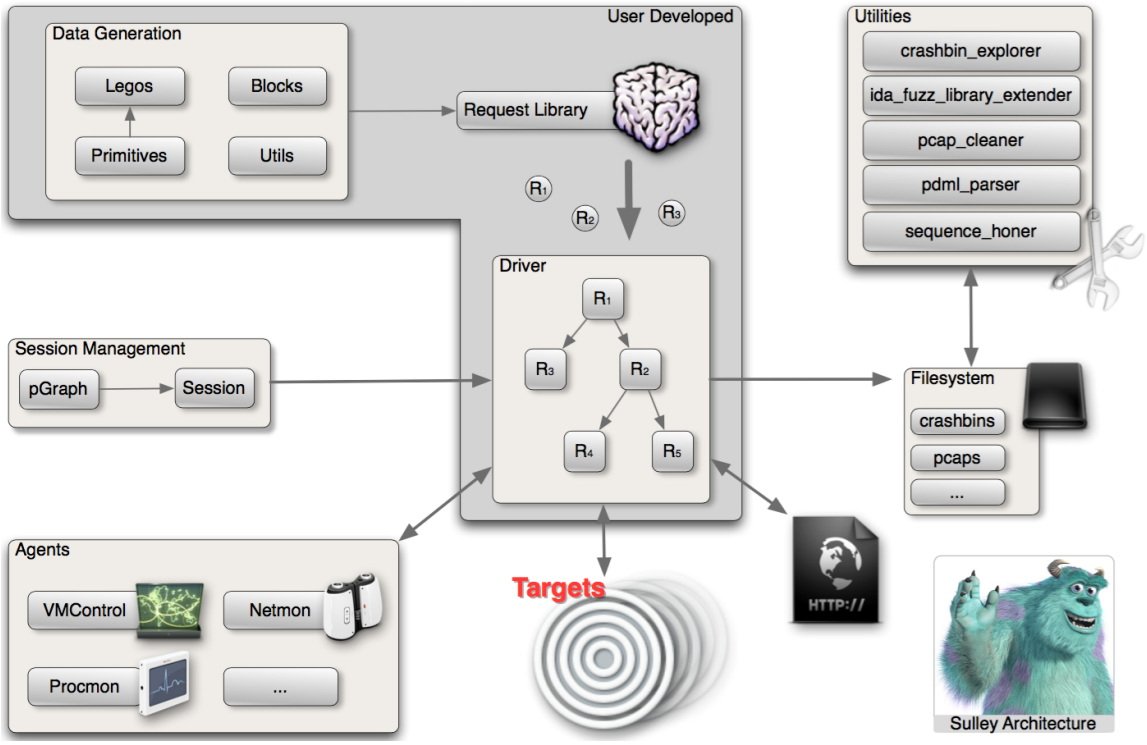

Figure 3.1: Sulley Architecture Diagram [68]

The Data Generation is used to build requests out of primitives and legos. Legos are complex types that can extend the framework. After the generation of requests, Ses-

sion Management/Driver can chain them together in a graph to form a session. The session class also provides a standalone web interface for monitoring and control.The Driver component is used to tie targets, agents and requests together. Agents are the interfaces with the targets for instrumentation and logging purposes and Utilities can perform a variety of tasks.

Sulley performs these steps:
1.Monitor the traffic or reverse some protocol parsing binary
2.Divide the target protocol into individual requests
3.Represent requests with various primitives.
4.Set up the connections between targets and agents
5.Instantiate a session and ties requests, agents and the targets together.
6.Fuzz and review results

3.4.1 Data Generation

The Data Generation is also called the Data Generator(DG), it utilizes a block-based approach to generate individual requests. This approach to protocol representation is simple, flexible and powerful. The figure below shows the components of the Data Generation:

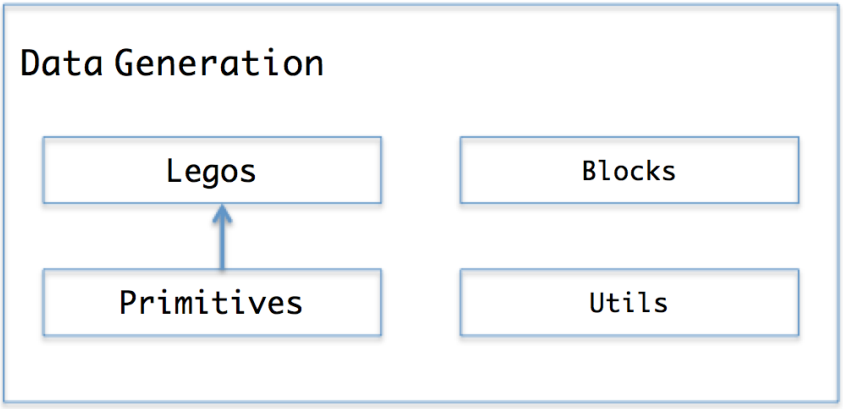

Figure 3.2: Data Generation

In Sulley, protocols are represented as a collection of primitives, blocks and block helpers. The 'name' optional argument of these elements gives users direct access to them without having to walk the stack.

Primitives include Static and Random Primitives, Integer Primitives and String and Delimiter Primitives. Static Primitives can add a static unmutating value of arbitrary length to the request and Random Primitives are used to generate random data of varying lengths. Integer Primitives have various simple types to deal with integer fields, such as s_char(), s_long(), s_double(), etc. In order to increase throughput, Integer Primitives default to a subset of potentially interesting values. Nevertheless,

you also can fuzz through the entire valid range if necessary. String and Delimiter Primitives can represent various string fields, such as E-mail addresses, usernames, passwords, etc. and support static sizes, variable padding and custom. With delimiters, strings are frequently parsed into sub-fields.

Primitives can be organized and nested within blocks. Any block can be associated with a group, encoder or dependency, which are powerful features. Grouping is used to tie a block to a group primitive to specify that the block should cycle through all possible mutations for each value within the group. Grouping is useful for representing a valid list of targets with similar argument structures. Encoder is a simple yet powerful block modifier and can connect a function with a block to modify the rendered contents of that block. Encoder also provides some functions like compression, etc.

Block helper is also an important aspect of Data Generation and has three helpers: Sizers, Checksums and Repeaters. Sizers can dynamically measure and render a block's size by taking the name of that block. Although Sulley will not fuzz size fields by default, we can enable the 'fuzzable' flag manually if necessary. Similarly, the Checksums helper can take the name of a block to calculate and render the checksum for it. Repeaters help to handle block repetitions and is useful for fuzzing multi-entry table parsers.

Sulley supports the creation of complex types called Legos and utilizes them for representing user-defined components such as IP addresses, hostnames, XML tags, etc. Legos can make fuzzing easy and more efficient.

3.4.2 Session Management

After defining various requests from the Data Generation, Sulley can tie them together in a session. Figure 3.3 shows the main components of Session Management:

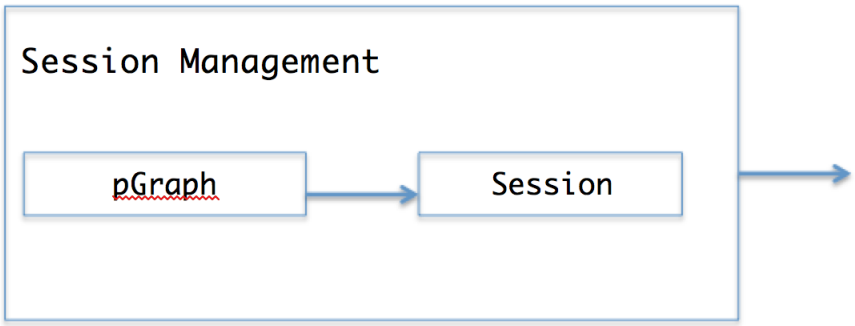

Figure 3.3: Session Management in Sulley

pGraph is a Python graph abstraction library and allows for simple graph construction, manipulation and rendering. The session class extends pGraph. One of the

major benefits of Sulley is its capability of fuzzing 'deep' within a protocol. Session class can achieve this function by linking multiple requests together in a graph. Here is an example from Sulley's official introduction document to demonstrate multiple paths and depths in a session:

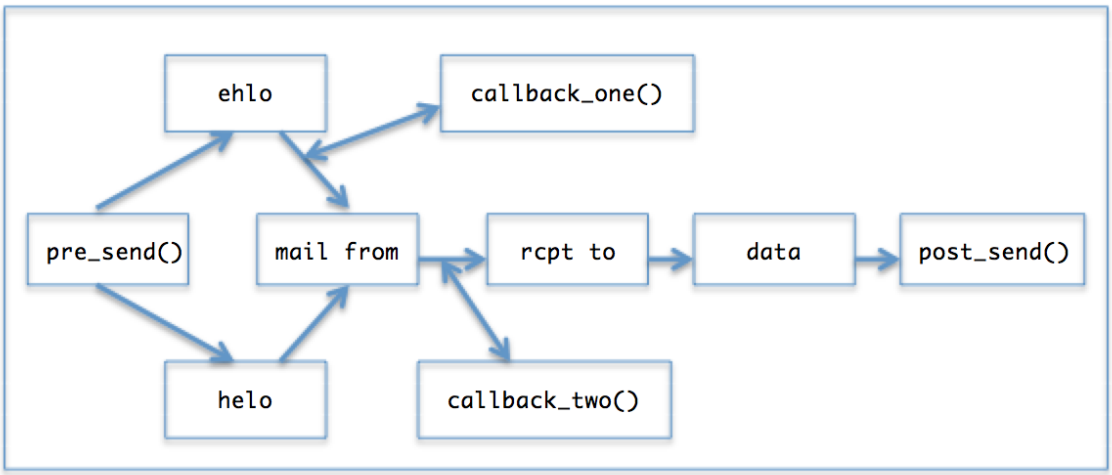

Figure 3.4: Multiple paths and depths in a session

The Session class can register pre and post send callbacks and assign a callback to each edge. Multiple network targets can be added into a session that automatically communicates with registered agents. The session class is responsible for walking the graph and fuzzing at each level.

3.4.3 Agents

Agents are a collection of various monitors and the interfaces with the targets for instrumentation and logging purposes. Figure 3.5 shows some monitors of Agents:

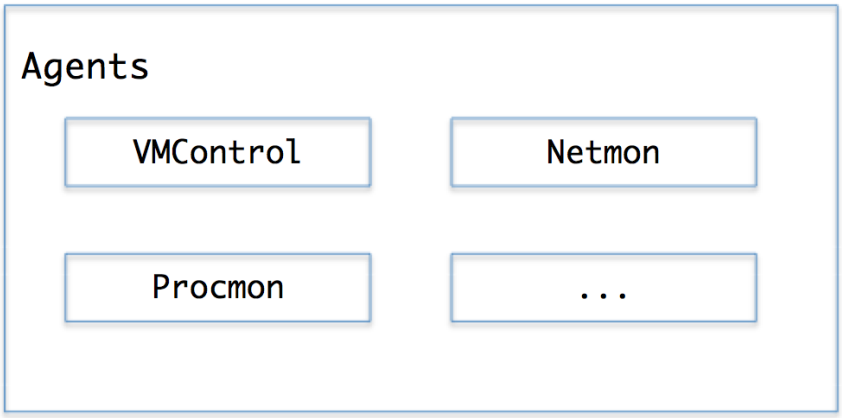

Figure 3.5: Agents in Sulley

The Netmon is a network monitor, it can monitor network traffic and save PCAPs to disk. This monitor is used to detect the network faults during the communication.

The Promon is a process monitor and needs to run on the fuzzing targets. It is responsible for monitoring the target health and detecting the faults which may occur in the target process during fuzzing. This monitor is hard-written to bind to TCP port 26002 and accepts connections from the Sulley session. After successfully transmitting an individual test case to the target, Sully contacts process monitor to determine if a fault was triggered. Detected faults are stored in a 'crash bin' file for postmortem analysis.

The VMWare control agent is forced to bind to TCP port 26003 and accepts connections from the Sulley session. This agent exposes a network API for a virtual machine instrumentation including the ability to start, stop, snapshot, etc. If a fault has been detected, Sulley can contact this agent and revert the virtual machine to a known good state.

3.4.4 Driver

The Driver component is used to tie targets, agents and requests together:
1. Import requests from the request list
2. Instantiate a session instance
3. Instantiate and bind target instances to the session
4. Tie the requests together to form a graph
5. Fuzz

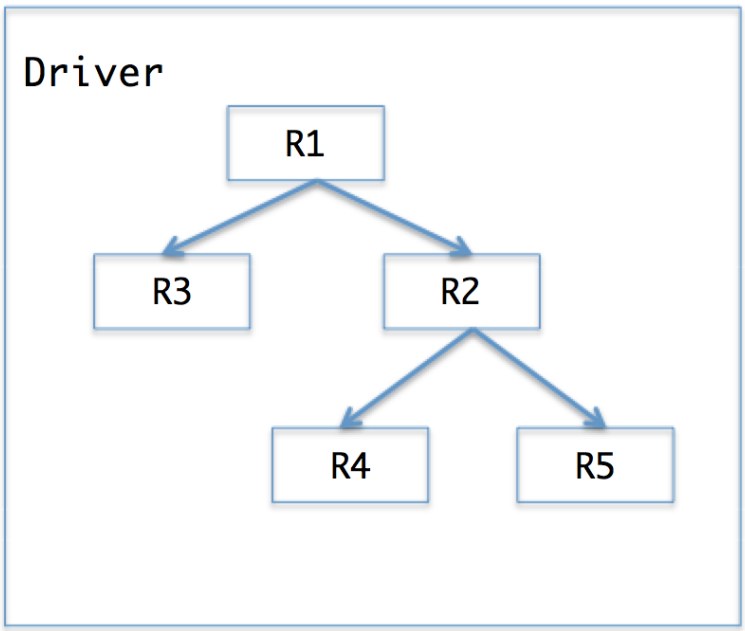

Figure 3.6: Driver in Sulley

The driver is entirely free form, though most of them will follow a simple and similar structure. It is also where edge and pre/post-send callbacks should be defined.

3.4.5 Utilities

As shown Figure 3.7, Utilities is a set of powerful tools that can carry out some specific tasks during the fuzzing process.

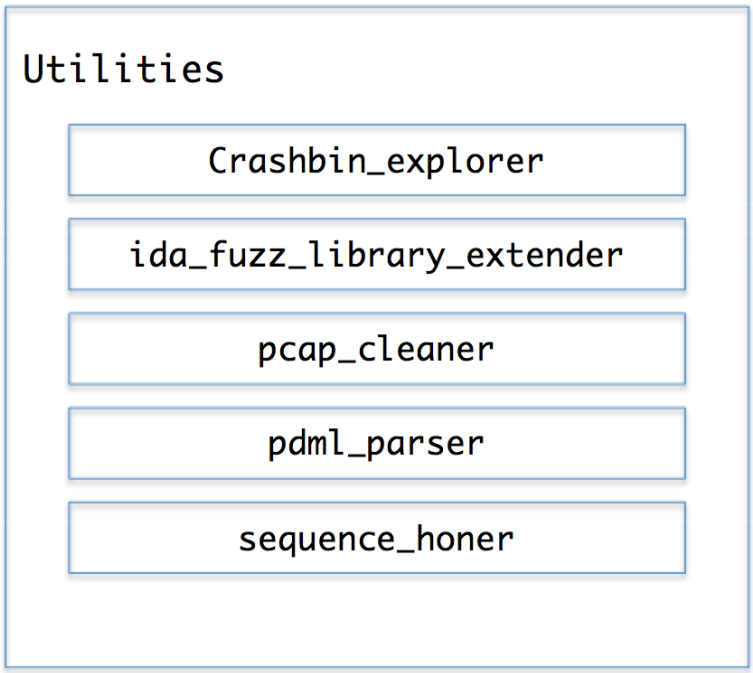

Figure 3.7: Utilities in Sulley

Here we give a brief introduction of three useful utilities:

1. The Crashbin_explorer is used to retrieve saved crash dumps and view the test case that caused a fault. It can list every location where a fault occurred.

2. The Pcap_cleaner iterates through a crashbin file and removes any PCAPs which are not associated with a fault. This is used to save the disk space.

3. The Pdml_parser can convert a PDML dump from Wireshark into a Sully request. This utility provides more convenience when cooperating with Wireshark.

4

Closed System Fuzzing

4.1 Test Targets

Because of the security agreement with CEVT, some key information is hided in this chapter. If you want to get more information please contact with CEVT.

From CEVT's internal documents, we found an interesting private protocol named the Internal Communication Protocol (ICP). ICP is a control channel for IP-based networks and it is designed for in-vehicle communication between IP-based ECUs. This means when applications want to communicate within the vehicle network between IP-based ECUs, they need this control channel to exchange data. This protocol is implemented on the application layer. It supports both TCP and UDP. While the client and server receive the data, they will validate whether there is any error in the data, header, length, payload etc. according to the ICP. If there is no error in the packets, then the packets will be accepted. This private protocol is still in the testing phase so it is a good fuzzing target. Section 5.2 'Testing strategy' gives more detailed information about ICP.

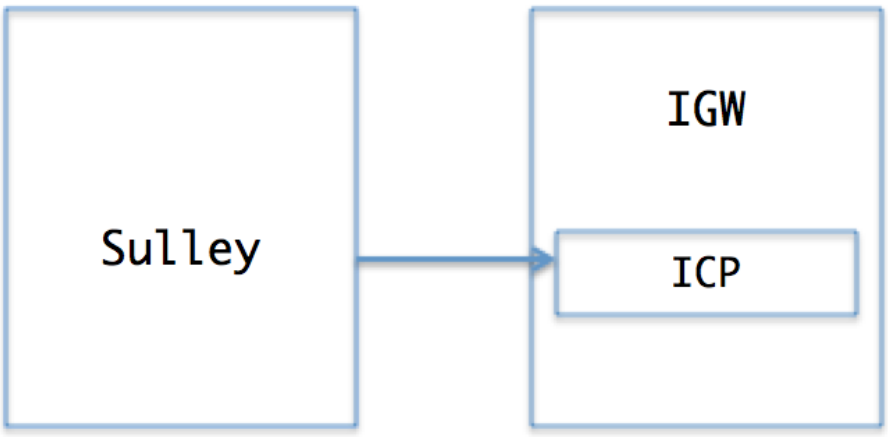

Figure 4.1: Fuzzing Target

After selecting ICP as the fuzzing target, we needed to find a suitable ECU as the hardware target. Finally the Internal Gateway (IGW) was chose as the fuzzing

hardware. Firstly, the ICP protocol running on the IGW can provide various services for internal communication. Secondly, the IGW can help the communication between different ECUs. Because of these two features, at last we chose the IGW as the hardware target.

4.2 Closed Systems

Each ECU works like a microcomputer and there are many services running on ECUs. Because of those features, the security issues, including the network and the system security, of ECU are crucial for those developers. So every ECU is designed as a closed system, which means that people cannot command the ECU to perform any function or try to modify the system settings through the outside network ports, such as the Ethernet or Wi-Fi. This feature can guarantee the securities of UCEs but it also put up a barrier for the fuzzing tools.

Some fuzzing tools need to run specific testing scripts on the fuzzing targets. For example, Sulley needs to run various Agents on the target hardware. When fuzzing an ECU, we may need to do some modification of fuzzing tools to guarantee the success of the fuzzing process. Sulley can pretend to be an ECU and send semi-valid data to testing targets and monitor their reactions. So fuzzing on an ECU is like performing a network attack on the target. What we concern is how to collect the information from the target ECU during the fuzzing process.

4.3 Implementation

After selecting Internal Communication Protocol (ICP) as the fuzzing target and the Internal Gateway (IGW) as the testing hardware, the next task is to modify the code of fuzzing tool Sully to implement fuzzing on CEVT's in-vehicle connectivity system. When applying fuzzing on a target, two main parts of Sulley need to be modified: the Data Generator (DG) and the Agents. Because the Agents are actually a collection of various monitors, we use Monitor (MO) to represent the Agents in the following contents. According to the pre-defined block construction, Data Generator can automatically create various semi-valid packages according to the predefined value range of those fuzzing blocks. We also can set the value of those blocks manually to ensure the created semi-valid package is the exactly one that we need. Data Generator can calculate the value of some fixed blocks automatically, such as length or CRC, so there is no worry that the semi-valid packages would be rejected by the header check of those blocks. Monitor part should run on the target hardware to detect the reactions of fuzzing target. Monitor can detect the process state and the network state of the fuzzing target. The figure below gives a example of how Sully works when fuzzing the FTP protocol running on a FTP server:

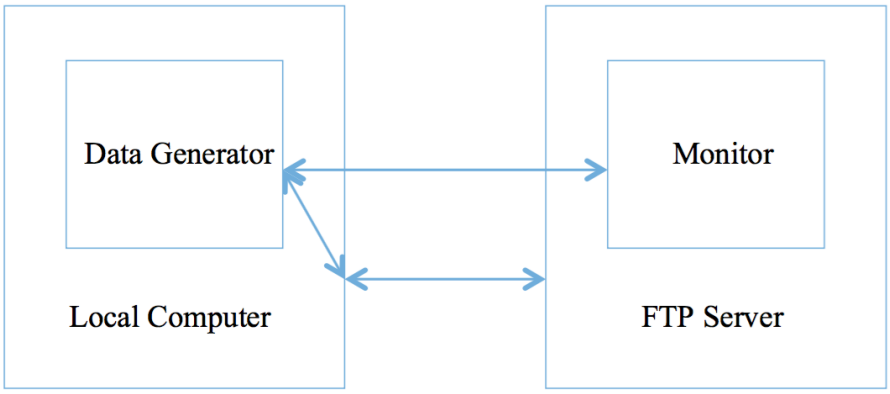

Figure 4.2: Fuzzing on a FTP server

At the beginning, the DG, which running on the local PC, sets up a TCP connection with the MO running on the FTP server. When the fuzzing starts, DG creates various semi-valid packages and sends them to the FTP server. At the same time, MO can detect the process state and network state of the FTP server. The process state includes the FTP protocol process and other pre-selected system processes. The network state records if the FTP server responds with some expected or unexpected packages. If MO detects those specific packages it will record them and send an alert to DG. The detailed information of how to perform fuzzing on a FTP server will be given in the following contents.

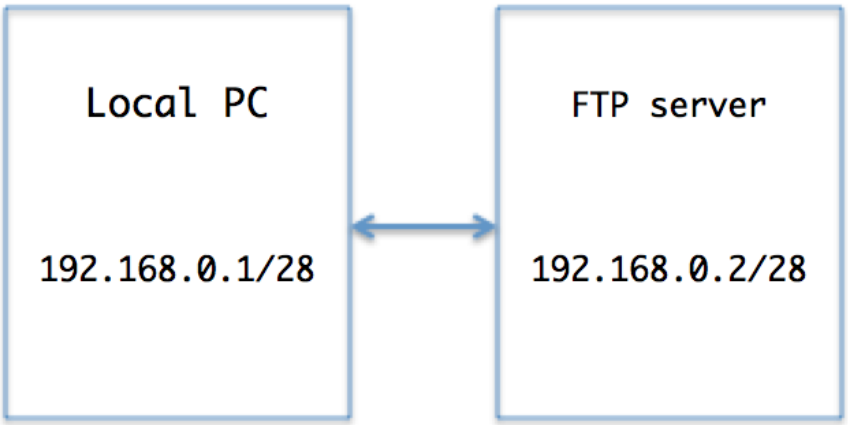

Figure 4.3: Setting up IP address

The first important step is that Sulley needs to set up a TCP communication between local computer and the Monitors running on the FTP server. In general case, a FTP server work on an internal network and have a private IP address. It can communicate with public networks through some techniques, such as NAT etc. If we

want to build a TCP communication between Sulley and the FTP server, we have to connect the local computer with the FTP server directly. After that we need to set a private IP address for the local computer and ensure this IP address is in the same subnet with the FTP server. For example, if the IP address of the FTP server is 192.168.0.2/28, the IP address of local computer can be set to 192.168.0.1/28. The Session Management component of Sulley can help control the TCP communication between Sulley and its Monitors.

Figure 4.4: Divide the protocol into blocks

Upon setting up a TCP communication with the FTP server, Sulley can try to create various semi-valid messages. Every protocol, including FTP, is structured and it can be divided into different blocks. Those blocks all have a fixed length so they are given a value range. Different values can represent different functions, such as various services or operations that the protocol can provide.

Some blocks can work independently, such as the length and the protocol version blocks of FTP. These blocks usually have a fixed value and if the value of the received message cannot match this correct value, the message will be discarded simply. So this type of blocks is not a good fuzzing target. Other blocks need to work together so that they can provide various services for different requests. For example, FTP server sets up different sessions for its clients respectively. The values of those blocks are dependent with each other, so the value combination of them can be used to create semi-valid messages. Fuzzing can check the reactions of the fuzzing target when it meets some conflicts in those value combinations. After the generation of those semi-valid messages, Sulley send them one by one to the FTP server through direct connection network between them.

As mentioned before, Sulley use the component Agents to collect the information from the fuzzing target. Agents actually are a set of various monitors. When fuzzing on a FTP server, the Process Monitor and the Network Monitor need to run on the FTP server.

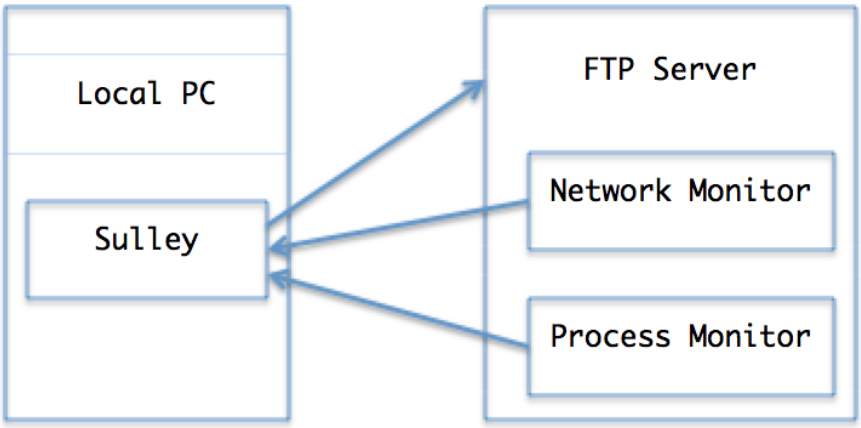

Figure 4.5: Monitors of Sulley

Process Monitor can gather the information of various processes running on the server. For example, it can monitor the state of FTP process according to the PID and other process information. If those processes have some problems, Process Monitor can record it send an alert to Sulley. Network Monitor can be used to check the messages received or sent by the FTP server. We can set up a message list for the Network Monitor and it can filter the messages according to this list. When meeting a required message on that list, the Network Monitor will record it and notify the Sulley. Network Monitor also can be used to detect the network failure.

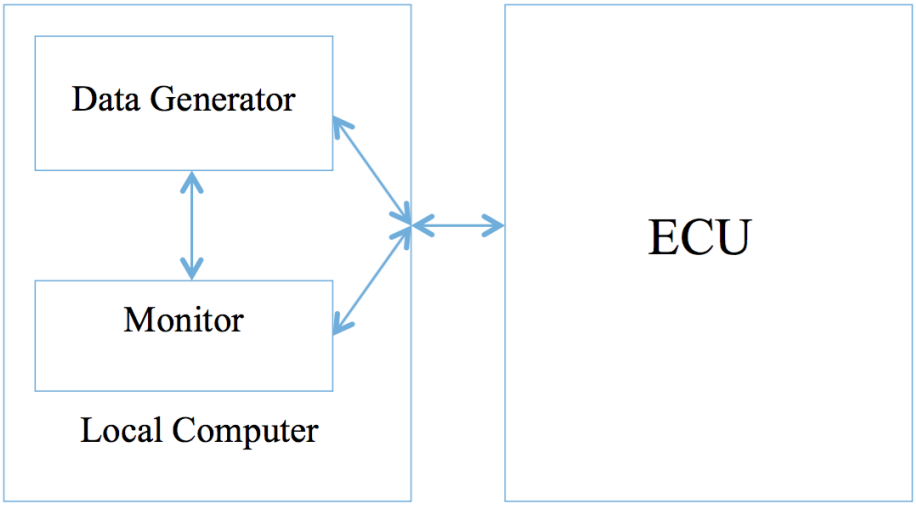

Figure 4.6: Fuzzing on an ECU

According to the example of fuzzing on a FTP server, the first work of fuzzing the ICP protocol is to analysis its blocks structure. The value combination of those related blocks could be used to generate the semi-valid messages. Since each ECU is a closed system, those Monitors cannot run on the ECU. We need to modify the

Monitor part and make sure it run on the local computer and detect the state of ECU according to the received messages from the ECU.

5

Case Study: CEVT Connected Cars

Because of the security agreement with CEVT, some key information is hiden in this chapter. Please contact with CEVT for more detailed information.

5.1 Fuzzing topology

Before applying fuzzing, we need to set up a fuzzing topology that can satisfy the requirements of fuzzing process and information collection. Fuzzing is used to test any system with structured input. It does not need to understand the structure of the system or the hardware. The only thing that fuzzing need to study is the structured input, so it is always used to test security issues of functions during the software development or various protocols. Fuzzing can divide the structured input into blocks and set up requests for those blocks, and then assemble those requests to create semi-valid messages.

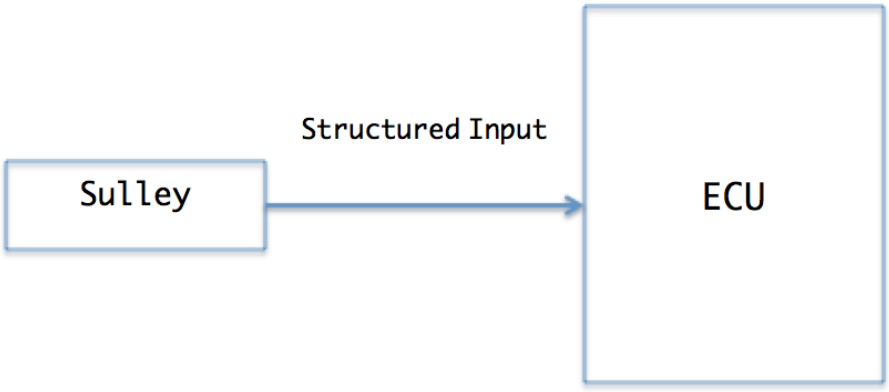

Figure 5.1: Fuzzing on systems with structured input

According to the protocol structure of ICP, the Data Generator (DG) component of Sulley can help generate semi-valid messages automatically. DG also can help calculate the value of some blocks, such as the CRC. This can improve the efficiency

of the fuzzing work.

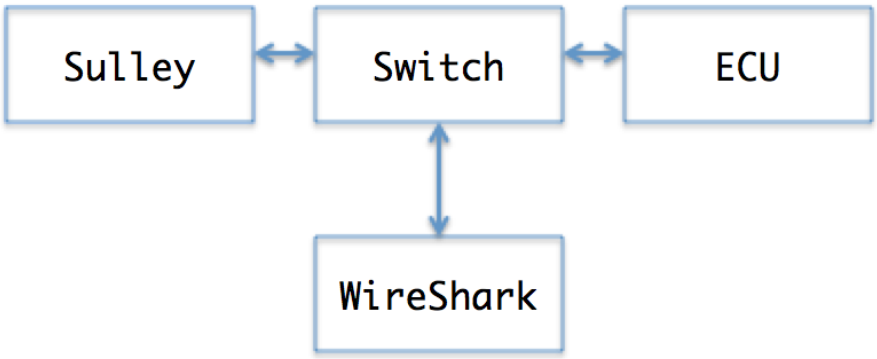

Figure 5.2: Fuzzing topology

It is easy of find that the direct connection between Sulley and ECU can meet all those requirement of fuzzing process. The Monitor part works on the local computer and also can collect information from this direct connection. But we still need more information to help the further analysis after fuzzing, so we put a switch between the Sulley and the target ECU. Through this switch, another computer can collect some key information by using some network tools, such as WireShark. The figure 5.2 gives a simple topology of fuzzing on an ECU.

5.2 Test Strategy

After all preparations of fuzzing, now we need to make a fuzzing strategy. The key part of the fuzzing strategy is how to create those semi-valid messages. First we need to study the ICP protocol. There is a lot of information in the protocol design document, such as the protocol structure, the functions of different blocks, etc. Those blocks that can corporate with each other can be the good fuzzing targets. Other blocks with fixed values cannot be taken into the consideration. Because delivering those semi-valid messages to the target is like to perform a network attack, we also need to discuss with the system engineer to avoid unexpected damages on the hardware.

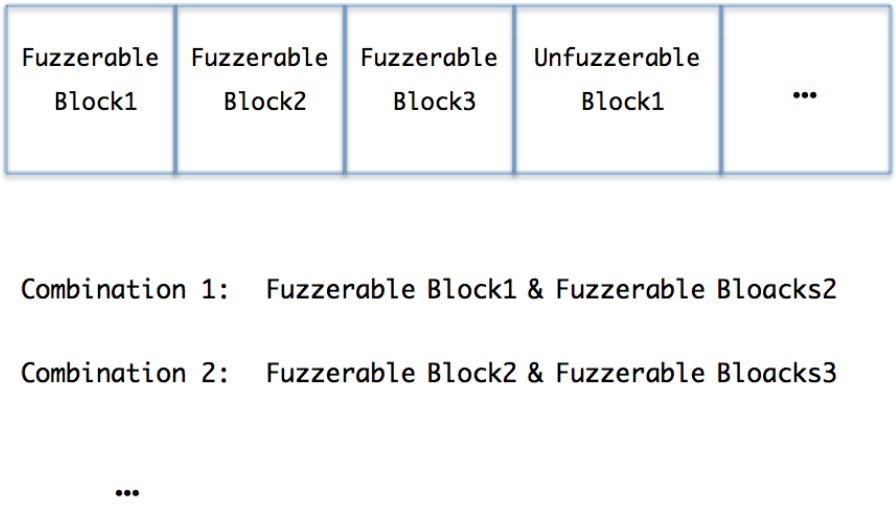

Figure 5.3: Selecting fuzzerable blocks

After selecting those fuzzerable blocks, we need to study how does the ICP handle those received messages. As shown in the figure 5.4, any protocol handles received messages according to a universal rule. The first step is to check some normal information, such CRC, protocol version, security information if existing, etc. In the second step, the protocol needs to check the information in header part. Because the structure of headers differentiates from various protocols, the detail check process is different. At last, the protocol needs to check the payload part if it is required.

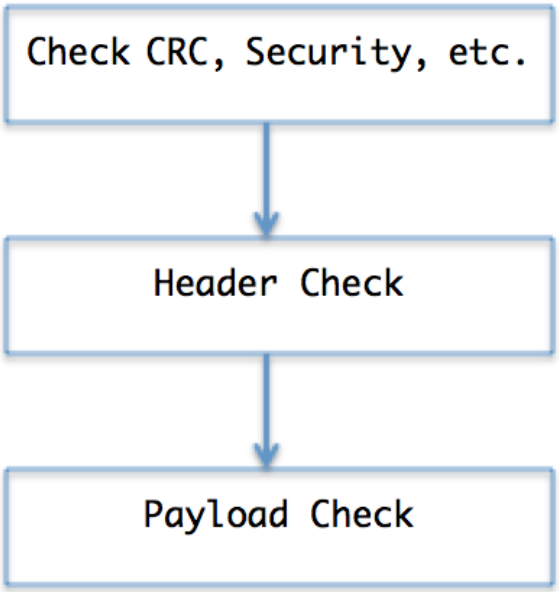

Figure 5.4: Message check flow

Analyzing how does the ICP handle the received messages can help us make a strategy for information collection during the fuzzing process. Sulley can understand how does the ICP react after receiving those semi-valid messages. So the Monitor part of Sulley need to filter the messages sent by the ICP and detect the useful messages according to this strategy.

5.3 Results

After making fuzzing strategies for data generation and information collection, we can perform fuzzing on the ECU easily. The table 5.1 gives an example of the chart for recording the combinations of those fuzzerable blocks and fuzzing results. Because there are a lot of block combinations, this chart can help avoid repetition work.

Combination of Blocks	Fuzzing Result
Block1 & Block2	Pass
Block2 & Block3	Pass
Block1 & Block2 & Block3	Pass
…	…

Table 5.1: Fuzzing results of blocks combinations

Except the block combinations, we also can set up some specific scenarios for fuzzing. For example, how does ICP react when receiving two identical request messages? We also need to create a list for the scenarios fuzzing.

Fuzzing Scenarios	Fuzzing Results
Scenario 1	Pass
Scenario 2	Pass
Scenario 3	Pass
…	…

Table 5.2: Fuzzing results of specific scenarios

Finally, we can start fuzzing and collect the information from the ECU. Because the security agreement with CEVT, the detailed fuzzing results cannot be shown in this public thesis, you can contact with CEVT to get this information.

6
Discussion

6.1 In-vehicle Security

A model known as CIA triad [69] is used to evaluate information security. The elements of the model are confidentiality, integrity and availability, which are considered as three most important part of security. In the following, potential vulnerabilities and countermeasures discussed based on the three security properties.

- **Confidentiality**

  Confidentiality is similar to privacy. To ensure confidentiality, sensitive data or information need to be prevented from the unauthorized person, organization or process [70]. Access to the sensitive data or information need to be restricted to the authorized readers.

  In in-vehicle network, since all the messages are broadcast on the automotive network, every ECU can hear these traffic. This causes a problem that attackers can get information by monitoring or hearing on one ECU. So it is important to have message identifier which can filter the traffic and data encryption to encrypt important data.

  To filter the traffic on ECUs, setting stateless firewall by only checking the IP/MAC address or the port numbers is not enough. It is important to set up a stateful firewall in the appropriate place. Stateful firewalls can be used to keep track of the state of the connections and ensure that all the passed packets exchanging in the connections are legitimate packets and match a known active connection. However, the disadvantage is that setting stateful firewalls increase the cost of the vehicle system. So choosing the appropriate places and number of stateful firewalls become an essential issue.

  In order to get access to the ECU to do some operations, such as updating the software, some security access keys should be provided to get permission from the ECU to enable the functions and avoid unwanted actions. To keep the confidentiality of the key, it should be controlled well that only limited people can get access to the key. Otherwise, it will become a security vulnerability.

- **Integrity**

  Data integrity is to maintain the accuracy, consistency and completeness of data during its entire life cycle [71]. Mechanism need to be applied on the in-vehicle networks to ensure that data cannot be modified or altered by unauthorized person. On CAN bus, CRC referring to Cyclic Redundancy Check is used to check the integrity of data. However, CRC is not secure enough, which becomes a potential vulnerability. User access controls and data permission can also be two countermeasures.

- **Availability**

  Availability means that the information should be available when it is needed. It is important to maintain all the hardware to guarantee that data can be accessed. Some recovery mechanism need to be set up, in case the data loss or interruptions by unexpected events, such as the natural disasters. Firewalls and proxy need to be set up on the ECUs in order to prevent the unreachable data and shutdown caused by denial-of-service (DoS) attacks. Dos can happen on CAN bus because of the Fault Confinement, which is function that could detect the fault nodes and disconnect these nodes from the network. It might happen that a good ECU is forced to shut down from the bus by an attacker.

6.2 Improvement on the Fuzzing Tool

There are some modifications which are needed to be performed on the Sulley before starting fuzzing. Since every ECU is a closed system, the Monitor(MO) of Sulley was modified to run on the local computer. The new Monitor listened on the network and waited for the keep-alive message from fuzzing target. When the keep-alive message disappeared from the network, the Monitor stopped the fuzzing process and output the last semi-valid message sent to the fuzzing target. It is a type of passive detection and if there is a network delay, the Monitor cannot give the exact message that causes the system problems. We have to seek the correct semi-valid message manually from the message list and try them one by one. Fortunately, some CEVT's private testing tools can detect system errors immediately so a more powerful Monitor could be developed from those testing tools in future work.

7
Conclusion

This thesis investigates how well fuzzing can be used to learn vulnerabilities in the CEVT's in-vehicle connectivity system. In order to achieve this goal, first we chose a suitable fuzzing tool to perform fuzzing on the CEVT's system. After the comparison of different fuzzing tools, Sulley was selected to be implemented on the system. Then we found an interesting private protocol Internal Communication Protocol (ICP) as a fuzzing target because this private protocol is still in the testing phase. The Internal Gateway (IGW) was also chosen to be a fuzzing hardware target. After the selection of fuzzing targets and fuzzing tools, we analyzed the protocol structure and prepared a careful fuzzing strategy for ICP. Because ECU is a closed system, we modified the Sulley to guarantee that fuzzing results can be collected correctly during the fuzzing process. After all these preparations, we applied fuzzing on CEVT's in-vehicle system and finally found some security vulnerabilities that were not found during the normal tests before. In future work, the integration between the fuzzing tool and CEVT's testing system should be done. Some promotions on Sulley's Monitor part also can help the fuzzing process.

Bibliography

[1] LYNK & CO offical website, https://www.lynkco.com.cn[Online]

[2] Brown, Eric (13 September 2016). "Who Needs the Internet of Things?". Linux.com. Retrieved 23 October 2016.

[3] Volkswagen, annual report 2017, "The car will soon be our second home.", https://annualreport2017.volkswagenag.com/magazine/connect-with-consumers/the-car-will-soon-be-our-second-home.html[Online]

[4] Auto Connected Car News, "Definition of Connected Car – What is the connected car?", http://www.autoconnectedcar.com/definition-of-connected-car-what-is-the-connec

[5] Tesla, Tesla Model 3, https://www.tesla.com/model3[Online]

[6] Keen Security Lab of Tencent, Car Hacking Research: Remote Attack Tesla Motors, 2016, http://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/

[7] Cyber attacks in connected-cars, https://blog.appknox.com/cyber-attacks-in-connected-cars/[Online]

[8] Miller C. and Valasek C. Remote exploitation of an unaltered passenger vehicle. Black Hat USA, 2015. Las Vegas.

[9] Kleberger P, Olovsson T, Jonsson E. Security aspects of the in-vehicle network in the connected car[C]//Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE, 2011: 528-533.

[10] Oehlert. Violating Assumptions with Fuzzing. IEEE Computer Society, Mar./Apr. 2005, pp. 58-62.

[11] China Euro Vehicle Technology AB, an innovation centre for Geely Group, http://www.cevt.se/[Online]

[12] Chalmers Library, How student work is published at Chalmers, http://www.lib.chalmers.se/publicering/att-publicera/studentarbeten/[Online]

[13] Fuzzing Wikipedia, https://en.wikipedia.org/wiki/Fuzzing/[Online]

[14] Kaner, Cem; A Course in Black Box Software Testing, 2004

[15] Barton P. Miller, Louis Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. Commun. ACM, 33(12):32–44, December 1990.

[16] Fuzzing – Mutation vs. Generation, http://resources.infosecinstitute.com/fuzzing-mutation-vs-generation/#gref/[Online]
[17] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. IEEE transactions on software engineering, 41(5):507–525, 2015
[18] Miller, Barton P., Louis Fredriksen, and Bryan So. "An empirical study of the reliability of UNIX utilities." Communications of the ACM 33.12 (1990): 32-44.
[19] Sutton M, Greene A, Amini P. Fuzzing: brute force vulnerability discovery[M]. Pearson Education, 2007.
[20] Godefroid P, Levin M Y, Molnar D A. Automated whitebox fuzz testing[C]//NDSS. 2008, 8: 151-166.
[21] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani. Synergy: A new algorithm for property checking. In Proceedings of the 14th Annual Symposium on Foundations of Software Engineering (FSE), 2006.
[22] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation), pages 213–223, Chicago, June 2005.
[23] Böhme M, Pham V T, Roychoudhury A. Coverage-based greybox fuzzing as markovchain[J]. IEEE Transactions on Software Engineering, 2017.
[24] Jared DeMott. Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing. In BlackHat and DefCon, 2007.
[25] Michal Zalewski. Pulling jpegs out of thin air. https://lcamtuf.blogspot.se/2014/11/pulling-jpegs-out-of-thin-air.html/[Online]
[26] Zalewski, Michal. "American fuzzy lop." 2017.
[27] Google Honggfuzz. 2017, https://github.com/google/honggfuzz/[Online]
[28] Google Syzkaller. 2017, https://github.com/google/syzkaller/[Online]
[29] Oehlert, Peter. "Violating assumptions with fuzzing." IEEE Security and Privacy 3.2 (2005): 58-62.
[30] Michael Sutton, Adam Greene, and Pedram Amini, Fuzzing: Brute force vulnerability discovery, Addison-Wesley Professional, 2007.
[31] Michael Sutton, Fuzzing - brute force vulnerability discovery, Presentation, RE- CON conference, Montreal, Canada, Friday June 16th 2006.
[32] Howden, W.E. (July 1978). "Theoretical and Empirical Studies of Program Testing". IEEE Transactions on Software Engineering. 4 (4): 293–298. doi:10.1109/TSE.1978.231514.
[33] Wikipedia, Attack Surface https://en.wikipedia.org/wiki/Attack_surface/[Online]
[34] Stein, Ralph (1967). The Automobile Book. Paul Hamlyn.
[35] Fowler, H.W.; Fowler, F.G., eds. (1976). Pocket Oxford Dictionary. Oxford University Press. ISBN 978-0198611134

[36] Elliott, Amy-Mae (25 February 2011). "The Future of the Connected Car". Mashable. Retrieved 22 July 2014.
[37] "Definition of Connected Car: What is the connected car? Defined". AUTO Connected Car. South Pasadena, California, United States: Apropos. Retrieved 22 July 2014.
[38] Greenough J. The connected-car report: The transformation of the automobile. Business Insider Intelligence, 2016.
[39] Staff, CAAT. "Automated and Connected Vehicles". autocaat.org. Retrieved 2018-05-01.
[40] "Intelligent Transportation Systems - Vehicle to Infrastructure (V2I) Deployment Guidance and Resources". www.its.dot.gov. Retrieved 2018-05-01.
[41] 3M, What is Vehicle-to-Infrastructure (V2I) Communication and why do we need it?
https://www.3m.com/3M/en_US/road-safety-us/resources/road-transportation-safety-center-blog/full-story/~/what-is-vehicle-to-infrastructure-v2i-communication-and-why-do-we-need-it/?storyid=021748d7-f48c-4cd8-8948-b7707f231795/[Online]
[42] jean.yoder.ctr@dot.gov (2016-10-26). "Vehicle-to-Vehicle Communication". NHTSA. Retrieved 2018-05-01.
[43] "Connected Vehicle Cloud Platforms, ABI Research". www.abiresearch.com. Retrieved 2018-05-01.
[44] "Intelligent Transportation Systems - Vehicle-to-Pedestrian (V2P) Communications for Safety". www.its.dot.gov. Retrieved 2018-05-01.
[45] "Future of Infrastructure: Vehicle-to-X (V2X) communication technology" (PDF). Siemens. Retrieved 1 May 2018.
[46] McKinsey and Company "Connected car, automotive value chain unbound" (PDF). Retrieved 20 April 2017.
[47] Wikipedia, In-car Entertainment,
https://en.wikipedia.org/wiki/In-car_entertainment/[Online]
[48] Wikipedia, Dashboard,
https://en.wikipedia.org/wiki/Dashboard/[Online]
[49] Wikipedia, Advanced driver-assistance systems,
https://en.wikipedia.org/wiki/Advanced_driver-assistance_systems/[Online]
[50] Schmid M. Automotive bus systems. Atmel Applications Journal: Issue 6, 2006.
[51] Renesas Electronics Corporation, “Introduction to CAN”,April 2010,
http://www.renesas.com/[Online]
[52] Daniel Mannisto, Mark Dawson: “An Overview of Controller Area Network (CAN) Technology”,April 2012,
http://www.parallax.com/dl/docs/prod/comm/cantechovew.pdf/[Online]
[53] Mary Tamar Tan, Brian Bailey, Han Lin. "Microchip AN2059: LIN Basics and Implementation of the MCC LIN Stack Library on 8-Bit PIC Microcontrollers".

[54] Wikipedia, Local Interconnect Network, https://en.wikipedia.org/wiki/Local_Interconnect_Network/[Online]
[55] "Lin Concept". LIN Overview. LIN Administration. Retrieved 28 October 2011.
[56] "How FlexRay Works". Freescale Semiconductor. Retrieved 21 March 2014.
[57] Wikipedia, MOST Bus, https://en.wikipedia.org/wiki/MOST_Bus/[Online]
[58] LIN vs CAN vs FlexRay vs MOST-Difference between LIN, CAN, FlexRay, MOST, http://www.rfwireless-world.com/Terminology/LIN-vs-CAN-vs-FlexRay-vs-MOST.html/[Online]
[59] Volvo Car Corporation, Complete Connectivity System
[60] Wolf, M., Weimerskirch, A., Wollinger, T.: State of the Art: Embedding Security in Vehicles. EURASIP Journal on Embedded Systems 2007, Article ID 74706, 16 (2007)
[61] Stephen Checkoway, Damon McCoy, Brian Kantor et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. USENIX Security, 2011
[62] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In D. Evans and G. Vigna, editors, IEEE Symposium on Security and Privacy. IEEE Computer Society, May 2010.
[63] Karl Koscher, Alexei Czeskis, Franziska Roesner et al. Experimental Security Analysis of a Modern Automobile. University of Washington, 2010
[64] Awesome - Fuzzing, https://github.com/secfigo/Awesome-Fuzzing/[Online]
[65] PeachTech, https://www.peach.tech/about/faq/[Online]
[66] Peach-fuzzer, https://www.peach.tech/products/peach-fuzzer/[Online]
[67] Evolution of Peach: From Project to Product, Peach Fuzzer, http://www.peach.tech/wp-content/uploads/Evolution-of-Peach_From-Project-to-Product-29July2015.pdf[Online]
[68] Sulley: Fuzzing Framework, Sulley Manual, http://www.fuzzing.org/wp-content/SulleyManual.pdf[Online]
[69] Perrin, Chad. "The CIA Triad". Retrieved 31 May 2012.

[70] Beckers, K. (2015). Pattern and Security Requirements: Engineering-Based Establishment of Security Standards. Springer. p. 100. ISBN 9783319166643.
[71] Boritz, J. Efrim (2005). "IS Practitioners' Views on Core Concepts of Information Integrity". International Journal of Accounting Information Systems. Elsevier. 6 (4): 260–279. doi:10.1016/j.accinf.2005.07.001. Retrieved 12 August 2011.