Coventry University

DOCTOR OF PHILOSOPHY

Systematic threat assessment and security testing of automotive over-the-air updates

Mahmood, Shahid

Award date:
2021

Awarding institution:
Coventry University

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of this thesis for personal non-commercial research or study
- This thesis cannot be reproduced or quoted extensively from without first obtaining permission from the copyright holder(s)
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Jul. 2025

Coventry University

Institute for Future Transport and Cities

Systematic Threat Assessment and Security Testing of Automotive Over-The-Air Updates

Shahid Mahmood

A thesis submitted in partial fulfilment of the University's requirements
for the degree of Doctor of Philosophy

July 27, 2021

Certificate of Ethical Approval

Applicant:

Shahid Mahmood

Project Title:

An Automated Cybersecurity Assessment Framework for Automotive OTA

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

10 February 2020

Project Reference Number:

P102815

This thesis is dedicated to the memories of my parents and especially to my dearest elder sister for her continued, unwavering support and encouragement to reach my goals. She has been a great source of inspiration for me, and there are no words to describe her contributions in my life.

Acknowledgement

Foremost, I would like to express my sincere gratitude to Allah Almighty for granting me the ability, strength, health and motivation to undertake and successfully accomplish this huge task. May the peace and blessings be on our beloved prophet Muhammad, his family and all of his companions.

I would like to offer my special thanks to my Director of Studies, Dr Hoang Nga Nguyen for his continuous support, patience, wisdom, and impressive knowledge and expertise that helped me immensely throughout my doctoral studies as well as during the process of producing this thesis. Without his persistent guidance and support, this thesis would not have been possible.

I am indebted to Prof. Siraj Shaikh for his highly useful, timely and invaluable guidance, expert and practical advice, and constructive criticism that I received throughout my research work. I have been impressed with his distinguished leadership and management skills. My heartiest appreciation goes to him for his incredible support, kindness, and encouragement.

My gratitude to all my (former and current) colleagues, friends, and well wishers who have assisted me throughout my studies over the years. I am particularly grateful for the insightful comments and suggestions offered by Dr. Jeremy Bryans.

Finally, I owe my deepest gratitude to my family for their unconditional love, patience, and most importantly their faith and confidence in my abilities that kept me motivated.

Abstract

Modern cars host numerous special-purpose, sophisticated computing and connectivity devices facilitating the correct functioning of various in-vehicle systems. These devices in the connected cars host complex software systems with more than 100 million lines of code, requiring regular and timely updates for functional enhancements and most importantly for fixing security-related bugs that could be exploited by adversaries to compromise the security of the vehicle. To replace the old mechanism for updating in-vehicle software which is expensive and inefficient for carmakers and inconvenient for the customers, Over-The-Air (OTA) software update system has emerged as an efficient, cost-effective and convenient solution for delivering software updates to automobiles remotely. While OTA offers several benefits, it introduces new security challenges that warrant immediate attention to carry out in-depth security analysis, as attackers can maliciously use the software update systems as attack vectors to undermine the vehicle security and safety. There are numerous studies investigating various aspects of the automotive cybersecurity; however, security testing of automotive OTA has not been covered adequately, with most of the prior work focusing on proposing improved techniques for securing automotive OTA updates. In order to ensure these update systems are effectively secure, thorough security assessment needs to be performed. To the best of our knowledge, there is currently no study that proposes or employs a systematic security testing approach for evaluating the security of automotive OTA update systems. This thesis closes this gap by presenting an in-depth security evaluation of Uptane framework, by using a structured threat analysis approach to constructing attack trees and employing a model-based security testing approach for generating effective security test cases. We implement a software tool that generates the security test cases by analyzing the structure of the attack trees and ultimately executing those test cases against the target system. We carried out several experiments mounting various attacks on the reference implementation of Uptane framework. While many of the experimental results showed that the framework is secure, providing effective protection against different threats and cyberattacks, some findings suggest that the reference implementation is vulnerable to the denial-of-service and eavesdropping attacks that can cause the system to fail in responding to legitimate update requests from clients and disclose sensitive information to malicious entities, respectively.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ACC Adaptive Cruise Control.

ACK Acknowledgement.

AES Advanced Encryption Standard.

AOM Aspect-Oriented Modeling.

AUTOSAR Automotive Open Software Architecture.

BC Blockchain.

CAN Controller Area Network.

CPS Cyber-Physical Systems.

CRC Cyclic Redundancy Check.

CSP Communicating Sequential Processes.

DFD Data Flow Diagram.

DoS Denial-of-Service.

ECDSA Elliptic Curve Digital Signature Algorithm.

ECU Electronic Control Unit.

EOF End of Frame.

FOTA Firmware Over-The-Air.

FTP File Transfer Protocl.

HARA Hazard Analysis and Risk Assessment.

HTTP Hyper Text Transfer Protocol.

HTTPS Hyper Text Transfer Protocol Secure.

ID Identifier.

IDE Identifier Extension.

IFS Inter-Frame Space.

IVI In-Vehicle Infotainment.

KES Keyless Entry System.

LIN Local Interconnect Network.

MAC Message Authentication Code.

MBST Model-Based Security Testing.

MITM Man-In-The-Middle.

MOST Media Oriented Systems Transport.

NIST National Institute of Standards and Technology.

OBD Onboard Diagnostic.

OCTANE Open Car Testbed and Network Experiments.

OEM Original Equipment Manufacturer.

OTA Over-The-Air.

PASTA Portable Automotive Security Testbed with Adaptability.

PBAC Policy-Based Access Control.

PTES Penetration Testing and Execution Standard.

RBAC Role-Based Access Control.

Repo Repository.

RSA Rivest–Shamir–Adleman.

RTR Remote Transmission Request.

SAND Sequential AND.

SDL Secure Development Lifecycle.

SecOC Secure Onboard Communication.

SFTP Secure File Transfer Protocol.

SIM Subscriber Identity Module.

SOF Start of Frame.

SOTA Software Over-The-Air.

SP Series Parallel.

STRIDE Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege.

SUT System Under Test.

TARA Threat Analysis and Risk Assessment.

TCU Telematics Control Unit.

TLS Transport Layer Protocol.

TMT Threat Modeling Tool.

TPMS Tyre Pressure Management System.

TUF The Update Framework.

UML Unified Modeling Language.

UNECE United Nations Economic Commission for Europe.

USB Universal Serial Bus.

VIN Vehicle Identification Number.

Wi-Fi Wireless Fidelity.

XML Extensible Markup Language.

Chapter 1

Introduction

1.1 Research Motivation

Modern vehicles are equipped with numerous sophisticated computing (known as Electronic Control Units or ECUs) and connectivity capabilities that enable and support correct functioning of various types of in-vehicle systems. With an increasing number of ECUs installed, today's luxury cars host complex software systems with more than 100 million lines of code [3, 4]. With that huge amount of code, regular and timely updates are inevitable for functional enhancements and most importantly for fixing bugs related to security issues that could potentially be exploited by adversaries to compromise the security and safety of the vehicle [5]. For the deployment of these updates, vehicles had traditionally been required to visit a service centre where an authorised personnel had to install the updates. This old mechanism for updating in-vehicle software is not only expensive and inefficient for carmakers, it is inconvenient for the customers as well (see Figure 1.1). For example, General Motors had to spend $4.1 billion on recalls in 2014 while their total net income for that particular year was $4 billion [6]. Another interesting related example is the work from Koscher et al. [7] which caused a recall of 1.4 million cars by automakers. Numerous vehicles were recalled for updates recently incurring huge financial costs for the automakers. Over-the-air software update system (see Figure 1.2) is emerging as an efficient, cost-effective and convenient way for delivering software updates to automobiles remotely allowing hassle-free delivery of critical updates in an economical and timely manner. While OTA offers several benefits, it introduces new security challenges that warrant immediate attention to carry out in-depth investigations, as attackers can maliciously use the software update systems as new attack vectors [8, 9, 10] to compromise the vehicle security. An insecure update mechanism can allow cyber criminals to undermine the security and safety of modern vehicles. There are numerous studies [7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21] investigating various aspects of the cybersecurity of automotive systems, exploring testing approaches and environments, threats, vulnerabilities, and mitigations, including many recent studies focusing on security issues of automotive OTA updates [22, 23, 24, 25]

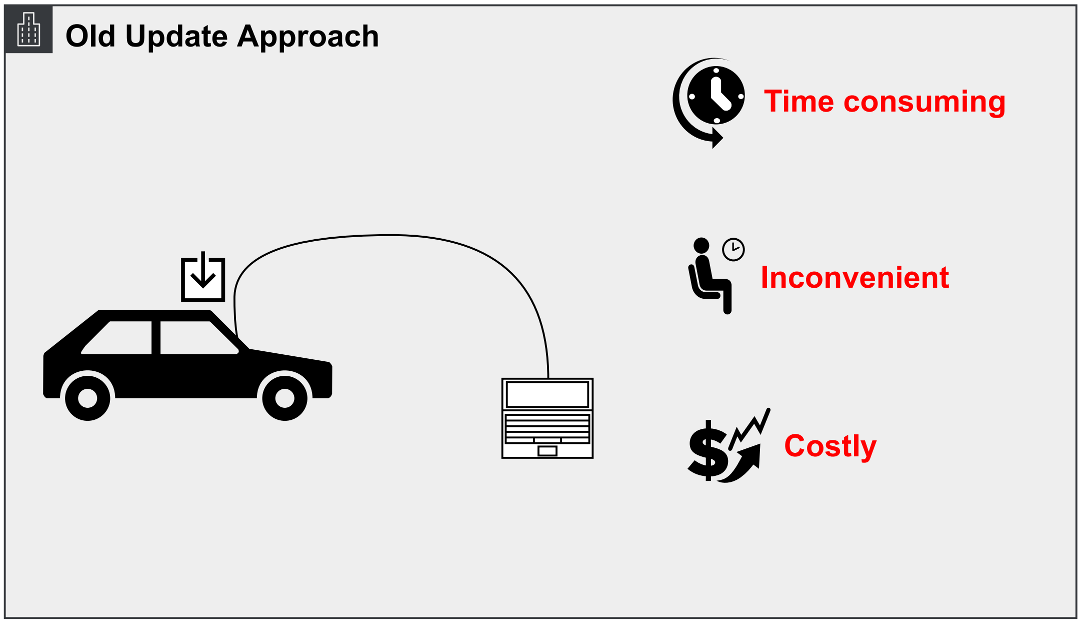


Figure 1.1: *This figure presents the traditional old-fashioned update process widely used, indicating some of its shortcomings.*

extensively; however, security testing of automotive OTA has not been covered adequately, In fact, most of the prior scientific studies tend to focus on proposing improved techniques for securing automotive OTA updates. Security evaluation of these solutions is imperative for validating their effectiveness. In order to ensure these update systems are secure and have effective and adequate protection in place against various cyberattacks and threats, thorough security analysis and evaluations need to be performed. To the best of our knowledge, there is currently no study that proposes or employs a systematic security testing approach for evaluating the security of automotive over-the-air update systems. This thesis attempts to close this gap by presenting a systematic security evaluation of Uptane Framework.

1.2 Research Questions, Aims and Objectives

As argued in [26], in contrast to an ad-hoc approach, which tends to suffer from a lack of clarity on the prioritization of security test cases, potentially leaving vulnerabilities undetected, a systematic security testing method increases the chances of revealing security loopholes in the system. In order to determine how to perform systematic security testing, we examined the literature by asking some basic questions, as summarized in the Figure 1.3. Where these questions helped us identify various prerequisites to a systematic security testing process and outputs from the major steps, the preliminary findings from the literature review suggested that the model-based security testing can assist with the systematic security testing of the OTA updates. These

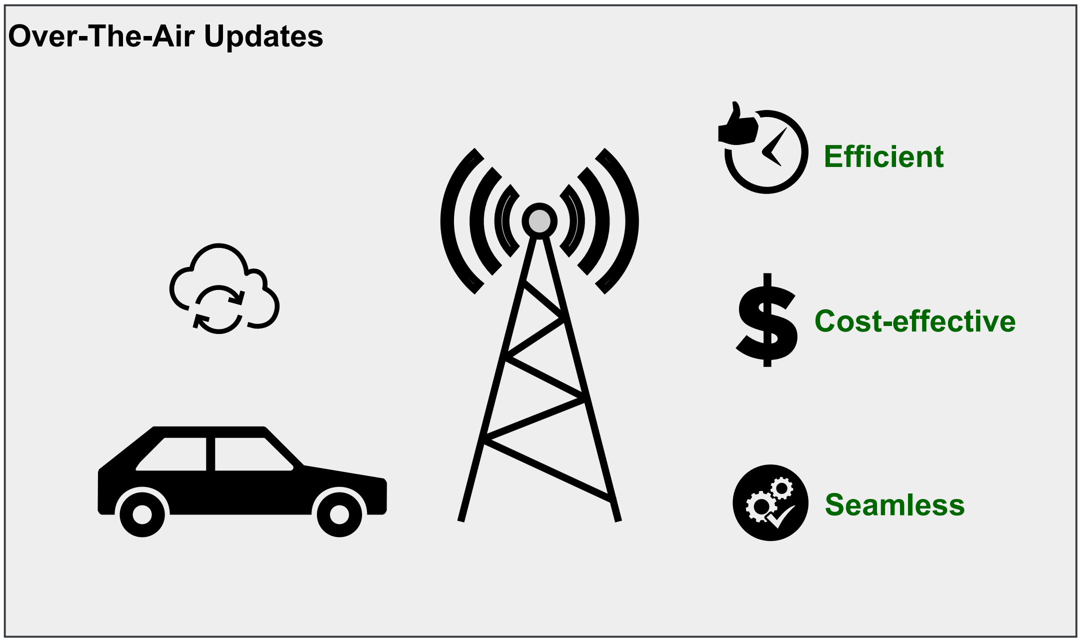


Figure 1.2: *This figure presents the modern, futuristic approach to update delivery: over-the-air updates. This method addresses the problems of the traditional approach.*

questions also enabled us to conceptualize and improve our overall threat analysis and security testing approach used in this study. Therefore, the key question investigated and addressed by this doctoral study is outlined below:

The Research Question How can Model-Based Security Testing (MBST) facilitate the systematic threat identification and executable test case generation for effective security testing of automotive systems that can help strengthen and improve the system security? This fundamental question was broken down into three sub-questions, as listed below:

1. How can model-based security assist with systematic threat enumeration?
2. How can the identified threats be leveraged to generate security test cases?
3. What specific tools and techniques can be employed for threat enumeration and test-case generation?

In order to address the core research question (and the associated sub-questions) introduced above, this doctoral research aims at applying the MBST approach for carrying out systematic threat assessment and security analysis of OTA updates, as MBST supports test case derivation and execution by leveraging the system models.

Based on the overall aim of the study and research questions, following objectives were identified for achieving the overall aim:

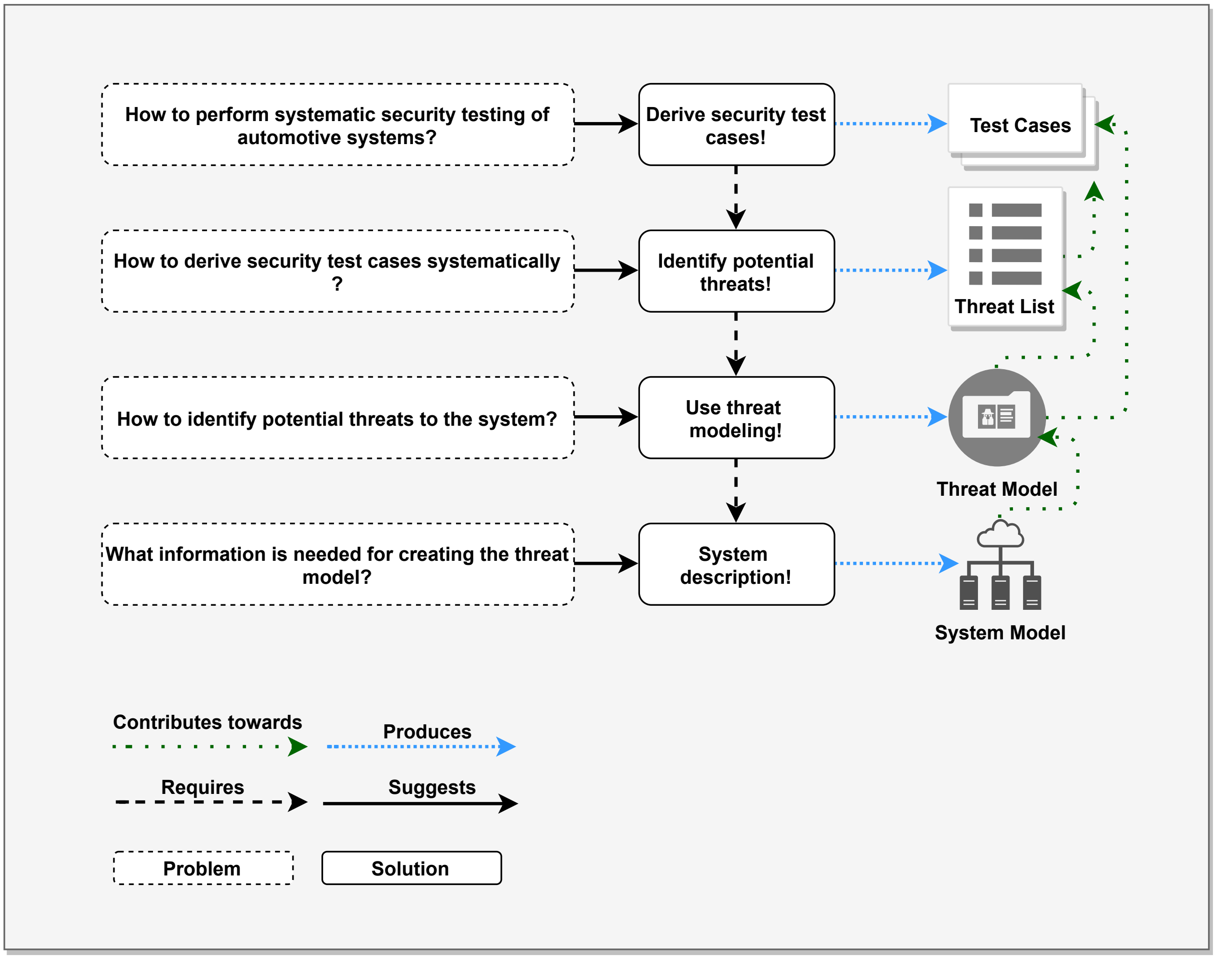

How to perform systematic security testing of automotive systems?
Derive security test cases!
Test Cases
How to derive security test cases systematically ?
Identify potential threats!
Threat List
How to identify potential threats to the system?
Use threat modeling!
Threat Model
What information is needed for creating the threat model?
System description!
System Model
Contributes towards
Produces
Requires
Suggests
Problem
Solution

Figure 1.3: *This figure shows the key questions that helped identify the core activities and outputs for performing a systematic threat analysis and security testing of the automotive OTA.*

Obj1: To conduct a critical review of the existing testing environments and methods for establishing the state of the art in automotive cybersecurity testing

Obj2: To apply a systematic threat analysis approach for threat enumeration and attack tree (a threat modeling technique; please refer to Chapter 2 for a detailed introduction of this technique) construction

Obj3: To design and implement a software tool capable of generating and executing security test cases from attack-tree analysis

Obj4: To validate the approach and the software tool by performing a comprehensive security analysis and testing of the OTA Updates (using the reference implementation of the Uptane Framework)

1.3 The Contributions of this Thesis

This research contributes to

A systematic threat analysis approach for constructing attack trees

Demonstrated by an in-depth analysis of OTA/Uptane, in Chapter 5 we show how our step-by-step threat analysis approach helps enumerate security threats and construct attack trees, which are validated through the system testing (a graphical overview of this contribution can be viewed in Figure 1.4). This contribution results from meeting the research objective 2 (please refer to Obj2 above).

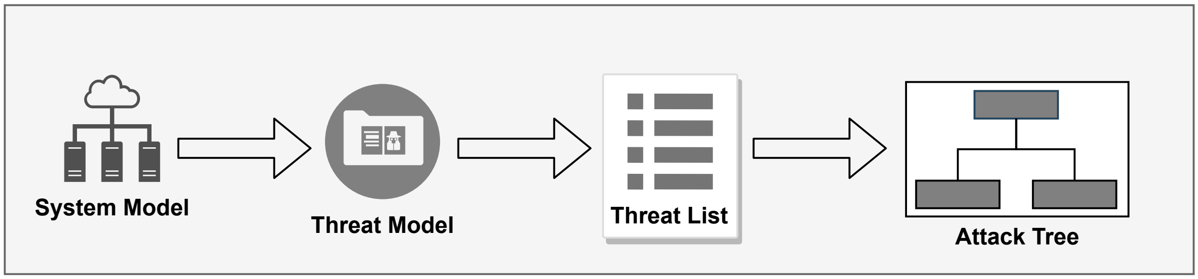


Figure 1.4: *Contribution 1: Overview of the threat analysis approach leading to the construction of an attack tree systematically.*

A model-based security testing approach based on attack tree to derive security test cases We introduce our model-based security testing approach in Chapter 6 that uses attack trees to derive effective security test cases by analyzing the structure of these attack trees (Figure 1.5 provides an overview of the approach). Based on the formal semantics of the attack trees, we use formal methods to prove the correctness of our test-case derivation approach. Furthermore, in order to validate the approach

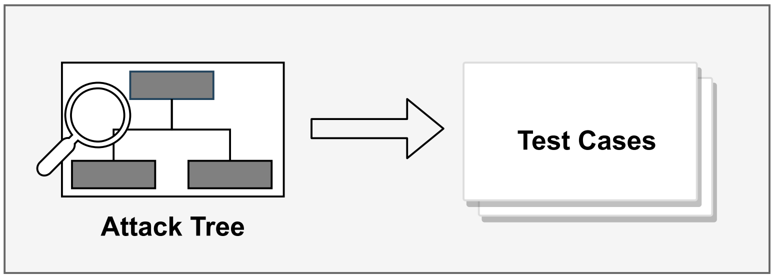


Figure 1.5: *Contribution 2: Overview of the model-based security testing approach for generating test cases by attack tree analysis.*

we demonstrate a number of cyberattacks using the generated security test cases. This contribution maps to the research objective 3 (please refer to Obj3 above).

An in-depth experimental security analysis of the reference implementation of the Uptane Framework

Meeting the research objective 4 (please see Obje4 above), a comprehensive security analysis of the Uptane Framework's reference implementation is presented in Chapter 7, which includes a complete threat analysis, attack-tree construction and various security test cases derived from the constructed attack trees as well as the security testing; highlighting the strengths and weaknesses of the reference implementation.

The automation of the test-case generation and execution by devising a software tool

The security test case generation and execution have been automated by designing and implementing a custom software tool (as evidenced in Chapter 6), which is capable of deriving correct security test cases by leveraging the attack-tree structure analysis approach we briefly introduced above. The tool executes all the generated test cases against the target system and generates a report showing the number of test cases executed and whether they succeeded. This contribution is linked with the research objective 4 (please see Obj4 above).

A comprehensive survey of the testbeds and testing approaches, providing a critical analysis of the testing environments and methods in automotive security testing

Finally, the literature review (as presented in Chapter 3) includes an extensive comparative analysis of the major testbeds and testing approaches (please refer to Obj1 above) proposed by the research community in automotive cybersecurity, presenting a critical analysis of the testing environments based on various relevant characteristics and factors.

1.4 Thesis Structure and Overview

Chapter 2 that follows this introductory chapter provides the reader with essential background information on automotive OTA updates, attack trees, and threat modeling, presenting:

- an overview of the automotive OTA updates
- key security issues, vulnerabilities & threats, security requirements, and some of the major proposed solutions to securing the OTA updates
- a detailed overview of the Uptane Framework is also included, introducing the server-side and client-side components (i.e., Image Repository, Director Repository, Time Server, Primary and Secondary ECUs etc.), metadata structures, roles, and specific procedures for full and partial verification of the metadata and images
- a dedicated subsection on the attack trees provides introductory information and two different formal semantics defined based on series-parallel graphs and sequences providing necessary background for the formal proof of our test-case generation approach presented in Chapter 6
- an introduction of the threat modeling providing overviews of some of the threat modelling approaches along with a description of STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of service, Elevation of privilege) and associated tool from Microsoft used in the threat analysis
- an overview of risk assessment and the description of the OWASP Risk Assessment approach, which has been used in this thesis determining the risk rating of the threats
- an overview of the MBST approaches proposed for security testing of embedded system.

Literature review is presented in Chapter 3, which includes the following:

- automotive cybersecurity in general, highlighting major attack vectors, such as Controller Area Network (CAN) bus, in-vehicle infotainment, and Onboard Diagnostic (OBD) port along with some relevant major security threats and countermeasures
- a comprehensive comparative analysis of the major automotive testbeds and testing approaches is also presented; all the testing setups are compared based on the following attributes:
    - adaptability
    - portability
    - fidelity
    - cost and safety implications

- a comparison of testbeds' various capabilities, i.e., supported communication protocols (i.e., CAN, LIN, FlexRay, MOST), and attack types (i.e., DoS, replay, spoofing etc.)
- finally, some of the widely used testing approaches in the domain are also described; in particular, overviews of automotive fuzz testing, automotive vulnerability scanning, and penetration testing are provided.

The next chapter (i.e., Chapter 4) introduces the methodology, by describing the following:

- an overview of the security testing approach and its major phases and corresponding activities
- an introduction of the step-by-step workflow of the approach by using in-vehicle infotainment system as an example to demonstrate the application of the approach
- this in particular includes showing the step-by-step process (i.e., System Decomposition) to identify critical assets of the system under test; and producing a suitable system description; this is followed by a demonstration of the threat enumeration process.

The Chapter 5 (titled *Constructing Attack Trees*) focuses on presenting the systematic approach for constructing attack trees based on the threats identified in the preceding phase. The approach is demonstrated by using adaptive cruise control system as an example target system. The process is clearly explained by guiding how to construct attack trees.

The following chapter (i.e., Chapter 6: Generating Test Cases) introduces the test-case generation process by using the attack trees constructed in the previous phase. Additionally, the correctness of the test-case generation approach is also proved in the chapter using formal methods. Moreover, the software tool for automated test-case generation and execution is also detailed. Finally, some example security test cases are generated and presented to show the approach in action.

In-depth security analysis and results of the security testing of the reference implementation of the Uptane Framework are presented in Chapter 7 by including the following:

- system decomposition is applied to prepare system description of the Uptane followed by a threat enumeration for identifying the security threats. A number of attack trees constructed by following the approach introduced in Chapter 5, which were in turn is used for generating various security test cases
- The results obtained from executing the security test cases are reported summarizing the outcomes of all experimental attacks on the reference implementation
- Key findings of the security analysis along with the limitations outlined before concluding the chapter.

Chapter 8 presents some relevant countermeasures for the selected experimental cyberattacks (for more information, see Chapter 7).

The last chapter, that is, Chapter 9 summarizes the research by highlighting the contributions of the thesis and indicating some future directions.

In addition to these core chapters, Appendices have been included to present some useful supplementary contents, providing additional information that has not been included in the main body of this thesis, which includes the following items:

- Appendix A: Source code of the software tool for generating security test cases.
- Appendix B: XML source code of the attack trees used for generating the security test cases included in the experimental security analysis of the reference implementation.
- C: Threat Modeling Reports generated by the threat modeling tool, providing the source information used for constructing the attack trees for the security analysis of the reference implementation.
- Finally, Appendix D includes the Ethics Documentation. A graphical overview is shown in the Figure 1.6.

1.5 Publications

As a result of this research program, the following publications have been produced. Hence, this thesis refers to/includes contents from these publications.

- Mahmood S, Fouillade A, Nguyen HN, Shaikh SA. A Model-Based Security Testing Approach for Automotive Over-The-Air Updates. In2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2020 Oct 24 (pp. 6-13). IEEE.
- Mahmood S, Nguyen HN, Shaikh SA. Automotive Cybersecurity Testing: Survey of Testbeds and Methods. InDigital Transformation, Cyber Security and Resilience of Modern Societies 2021 Mar 24 (pp. 219-243). Springer International Publishing.

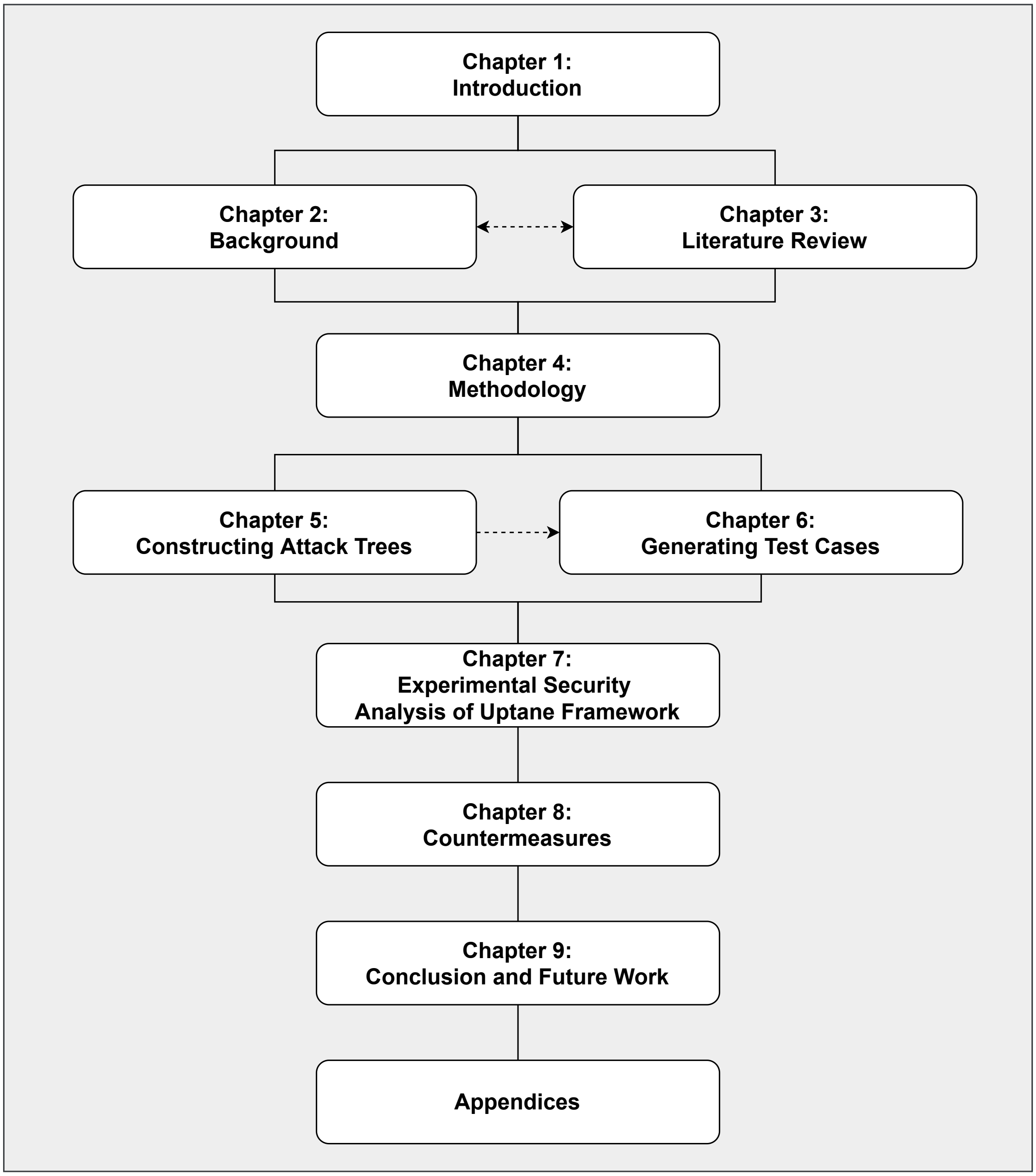


Figure 1.6: *Graphical overview of the thesis.*

1.6 Chapter summary

This chapter presented an introduction of this doctoral research study, outlining the research motivation (i.e., why systematic security testing of the Uptane is essential), questions (i.e., how the MBST can be effectively employed for the systematic security testing, etc.), aim, and objectives. Additionally, it summarized the key contributions of the thesis, which briefly include the following:

- A systematic threat analysis approach for constructing attack trees

  –Demonstrated by in-depth analysis of the Uptane

  –Constructed attack trees validated through system testing

- A model-based security testing approach based on attack trees to derive security test cases

  –Proved the approach's correctness by using formal methods

  –Demonstrated attacks using the generated test cases

- In-depth experimental security analysis of the Uptane Framework

- Automation of the test-case generation and execution by implementing a custom software tool

- A comprehensive comparative review of the testbeds and testing approaches published in the relevant literature

Finally, this chapter is concluded by providing an overview of the thesis along with listing the publication that resulted from this research program.

Chapter 2

Background

In this chapter, we present some background to the research work presented in this thesis. Starting with an overview of the automotive OTA, we describe some use cases, security issues, security requirements, some proposed techniques for secure delivery of the OTA updates, as well as an extensive introduction to the Uptane Framework. In addition to a detailed overview of the attack trees along with their formal semantics, we also provide a brief introduction to the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) threat modeling approach. Finally, an overview of the model-based security testing approaches employed by automotive security researchers is briefly discussed.

2.1 Automotive OTA Security

2.1.1 Overview

The phrase *Over-The-Air or Software Over-The-Air (SOTA)* refers to the method of delivering software updates using a WiFi, Bluetooth, or cellular network link remotely. While the acronym *SOTA* is generally used for referring to all types of software updates, the phrase *Firmware Over-The-Air (FOTA)* is used specifically to refer to the deployment of a firmware to the target device [27]. In most cases, the FOTA is used to deliver an update that involves replacing the existing firmware on the device. (more precisely, an ECU). The three main components of the OTA update system generally include: backend cloud servers, ECU or other similar devices in the vehicle and a suitable communication link.

OTA updates have been around for several years in the software industry for the deployment of critical bug fixes and functional enhancements to both the operating systems and application programs in laptops and other handheld computing and communication devices [28]. In the automotive industry, OEMs (Original Equipment Manufacturers) are increasingly embracing the OTA technology for delivering updates to embedded devices in the connected vehicle [29]. Car manufacturers are predicted to save $35 billion in year 2022 by relying on OTA technology for sending updates remotely [30].

Use Cases

Some of the major use cases for automotive updates are outlined below[31]:

- **Bug fixing:** In order to comply with legal and regulatory requirements, automakers can leverage SOTA technology for delivering updates addressing safety and/or security issues in an economical and speedy way, eliminating the need for vehicle recalls. This type of SOTA update often involves fixing safety-critical faults.
- **Quality improvements:** SOTA updates can be used by automobile manufacturers for quality enhancements, such as improving fuel efficiency.
- **Research and development:** SOTA technology can also be utilised for collecting useful information about various aspects of the vehicle which could be highly useful for future developments. This may include gathering data about performance or other technical issues.

2.1.2 Vulnerabilities and Threats

Software update systems have been attacked and compromised by cybercriminals (for example, [32, 33, 34, 35, 36]) for delivering and installing malware on computer and mobile systems. The World Forum for Harmonization of Vehicle Regulations (WP.29) - an international regulatory forum within the institutional Framework of the United Nations Economic Commission for Europe (UNECE) - has introduced new regulations to be implemented by the member states from January 2021. These regulations are concerned with regulating the automotive cybersecurity, automotive cybersecurity management systems, automotive OTA updates and automotive OTA updates management systems [37]. A revised draft proposal for these regulations has recently been published by UNECE [2], which identifies major threats (as shown in Table 2.1) to automotive update procedures along with relevant vulnerabilities and attack methods.

A more comprehensive threat model has been presented in [38], as can be seen in Figure 2.1, that provides a graphical overview of various security threats to the OTA system that can potentially be used by malicious entities for compromising the security and safety of connected vehicles. Each threat has one or more types of related attacks that can be used by hackers, ranging from reading the contents of an update to gaining the vehicle control. Attackers can have one or more attack goals, as described in [38] and [39]. A summary of these goals is presented below:

- Read the contents of updates to discover confidential information, reverse-engineer firmware, or compare two firmware images to identify security fixes and hence determine the fixed security vulnerability

Table 2.1: *Automotive OTA update security threats and vulnerabilities. Adapted from [2].*

Threat	Vulnerability or Attack Method
Misuse or compromise of update procedures	• Compromise of OTA update procedures by fabricating system update program or firmware • The software is manipulated prior to the update process • Cryptographic keys compromised for delivering malicious update
Deny legitimate updates	Denial of service attack against update servers or network to block the delivery of critical updates

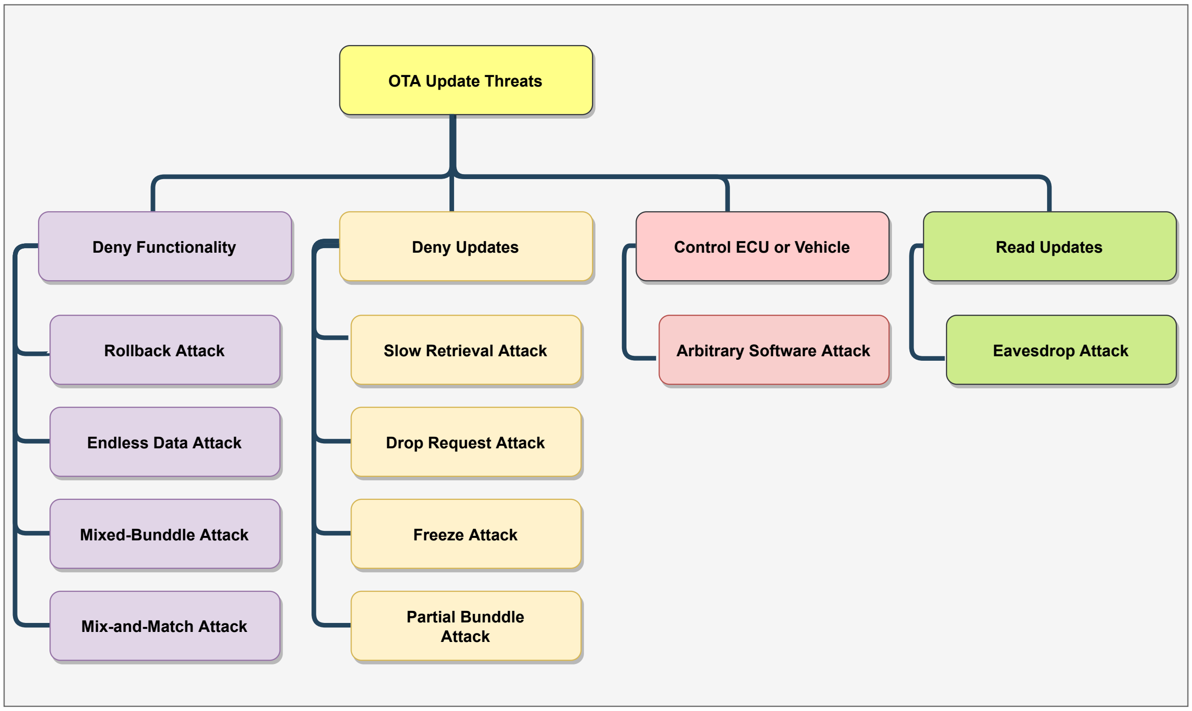


Figure 2.1: *Graphical overview of major threats to OTA Update System along with corresponding potential attacks.*

- Deny installation of updates to prevent vehicles from fixing software problems
- Cause one or more ECUs in the vehicle to fail, denying use of the vehicle or of certain functions
- Control ECUs within the vehicle, and possibly the vehicle itself

2.1.3 Security Requirements

Security of OTA updates is critical and effective defense and mitigation against different potential threats needs identification and understanding of security requirements of OTA update ecosystem. Halder et al. [40] identify various security requirements for OTA updates, as summarised in Table 2.2.

2.1.4 Secure OTA Update Methods and Techniques

There is no doubt that in the future all or most updates to the connected cars will be delivered using OTA technology [38]; thus, the security of these updates is paramount. A number of studies have been published proposing different solutions for securing OTA updates. We provide an overview of some of these solutions in the following subsections.

AiroDiag: An Over-The-Air Diagnostics and Updates System

Mansour et al.[41] propose AiroDiag - an over-the-air system - for performing automotive diagnostics and software updates. Figure 2.2 provides an overview of the major components of the system on both the client and OEM sides. The AiroDiag client monitors and sends the diagnostics (e.g., faults and performance) information to the OEM, which can be used to advise the customer of any detected issues or new updates to be installed. The database on the OEM backend holds both the vehicle information (such as, vehicle manifests detailing what updates are already installed on the vehicle) as well as the authentication communication keys for each car (tagged with a unique manufacturer ID) which has AiroDiag installed on it. This authentication key is supplied to the server when the client requests to establish a connection for verifying the client is a trusted entity. ECUs may either use CAN or serial communication for communicating with the AiroDiag component on the vehicle. Once connected with the server, the client sends a list of all the currently installed software on each ECU. The server compares the received list with the one it contains in the database and informs the client of the new updates if applicable. The driver is shown an alert to ask whether to proceed with the downloading and installation of the updates. The AiroDiag client on the vehicle side will proceed with downloading the software updates if the driver allows it to do so and stores all the updates on a non-volatile memory (such as an SD card), and flashes them on corresponding ECUs once download is complete. In order to ensure the customer's privacy is not compromised, an encrypted channel is used

Table 2.2: *A summary of the automotive OTA update system security requirements for each of the major components. While this table presents key requirements for the automotive OTA update, it is not exhaustive by any means.*

Component	Security Requirement
Security of the update package during transit	Security of update package should be ensured so that its integrity, confidentiality, authenticity, and freshness are intact while it is being transmitted from the backend to the vehicle.
Security of update package while stored	The update package must be protected against any security breaches affecting its integrity, authenticity, confidentiality, and freshness while it is stored prior to the installation.
Update authorisation verification	The target ECU/device needs to be verified to ensure whether it is authorised for receiving and installing the update package.
Protection of the update installation	Traceability of the delivery and installation of the update must be ensured.
Protection against denial of service	The updates must be available to the ECUs for installation when required; therefore, the backend servers must be protected against any security breaches (e.g., DoS) affecting the availability of the updates.

between the vehicle and OEM system as well as all the features offered by AiroDiag must obtain driver's approval before proceeding with any action.

The authors of the AiroDiag implement it to simulate the update process and share the results showing the time taken by the OTA update procedure. Advanced Encryption Standard (AES) is used for all the communication between the server and client sides. Some of the key considerations highlighted about some limitations include the long boot time (due to the Ubuntu OS installed) taken by the AiroDiag client, and potential of cyberattacks.

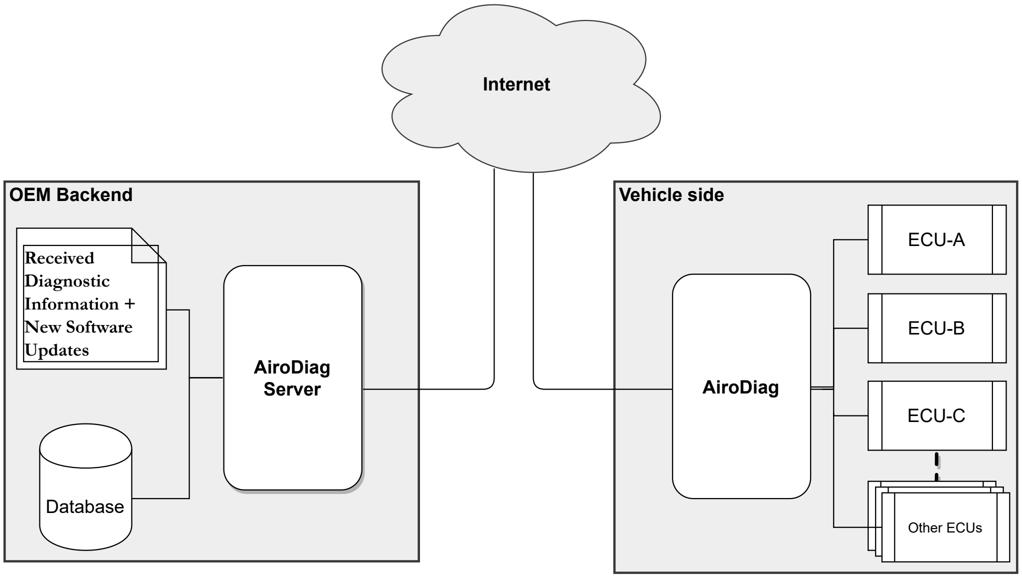


Figure 2.2: *An overview of the AiroDiag architecture (adapted from [41]) showing the components on the server side and client side.*

An Integrated Approach for Securing the OTA Software Updates

A method using a combination of enhanced cryptography and image stenography techniques for securing automotive OTA software updates has been proposed by Mayilsamy et al. [42]. The authors use the customised RSA crypographic algorithm for encrypting the update data, which in turn is embedded along the edge of the image using the Least Significant Bit technique. They use fuzzy logic for the detection of the edges. For the verification of the authenticity of the source of the software update, they leverage Hash algorithm. The proposed method provides two levels of security for secure OTA updates for automobiles: while the modified RSA algorithm is used to provide the first level of security; second level of security is achieved by using image stenography technique (using fuzzy logic for the edge detection). Based on the evaluation results of their proposed method, the authors conclude that their proposed method showed better results in terms of the security as compared to conventional cryptographic tech-

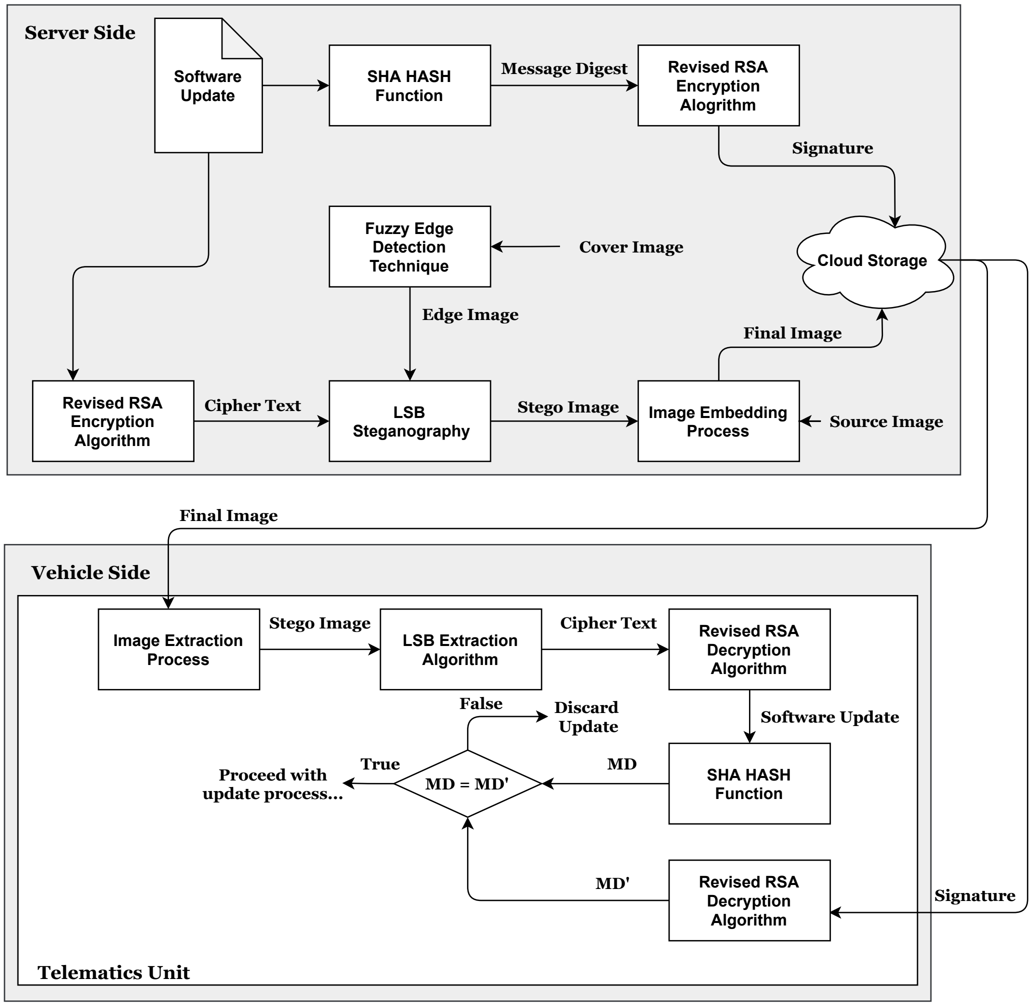


Figure 2.3: *An overview secure automotive OTA updates systems using enhanced cryptography and image stenography techniques (adapted from [42]).*

niques. However, a major limitation of their proposed method is the performance when it comes to encryption/decryption time.

OTA Update Security Using Blockchain Techniques

For effective security protection of OTA updates for connected vehicles, Steger et al. [43] propose a secure architecture employing Blockchain (BC) technology. The proposed architecture ensures the confidentiality and integrity of the software updates as well as privacy of all the entities involved in the system by providing a secure and trustworthy interconnection between them, which relies on a Lightweight Scalable Blockchain for addressing the inherent limitations (i.e., high resource consumption and high latency) of traditional underpinning consensus algorithm in order to meet the special requirements of the intrinsically resource-constrained embedded systems. Instead of relying on a network model involving central management, this BC-based architecture uses a distributed environment where each participating stakeholder forms a cluster, which constitutes a cluster head and a number of cluster members. A network overlay is used to interconnect all the cluster heads. The software provider distributes the new software and/or updates to the OEM which forwards them to the local software update providers, and ultimately to the target vehicles for installation on the target ECUs. Cloud storage acts as a secure repository to hold the updates received from the software providers or OEMs. These cloud repositories are secured by advanced authentication mechanisms to ensure only authorized entities are able to access, modify, and download software images. The authors evaluate their proposed architecture by means of a proof-of-concept implementation of the OTA update system, comparing it with certificate-based system. The results demonstrate the proposed BC-based architecture for OTA updates is better than the traditional certificate-based systems in terms of performance.

Although Blockchain technology can effectively be used to guarantee the integrity, authenticity, and confidentiality aspects of the OTA updates, increased complexity stemming from its inherent distributed architecture and redundancy/replication requirements raise cost, time, and effort concerns. Moreover, more advanced, update repository-related attacks, such as slow retrieval attack, freeze update attack, endless data attack, etc. need further attention, introducing more complexities. Finally, this particular solution does not consider customization requirements for delivering required updates to the vehicle based on previously installed updates.

All the solutions described above focus on the confidentiality and integrity of the update contents by using cryptographic and Hash function techniques and technologies, While such techniques have their own merits, they do not provide a comprehensive coverage of all the threat types that can compromise the security of such systems. (as shown in the Figure 2.1).

The Uptane Framework

Uptane, developed by US researchers in collaboration with automotive industry stakeholders, is an automotive software update framework, which is claimed to address automotive-specific security flaws, and provide protection against a wide range of security attacks, offering both the security and customisability of updates for different vehicles depending on their particular needs.

As shown in Figure 2.5, Uptane Framework has three core components: the Image Repository, the Director Repository, and the Time Server.

Image Repository

The Image Repository holds all the images deployed by the OEM along with metadata files for proving the authenticity of the hosted images. OEMs use offline keys for signing

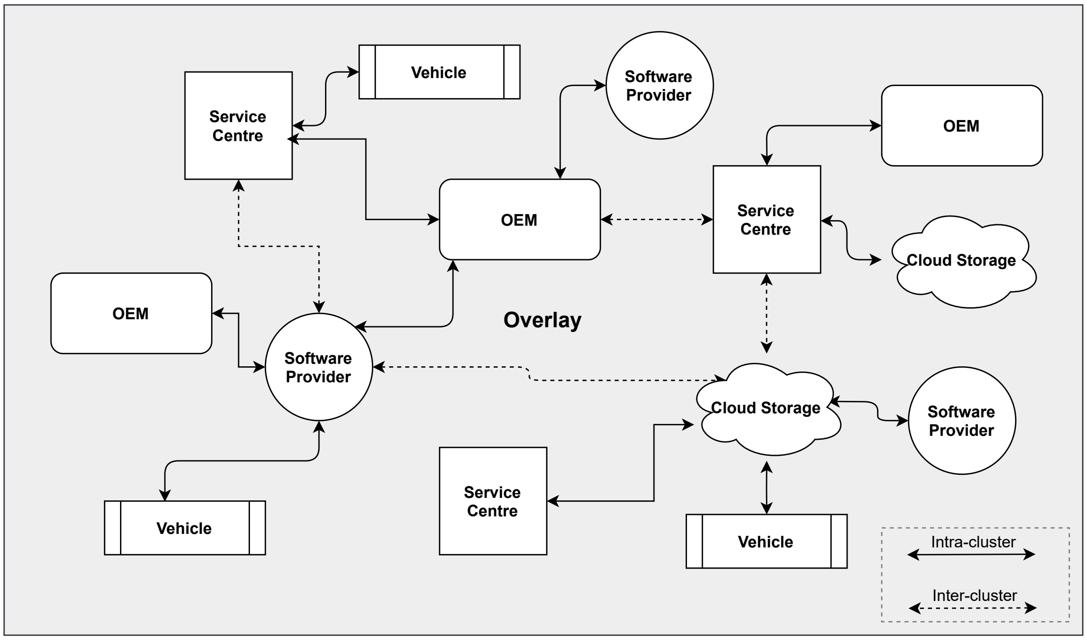


Figure 2.4: *Blockchain-based automotive OTA Architecture (adapted from [43]) for ensuring the confidentiality, integrity of software updates as well as protecting the privacy of the users.*

metadata stored on the repository as a protective measure against any attempt from attackers to tamper with this metadata. Some of the mandatory capabilities that the Image Repository needs include the following:

- An interface for image and metadata download (that *SHOULD* be public) *SHALL* be exposed.
- Authorization *SHALL* be required for writing images and metadata.
- A method enabling authorized users to upload images and associated metadata *SHALL* be provided.
- The repository *SHALL* check whether the user has the appropriate set of permissions for writing images and metadata for the specific images by checking the chain of delegation.
- The repository *SHALL* include appropriate storage mechanism allowing authorized users to store files with unique file names. It *SHALL* also enable the users to retrieve the stored files. The storage mechanism *MAY* be file-system, key-value store or a database.
- Read access *MAY* require authentication.

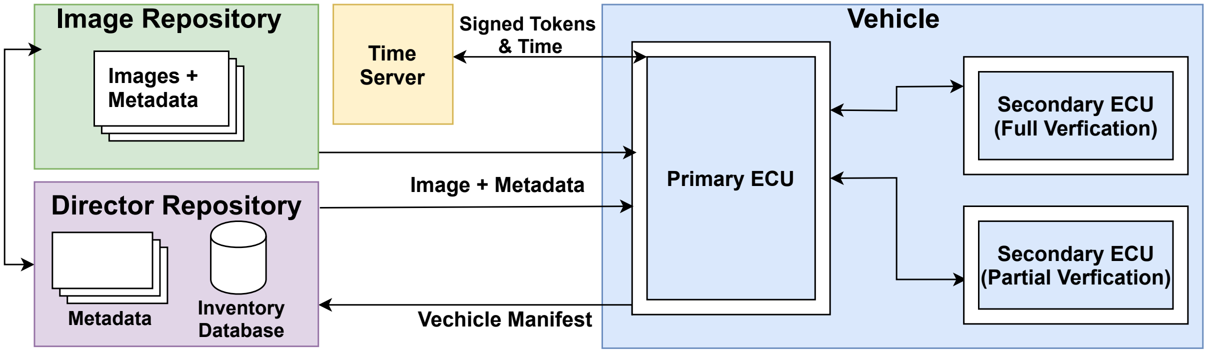


Figure 2.5: *An overview of the Uptane Framework, illustrating the interconnections and flow of information among the Time Server, Image Repository, Director Repository, Primary ECU and Secondary ECU.*

Director Repository

The Director Repository is responsible for tracking and determining what updates to deliver to each ECU based on the current status of the repository and currently installed updates. Based on the information contained in the signed manifest provided by the vehicle, Director Repository determines and prepares appropriate update packages for the vehicle. A vehicle manifest informs the Director of its previously installed versions of the updates. Unlike Image Repository, Director Repository uses online keys for signing metadata. Director Repository also needs to meet some mandatory and optional requirements as outlined below:

- An interface *SHALL* be exposed for *Primaries* to download metadata and upload vehicle version manifests. This interface *SHOULD* be public.
- The Director *MAY* encrypt images if need be, either using on-the-fly encryption or storing the encrypted images on the repository.
- For enabling the automated services to write metadata, the Director Repository *SHALL* implement appropriate storage. It *MAY* use file-system, key-value store, or a database.

Furthermore the Director Repository is required to comply with the following six-step process for directing the installation of updates on the target cars:

1. The vehicle *SHOULD* be identified by the Director, which *MAY* be done by determining the unique vehicle identifier received from the Primary ECU. Moreover, other mechanisms (e.g., 2-way TLS with unique client certificates) *MAY* also be used.
2. The Director queries its inventory database for relevant information about a specific ECU in the vehicle by using the vehicle identifier.

3. Manifest accuracy check *SHALL* be performed by Director by comparing the received manifest with the one in its inventory database. The request *MAY* be discarded should any checks fail.

   While an implementer *MAY* include additional checks, at least following checks *SHALL* be carried out:

   - All ECUs contained in the manifest are also recorded in the inventory database.
   - The sender Primary ECU's key matches the signature of the manifest.
   - Each Secondary ECU's contribution to the manifest signature matches its corresponding key.

4. In order to prevent replay attack, the Director *SHALL* check whether the nonce or counter in each ECU version report has been used before, the request *SHOULD* be dropped if the nonce or counter is being reused.

5. The information contained in the vehicle version manifest is used by the Director to determine whether new updates need to be installed based on the versions of currently installed software. The Director *MUST* consider any conflicts or dependencies between the images and *SHOULD* employ appropriate, well-established dependency resolution techniques.

6. Images *MAY* be encrypted by the Director if required by the target ECUs.

7. Metadata, comprising of Targets, Snapshot, and Timestamp, is generated for the desired images by the Director to be sent to the Primary ECU. The metadata generation takes into account the dependency resolution.

A private inventory database *SHALL* be used by the Director for storing vehicle and associated ECUs information, which *MUST* record a unique identifier (e.g., a VIN) for each vehicle. Additionally, following information *MUST* be recorded for each ECU:

- A unique identifier, such as a serial number
- An ECU key (which can either be symmetric or asymmetric; only public key to be stored if the latter is used)
- An identifier for ECU key
- An indication of whether the ECU is a Primary or Secondary

Furthermore, additional information pertaining to the ECUs and vehicles may be recorded in the inventory database. In particular, a hardware identifier *SHOULD* be used to prevent the Director from directing an ECU to install incompatible firmware.

Time Server

As time is a critical aspect in automotive software updates, knowledge of current, accurate time is crucial for ECUs to provide protection against freeze attacks, which involves sending the same update indefinitely even when new updates are available. Many ECUs are unaware of current time because they do not have builtin clocks, this is where the Time Server plays an important role in providing accurate time to the vehicle in a cryptographically secure manner, which helps ECUs defend time-related attacks. The Uptane standard includes the Time Server as an optional component, and if it is implemented the following requirements must be fulfilled:

- In response to the token sequences received from the vehicle, the Time Server is required to provide one or more signed responses along with the time and these tokens. It *MAY* generate a single time attestation composed of the current time and all the tokens. Alternatively, it may generate multiple time attestations each comprising of the current time and one or more tokens. The response should contain all the tokens.

- In order for the Primaries to communicate with it, the Time Server will expose a public interface, which *MAY* rely on any transport control protocol, such as HTTP, HTTPS, FTP, or SFTP.

- The Time Server's key is rotated by listing a new key in the Director's Root metadata, as is the case with other roles' keys. Additionally, the key must also be listed in the custom field of the Targets metadata of the Director Repository.

Uptane Roles

Uptane repositories rely on different roles, each responsible for signing different type of metadata as explained below:

- **The Root Role** is responsible for signing metadata, used for distributing and revoking public keys for the verification of the Root, Targets, Timestamp and Snapshot metadata.

- **The Snapshot Role** is concerned with signing metadata indicating the images released by the repository at the same time.

- **The Targets Role** has the responsibility of signing metadata (e.g., cryptographic hashes and file size etc.) that verifies the image.

- **The Timestamp Role** is used to sign metadata to indicate the availability of new metadata or images on the repository.

Uptane Metadata Structure and Types

To ensure security, Uptane Framework relies on a properly generated metadata having a specified structure. There is no particular format or encoding mandated by the Uptane standard for the metadata. Any encoding scheme complying with the standard requirements can be used by the implementers. However, since the metadata verification involves string comparisons; therefore, the Uptane standard mandates the use of Unicode Format for Network Interchange for encoding all strings. Table 2.3 provides a summary of the four different types of metadata generated by Uptane Framework, each corresponding to the metadata roles described earlier. In addition to the characteristics (unique to each metadata type) described in the Table 2.3, there are some common characteristics shared by all types of the metadata. All public keys, represented by a public key identifier, can be composed of all or either of the following:

- The public key's value, which *MAY* be formatted as a PEM string
- The cryptographic algorithm (e.g., ECDSA or RSA) used by the public key
- The signature verification scheme used, such as ecdsa-sha2-nistp256 or rsassa-pss-sha256
- All the four Uptane roles have a common structure, which includes metadata payload to be signed and an attribute that contains the signature of the payload

Finally, there may be additional custom metadata belonging to the Targets files (for more on this, please refer to the official design documentation at [39]).

Primary and Secondary ECUs

A primary ECU is typically the one that is more capable in terms of storage capacity and connectivity as compared to a Secondary ECU which needs help from the primary ECU for receiving and installing software updates. A Primary ECU directly communicates with the Director Repository in order to download metadata and firmware images, carry out verification to verify the authenticity and integrity of updates, and finally distribute the downloaded updates to the Secondary ECU. A Secondary ECU, depending on its capabilities, performs full or partial verification of the image against the metadata, and installs it if the verification succeeds.

Uptane-compliant ECUs *SHALL* be capable of downloading and verifying image metadata and image binaries before the installation of a new image. Moreover, ECUs *MUST* have a secure way for verifying the current time. In order to detect and act against any attempt of a slow retrieval attack, ECUs *SHOULD* monitor the metadata and image binaries download speed. The Director repository *SHOULD* be informed (e.g., by communicating this information in the next vehicle manifest) if the download speed drops the specified minimum threshold level. The Uptane Framework design documentation specifies the following key *build-time* requirements for ECUs:

Table 2.3: *This table summarizes the four different types of metadata used by the Uptane Framework.*

Metadata Type	Description
Root Metadata	• Responsible for the distribution of public keys of top-level Root, Targets, Snapshot, and Timestamp roles • *SHALL* contain two attributes: – A representation of the public keys of all four roles and each key *SHALL* have a unique public key identifier – An attribute mapping each role to its public key and the required signatures threshold for this role
Targets Metadata	• Contains all the information (i.e., filename, file sizes, hashes) pertaining to the images to be installed on ECUs • May also contain metadata about delegations, which allow one Targets role to delegate its authority to another
Snapshot Metadata	• Lists filenames and version numbers of all Targets metadata files, providing protection against mix-and-match attacks • *MAY* also list filename and version number of the Root metadata for backward compatibility with the TUF
Timestamp Metadata	*SHALL* contain: • the filename and version number of the latest Snapshot metadata • One or more hashes of the Snapshot metadata file as well as the hashing function used

1. At the time of manufacture or installation, a reasonably recent copy of the required Uptane metadata should be provisioned to allow the ECU to determine if the remote repository is legitimate when downloading the metadata for the first time. In order to minimize the possibility of replay and rollback attacks, Secondary ECUs with partial verification capability *MUST* have the Root and Targets metadata from the Director repository. They *MAY* also contain the metadata from other roles or Image Repository if necessary.

2. Secondary ECUs with the ability of full verification, *MUST* have full set of metadata from both repositories, which is comprised of Root, Targets, Snapshot, and Timestamp. Additionally, they are required to have repository mapping metadata.

3. They must also have either current time or securely attested reasonably recent time.

4. An ECU key, which is a private key and unique to the ECU. This key is used for signing the ECU version report and decrypting the firmware image.

Vehicle Version Manifest

The vehicle version manifest, a metadata structure, *MUST* have the information detailed below:

- An attribute encompassing the payload signature(s), each of which is specified by:
  - The public key identifier of the key used for signing the payload
  - The method used for signing, such as ed25519
  - The payload's hash to be signed
  - The hash function employed, such as SHA-3 256
  - The signature of the hash

- A payload comprising of the following pieces of information:
  - VIN or a unique identifier of the vehicle
  - The unique identifier or serial number of the Primary ECU
  - The ECUs version reports list. Primary ECU's version report should also be included in this list.

ECU Version Report

This report is very similar to the vehicle version manifest outlined above. That is, it is a metadata structure containing the signature(s) specified by an attribute (identical to the one listed above) and a payload including the following:

- A unique identifier (e.g., a serial number) for the ECU
- Currently installed image's filename, length, and hashes
- An indication of any security issues detected
- The latest time that can be verified by the ECU at the time of generation of this report
- A counter or nonce for preventing replay of the ECU version report. This can be a cryptographically generated nonce similar to the one generated for Time Server. This value *MUST* not be reused.

Full vs. Partial Verification of Metadata

As indicated earlier, metadata verification can either be full or partial, depending on the capabilities of the ECUs. Figure 2.6 shows the step-by-step process of full verification. The process begins with loading and verifying the current time (or most recent attested time), leading to the step two which downloads the metadata form the Director Repository and carries on with checking if the metadata is correct. If either the download process or the check fails, the other steps are skipped and any errors are reported. All the subsequent steps follow the same pattern, that is, to proceed with the next step if the current step is successful, skip all remaining steps and jump to the error reporting, otherwise. The outcomes of the step four and step six may result in ending with no updates and exiting the process by skipping all the following steps. Each step is concerned with downloading and/or performing checks on different types of metadata. While full verification has several steps, partial verification process, in contrast, only includes a couple of steps, as it is limited to loading and verifying the current time or securely attested most recent time followed by downloading and checking Targets metadata from Director Repository. However, partial verification may optionally go beyond these required basic checks (if secondary ECUs are capable of doing so) by including additional checks. If all the checks and verification steps proceed successfully, the firmware binaries are downloaded and installed on the target ECU. We have provided an overview of the full and partial verification procedures in this section, for a more in-depth and extensive description please see the official design documentation of Uptane Framework that can be found at [39].

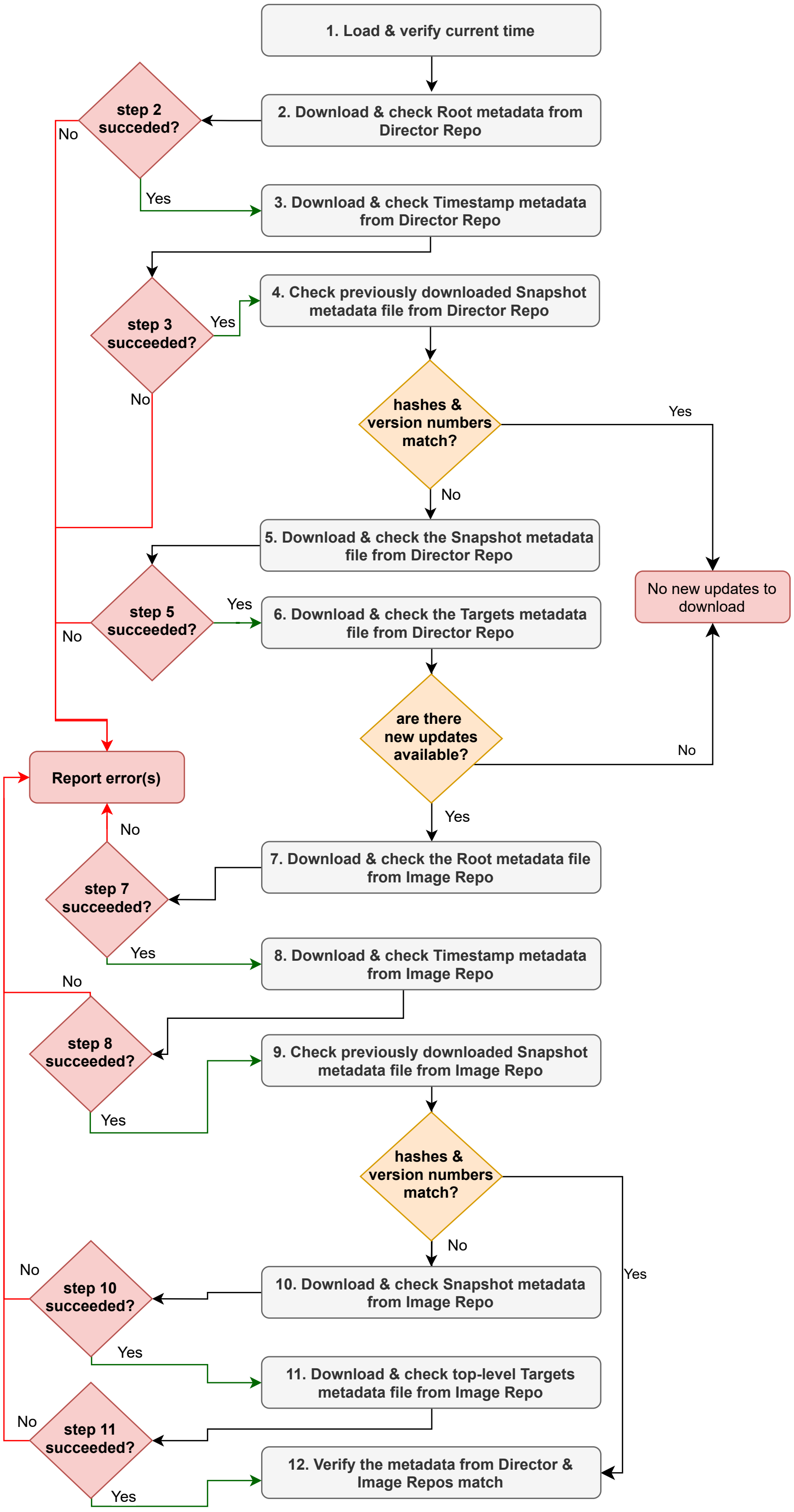

1. Load & verify current time
2. Download & check Root metadata from Director Repo
step 2 succeded?
No
Yes
3. Download & check Timestamp metadata from Director Repo
step 3 succeeded?
Yes
No
4. Check previously downloaded Snapshot metadata file from Director Repo
hashes & version numbers match?
Yes
No
5. Download & check the Snapshot metadata file from Director Repo
No new updates to download
step 5 succeeded?
Yes
No
6. Download & check the Targets metadata file from Director Repo
are there new updates available?
No
Yes
Report error(s)
No
7. Download & check the Root metadata file from Image Repo
step 7 succeeded?
Yes
8. Download & check Timestamp metadata from Image Repo
No
step 8 succeeded?
9. Check previously downloaded Snapshot metadata file from Image Repo
Yes
hashes & version numbers match?
No
Yes
No
step 10 succeeded?
10. Download & check Snapshot metadata from Image Repo
Yes
11. Download & check top-level Targets metadata file from Image Repo
No
step 11 succeeded?
12. Verify the metadata from Director & Image Repos match
Yes

Figure 2.6: *This diagram provides a graphical representation of the full verification (usually performed by the Primary ECU) of the metadata. The process proceeds with the successive step if the current step is successful, otherwise error(s) causing the failure are reported.*

2.2 In-Vehicle Networks and Systems

Most cars today come equipped with a variety of computing devices, known as Electronic Control Units (ECUs). A typical modern vehicle may contain a number of different ECUs, each of which has unique responsibilities for performing one or more functions of the vehicle. For example, one ECU may be responsible for detecting whether there is a passenger present in the vehicle, whereas another one may be monitoring the tyre pressure. In order to perform their duties correctly, these ECUs need to communicate with each other as well as external world [14].

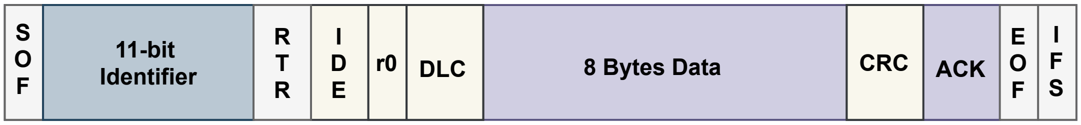


Figure 2.7: *CAN 2.0A data frame structure (redrawn from [44]).*

For local communication, ECUs rely on various automotive networking technologies including Controller Area Network (CAN, see Figure 2.8 for an example), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), and FlexRay. Each of these technologies has been designed to meet specific needs of a particular automotive application. For instance, MOST is a high-speed network technology to support audio, video, and voice data communications. Whereas, LIN is used in automotive applications requiring low network bandwidth and speed, such as mirror control and door lock/unlock features [45].

2.2.1 CAN Bus

The latest of the several versions of the CAN specification was released in 1991 by Bosch, which is version 2.0. There are two parts of the CAN specification, referred to as CAN 2.0A and CAN 2.0B, the former containing an 11-bit identifier as compared with the latter having a 29-bit identifier. The CAN specification breaks the messages into data frames, comprising of a certain structure as shown in the Figure 2.7. A brief definition of each term, representing a certain part of the frame, is given below:

- **Start of Frame (SOF)** Marking the Start Of Frame, SOF consists of a single dominant bit, used to synchronize the nodes on the bus.

- **11-bit Identifier** is used to express the priority of the message, where the lowest value denotes the highest priority and vice versa.

- **Remote Transmission Request (RTR)** is a single bit, which is dominant when information from another ECU is needed. The request is received by all the nodes; however, the target node is determined by the identifier. While all nodes receive the data contained in the response, it is consumed by the relevant nodes only.

- **Identifier Extension (IDE)** is also a single bit, which indicates a standard CAN identifier is being sent with no extension.

- **Eight-Bytes Data**, as the name suggests, this field represents the data (up to 8 bytes long) to be transmitted.

- **Cyclic Redundancy Check (CRC)** carries the 2 bytes of BCH error-correcting checksum for error detection purposes.

- **Acknowledgement (ACK)** is a recessive bit (it is two bit long, first is used for acknowledgement and the second one for a delimiter), which is overwritten to become a dominant bit in the original message by every node receiving an accurate message to indicate an error-free message has been transmitted. The message is discarded by a receiving node if an error is detected, thereby leaving this bit in the recessive state, causing the sending node to resend the message after re-arbitration. This is how the Integrity of the data is acknowledged by each node.

- **End of Frame (EOF)** (a seven-bit field) indicates the end of the frame. It disables bit stuffing and indicates a stuffing error when in dominant state. A bit of the opposite logic level is stuffed into the data when during the normal operation five bits of the same logic appear in succession.

- **Inter-Frame Space (IFS)** is comprised of seven bits, it contains the time needed by the controller to move a received frame to the appropriate position in a message buffer area.

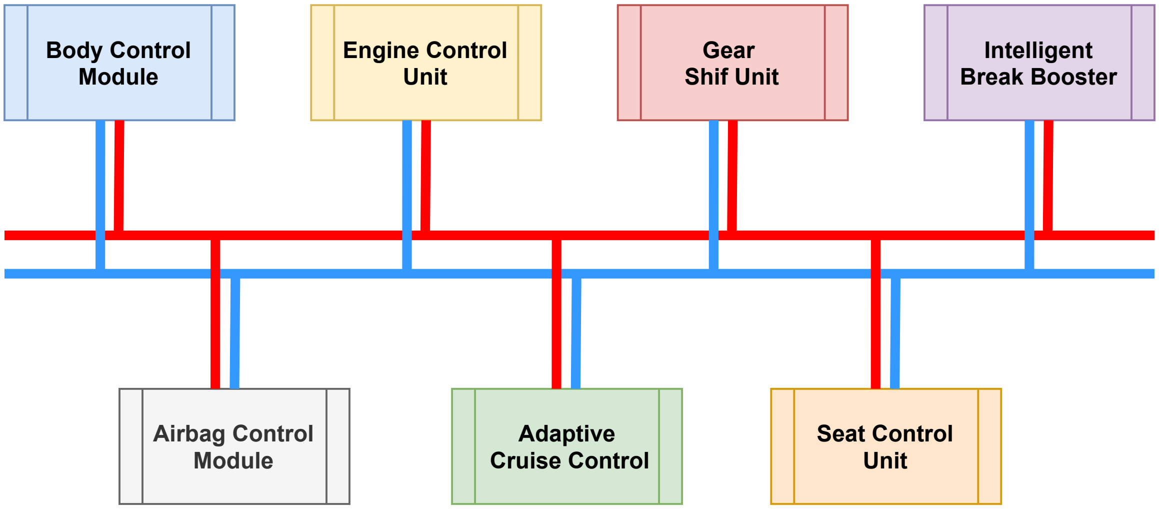


Figure 2.8: *An illustration of the CAN bus with some typical ECUs found in a modern car.*

2.2.2 LIN

Founded in the late 1990s by five automobile manufacturers (Audi, BMW, Mercedes-Benz, Volkswagen Group and Volvo Cars), Local Interconnect Network (LIN) is an inexpensive vehicular communication protocol that utilises what is commonly known as a master-slave model, wherein a single bus is shared by a master node and up to 16 slave nodes. A slave node cannot send a message unless it has previously been asked by the master node to do so. Some of the key characteristics of the protocol include:

- Up to 20 kbit/s speed
- Variable length of data frame
- Time-synced multi-cast reception
- Ability of faulty nodes detection
- Flexible configurations
- Error-detection and checksum capabilities
- Low-cost silicon implementation
- Guaranteed latency times

As noted above, LIN is primarily used in the applications where safety and performance are not key considerations.

2.2.3 FlexRay

Developed by FlexRay Consortium, FlexRay is an automotive communications protocol, offering enhanced data rates (up to 10 Mbit/s) to enable applications requiring higher speed. Figure 2.9 shows format of the FlexRay frame.

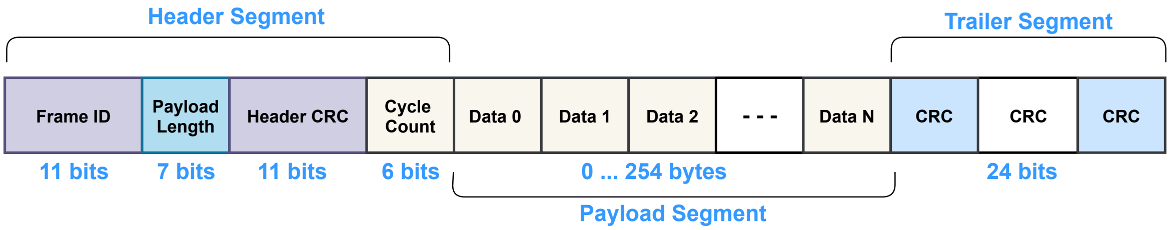


Figure 2.9: *FlexRay Frame Format (redrawn from [1]).*

Listed below are some of the key features of this vehicular protocol:

- Supports Party Line and Start topologies

- High-speed serial communication
- Fault-tolerant communication
- Supports both time-triggered and event-triggered schemes

While FlexRay was introduced as a successor to CAN, its widespread adoption is adversely affected by the higher production costs.

2.2.4 MOST

Media Oriented Systems Transport (MOST) is used to carry multimedia data into the car via optical fiber. MOST is a synchronous network offering multiple data channels as well as a control channel used to set up which data channels a sender and a receiver will use. Synchronous data channels are used to transfer streaming data (such as audio or video signals) but MOST also implements asynchronous data transfer mechanisms (for example while retrieving data from the Internet) by using some dedicated channels. It offers rates going up to 24Mb/s.

MOST is a high-speed (up to 24 Mbit/s data rate) multimedia network technology that uses a ring or daisy-chain topology for transferring multimedia (i.e., video, audio, voice, etc.) data via plastic optical fiber.

Being a synchronous network type, MOST provides the ability to transfer streaming data (such as video and audio) using synchronous data channels.

We outline some of the main characteristics of this technology below:

- Ability to manage up to 64 in a ring topology
- Plug and play capability - allowing easy installation/removal of MOST devices
- Support redundant double ring configuration for safety-critical applications
- Timing Master - maintains continuous MOST frames into the ring.

2.2.5 In-vehicle Infotainment

An In-Vehicle Infotainment (IVI) or automotive infotainment system is an integrated unit (a combination of hardware and software), providing information services and entertainment functionality to the driver and other vehicle occupants for an enhanced in-vehicle experience.

While legacy entertainment systems were quite simple (consisting of cassette/CD players and radios), modern IVIs offer a sophisticated set of capabilities, including navigation systems, USB/Bluetooth/Wi-Fi connectivity, and video/DVD players, touchscreen displays etc. Following is a brief overview of the various components of a typical IVI system:

Integrated Head Unit: Mounted on the car's dashboard, a head unit provides a user-friendly HMI (Human-Machine Interface) by means of a high-resolution touch-screen display unit, acting as the control centre for the infotainment system.

Heads-up Display Integrated with car's windshield, a heads-up display is transparent screen - a latest technology that shows real-time vehicle information (speed, digital cluster, navigation maps etc.) to the driver.

Digital Instrument Cluster Replacing the old-fashioned, static displays of the vehicle's instruments with the digital instrument clusters for displaying speed and tachometer, speedometer, odometer etc.

2.2.6 Onboard Diagnostic (OBD) Port

Modern vehicles have Onboard Diagnostic (OBD) ports inside them that are used for ECU firmware updates, vehicle repairing and inspections. Implementation of these ports is obligatory since 1998 in the USA and since 2001 for gasoline-powered vehicles and since 2003 for diesel-powered vehicles in the EU respectively [46]. Onboard Diagnostic is mainly used for reporting the data gathered by various sensors in the car to the outside world, providing information on the health status of the vehicle. This information is often used by service providers for fixing any reported problems [47]. Since inexpensive OBD dongles are readily available in the market, attackers can leverage them as an entry point for breaking into in-vehicle networks.

2.3 Attack Trees

Originally proposed by Schneier [48], attack trees are a formal, structured approach for describing threats to a system by means of multi-levelled, hierarchical diagrams facilitating the visualisation of various ways attackers can use to attack a system or an asset. An attack tree is drawn following a top-down approach; that is, with the root node at the top of the tree representing an overall goal of the attacker. Primitive actions carried out by the attackers are depicted by the lowest level of nodes, called leaf nodes. Inner nodes sitting between the root node and leaf nodes represent the sub-goals of the attacker that contribute to the overall goal.

The root node and each intermediate node specify the refinement of their corresponding subtrees (see Figure 2.10). The refinement can either be disjunctive (**OR**) - depicted by no symbol, conjunctive (**AND**) - depicted by an arc, or a variant of the conjunctive refinement, the Sequential AND (**SAND**) - depicted by an arc with an arrowhead. The disjunctive or OR refinement is used to specify that at least one of the steps (leaf nodes) is complete in order for the parent node to be considered complete. Conjunctive refinement, on the other hand, requires that all steps are complete

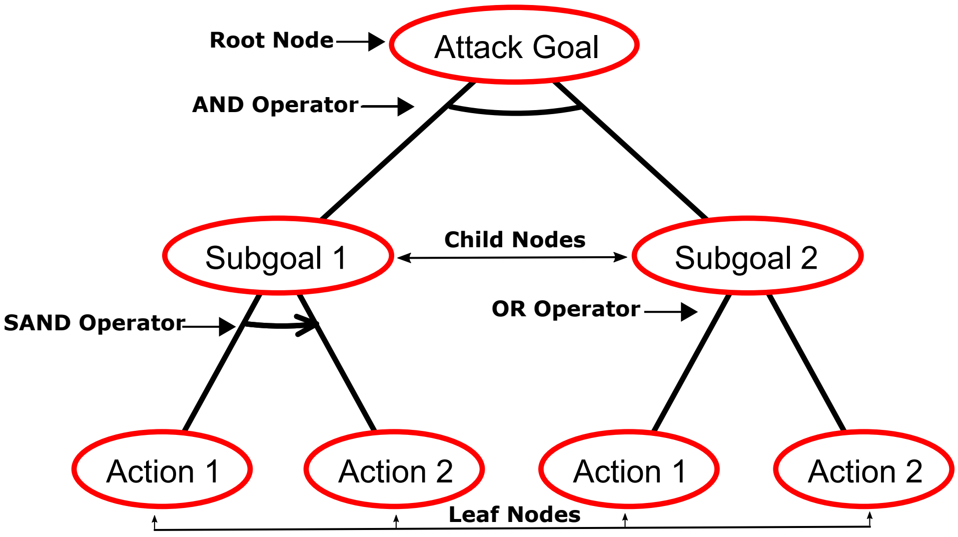


Figure 2.10: *Example Attack Tree: This example attack tree depicts a root, two children, and four different leaf nodes. Annotations indicate the type of refinement (i.e., AND, OR, SAND operators), type of role that is assumed by each of the node types.*

before an attack can be considered complete. Finally, steps of a **SAND** tree must be performed in the specified order for the attack to succeed.

Definition 2.3.1. With $\mathbb{A}$ denoting a set of potential atomic actions of the attacker, the elements of attack trees are $\mathbb{A} \cup \{\mathbf{OR}, \mathbf{AND}, \mathbf{SAND}\}$; the following grammar can be used to construct an attack tree (where $a \in \mathbb{A}$):

$$t ::= a \mid \mathbf{OR}(t, \ldots, t) \mid \mathbf{AND}(t, \ldots, t) \mid \mathbf{SAND}(t, \ldots, t)$$

The set of all attack trees is denoted by $\mathbb{T}$, which is consistent with the formalisation of the attack trees proposed in [49, 50].

Example 1: Figure 2.11 presents an example attack tree with the overall goal of delivering a malicious update to an IoT device. It is to be noted that the overall attack tree is a **SAND** tree with the first child node (on the left) being an **AND** subtree and the second a leaf node (representing an atomic attacker action). The sequential **AND** refinement has been applied because the ordering of the actions execution is important. The **AND** subtree is composed of two children nodes, one of which is in turn is an **OR** subtree and the other one a leaf node. As preparing the malicious update needs access to the cryptographic keys for signing the software (SW) as well as crafting (i.e., writing the malicious program) the update; hence, an **AND** or conjunctive refinement was the most appropriate choice. Note that both of these actions can be performed in any order. Since compromising cryptographic keys can be accomplished in two different ways (i.e. by bribing the admin or stealing the keys); therefore, a disjunctive (or an **OR**) refinement has been used to express this scenario. Following is a representation of the attack tree displayed in Figure 2.11, which has been written using the term notation introduced above. Please note that for convenience, only keywords have been used to denote attacker atomic actions (i.e. $a \in \mathbb{A}$).

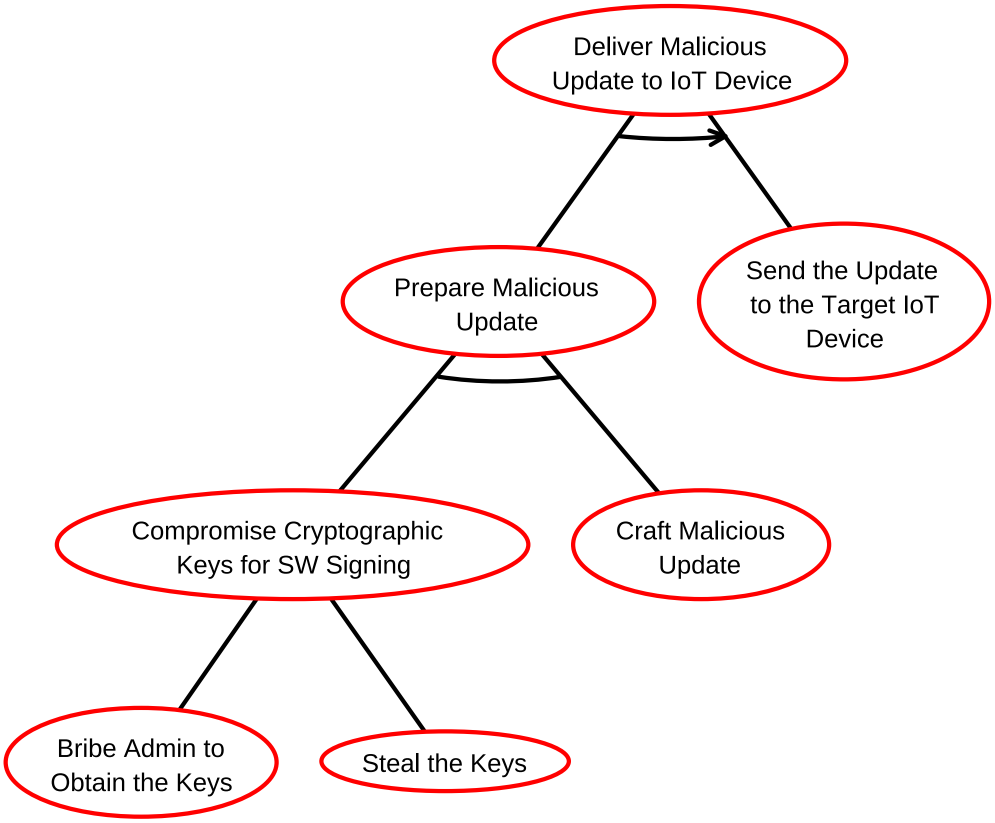

Deliver Malicious Update to IoT Device
Prepare Malicious Update
Send the Update to the Target IoT Device
Compromise Cryptographic Keys for SW Signing
Craft Malicious Update
Bribe Admin to Obtain the Keys
Steal the Keys

Figure 2.11: *An example **SAND** attack tree constructed following the grammar defined above. This overall **SAND** aims at delivering a malicious update to ta an IoT device.*

$$t = \textbf{SAND}\left(\textbf{AND}\left(\textbf{OR}\left(Bribe\ Admin, Steal\ Keys\right), Craft\ Update\right), Send\ Update\right)$$

Prior studies have provided formal semantics for attack trees, some of which include multiset semantics introduced by Mauw and Oostdijk in [50], set semantics defined by Bossuat and Kordy in [51], path semantics formalised by Audinot et al. in [52], serial-parallel graph semantics presented by Jhawar et al. in [49], and sequence semantics by Mantel and Probst introduced in [53]. In order to present the notion of how attack trees can be formally expressed and interpreted, we present the SP Graph Semantics in the following subsection, and the Sequence Semantics in Chapter 6 (where, which are more closely related to our own work presented in this thesis for deriving test cases by attack-tree analysis. It is noteworthy that, as noted in [54], these semantics (i.e. GP Graph and Sequence semantics) are equivalent.

2.3.1 SP Graph Semantics

Attack trees can be interpreted as a set of series-parallel (SP) graphs [49] for the purpose of defining the semantics of the attack tree. With two unique vertices source and sink (where source is a vertex with no incoming edges and sink is a vertex that has no outgoing edges), an SP graph is an edge-labelled directed graph that can be constructed with two operators for sequential and parallel composition of graphs. A definition of the source-sink graphs is required before defining SP graphs. Following formal definitions of source-sink graphs and SP graphs have been used from [49]:

Definition 2.3.2. A source-sink graph over $\mathbb{A}$ is a tuple $G = (V, E, s, z)$ where V is a set of vertices, E is a multiset of edges with support $E* \subseteq V \times \mathbb{A} \times V$, $s \in V$ is a unique source and $z \in V$ is a unique sink, and $s \neq z$.

The sequential composition of two source-sink graphs G and G' (denoted by $G \cdot G'$) is the graph that results from taking the disjoint union of G and G' and concatenating the sink of G with the source of G'. Hence, if we denote the disjoint union by $\dot{\cup}$ and the multiset of E with $E^{[s/z]}$, where vertices s replace z, we can then define $G \cdot G'$ as:

$$G \cdot G' = (V\{z\} \dot{\cup} V', E^{[s'/z']} \dot{\cup} E', s, z')$$

Denoted by $G \mid\mid G'$, parallel composition is similar to the sequential composition; the only difference is that two sources and two sinks are identified. This can be formally defined as:

$$G \mid\mid G' = (V \ \{\text{s, z}\} \dot{\cup} V', E^{[s'/s, z'/z]} \dot{\cup} E', s', z')$$

Now that the definition of the source-sink graph has been presented, we proceed with providing the definition of the SP graphs, which will help us define the attack-trees formal semantics by interpreting them as SP graphs.

Definition 2.3.3. The set $\mathbb{G}_{SP}$ over $\mathbb{A}$ is defined inductively by these rules:
For $a \in \mathbb{A}$, $\xrightarrow{a}$ is an SP graph,
If G and G' are SP graphs, then so are $G \cdot G'$ and $G \mid\mid G'$.

Example 2: The following construction corresponds to the SP graph shown in the Figure 2.12.

$$\xrightarrow{a} . \left(\left((\xrightarrow{b} \mid\mid \xrightarrow{c} \mid\mid \xrightarrow{d}) . \xrightarrow{f} \right) \mid\mid \xrightarrow{e} \right)$$

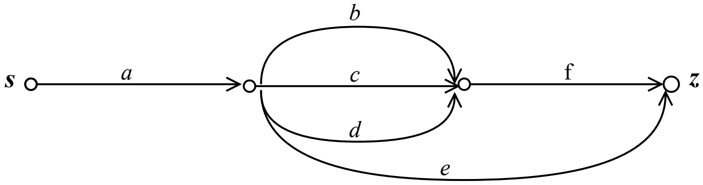


Figure 2.12: *An example SP graph, with the source represented by* s *and sink represented by* z*.*

The formal definition presented below shows how the full SP semantics of a given attack tree can be defined.

Definition 2.3.4. The SP graph semantics for attack trees in $\mathbb{T}$ can be given by the following function:

$$[\![\cdot]\!]_{SP} : \mathbb{T} \rightarrow P(\mathbb{G}_{SP})$$

This is defined recursively. If $a \in \mathbb{A}$, $t_i \in \mathbb{T}$, and $1 \le i \le k$, then

$[\![a]\!]_{SP} = \{\xrightarrow{a}\}$
$[\![\mathbf{OR}(t_1, ..., t_k)]\!]_{SP} = \bigcup_{i=1}^{k} [\![t_i]\!]_{SP}$
$[\![\mathbf{AND}(t_1, ..., t_k)]\!]_{SP} = \{G_1 \,||\, ... \,||\, G_k \mid (\, G_1, ..., \, G_k) \in [\![t_1]\!]_{SP} \times ... \times [\![t_k]\!]_{SP}\}$
$[\![\mathbf{SAND}(t_1, ..., t_k)]\!]_{SP} = \{G_1 \, \cdot ... \cdot \, G_k \mid (G_1, ..., \, G_k) \in [\![t_1]\!]_{SP} \times ... \times [\![t_k]\!]_{SP}\}$
where $[\![t]\!]_{SP} = \{G_1, ..., G_k\}$ corresponds to a set of possible attacks G_i

Example 3: The following SP semantics corresponds to the attack tree *t* presented in the Figure 2.13. As can be seen, this overall **AND** attack tree has two subtrees, each with two different leaf nodes for representing atomic actions. We use this simple attack tree to show SP semantics of the attack tree. Since the overall tree is an **AND**, hence both of its children subtrees must be complete. It can be observed that the **OR** subtree on the left has two leaf nodes denoted by a and b, representing two different possible ways to achieve the subgoal of this particular subtree. The **SAND** tree on the right also has two leaf nodes denoted by c and d, both of which must be performed in the given order in order for the parent subtree to be considered complete. Overall, either of the actions from the first subtree (i.e., **OR** subtree) needs to be combined with both of the actions in the second subtree (i.e., **SAND** subtree) to achieve the overall goal of the parent **AND** tree.

$$[\![t]\!]_{SP} = \{\xrightarrow{a} \,||\, \xrightarrow{c} . \xrightarrow{d}, \xrightarrow{b} \,||\, \xrightarrow{c} . \xrightarrow{d}\}$$

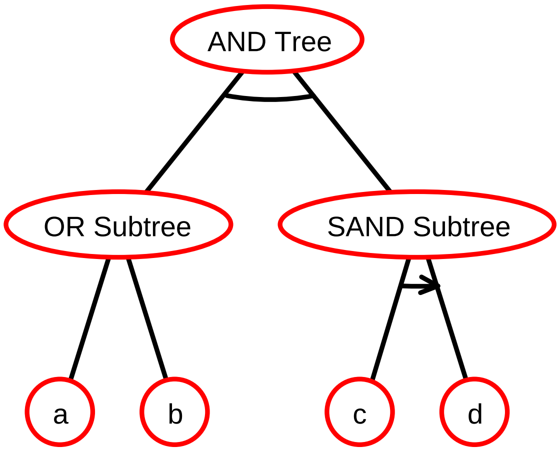


Figure 2.13: *An example* ***AND*** *with two different subtrees. See associated SP semantics above.*

2.4 Threat Modeling

Threat modeling is a security analysis technique that helps identify risks by using abstractions [55]. It plays a vital role in automotive security engineering by facilitating in the identification of potential threats and relevant defensive mechanisms. In particular, threat modeling helps model the system and its trust assumptions as well as aid in modeling adversaries in order to understand their motivations, capabilities, tactics, techniques, and procedures.[56]. Several threat modeling methods and techniques have been proposed, some of which are briefly described below:

2.4.1 CORAS

[57] - based on Australian Risk Management Standard AS/NZS 4360:2004 - is a threat modeling and specification language, compromising of five main activities: 1) establishing the context, 2) identifying risks, 3) analysing risks, 4) evaluating risks, and 5) treating those risks. It uses specialized UML use case diagrams for modeling threats and undesirable behaviours.

2.4.2 PASTA

PASTA or Process for Attack Simulation and Threat Analysis [58] is a seven-stage threat modeling framework aiming at providing an attacker-centric view of the system, which can aid in developing relevant, effective mitigation strategies against cyber threats ad attacks. The seven stages of the PASTA include: 1) define the objectives, 2) define the technical scope, 3) decompose the application, 4) analyse the threats, 5) analyse vulnerabilities and weaknesses, 6) model the attacks, and 7) analyse risk and its impact.

2.4.3 T-MAP

T-MAP [59] is an attack-path-analysis based threat modeling method for the quantification of the security threats based on the total severity weights of the relevant security paths for commercial-off-the-shelf (COTS) systems. UML class diagrams are used for developing the attack path models. Four different class diagrams are generated for each step: access class diagram, vulnerability class diagram, target asset class diagram, and affected value class diagram.

2.4.4 STRIDE

STRIDE is a cybersecurity threat identification and classification model providing a structured approach for grouping threats into six threat categories: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege [60], as summarized in Table 2.4. While the threat modeling methodologies/systems introduced above have their own merits, in this study we use STRIDE for the following key reasons: Firstly, STRIDE is a mature, well-known threat modeling tool that is extensively used in both automotive industry and research settings. Secondly, STRIDE model is an extension of the familiar and famous CIA triad (Confidentiality, Integrity, Availability). Finally, STRIDE has an associated threat-modeling tool, as explained in detail below (providing the ability to use specialised, extensible automotive-specific templates) that we leverage in this study for threat enumeration, which contributes to our attack-tree construction process.

2.4.5 Threat Modeling Tool

Microsoft's Threat Modeling Tool (TMT), which was originally developed for supporting its well-known Secure Development Lifecycle (SDL) process for a systematic identification of various security threats during the initial phases of the software development lifecycle [61]. TMT, as its name implies, allows security analysts to model the target system by employing one of the standard notations, known as Data Flow Diagram (DFD), for visualising system components, information flows and trust boundaries for the identification and classification of various threats using the STRIDE threat classification model. The threat modeling tool has been around for several years, and its adoption and application has not been confined to secure software engineering only, it is also widely used to support security assessments in other domains, including automotive cybersecurity testing, such as [56, 62, 63, 28].

Figure 2.14 shows the standard symbols used for modeling data flow diagrams. A data flow depicted by a one-sided arrow, is used to represent flow of information from one element to another. A process, depicted using a circle, is used to represent computations or running programs. A data store is depicted by two horizontal parallel lines and is used to represent files, databases, and registry keys. External Interactors, also known as external entities, are external actors, providers, and consumers of the

Table 2.4: *An overview of the STRIDE model summarizing threat categories and relevant affected security proprieties.*

Threat		Definition	Property
Spoofing identity	S	Assuming the identity of a human or non-human system entity for achieving a malicious goal	Authentication
Tampering of data	T	Making unauthorized changes to data or code	Integrity
Repudiation	R	Refusing to accept the responsibility of a performed action	Non-repudiation
Information disclosure	I	Disclosing confidential information to unauthorized parties	Confidentiality
Denial of service	D	Causing disruption to a system service so users cannot access or use it	Availability
Elevation of privileges	E	Obtaining higher level of privileges than originally granted	Authorization

data from the system. Finally, a trust boundary, depicted by a dotted line, is an extension to the standard DFDs for showing separations between trusted and untrusted components.

2.5 Risk Assessment

The process involving identification, estimation, and prioritization of risks to the organizational operations, assets, and individuals is commonly referred to as *risk assessment*. There are two key factors that are used to determine the level of risk: *likelihood* and *impact*. Likelihood - a weighted risk factor - that refers to the probability of a given threat's capability of exploiting one or more vulnerabilities. Whereas the magnitude of the harm or damage is referred to as the impact [64].

2.5.1 Assessment Approaches

While the overall security risk assessment process is generally well defined, its subprocesses can be implemented in different ways based on the context and specific needs of a particular organization. As a result of this flexibility, a plethora of risk assessment

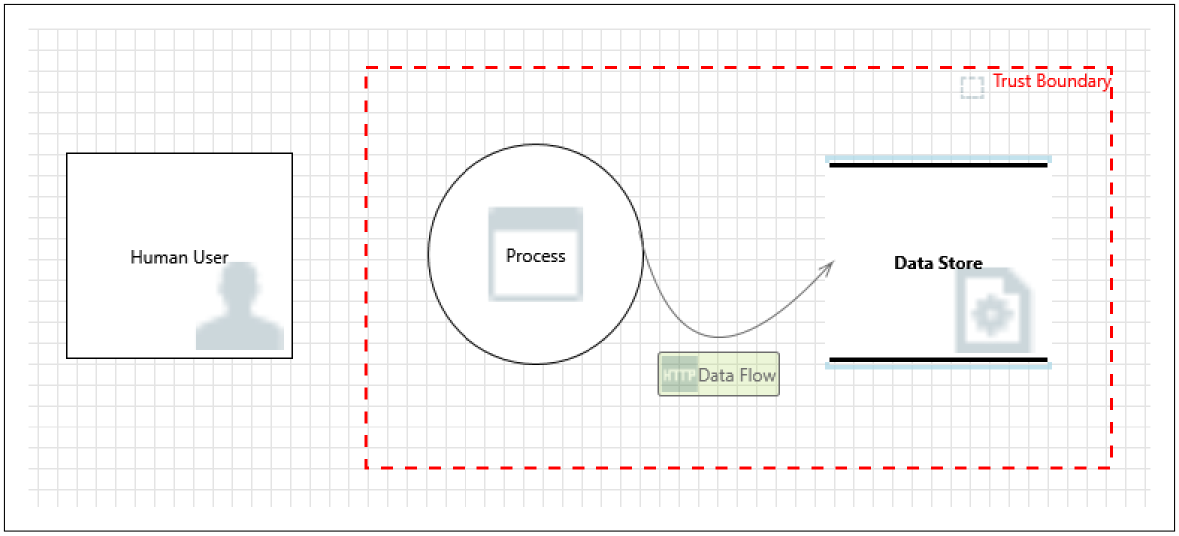


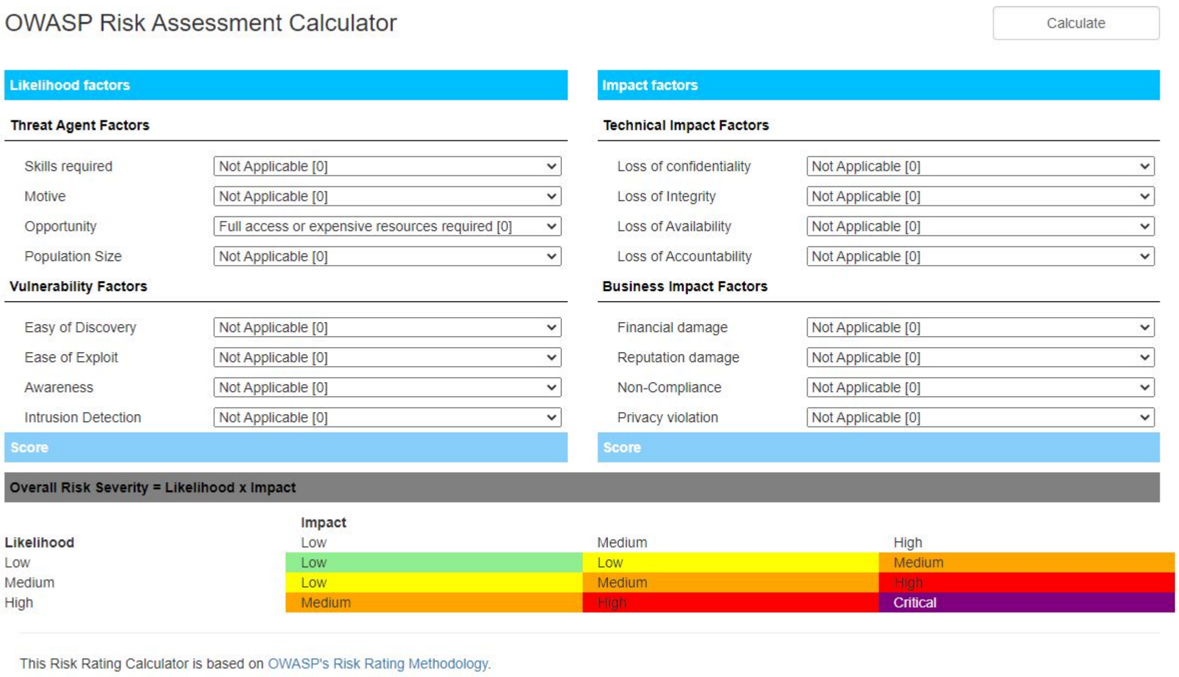
Figure 2.14: *This figure shows different symbols used in Microsoft's Threat Modeling Tool for drawing Data Flow Diagrams in order to enumerate various potential threats.*

tools, techniques and approaches have been introduced, including the well-known NIST SP800-30, OCTAVE, and OWASP Risk Rating Methodology. Some risk assessment approaches (such as NIST SP800-30) are threat oriented - i.e. as first steps, they involve identification of the threat sources/events followed by identification of exploitable vulnerabilities along with the probability and impact of the threat events. In contrast, other approaches (such as OCTAVE) are asset oriented - i.e., they focus on identifying critical assets before determining how those assets can be targeted and the magnitude of the resultant damage [65].

There are numerous methods (i.e., quantitative, qualitative, semi-quantitative) that can be used for the assessment of the risk and its associated contributing factors, each with certain merits and limitations. Approaches employing quantitative methods use numerical values - where the meaning and proportionality of the values are used consistently both inside and outside of the assessment context. While these types of assessments are proved to be effective for cost-benefit analyses of alternative risk treatment options, the meaning of the assessment results may not always be clear and may suffer from subjective interpretations. Qualitative approaches, on the other hand, assess the risk by employing non-numeric categories (e.g., low, moderate, high). While such approaches are effective in terms of communicating the assessment results to the stakeholders/decision makers, the range of values in qualitative approaches is often small as compared to quantitative assessments, resulting in difficulties of prioritization/comparison of reported risks. Annotations and use of tables can enhance the repeatability and reproducibility of the qualitative assessments. Finally, semi-quantitative approaches - offering the benefits of both the quantitative and qualitative approaches - use scales, bins or representative numbers whose meanings and values are not maintained in other contexts [64].

2.5.2 OWASP Risk Assessment Tool

This study uses the OWASP risk assessment tool (shown in Figure 2.15) [66] - a web-based tool that uses OWASP Risk Rating Methodology [67], which is comparatively simple, easy to use, and can be customised. Similar to the risk assessment approach introduced in the ISO/SAE 21434 standard, this tool calculates the overall risk severity by considering various factors (i.e. threat agent factors, vulnerability factors, technical impact factors, and business impact factors). A brief overview of each of these factors is presented below.



OWASP Risk Assessment Calculator

Likelihood factors

Threat Agent Factors

Skills required: Not Applicable [0]
Motive: Not Applicable [0]
Opportunity: Full access or expensive resources required [0]
Population Size: Not Applicable [0]

Vulnerability Factors

Easy of Discovery: Not Applicable [0]
Ease of Exploit: Not Applicable [0]
Awareness: Not Applicable [0]
Intrusion Detection: Not Applicable [0]

Score

Impact factors

Technical Impact Factors

Loss of confidentiality: Not Applicable [0]
Loss of Integrity: Not Applicable [0]
Loss of Availability: Not Applicable [0]
Loss of Accountability: Not Applicable [0]

Business Impact Factors

Financial damage: Not Applicable [0]
Reputation damage: Not Applicable [0]
Non-Compliance: Not Applicable [0]
Privacy violation: Not Applicable [0]

Score

Overall Risk Severity = Likelihood x Impact

	Impact		
Likelihood	Low	Medium	High
Low	Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

This Risk Rating Calculator is based on OWASP's Risk Rating Methodology.

Figure 2.15: *OWASP Risk Assessment Tool.*

Probability or Likelihood Factors: These include attacker's skill level (i.e., level of technical expertise of the threat agent), motive (i.e., how motivated the adversary is), opportunity (i.e, what resources are required for the exploitation of the vulnerability), population size (i.e., how large is the threat agents group), ease of discovery (i.e., how easy is the discovery of this vulnerability by the threat agent group), ease of exploitation (i.e., how easy it is for the threat agents group to exploit this vulnerability), awareness (i.e., how well the adversaries know this vulnerability), and intrusion detection (i.e.,likelihood of the detection of the exploitation) [68].

Impact Factors: These factors include loss of confidentiality, integrity, availability, and accountability of the concerned asset. Other factors include the extent of the financial and reputation damage as well as any non-compliance and privacy violations.

2.6 Model-Based Security Testing

MBST is concerned with specifying, documenting and generating security test objectives, test cases, and test suites (where a test case involves validating whether the system is working as expected, and a test suite is simply a set of such test cases grouped together for execution purposes [69]) in a systematic and efficient manner [70]. MBST primarily uses models to verify if the target system meets its security requirements [71]. Although, MBST is relatively a new area of research, there are few studies employing it for the security testing of embedded and concurrent systems (involving cyber-physical components), a couple of which are presented below:

Santos et al. [72] propose their automotive cybersecurity testing framework, which uses Communicating Sequential Processes (CSP) for representing the models of the vehicle's bus systems as well as a set of attacks against these systems. CSP - a language with its own syntax and semantics - is a process-algebraic formalism used to model and analyze concurrent systems. Using CSP, they create architectures of the vehicle's network and bus systems along with the attack models. One of the key challenges that authors claim to address in their work is the scalability of the testing in distributed environments. Their system model is comprised of networks, bus systems connected to each network, and the gateways. Additionally, network parameters, such as latency can also be modelled. An attack model is also created, defining the attackers' capabilities as channels. An attacker's capabilities may include command spoofing, communication disruption, eavesdropping and influencing behaviours of the system. According to the authors, the ability for a detailed definition of the scope of the attack and test cases is a key advantage of using these models for security testing.

Wasicek et al. [73] present Aspect-Oriented Modeling (AOM) as a powerful technique for security evaluation of Cyber-Physical Systems (CPS), especially focusing on safety-critical elements in automotive control systems. AOM is based on the ideas inspired by aspect-oriented programming, which is concerned with crosscutting aspects being expressed as concerns (e.g., security, quality of service, caching etc.) [74]. Aspect-oriented modeling is used to express crosscutting concerns at a higher level of abstraction by means of modeling elements [75]. The technique presented by [76] models attacks as aspects, and aims at discovering and fixing potential security flaws and vulnerabilities at design time, because it becomes highly costly to find and fix the bugs if they are discovered later in the development life-cycle stages for automotive systems. Some of the main benefits that can be achieved by using AOM for security assessment of automotive systems include: separation of functional and attack models into aspects allows domain experts to work on different aspects without any interference; real-world attack scenarios involving high degree of risks can be modelled easily;

general models can be reused in other systems. An automotive case study is presented by the authors, involving the adaptive cruise control system as an example. They use a special modeling and simulation framework, called Ptolemy II, for developing their models. The authors intended to explore the effects of attacks on the communication between two vehicles. A discussion of four different attacks (i.e., man-in-the-middle, fuzzing, interruption, and replay) is presented.

Both of the works presented above rely on the models of the systems rather than performing any practical testing involving test scripts to be executed against the target systems, providing no observable insights on any behavioural/functional changes in the system under test. The approach applied in this thesis, on the other hand, uses threat models for automated derivation and execution of security test cases by combining the MBST with penetration testing.

2.7 Chapter summary

In this chapter, we provided relevant and essential background information for the work presented in this thesis, exploring various relevant areas; starting with an extensive overview of the automotive OTA updates, we described various aspects of this modern technology by citing some the key developments in the area; following this, we introduced the attack trees along with two types of formal semantics defined in the literature, as they provide the basis for our test case generation approach; additionally, a brief overview of the threat modeling, focusing on STRIDE methodology and related tool that are used extensively for the threat enumeration in this work is also provided; finally, we included a summary of a couple of studies on MBST approaches proposed for the security testing of embedded and safety-critical systems, highlighting their limitations.

Chapter 3

Literature Review

As suggested by its title, this chapter presents an overview of the findings, insights, and state-of-the-art from the automotive cybersecurity domain as a result of the literature review conducted over several months of this doctoral research. To begin with, a number of major intrusion points exposed by a connected car are described along with an overview of the relevant security threats and associated countermeasures and mitigations, citing new developments in the field. Following this, a comprehensive survey of the testbeds and testing approaches proposed in the related scientific literature is presented, by critically analysing their characteristics and highlighting the merits and demerits of each. Figure 3.1 presents a graphical overview of this chapter's structure, providing a breakdown of the sections and corresponding subsections.

3.1 Automotive Cybersecurity: State of the Art

Software flaws or vulnerabilities in the vehicle can lead to serious consequences, ranging from incidents of information theft to life-threatening situations. Numerous previous studies show how a vehicle can be maliciously controlled by exploiting one or more weaknesses in its software systems. Koscher et al. [7] demonstrate that it is possible for an adversary to maliciously influence a car's behaviour (e.g., engaging or disengaging its brakes) if they are able to access the car's internal network.

A connected vehicle may have several internal and external connections for accomplishing various important tasks. While these connections support correct functioning of different applications in the car, they can be exploited by cybercriminals to launch cyberattacks targeting various digital systems in the vehicle. Some of the external connections to the vehicle include cellular network, Wi-Fi, Bluetooth, Keyless Entry System (KES), and Tyre Pressure Management System (TPMS). Whereas, OBD, USB, and in-vehicle infotainment are some of the internal connections [77]. This section provides an overview of some of the common cybersecurity threats and risks faced by modern automobiles.

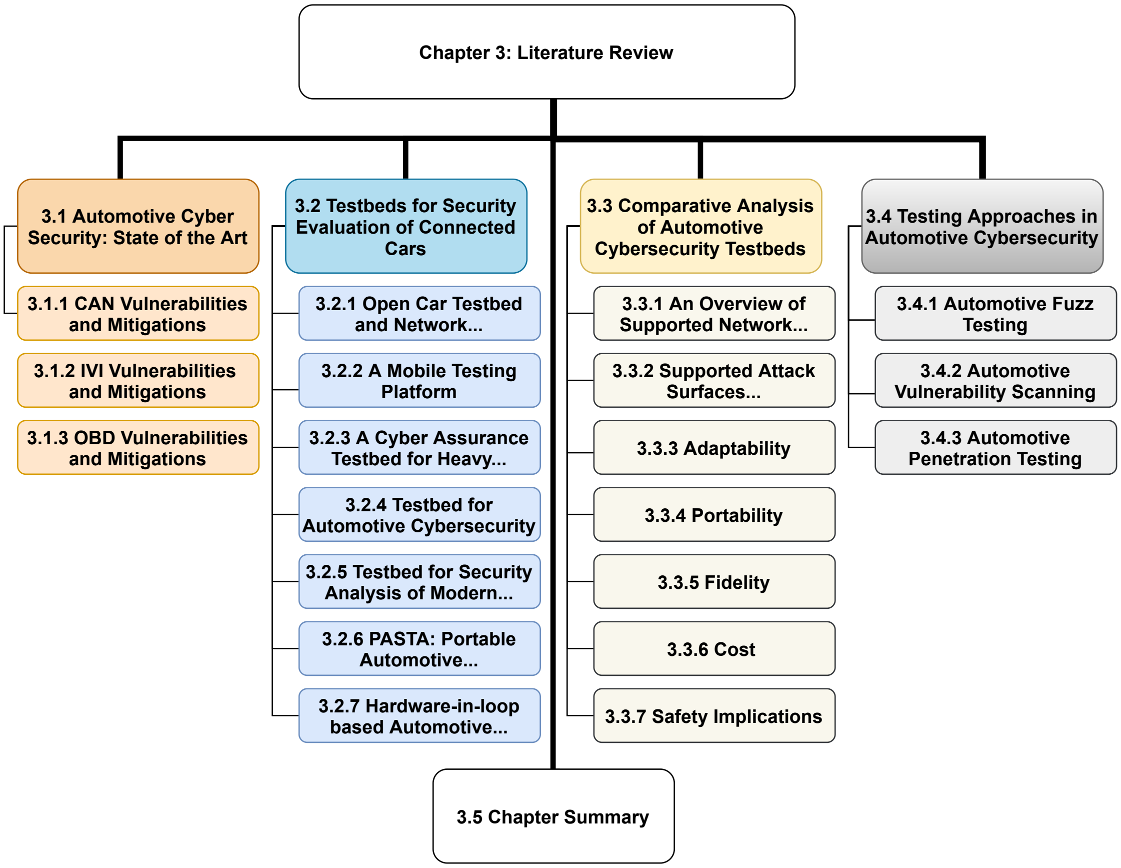

Chapter 3: Literature Review
3.1 Automotive Cyber Security: State of the Art
3.1.1 CAN Vulnerabilities and Mitigations
3.1.2 IVI Vulnerabilities and Mitigations
3.1.3 OBD Vulnerabilities and Mitigations
3.2 Testbeds for Security Evaluation of Connected Cars
3.2.1 Open Car Testbed and Network...
3.2.2 A Mobile Testing Platform
3.2.3 A Cyber Assurance Testbed for Heavy...
3.2.4 Testbed for Automotive Cybersecurity
3.2.5 Testbed for Security Analysis of Modern...
3.2.6 PASTA: Portable Automotive...
3.2.7 Hardware-in-loop based Automotive...
3.3 Comparative Analysis of Automotive Cybersecurity Testbeds
3.3.1 An Overview of Supported Network...
3.3.2 Supported Attack Surfaces...
3.3.3 Adaptability
3.3.4 Portability
3.3.5 Fidelity
3.3.6 Cost
3.3.7 Safety Implications
3.4 Testing Approaches in Automotive Cybersecurity
3.4.1 Automotive Fuzz Testing
3.4.2 Automotive Vulnerability Scanning
3.4.3 Automotive Penetration Testing
3.5 Chapter Summary

Figure 3.1: *Graphical Overview of the Chapter.*

3.1.1 CAN Vulnerabilities and Mitigations

Although CAN is a robust and fault-tolerant network technology, it lacks any security mechanisms because it has not been designed with security in mind [78]. Therefore, it is vulnerable to numerous cybersecurity threats (such as, eavesdropping/man-in-the-middle attacks compromising confidentiality and privacy, denial-of-service attacks affecting availability of the information and services, spoofing attacks compromising integrity and authenticity) which have been reported in many existing studies, such as [79, 19, 80].

Chen and Lin [81] present two different DoS attacks on automotive CAN bus by exploiting the major inherent flaws in the CAN protocol for demonstarting the impact of such attacks on the CAN bus efficiency. Their experimental results suggest that the attacks had significant impact on the bus efficiency. Furthermore, they also conclude that the entire CAN network could be jeopardized by a single attacker by injecting fake can messages repetitively. While the first attack involves sending CAN messages by using a defined identifier with invalid data, the second one includes sending irregular data with the smaller CAN message ID. Their experimental results show that both attacks were successful in blocking the operation of the CAN bus. Consider another example attack wherein an adversary could gain access to the internal network of the vehicle remotely followed by injecting messages in order to compromise and control an ECU, and even the vehicle itself. Once an ECU is compromised, the attacker can potentially control the safety-critical functions of the car, such as braking, acceleration, and steering. It is however important to note that safety-critical components usually reside on a network that is separate from other non-critical components. Nevertheless, hackers may still be able to break into these networks by leveraging a gateway ECU [16].

Other examples of CAN bus exploitation include installation of a malicious diagnostic device to send packets to the CAN bus, using CAN bus to start a vehicle without a key, leveraging the CAN bus to upload malware, installing a malicious diagnostic device in order to track the vehicle and enable remote communications directly to the CAN bus [82].

Several solutions have been proposed to provide protection against security threats to the CAN bus, which include employing encryption for preventing information disclosure and tampering, authentication mechanisms for dealing with spoofing attacks as well as network segmentation and intrusion detection and prevention systems as protection against DoS attacks. Changing network topology can also be a useful strategy whereby the critical and non-critical ECUs are separated by placing in different networks. However, one particular concern associated with this network segmentation approach is the fact that these networks are interconnected using a central gateway unit, which could be compromised to send malicious messages to the ECUs residing on different networks [83].

Part of the well-known standard Automotive Open Software Architecture (AUTOSAR),Secure Onboard Communication (SecOC) is an AUTOSAR module that released in version 4.2 of the specification, which offers protection against tampering, spoofing and replay attacks. It may use either symmetric or asymmetric cryptography for ensuring the authenticity and integrity of the messages. In order to ensure authenticity of the message sender, SecOC uses either digital certificates or Message Authentication Code (MAC) [84].

Scientists are constantly striving to discover novel ways or applying existing approaches in other domains for developing effective solutions that can help strengthen the system security. One example of such efforts is a solution for monitoring and detecting cybersecurity attacks on CAN bus, proposed by Kalutarage et al. [85], a contextualized anomaly detector, which employs message sequence modeling using n-gram distributions, and relying on benign data for threshold estimation and training purposes. The authors evaluate their proposed solution by means of two spoofing attacks targeting RPM and gear gauge messages. The experimental results demonstrate their approach was able to detect the attacks with hundred-percent accuracy using a small fraction of a second.

3.1.2 IVI Vulnerabilities and Mitigations

Infotainment systems, one of the major attack vectors in connected cars (as depicted in the Figure 3.1), are growing both in terms of their capabilities and popularity. As more and more features are being added to infotainment systems, this will likely to increase the number of new vulnerabilities, attack vectors, and threats that can undermine the privacy and safety of the vehicle and its occupants. Typically, an IVI is interconnected with the CAN bus for communicating with other devices. From cybersecurity perspective, this connectivity may have serious implications. Prior studies have evidenced that cybercriminals can target automotive infotainment systems for mounting sophisticated attacks on automobiles [86].

An attacker can exploit weaknesses in the infotainment system or can use it as an entry point to gain access to in-vehicle network, thus to safety-critical features of the vehicle. Some possible use cases include utilising a remote connection to the infotainment system for exploiting the application in the IVI responsible for handling incoming calls, accessing the Subscriber Identity Module (SIM) through the IVI, installing malicious code on the infotainment system, putting the infotainment console into debug mode, using a malicious application to access the internal CAN bus network, using a malicious application to eavesdrop on actions taken by vehicle occupants. Mitigating strategies for addressing the security weaknesses include the following [87]:

- USB sticks with only supported file systems should be allowed to connect
- Security permissions for mounted USB sticks should be limited to read-only
- USB configurations should be restricted to enable essential USB services only
- Updates should always be signed or encrypted
- Authentication of the update procedure should be ensured
- Security of the key storage should be ensured
- Failed updates should be rescued by fall back mechanism
- Onboard applications should only be installed from trusted and official sources
- High-risk applications should be isolated using containers or virtual machines
- Strict access models (e.g., RBAC or PBAC, etc.) should be applied by dividing different security domains for effective application management
- Proper configurations of wireless protocols should be ensured
- Network routing should be restricted to pre-defined normal behaviour
- Disable all Bluetooth profiles that are not in use.

Table 3.1: *Attack surfaces and security threat associated with In-Vehicle Infotainment interfaces.*

Component/attack surfaces	Threat/attack vectors
USB Port	• Malware injection • Malicious firmware updates • Port scanning (USB-to-Ethernet)
Wireless connectivity (Wi-Fi, Bluetooth, Cellular, GPS)	• Packet sniffing • Protocol-specific exploits • Jamming • GPS tracking, spoofing • MITM
Multimedia playback	• Tampering with media services, Bluetooth, and Wi-Fi stacks
External diagnostic	• Fuzzing attack
Onboard applications	• Application related vulnerabilities

3.1.3 OBD Vulnerabilities and Mitigations

Nilsson and Larsan in [88] demonstrate how a virus can be injected on to the CAN bus through the OBD port that issues some messages for controlling some aspects of the vehicle behaviour (e.g., locks, brakes, etc.) if certain conditions are found to be true. Unlike the attack mentioned above in [88], which requires physical access to the vehicle, many modern automobiles allow remote access to these dongles via Wi-Fi connections from a computer, allowing adversaries to launch cyberattacks remotely. As reported in a survey [89], more than 50% of the surveyed dongles, were found to be containing vulnerabilities (e.g., exposed keys, weak encryption), which can be exploited by cybercriminals to compromise the security of a vehicle.

Intrusion detection systems and firewalls are some of the major countermeasures used for dealing with automotive OBD-specific security threats and cyberattacks. One of the recent studies aiming at devising innovative solutions to address security challenges in this area includes an OBD anomaly detector proposed by Rumez et al. [90], which is capable of detecting anomalies in automotive diagnostic applications by leveraging a statistical language model. The authors use two different n-grams models (i.e., byte-based and sequence-based) to analyze the incoming diagnostic frames for determining whether the sequences and embedded bytes are contextually legitimate and valid.

By compromising one of the connections listed above, cybercriminals can potentially attack in-vehicle systems in order to take over a vehicle remotely, shut down it, unlock it, track it, thwart its safety systems, install malware on it, or spy on its occupants. For example, an adversary can access the vehicle's internal network or the remote diagnostic system remotely by means of cellular connection. Similarly, an attacker can exploit the Wi-Fi connection for gaining access to the vehicle network (from up to 300 yards), intercepting data traffic of the Wi-Fi network, breaking the Wi-Fi password and more [77].

3.2 Testbeds for Security Evaluation of Connected Cars

Systematic cybersecurity evaluation of automotive systems is a non-trivial, critical task. Comprehensive security assessment requires a disciplined and well thought out approach. As opposed to an ad-hoc testing approach which often suffers from subjective prioritization of test cases leaving numerous undiscovered vulnerabilities in the system, a methodical approach increases the chances of detecting more flaws. Both the testing approaches and the testing environments play crucial role for discovering security loopholes in a system; we conducted a comprehensive survey of the automotive testbeds and testing approaches, which is presented below. In this section we present an overview of seven different automotive cybersecurity testbeds that have been proposed in the last ten years.

Testbeds can generally be categorised in three different types: simulation based, hardware based, and hybrid. Simulation-based testbeds rely solely or substantially on software to simulate the behaviour of ECUs and in-vehicle networks. Since they do not include real cyber-physical components, simulation-based testbeds are generally cheaper to build, and provide a safer environment for the testers. Hardware-based testbeds, on the other hand, include real or emulated hardware components. As opposed to software-based testbeds, hardware-based testbeds enable testers to study interactions between components through physical inputs and outputs. Hybrid testbeds include both software and hardware components, offering strengths of simulation-based and hardware-based testbeds.

Table 3.2 presents an overview of the surveyed testbeds, indicating whether they are simulation-based, hardware-based, or hybrid. Mobile testing platform from [91] is the only testbed that uses real physical components and a vehicle (go-cart) for investigating cybersecurity threats. OCTANE and the Testbed for Security Analysis of Modern Vehicle Systems are hybrid testing environments. All other testbeds rely on virtual/software components only.

Table 3.2: *Types of Automotive Cybersecurity Testbeds*

Name of Testbed	**Test Platform**	**Year**	**Reference**
Open Car Testbed and Network Experiments (OCTANE)	Hybrid	2013	[92]
Mobile Testing Platform	Hardware	2015	[91]
Cyber Assurance Testbed for Heavy Vehicle Electronic Controls	Simulator	2016	[93]
Testbed for Automotive Cybersecurity	Simulator	2017	[94]
Testbed for Security Analysis of Modern Vehicle Systems	Hybrid	2017	[95]
Portable Automotive Security Testbed with Adaptability (PASTA)	Simulator	2018	[96]
Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed	Simulator	2019	[97]

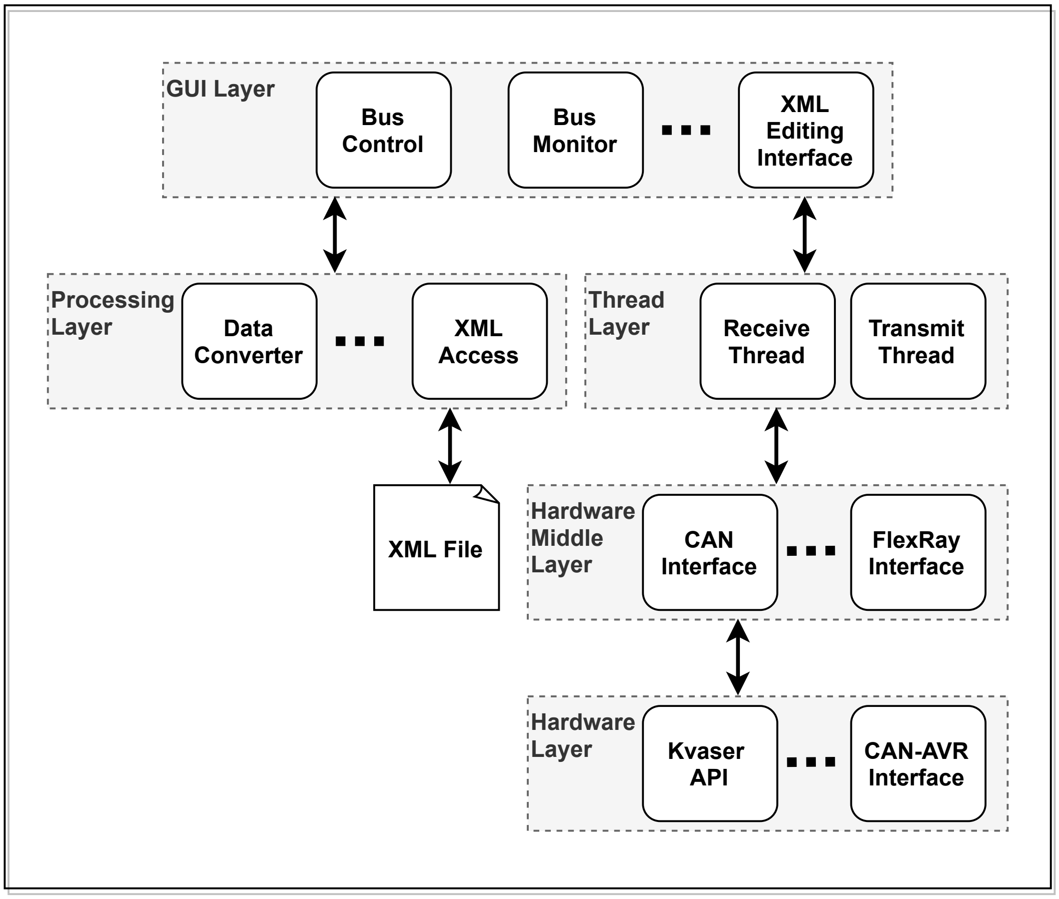


Figure 3.2: *OCTANE: Software package architecture. (redrawn from [98])*

3.2.1 OCTANE

OCTANE [92] includes a hardware framework and a software package providing capabilities to reverse engineer and test automotive networks. In particular, the tool can be used for fuzz testing various proprietary vehicular network protocols.

The software package (see Figure 3.2 for the software package architecture) allows transmission and monitoring of CAN messages for general purpose network diagnostic and debugging as well as automated replay testing of Electronic Control Units (ECUs), whereas the hardware framework assists in setting up hardware components of the automotive networks for two main different configurations: lab setup and real-world setup. The hardware framework outlines a structured step-by-step approach to set up a particular environment without prescribing any specific type of hardware components.

The testbed has been designed to enable entry into automotive cybersecurity testing research and teaching in a safe and cost-effective way. In order to maintain clear separation of concerns between software and hardware components, the hardware middle layer plays a pivotal role. This makes it easy to add a new hardware adapter to replace the existing one without affecting other layers. In order to enable adaptability and flexibility, the software package has been designed using a layered architecture consisting of a presentation layer, a business layer composed of a processing layer and a thread layer, a hardware middle layer, and a hardware layer.

This testbed allows security testing of various vehicular network protocols including CAN, LIN, MOST and FlexRay. An appropriate adapter needs to be used when working on a specific network technology. One of the main limitations of the testbed is that it only uses the OBD port as an attack surface. The testbed is not capable of testing vulnerabilities related to wireless connectivity. We observe that although the source code is available to download on the Google Code platform, we were unable to find any related documentation.

3.2.2 A Mobile Testing Platform

Miller and Valasek [91] implemented a mobile testbed by modifying a go-cart to emulate a real vehicle. They equipped it with various ECUs and sensors, which help study the behaviour of actual devices in an economical way. As compared to a real vehicle, there is low financial risk involved, because the go-cart is much cheaper than a real vehicle. However, risk of physical injuries is still present as it is a moving vehicle.

One of the major capabilities of the real vehicle they included was a power steering control module (PSCM) on the go-cart. Additionally, they integrated different sensors including proximity and speed sensors. A real pre-collision system was also incorporated for actual distance readings while the vehicle is in motion. While using a moving vehicle instead of a bench setup certainly enabled the testers to study the behaviour of the moving vehicle, which is generally not possible with a setup on a bench, there are some shortcomings as well that the original developers of the environment identified, as outlined blow:

While real components were used in the go-cart, they may not represent the complete functionality and behaviour of an actual car. For example, the developers note that PSCM does not work properly after some right and left turns, as it enters its final state. Another limitation reported by authors is steering wheel radius that does not allow steering to be controlled by the CAN bus. This limitation was also a hurdle for auto-park capability, which otherwise could have been realized. Finally, remodelling of the go-cart vehicle is another key challenge for the researchers who may be interested in using this setup. To summarize, while this mobile testing platform enables the tester to evaluate the impact of of cyberattacks on the moving vehicle cost-effectively, it has some considerable limitations as well.

3.2.3 A Cyber Assurance Testbed for Heavy Vehicle Electronic Controls

This testbed [93] has mainly been proposed for cybersecurity testing of heavy vehicles remotely. It primarily supports J1939 networks that are found in heavy vehicles

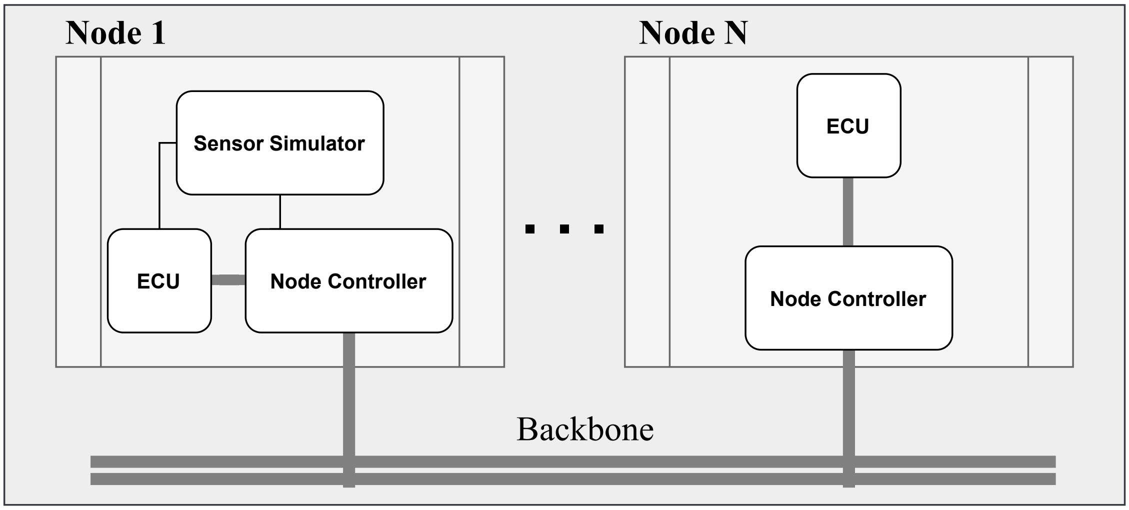


Figure 3.3: *Conceptual diagram of the testbed showing the isolation of the modules from the backbone (adapted from [93]).*

including buses and trucks. Authors used real ECUs, Linux-based simulated node controllers for their testing setup. The testbed components include an electronic brake controller, engine control module, backbone, node controller, power supply, switch, sensor simulator, and telematics along with associated antennas. Figure 3.3 presents a conceptual diagram of the testbed showing the component arrangement that involves isolating each module from the backbone.

The testbed allows the researcher to study and manipulate the network traffic by providing various features. For example, one of the distinctive characteristics of this testbed is the capability of remote experimentation, which allows the analyst to access the data remotely without any physical interaction with the vehicle. For this purpose, the authors of the testbed introduced a custom-built five-layer application, as shown in the Figure 3.4.

The five layers are web interface, experiment processing, experiment logic, CAN data processor and a database for experiment and J1939 data. The web interface layer allows the tester/researcher to interact with ECUs for monitoring and modifying network traffic. Experiment processing layer is responsible for converting the CAN messages into a human readable format. The database layer is mainly used for storing the experiment data.

3.2.4 Testbed for Automotive Cybersecurity

Fowler et al. [94] built a testbed for automotive cybersecurity testing consisting of an established industry, real-time CAN simulator from Vector Informatik.

The simulator along with its associated software CANoe is widely used in the automotive industry primarily by automakers for the development and testing of ECUs. The simulator provides CAN data traffic monitoring, capturing, and analysis capa-

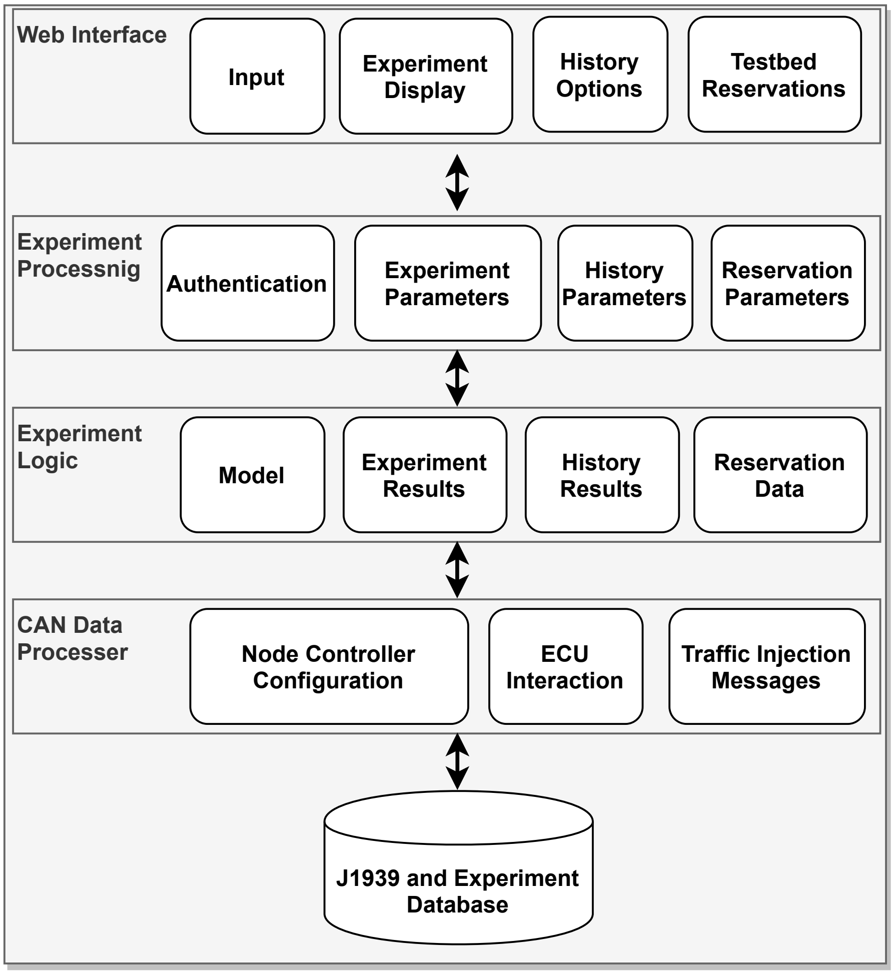


Figure 3.4: *Five-layer software architecture of the remote interface to the testbed (adapted from [93]).*

bilities, which help in reverse engineering of vehicles. To validate the testbed, CAN message-injection was performed by using a Bluetooth-enabled dongle connected to an OBD port on the simulator. The messages were successfully injected validating the correct functioning of the testbed. The description presented in the paper is limited to some high level information only without going into details about the architecture and other characteristics of the testbed.

3.2.5 Testbed for Security Analysis of Modern Vehicles

Zheng et al. [95] developed a prototype of their proposed testbed (as shown in Figure 3.5), which was built around a real-time CAN bus simulator using dedicated hardware from National Instruments and a simulated vehicular infotainment system (using LabVIEW software).The testbed is able to capture CAN messages for security analysis and can inject malicious messages through simulated infotainment system.

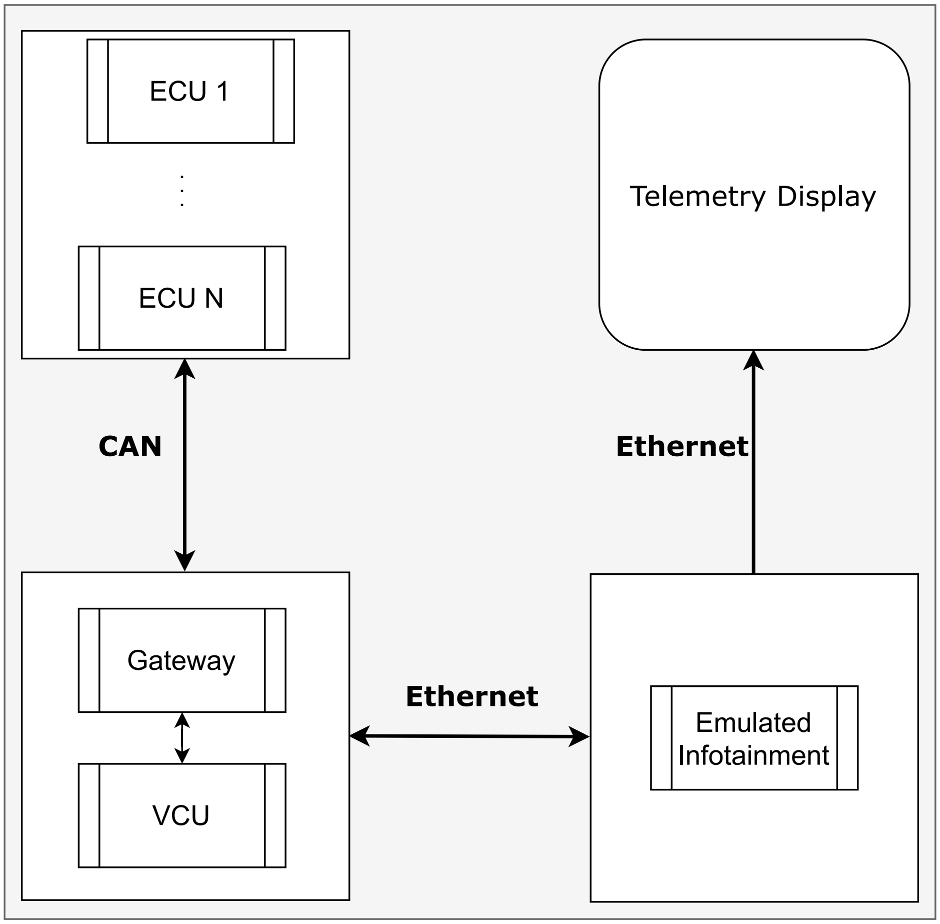


Figure 3.5: *High-level architecture of the testbed for modern vehicle security analysis (adapted from [95]).*

It is argued by the authors that while use of real vehicles or vehicle components for testing is more effective and produces accurate results, such test environments provide little or no flexibility in terms of their configurations. The proposed testbed by Zheng et al. is reconfigurable, enabling the testbed to replicate many test configurations. Furthermore, the testbed is able to reproduce the complexity of interconnected ECUs in the in-vehicle network.

The authors performed a denial-of-service attack targeting the CAN bus by leveraging the emulated infotainment system as an entry point into the in-vehicle network. A dump containing a large number of previously captured CAN messages was injected causing the CAN bus to fail to operate properly by rejecting legitimate CAN messages. This testbed is reconfigurable, inexpensive to reproduce, and provides a safe environment for automotive cybersecurity testing. However, being largely a simulated environment its obvious limitation is the lack of physical input and output ports which seriously affects security evaluation requiring these ports.

3.2.6 Portable Automotive Security Testbed with Adaptability: PASTA

Portable Automotive Security Testbed with Adaptability (PASTA) [96] is another automotive security testbed with a special focus on white-box ECUs, high adaptability and portability. Authors explain why white-box ECUs can be more effective when it comes to automotive cybersecurity testing. First of all, white-box ECUs provide the ability to observe their inputs and outputs as well as disassembly of the ECU programs without involving any suppliers or OEMs. Secondly, ECUs can be reprogrammed and rearranged in a number of different configurations in the automotive networks allowing evaluation of the security technology against cyberattacks. Finally, the ability to modify different parameters, such as CAN ID, payload, or transmission cycle enables the reproducibility of a commercial vehicle.

The authors outline the requirements that they considered while designing their proposed testbed. The first factor is the cost of the testbed, which is typically very high when involving a real vehicle containing a variety of ECUs and in-vehicle networks. High financial cost is one of the barriers to automotive cybersecurity research. To minimize the cost of their testbed, they eliminated expensive sensors and other similar components including simulators such as speed, angle of tyres, and status of headlights. Authors argue that such expensive components are not essential for cybersecurity testing.

Portability of the testbed is another key consideration; PASTA has been designed with portability in mind, its compact size allows it to be easily carried to different places for demonstrating research experiments and results. Another aspect of the testbed is the generalizability of the vehicle to ensure the testing is not restricted to a specific make and model. All the testbed components have been fitted in an attache case.

Safety is also critical aspect, especially when the testing involves physical subsystems, such as actuators, which can behave in an unexpected way causing injuries to the researchers or any other parties involved. Such safety risks can be addressed by using an emulated actuator instead of a real one, for example. Finally, the target testbed must be designed in a way that it supports the learning of all the stakeholders, especially software developers, because software bugs and flaws in the software design due to human error often result in catastrophic consequences.

PASTA, according to its designers, meets all the requirements outlined above. The testbed can be customized to fulfill specific needs of a researcher, as it has been designed using non-proprietary technologies.The testbed allows testers to use custom security technology and provides the flexibility to design the in-vehicle network as per their particular requirements.

While PASTA is safe, flexible, portable and adaptable, it has some shortcomings too. Its software vehicle simulator is not able to replicate a vehicle's behaviour accurately. As the designers of PASTA have noted that speed of the vehicle reaches 199 km/hour in a very short time when the acceleration is applied, which is obviously not reflective of true behaviour of a real vehicle. The software needs tweaking to resolve this issue. Another limitation of PASTA is that it currently supports CAN protocol only. Other protocols such as LIN, FlexRay, and MOST are not supported. OBD-II port and a tapped CAN cable are the only physical intrusion points that PASTA provides for launching attacks on the CAN bus. Moreover, it currently lacks attack surfaces such as Bluetooth, WiFi, and cellular networks. Finally, the software architecture for the implementation of ECUs environment is not Automotive Open System Architecture compliant.

3.2.7 Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed

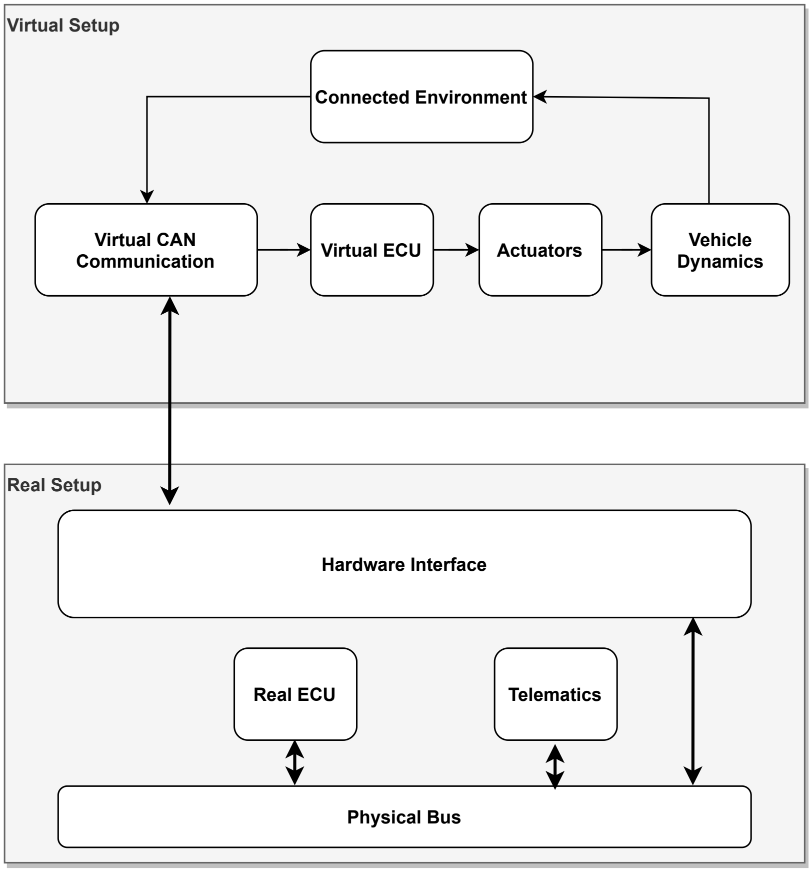


Figure 3.6: *Simplified architectural view of the proposed hardware-in-the-loop based testbed (adapted from [97]).*

Oruganti et al. [97] propose a testbed for automotive cybersecurity testing, as presented in the Figure 3.6. Authors report their current progress towards the development of their testbed which will include hardware-in-loop components. The current testbed is completely a virtual setup, thus limited to a software simulation only. Using this virtual testbed, the authors demonstrate a GPS location spoofing attack on a virtual vehicle. The authors list essential elements of the testbed that should be present, which include connectivity, vehicular networks, controller modeling and algorithm implementation, hardware-in-loop and telematics. Each of these subsystems allows the cybersecurity evaluation and validation of a connected car for a range of attack surfaces and attack vectors.

3.3 Comparative Analysis of Automotive Cybersecurity Testbeds

This section compares the reviewed testbeds based on their various characteristics, such as adaptability, portability, fidelity and cost. An overview of other capabilities (e.g., types of attacks, attack surfaces, attack targets, and communication protocols supported) of the testbeds have also been discussed.

Each testbed type has its own strengths and limitations. Ideally, a testbed should be able to reproduce the behaviour of a real vehicle as accurately and faithfully as possible (fidelity), adaptable, portable, safe, and inexpensive to construct, as explained in [96]. In following subsections, we provide an overview of the supported vehicular network protocols, support attack surfaces and types along with a description of how and to what extent each of the testbeds meets the requirements of adaptability, portability, fidelity, safety, and cost-effectiveness.

Adaptability is a measure of a testbed's ability to support different testing configurations, i.e. how well a testbed can adapt to different testing scenarios and applications. A testbed with a *High* adaptability relies on standard, general-purpose components that enable such testbeds to support the security testing of most in-vehicle network types (i.e., CAN, MOST, LIN, FlexRay, etc.) as well as of different types of ECUs. A testbed comprising of both general-purpose and special-purpose components could be considered to have a *Medium* adaptability, whereas a testbed built using proprietary, special-purpose devices offers a *Low* adaptability. Portability refers to how easy it is for the testbed to be carried around. Testbeds constructed using small, lightweight and/or virtual components and those not relying on heavy cyber-physical components (e.g., actuators) offer *High* portability. While testbeds rated *Medium* for portability, containing a combination of both virtual and some lightweight physical components/devices, can be moved around, movement could be challenging depending on the nature of the physical components involved. We assigned a *Low* portability rating, indicating the inability and/or complexities stemming from the physical characteristics (i.e., dimensions and weight) of the testbed components. Fidelity of testbed can be measured by evaluating how accurately it can imitate the behaviour of a real vehicle in response to a cyberattack. Real hardware-based devices with multiple input/output capabilities can deliver a high accuracy of the behavioural reactions of the system under test, hence can be given a *High* rating for the fidelity. Testing environments involving cyber physical components often have safety implications for the security analysts and any hardware/equipment involved; therefore it is vital to determine how a given testbed ensures the safety of testers and the equipment involved in the testing. We rate a testbed to be highly safe if it does not include any cyber-physical components that can potentially cause physical harm (i.e., the testing environment is virtual, relying on simulations only). On the other hand, a testing setup involving both virtual and physical components can be rated *Medium* for safety, and can be assigned a *Low* rating if all or most of the components are physical. Finally, cost of the entire setup can be compared based on whether it contains real physical devices, only virtual components or a mix of both types. In addition to the hardware devices, software components (e.g., open-source vs commercial) may also have an impact on the overall cost of the testing setup. A virtual testbed with free, open source software applications will have a *Low* cost, as compared with the ones involving simulated and real devices/components, which can be rated *Medium* for the cost factor. Hardware-based testbeds running commercial software applications are more expensive, thus a *High* cost rating has been assigned. Table 3.5 provides a comparison of how adaptable, portable, accurate, safe and costly each testbed is. While a testbed offering *High* adaptability, portability, fidelity, and safety is the best as compared with the ones with *Medium* or *Low* values for these attributes, testbeds having *Low* cost value are comparatively better than the ones with *High* or *Medium* cost.

3.3.1 An Overview of Supported Network Protocols

Modern cars have multiple network types for facilitating various applications. Not all testbeds that have been surveyed offer support for testing all types of communication standards. Table 3.3 gives an overview of the protocols supported by each testbed. As can be noticed, OCTANE is the only testbed that claims to support testing for all major vehicular network protocols. All other testbeds do not cover any protocol other than CAN. This means they are unable to support study of threats/attacks related to other network standards found in modern automobiles.

3.3.2 Supported Attack Surfaces, Types of Attacks, and Attack Goal

Table 3.4 highlights types of attack surfaces exposed by each testbed, types of attack supported or demonstrated, and attack target or goal. OBD-II port is the most popular choice as an entry point into the in-vehicle network. This is probably due to the fact that all cars do have an OBD port, (since it is legal requirement to have one) OBD scanners are cheap and easily available in the market.

Table 3.3: *Overview of what types of in-vehicle network protocols are supported by each testbed for cybersecurity testing (* FLR stands for FlexRay and Ref. for Reference).*

Testbed Name	CAN	LIN	FLR*	MOST	Ref.*
Open Car Testbed and Network Experiments (OCTANE)	✓	✓	✓	✓	[92]
Mobile Testing Platform	✓	n/a	n/a	n/a	[91]
Cyber Assurance Testbed for Heavy Vehicle Electronic Controls	✓	n/a	n/a	n/a	[93]
Testbed for Automotive Cybersecurity	✓	n/a	n/a	n/a	[94]
Testbed for Security Analysis of Modern Vehicle Systems	✓	n/a	n/a	n/a	[95]
Portable Automotive Security Testbed with Adaptability (PASTA)	✓	n/a	n/a	n/a	[96]
Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed	✓	n/a	n/a	n/a	[97]

Table 3.4: *An overview of the types of exposed attack surfaces, types of attacks, target and/or goal of the attacks supported by each testbed.*

Testbed Name	Attack Surface	Attack Type	Attack Target/Goal	Reference
Open Car Testbed and Network Experiments (OCTANE)	OBD II Port	Message sniffing, DoS, Replay	N\A	[92]
Mobile Testing Platform	OBD II Port	CAN Message Injection	Take over vehicle control	[91]
Cyber Assurance Testbed for Heavy Vehicle Electronic Controls	ECU, Ethernet, USB	Brute force, DoS	Evaluation of Seed\Key, Exchange Strength and, Intrusion Detection System	[93]
Testbed for Automotive Cybersecurity	OBD II Port	CAN Message Injection	Comfort Subsystem Manipulation (e.g., headlamp ON/OFF)	[94]
Testbed for Security Analysis of Modern Vehicle Systems	Infotainment Gateway	Can Sniffing, Code Injection, DoS	Control vehicle maneuver	[95]
Portable Automotive Security Testbed with Adaptability (PASTA)	OBDII Port/Clipping Area	Code Injection/Execution	N\A	[96]
Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed	Navigation System	GPS Spoofing	Spoof GPS Location	[97]

Similarly, most popular type of attack is message/code injection. This is due to the fact that CAN protocol does not have an authentication or other security mechanism capable of identifying and rejecting malicious contents. Since many testbeds lack support for wireless/remote attack surfaces, they are only confined to testing attack scenarios assuming physical access to the vehicle.

3.3.3 Adaptability

In-vehicle networks and ECUs are the major targets of cyber attacks, their security testing is important to identify and fix any security issues. Unfortunately, due to copyright restrictions and closed-source proprietary ECU technologies, it is very difficult to perform security testing on commercial ECUs. In addition, testing specific ECUs does not provide insights and results that can be helpful when testing the ECUs from different manufactures. Similarly, it is also important that the testbed is adaptable to a variety of testing configurations (e.g., with different vehicular network types) and not confined to a particular technology.

For instance, PASTA [96] includes white-box or programmable ECUs that allow the researcher to program an ECU to replicate the behaviour of a specific ECU, with the knowledge of internal implementation, which is usually not possible with real proprietary ECUs. Also, because it is largely software-based setup, it can be used for different testing configurations.

The layered-based design of the OCTANE [92] allows it to be adapted to different testing setups by replacing hardware components in the hardware middle layer without affecting other layers. Furthermore, its bespoke software package can be modified to extend its capabilities according to specific testing scenarios. The prototype testbed proposed by Zheng et al. [95] has a flexible architecture allowing additional ECUs to be added easily. The setup can also be used for testing and investigating attacks launched via remote connections. Since it is entirely software-based, the testbed presented by Oruganti et al. [97] is adaptable to various testing configurations, as virtual components can be easily added or removed in the software environment.

Daily et al. have relied on actual ECUs and sensor simulations primarily focusing on J1939 based networks, which are specifically designed for heavy vehicles, such as trucks and buses. Although, the authors do not explicitly consider or discuss adaptability of the testbed, based on the information provided, it seems probable for the testbed to be adapted to various testing configurations.

3.3.4 Portability

A key factor to consider while designing or using a testbed is the portability. Sometimes it may be necessary to carry the testbed to a different location for demonstration purposes (e.g., in a conference or workshop). A testbed with compact or virtual components is obviously easy to carry around as opposed to the ones that include large actual components. Below we describe how each of the reviewed testbeds support portability.

Table 3.5: *A comparative overview of the reviewed testbeds based on adaptability, portability, fidelity, safety and cost*

(• • • = High, • • = Medium, • = Low)

Testbed Name	Adaptability	Portability	Fidelity	Safety	Cost
Open Car Testbed and Network Experiments (OCTANE) (real-world setup)	•	•	• • •	•	• • •
Open Car Testbed and Network Experiments (OCTANE) (lab setup)	• •	• •	• •	• •	•
Mobile Testing Platform	•	• •	• •	•	• •
Cyber Assurance Testbed for Heavy Vehicle Electronic Controls	•	• •	• •	• • •	•
Testbed for Automotive Cybersecurity	• •	• •	•	•	• •
Testbed for Security Analysis of Modern Vehicle Systems	• •	• • •	•	• • •	•
Portable Automotive Security Testbed with Adaptability (PASTA)	• • •	• • •	• •	• • •	• •
Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed	• •	• • •	•	• • •	•

OCTANE has two types of main setups: lab based and real world. The lab-based testing environment typically relies on small components and does not involve real vehicle. So, it is possible to carry the lab setup as necessary with ease. However, in the case of a real-world testing setup, the portability depends on the actual components involved. The portability will be affected if, for example, a real car or heavy components are used. PASTA has been designed to be portable, so all its components are able to fit in a briefcase allowing high degree of mobility. Instead of using a real vehicle, a simulated or scale model of a real vehicle is a key factor in allowing this testbed to be more portable. The testbed from Zheng et al. contains simulated and emulated components so it should be easy to relocate if required.

Similar to the Zheng et al., the testbed from Oruganti et al. is purely a software-based environment which allows it to be moved around easily. The cyber assurance testbed by Daily et al. does not use a real vehicle, can be accessed remotely and uses simulated components with real ECUs, hence it satisfies the portability requirements.

3.3.5 Fidelity

Fidelity of a testbed refers to its ability to accurately reproduce the behaviour of a real vehicle or components in response to a specific event. To achieve high degree of fidelity, real vehicle and/or real hardware components must be included in the test. Software-based testing environments cannot faithfully reflect the conditions of a real car. Thus, fidelity of the test results is directly linked to the type of systems/components involved in the testing. Most importantly, complex interactions among various ECUs and other cyber-physical components inside the vehicle cannot be simply reproduced with high accuracy in a virtual environment.

The software vehicle simulator used in PASTA, for example, reaches 199km/h in a very short time which does not mimic the actual behaviour of the vehicle. It can be concluded that while virtual, software-based testbeds have their own merits, they do not generally replicate actual behaviour of a real vehicle.

3.3.6 Cost

A virtual or software-based testbed is generally cheaper than a testbed which includes cyber physical components. OCTANE, and mobile testing platform (involving go-cart) rely on physical components, they are therefore more expensive. On the other hand, Zheng et al. Fowler et al. Oruganti et al. are software-based testbeds their cost is comparatively lower. PASTA and Daily et al. both contain ECUs, their cost will be higher than the pure software-based testbeds.

3.3.7 Safety Implications

Testing real vehicles help study the actual impact and behaviour of a vehicle as a result of a cyber attack. However, this has serious safety implications for the researcher and the vehicle under test. Physical safety of all stakeholders as well as of all the components/equipment involved must be the top priority. We look what safety implications each of the reviewed testbeds may have. In general, safety risks are high when a real vehicle or large cyber-physical components are used in the testing. The risk is even higher when the testing involves a moving vehicle on the road. While designing a new or using an existing testbed, it is advisable to carefully consider any safety issues that can potentially surface.

OCTANE has two testing environments - laboratory-based and real world. In the lab-based setup, there are virtually no concerns related to human safety as it is a controlled environment with no real vehicle involved. The real-world testing setup potentially can lead to situations that can affect safety of both the vehicle and the testers.

Since [96], [95], [97] are primarily simulation based, these testbeds do not raise any safety concerns for the testers/researchers. Similarly, because the testbed from Daily et al. [93] is remotely accessible, it is safe to use.

It can be noticed that while software-based testbeds are generally more adaptable, portable, inexpensive, and safe, they however lack physical inputs and outputs (I/O) which may not be useful in the scenarios where evaluation of physical I/O is essential. Moreover, software-only testbeds do not provide accurate results and are often unable to reproduce the behaviour of actual systems.

3.4 Testing Approaches in Automotive Cybersecurity

While testbeds play a key role in security assessment of in-vehicle computing systems, effective testing methods are equally crucial for successful security evaluation of these systems. Knowledge of different testing approaches can be useful in choosing and applying the best possible technique for optimal results. We present a survey of four different automotive cybersecurity testing approaches here, as at the time of this writing, there is no existing work presenting such a survey.

Interconnected computing components (i.e. ECUs) in a modern vehicle control various features including safety-critical functions, such as airbags, braking, acceleration etc. Attackers can exploit security loopholes in these systems to take over control, steal information, or cause damage to the vehicle and/or its occupants. Prior studies [15, 14, 7, 99] discuss different attack scenarios that are possible and practical. Therefore, thorough and systematic testing of automotive components is paramount.

There are effective approaches employed by cybersecurity testers, professionals and researchers, which help detect potential security weaknesses in automotive systems. Following subsections discuss some major cybersecurity testing approaches.

3.4.1 Automotive Fuzz Testing (Fuzzing)

Fuzz testing or fuzzing is used to discover new vulnerabilities by exposing the system to invalid, malformed, or unexpected inputs and the target system is monitored for any unusual behaviour, which may cause the system to crash. Fuzzing involves three main steps [100, 101]:

1. Preparing the input
2. Delivering the input to the target
3. Observing the behaviour of the target

Fuzzer, a software application specifically designed for performing fuzz testing, is used for bombarding the system under test with a huge number of automatically generated data values. The software then observes system's behaviour to see any reactions to the input data. The input values are crafted either from existing valid input dataets or from a prescribed set of values.

While fuzzing has been around since 90s, and widely used as an effective testing technique in other domains for vulnerability discovery, it is not very popular in automotive security testing yet. This is probably due to the presence of specific challenges that require some adjustments for successful application of the technique to automotive security assessment. For example, monitoring of the system for unusual behaviour is crucial, but since the same interface is usually used for both the fuzz massage-injection and monitoring purposes, this means the internal reactions of the system might not be visible to the observers. A virtual testing environment with adequate support for observing the reaction of the target ECU can be an effective solution to this challenge as Bayer et al. report in [102].

In their study, Fowler et al. [103] describe a basic experimental attack they performed on a virtual vehicle using fuzz testing with a custom-built fuzzer. OBD port was used to interface the fuzzer with the CAN bus. The attack involved locking/unlocking the door lock of the virtual vehicle by injecting messages onto CAN bus. This was achieved by injecting random CAN messages for a short period of time. Based on their experience by executing the attack successfully and influencing the behaviour of the vehicle, the authors conclude that the fuzzing can be useful in reverse engineering of CAN messages as well as causing disruption to the vehicular networks. Most importantly, they note that the fuzzing can be detrimental for the vehicle under test.

In a more recent work [104], Fowler et al. emphasize the importance and usefulness of fuzzing (and other security testing methods), especially, when it is performed prior to production for allowing the discovery and fixing of bugs, which can lead to serious security issues in the early phases of the system development.

3.4.2 Automotive Vulnerability Scanning

Automotive vulnerability scanning focuses on testing the system for existing known weaknesses in the system to ensure that the system is protected against known threats. An automotive system is typically scanned for identifying known weaknesses in the source code, ICT infrastructure and networks by using a regularly updated database of known vulnerabilities.

Vulnerability scanning can be performed in several different ways, depending on the types of target weaknesses for which the system is being examined. For example, in order to verify whether certain software flaws (e.g., buffer/heap overflows) present in the software, static and dynamic analyses can be performed on the source code. Various interfaces including Wi-Fi, cellular network, and Ethernet can be scanned for open ports and running services in automotive systems. In particular, in-vehicle networks, such as CAN and on-board diagnostic port should be scanned. Finally, analysis of the entire system specifically focusing on various configurations to verify if there are any loopholes that can be leveraged by adversaries to compromise the system. [105].

Vulnerability scanning of an automotive infotainment system is presented in a recent study [106] by Josephlal and Adepu. The infotainment system used in the study has various connectivity interfaces including Wi-Fi, Bluetooth, USB port, CAN and others. The authors used different tools (e.g., Nmap, Nessus) to support their experiment involving a attack vector analysis and vulnerability scanning of the infotainment system. The scan was able to detect various types of vulnerabilities of varying levels of risks. In particular, IP address of the infotainment system, an infotainment service running on a certain port, as well as a number of information leaking vulnerabilities were identified.

In addition to the vulnerability scan described above, they also report different attacks including a denial-of-service attack they conducted using a malicious smartphone app.

3.4.3 Automotive Penetration Testing

Penetration testing, in general, is a security assessment approach which is usually adopted by security testing professionals to carry out security testing from the perspective of an attacker to discover security weaknesses in a system. While there are different variants of the approach, it generally has the following key stages as outlined in the National Institute of Standards and Technology (NIST) Guide to Information Security Testing and Assessment [107]:

1. Planning - this phase is concerned with collecting as much information as possible about the target system as well as the boundaries and relevant components involved in the testing.

2. Discovery - in this phase, all the available public external interfaces of the system are systematically discovered and enumerated.

3. Attack - in order to test the identified interfaces, a series of attacks are launched on the system by exploiting the found vulnerabilities.

4. Reporting - the reporting takes place simultaneously with other three steps. Documentation of the findings is done in this phase.

When the tester has no or limited knowledge of the system under test, they largely depend on publicly available information of the target system. In this case, the target system is treated as a black box, as such the specification of the system is not accessible. In contrast, when the tester has detailed knowledge of the system, the system can be referred to as white box, as the internal details of the system are known to the tester. Whereas, the system may be considered a grey box when the tester has partial information about it [108]. Black-box approach is the most appropriate choice for automotive cybersecurity assessment due to the unavailability of the functional specifications of in-vehicle systems.

Durrwang et al. [109] introduce their approach that combines safety (Hazard Analysis and Risk Assessment) and threat (Threat Analysis and Risk Assessment) analyses for supporting penetration tests for the enhancement of automotive security evaluation. They introduce a method that uses attack trees for deriving security test cases. Their test-case derivation approach relies mainly on safety and threat analyses. Experiments involving penetration testing against an automotive safety-critical, airbag ECU are conducted by the authors for demonstrating and evaluating their testing approach. As the test cases are derived from threat and safety analyses, adequacy and effectiveness of the derived test cases rely on the quality of these analyses. Furthermore, the main focus of the approach is on the identification and testing of the security threats affecting the safety of automobiles. Lastly, the experimentation has been limited to threats targeting a single ECU in the vehicle.

Penetration Testing and Execution Standard (PTES) [110] defines the key stages or phases of the penetration testing as follows:

1. Pre-engagement Interactions

2. Intelligence Gathering

3. Threat Modelling

4. Vulnerability Analysis

5. Exploitation

6. Post Exploitation

7. Reporting

A Framework for systematic security testing of automotive Bluetooth interfaces is proposed by Cheah et al. [111], which relies on a proof-of-concept tool, threat modeling (using attack trees), and a penetration testing approach. While the testing approach helped the authors discover various vulnerabilities in the automotive Bluetooth interface, security evaluation was confined to a single scenario with the goal of data extraction from the vehicle. Additionally, authors did not use or explain a systematic approach for constructing the attack tree they used, they relied on a predefined attack tree.

3.5 Chapter summary

As a result of conducting the literature review, this chapter presented an overview of the state-of-the-art in automotive cybersecurity by exploring various relevant technologies, security challenges, protective mechanisms, new developments, testing environments, and techniques proposed in the relevant literature. An in-depth review of the existing scientific literature addressing various aspects of the automotive cybersecurity domain helped us discovered that there were no prior studies adequately investigating the systematic security analysis and testing of the automotive over-the-air updates. Moreover, by conducting a comparative analysis of the key security testing approaches with a particular focus on their strengths and weaknesses, we have been able to refine our testing approach by realising that while penetration testing is one of the most widely employed testing approaches, it has certain limitations, which can be overcome by combining it with model-based security testing techniques. Finally, another useful contribution of the literature review includes key insights and practical lessons for constructing a cost-effective, safe, portable, and adaptable testing environment for the security analysis of the automotive security evaluations.

Chapter 4

Methodology

In this chapter, we present the testing approach we adopted in this study (initially introduced in [112] that relies on the custom-built software tool for generating and executing test cases automatically. First, a brief overview of the approach is presented followed by a detailed description of each phase and relevant activities. In order to demonstrate the applicability of the approach, we provide an example, illustrating the workflow of all the phases involved.

4.1 Systematic Security Testing Approach

The testing approach we employ in this study is inspired by the Penetration Testing and Execution Standard (PTES) [110] and some of the ideas presented in [111, 113, 62, 114]. Our work diverges from the approaches mentioned above in many ways: Our study focuses on carrying out the systematic security testing of automotive OTA updates using a model-based security testing approach based on attack trees. We employ a structured threat enumeration approach along with threat modeling with attack trees (which is one of the recommended threat modeling approaches suggested by SAE 3601 standard [115]) for systematic derivation of executable security test cases. Our approach encompasses a step-by-step method for constructing attack trees. A custom software tool capable of automatic test-case derivation and execution has been implemented that analyzes the structure of the attack trees to derive effective test cases. We combine model-based security testing techniques with penetration testing for better threat identification, systematic derivation of security test cases, and automated test-case generation and execution against the target system.

A graphical overview of the approach is presented in Figure 4.1. The first phase *Information Gathering* is concerned with gathering information about the target system insofar as possible. System decomposition in this phase refers to the process of identifying major components and their internal and external interfaces along with all interactions. Description of the system (preferably a visual model) is the output from this phase, which is subsequently used in the next phase *Threat Assessment* for exam-

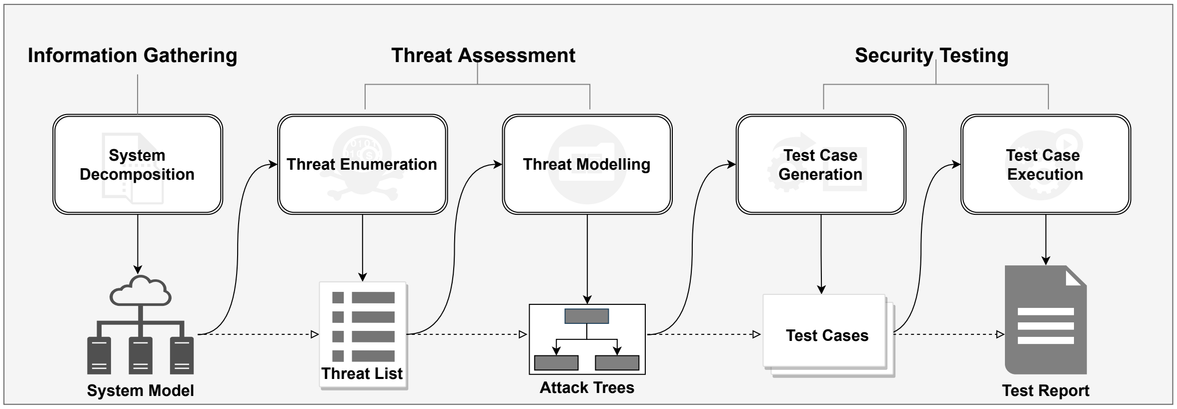


Figure 4.1: *An overview of the threat assessment and security testing approach used for automotive OTA update system, showing key phases, inputs and outputs of each phase.*

ining the system in order to determine what potential threats/vulnerabilities can be leveraged by cybercriminals for compromising system security. Based on the information provided by the preceding phase, *Threat Enumeration* helps generate a *Threat List*, which in turn assists *Threat Modeling* activity in the same phase. Since the generated threat list is usually limited to providing high-level descriptions of the potential threats, identification of the specific attack actions/steps can be accomplished by creating *Attack Trees*. Subsequent steps in the following phase (i.e., *Security Testing*) utilise these attack trees for deriving test cases, preparing test scripts, and finally executing them against the target system. Our prototype software tool analyzes the structure of each attack tree to identify and extract executable test cases/scripts. The final output of this systematic process is a *Test Report* providing a brief summary of the executed test cases. The section that follows highlights the differences and commonalities between our approach and PTES.

4.1.1 Inspirations and Adaptations

Figure 4.2 shows an overview of the PTES, which consists of seven different stages. As can be seen, while our testing approach has many commonalities with the PTES, there are some notable differences as well. Most importantly, our approach does not strictly include all the stages/activities of the PTES, instead it adapts some of the best practices/ideas by avoiding complexities and irrelevant processes/activities. To begin with, the *Information Gathering* phase in our approach is similar to *Intelligence Gathering* stage of the PTES methodology, as both aim at gathering information about the target system. Since PTES is a set of guidelines, it recommends the use of threat modeling (without advocating any particular approach, though) for threat identification and risk assessment. In particular, it breaks down traditional threat modeling comprising of assets and attackers into business assets and business processes, and threat communities and their capabilities, respectively. While our approach relies on the threat modeling for identifying potential threats in very similar manner, it goes beyond this fundamental purpose by leveraging threat modeling to facilitate the automated test-case generation and execution process (as detailed in the Chapter 5). *Threat Assessment* can be considered the counterpart of *Vulnerability Analysis* stage in PTES to some extent. Activities similar to the ones carried out in *Exploitation and Reporting* stages are performed by *Security Testing* phase of the approach employed in this study. Finally, our testing approach has no equivalent to the *Pre-engagement* and *Post-exploitation* stages. We describe all the phases/activities of our approach with more details in the following subsections.

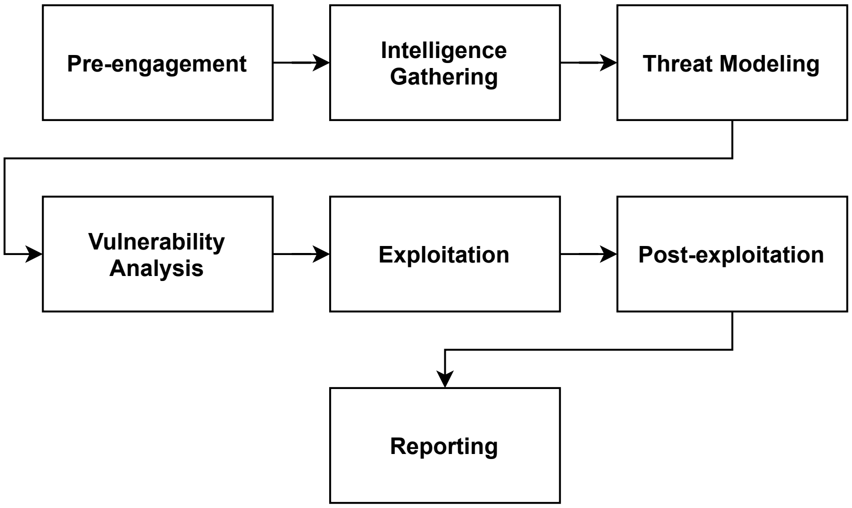


Figure 4.2: *Overview of the Penetration Testing and Executing Standard (PTES) Methodology (adapted from [110]).*

4.2 Information Gathering

This phase is concerned with learning about the target system from the perspective of an adversary to identify relevant physical and logical (i.e., hardware and software) assets/components for a comprehensive security analysis. In particular, all cyber assets - including all the computing and communication devices capable of storing, processing, and receiving/transmitting information, along with associated software components (both system and application) - must be identified that can potentially be the target of cyberattacks. It is worth mentioning that while human assets should also be identified and documented if applicable, we do not consider such assets to be relevant, as test cases constituting social engineering attacks cannot usually be automated. It consists of one major activity that we refer to as *system decomposition*, which is detailed below.

4.2.1 System Decomposition

One of the key elements of the effective security testing is the understanding and knowledge of technical aspects of the system under test. However, in most cases, design specifications and implementation details of the in-vehicle digital systems are not readily available due to commercial reasons and the obscurity of subsystems; therefore, such information may need to be gathered from various other sources, including publicly available technical documentation, (which may include technical guides and technical manuals) and several times by directly observing (and if practicable, may be by reverse engineering) the system/component.

This process is concerned with finding as much information about the system as possible. In particular, identification of core hardware and software system components, interfaces, interrelationships among components, communication devices, technologies, protocols, and key processes can be highly useful in revealing associated security threats. A step-by-step approach to this can be similar to the one as illustrated in Figure 4.3. A large, complex system can be broken down into subsystems, each of which can further be divided into components and finally the components into sub-components. The overall purpose of this process is to identify critical, security-relevant assets leading to threat identification to test case generation, and ultimately to an in-depth security analysis to establish whether the system has adequate and effective security controls, protections against those threats. In the subsection that follows, an example is provided for illustrating system decomposition approach.

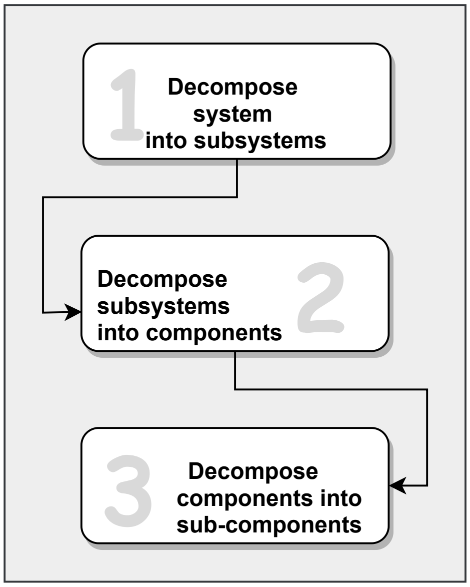

1
Decompose system into subsystems
Decompose subsystems into components
2
3
Decompose components into sub-components

Figure 4.3: *A graphical overview of the system decomposition steps. The system is decomposed into subsystems, subsystems into components, and finally components into sub-components.*

4.2.2 Automotive In-Vehicle Infotainment: Example of System Decomposition

An In-Vehicle Infotainment or an automotive infotainment system is an integrated unit providing information services and entertainment functionality to the driver and other vehicle occupants for an enhanced in-vehicle experience.

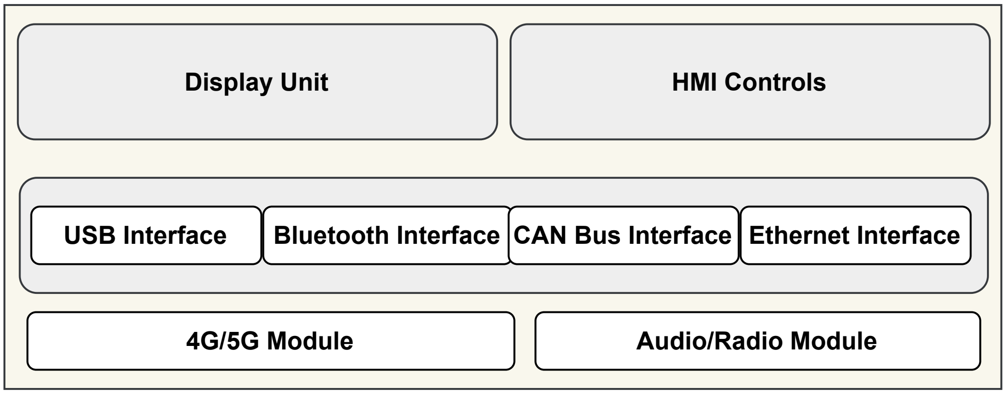


Figure 4.4: *High-level graphical view of the in-vehicle infotainment system.*

Three broad types of the services that a typical IVI can provide include entertainment services, communication services, and information services. Entertainment services offer the capabilities of playing audio and video media both through a CD/DVD player or via a USB device. Communication services allow the driver to make and receive phone calls, use messaging services, and issue voice commands for operating the system. For an enhanced hands-free experience, the driver can use a Bluetooth headset. Information services provide a variety of highly useful information including vehicle related information, such as fuel levels, total distance travelled, whether the doors are open, and the health status of various devices.

Step 1: Decompose system into subsystems Figure 4.4 presents the system view of the IVI without providing any details of the subsystems or components. With the first pass of our investigation, we discovered five different layers of the same IVI by decomposing it into subsystems as shown in Figure 4.5. It is apparent that while the information provided by this view is more meaningful and detailed as compared to the limited details expressed by the overall view of the system, this is still by no means a comprehensive description of the system either. Nonetheless, it certainly lays the foundation for deeper examination of each individual subsystem.

Step 2: Decompose subsystems into components Based on the subsystems identified in the Step 1, further exploration enabled us to discover associated components for each subsystem, as depicted in the Figure 4.6. For instance, we were able

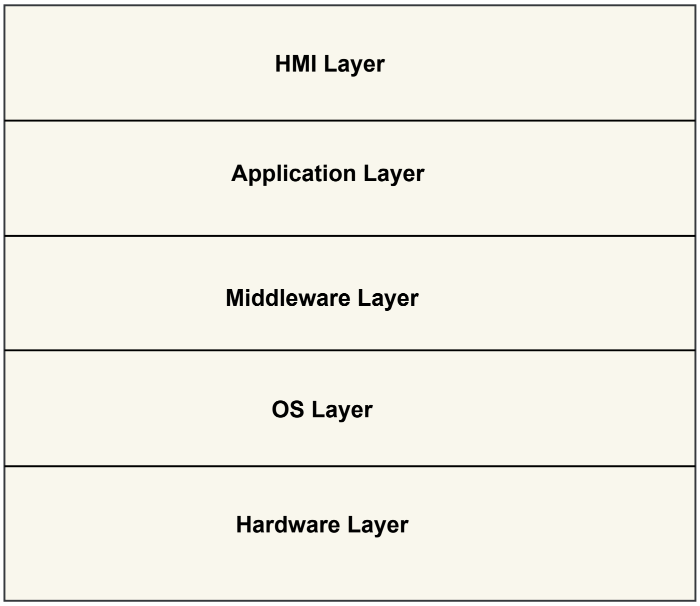


Figure 4.5: *The IVI decomposed into subsystems, represented as five different layers.*

to identify CPU, Memory, Storage, CAN, Wi-Fi and BT (Bluetooth) components by decomposing the hardware layer of the IVI. It is evident that this evolutionary system/asset discovery approach is effective for understanding the target system with appropriate level of details.

Step 3: Decomposing components into sub-components Ultimately, in this final step of the system decomposition, information discovered during the preceding two steps should form the basis for determining critical cyber assets along with associated potential intrusion points, attack surfaces by decomposing each component into sub-components (if applicable). As an example, decomposition of the OS Core component of the OS Layer may result in the discovery of specific kernel and hardware abstraction layer (HAL) sub-components, which can be further investigated for determining if there are any known, reported vulnerabilities for the specific version of the kernel being used. Additionally, with more detailed information of the system, identification of the component interactions and information flows should be a straightforward task.

All the information gathered at the end of the system decomposition activity should be helpful in producing the system description, preferably in the form of a graphical model. The rationale for this preference is firstly for minimizing/eliminating the potential confusions/misunderstanding caused by the use of natural languages that tend to be inherently ambiguous, and secondly conversion from one graphical model to another would be easy. The system description/model should help express both the architectural and behavioural aspects of the system if possible, as this will streamline the process of crafting a DFD diagram using Threat Modeling Tool in the Threat

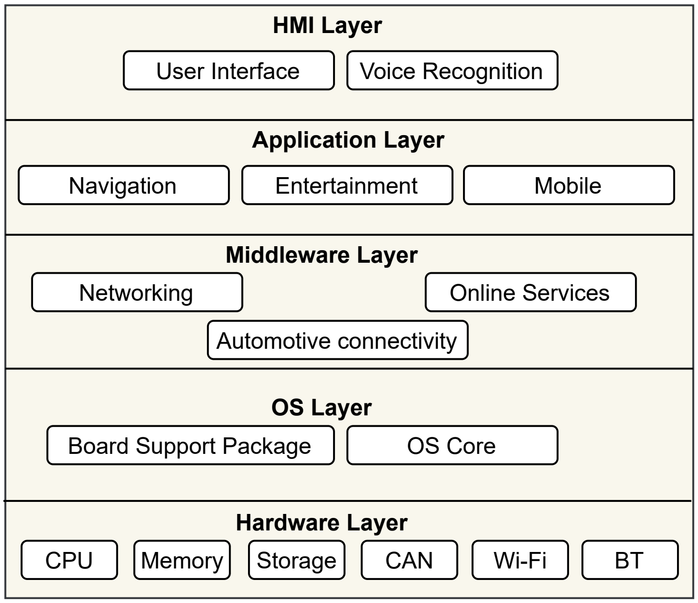


Figure 4.6: *This diagram (adapted from [116]) presents the component-level overview of the IVI by decomposing its subsystems into components.*

Assessment phase. Following the system decomposition, we rely on the standard and widely used modeling technique: Unified Modeling Language (UML) for producing the system model. Once the system model has been created in this phase, we can proceed with the next important activity: identification of the threats.

4.3 Threat Assessment

This phase is comprised of two major activities: threat enumeration and threat modeling. Both activities are crucial for the systematic and automated test case generation and execution process, as each plays an important and distinct role in the process. Threat enumeration involves identifying all potential threats to the system by using the well-known, standard approach STRIDE supported by its associated threat modeling tool, which is capable of enumerating various types of threats in a structured way. Threats identified by the threat enumeration process lay the groundwork for attack tree construction in the threat modeling process. Our automated test case generation and execution tool relies on attack trees for generating concrete security test cases by analyzing the structure of attack trees (for further details, see Section 5.2). The information pertaining to the target system discovered in the *Information Gathering* phase, is leveraged for automated threat identification in this phase. Details of each of the activities are presented in the following subsections.

4.3.1 Threat Enumeration

There are several approaches to identifying security threats, the popular SAE J3601 standard for automotive cybersecurity [115] suggests threat modelling for identifying security threats. For the structured threat identification approach, we employ Microsoft's Threat Modeling Tool. The system model produced in the *Systems Decomposition* process (as shown in Figure 4.1), serves as an input to creating data flow diagram using TMT. While all or most of the relevant components identified in the preceding phase are more likely to become processes, their interrelationships are represented using the data flows. A data store, as its name suggests, is used to denote files and databases etc. Once the diagram is complete, a report of the potential threats associated with each element of the system can be generated. Potential threats identified in the generated report are categorized into different threat groups by the tool using Microsoft's threat classification model STRIDE. Furthermore, the report includes descriptions of each threat and potential attack methods employed by adversaries. An example demonstrating threat enumeration process is presented in the following subsection.

4.3.2 IVI System: Example of Threat Enumeration

This section presents a simple example of threat enumeration process by using the IVI system introduced in the *Information Gathering* phase above.

To start with, threat enumeration process requires a data flow diagram of the target system as an input to identify potential threats. As indicated earlier, Microsoft's Threat Modeling Tool is used for that purpose, which provides all the necessary tools of the trade for creating the data flow diagram. The reason for using this particular application is its unique capability of generating a threat report from the diagram, which is not possible with other diagramming tools. In order to ensure the identified threats are most relevant, an automotive template developed by NCC Group [117] has been used, which offers automotive-specific stencils to draw data flow diagrams in Threat Modeling Tool.

Recall that a number of components were identified including USB, Wi-Fi, and HMI. Figure 4.7 displays the data flow diagram showing the IVI system in the centre and other components placed around it. In addition to the components listed above, a TCU, an IVI System Data Store, and a Navigation Maps Data Store have also been included to this data flow diagram to depict various connectivity devices/interfaces and data stores. Bidirectional data flows have been used to denote the two-way communication between components. Depending on the purpose and required level of detail, data flow diagrams can be created at various levels of abstraction. In this example diagram of the IVI system, connectivity components/interfaces have been the main focus. For example, a data flow diagram with a lower level of abstraction may decompose the IVI system into the layers listed in the Figure 4.6.

Figures 4.8 and 4.9 show two pages extracted from the threat report generated by the TMT based on the diagram displayed in Figure 4.7, presenting a summary of the

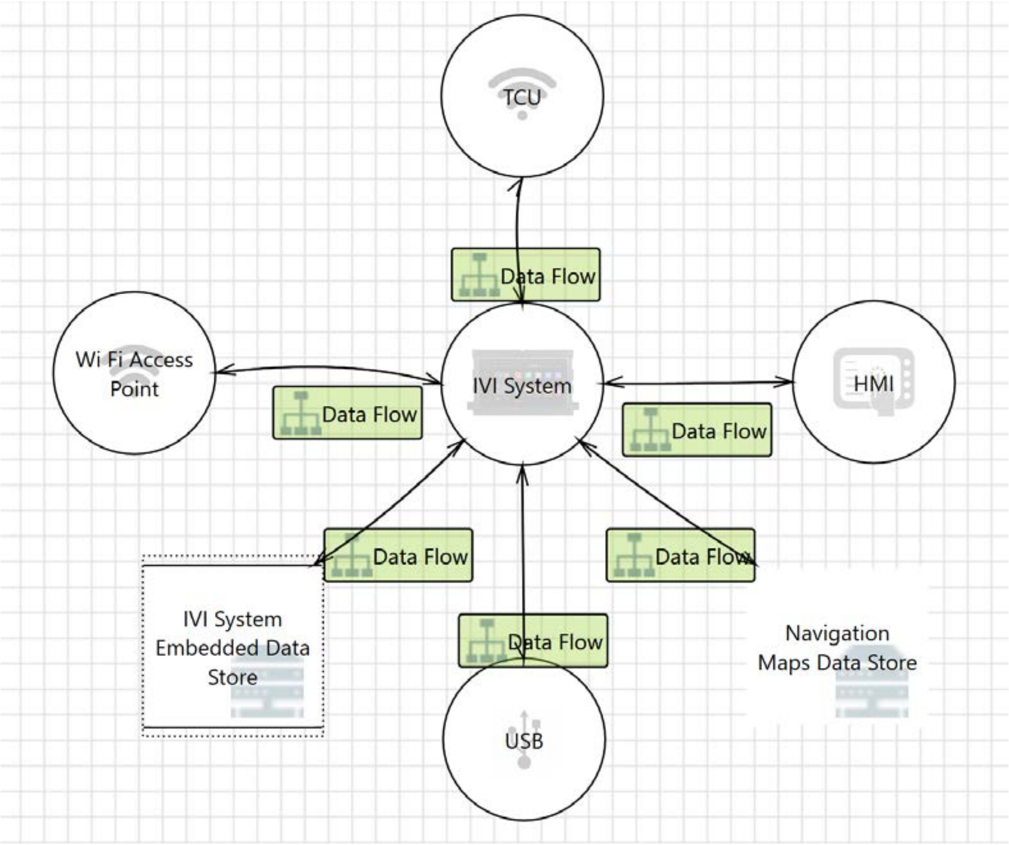
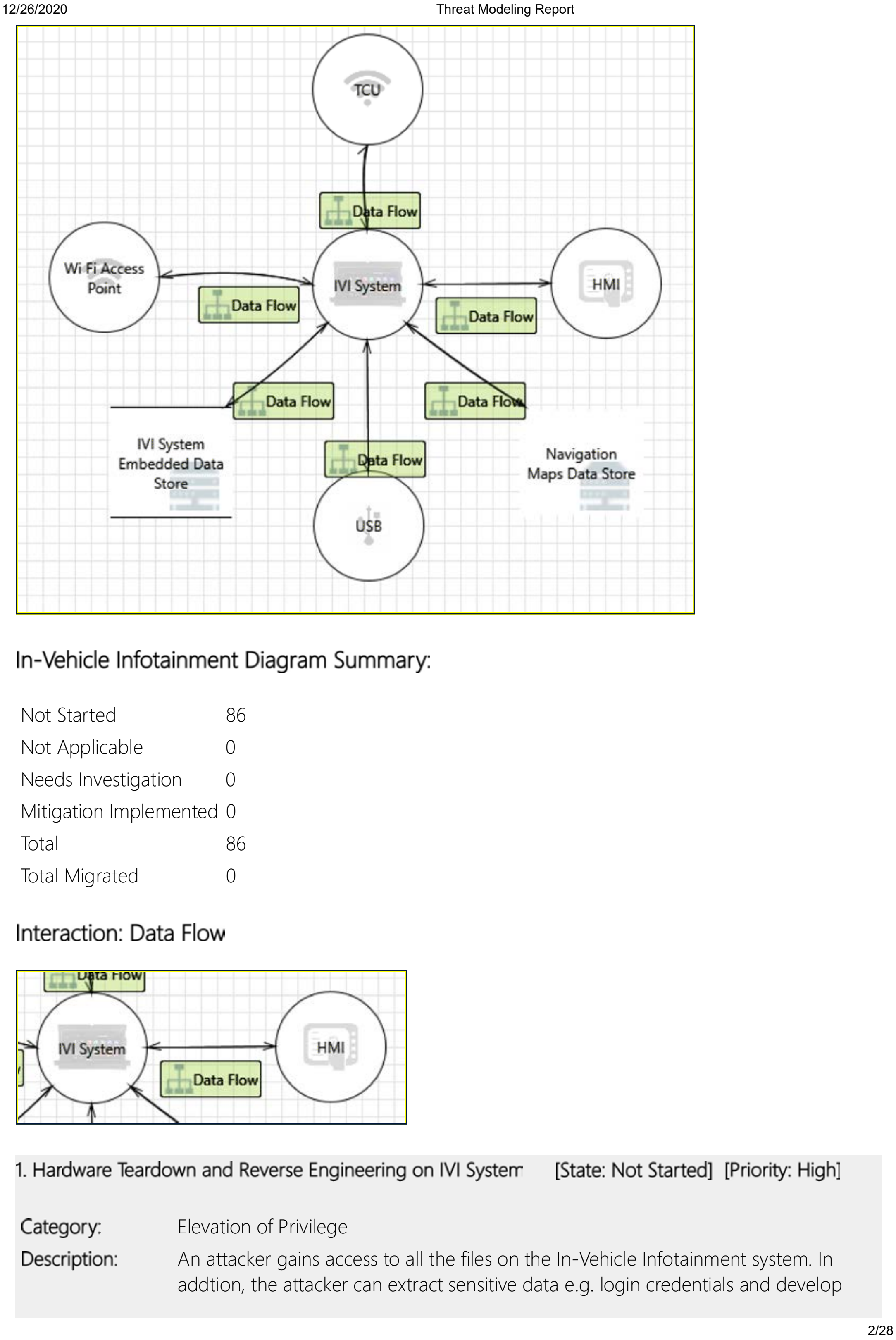


Figure 4.7: *Example data flow diagram depicting the infotainment system and its associated components identified in the previous phase.*

threat report as well as the details of five different threats (see Appendix C for the complete threat modeling report). In addition to the title of the threat, other useful pieces of information are included for each identified threat entry including priority, justification, description, the STRIDE category, and possible attack method(s). We use most of this information in the *Threat Modeling* phase for constructing attack trees.

4.3.3 Threat Modeling

While the threat report generated by the TMT enumerates all identified threats that can potentially be leveraged by cybercriminals to compromise the system security, it is often limited to a high-level description of the threat and attack methods, providing no information about the specific steps/actions performed for compromising the system security [118]. Effective security testing requires a good understanding of the different ways employed by the adversary to carry out these attacks. Attack trees can effectively assist with identifying specific techniques and associated actions performed by an attacker. Attack tree construction process requires a clear identification of the



12/26/2020 Threat Modeling Report

In-Vehicle Infotainment Diagram Summary:

Not Started	86
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	86
Total Migrated	0

Interaction: Data Flow

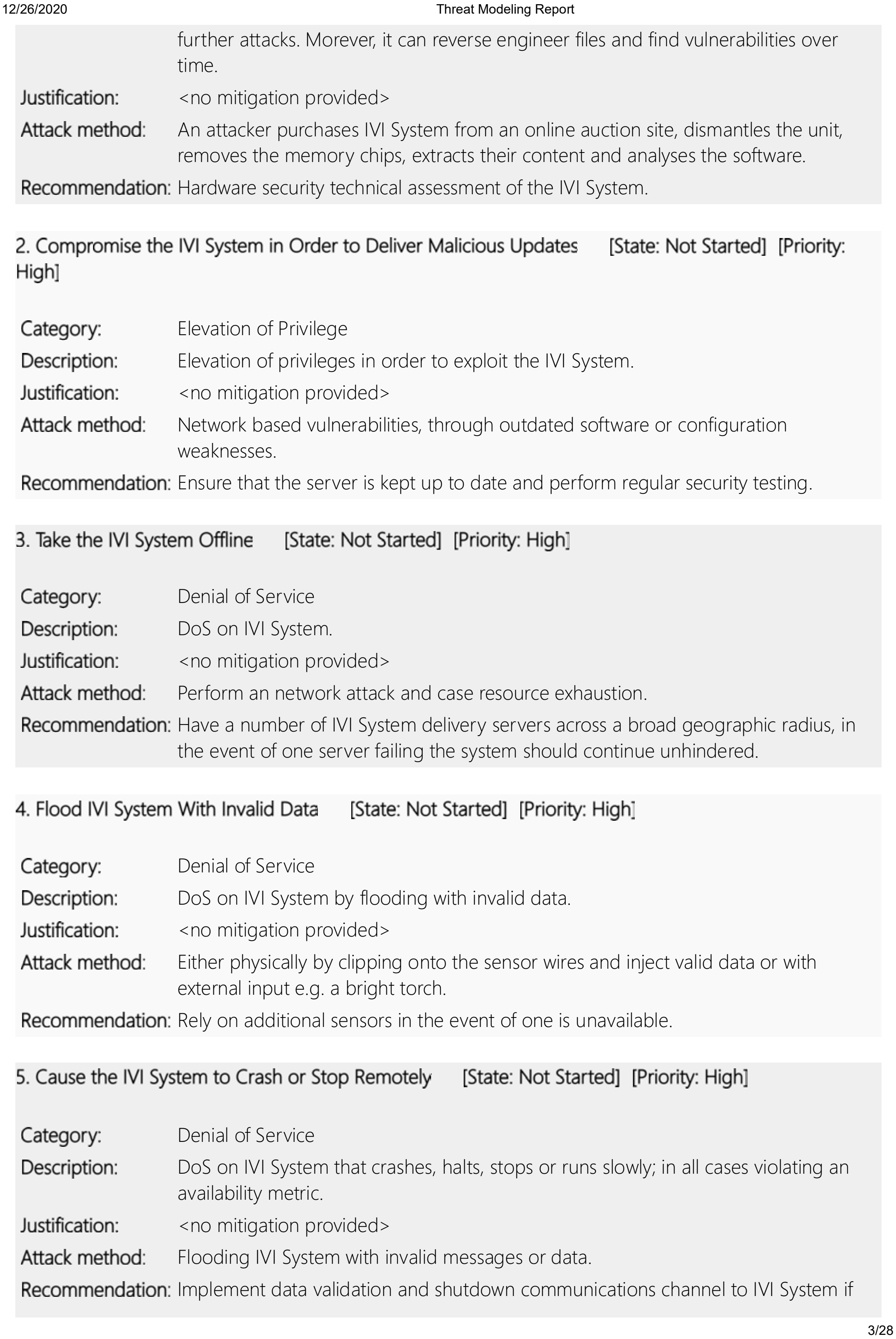
1. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop

2/28

Figure 4.8: *A summary page extracted from the Threat Report generated by the Threat Modeling Tool. To view complete threat report, please see Appendix C.*



12/26/2020 Threat Modeling Report

further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

2. Compromise the IVI System in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

3. Take the IVI System Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of IVI System delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

4. Flood IVI System With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

5. Cause the IVI System to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if

3/28

Figure 4.9: *Page two extracted from the Threat Report generated by the Threat Modeling Tool, showing details of various identified threats for the IVI system.*

attacker's goal, which serves as the root node of the attack tree. This is then followed by identifying subgoals and specific attack techniques that can help achieve the overall attacker's goal. A comprehensive description of the threat modeling (constituting attack tree construction process) is given in the next chapter.

4.4 Test Case Generation and Execution

All the activities carried out in the preceding phases contribute towards producing the building blocks required for generating security test cases. Attack trees constructed in the *Threat Assessment* phase facilitate the automated test case generation and execution. Chapter 6 presents an extensive description of the test case generation and execution process and its associated bespoke software tool. The tool has been developed for the automation of the test case generation and execution, and is capable of deriving test cases by performing attack-tree structure analysis.

4.5 Chapter summary

This chapter presented a detailed overview of the systematic testing approach adopted in this thesis with relevant examples of key activities to showcase its application and effectiveness. The approach, inspired by the well-known Penetration Testing and Execution methodology and model-based security testing approach, is comprised of three major phases, each constituting different activities performed in a step-by-step manner to derive effective security test cases. The *System Decomposition* activity in *Information Gathering* divides the target system into subsystems, components and subcomponents for the identification of cyber assets in the target system. This is followed by the *Threat Enumeration and Threat Modeling* activities in the phase *Threat Assessment* aiming at identifying relevant potential security threats by using the STRIDE methodology and refining the identified threats using attack trees, respectively. Finally, the *Security Testing* phase concludes the testing process by generating and executing the automated security test cases by leveraging the software tool that relies on attack trees for the test case derivation.

Chapter 5

Constructing Attack Trees

The preceding chapter has introduced the overall security testing approach we employ in this thesis, which includes various phases and related activities. This chapter details the approach to constructing attack trees based on the threats identified/enumerated in preceding steps of our security testing approach. We first present an overview of the approach for constructing attack trees by outlining key activities that are prerequisites to the attack-tree construction, leading to the process of building attack trees in a systematic manner. Finally, we use an example to showcase our step-by-step approach for creating attack trees. The contribution of this chapter is the systematic approach for constructing attack trees.

5.1 A Step-by-Step Approach for Constructing Attack Trees

While the attack tree is a mature and widely used threat modeling approach that security analysts used to depict security threats, prior studies (e.g. [119, 120, 121]) employing this technique do not specify/describe a method for constructing attack trees. We address this gap by introducing a step-by-step approach for building attack trees systematically. The attack-tree construction process is facilitated by applying a top-down approach, that is, the major steps for constructing the attack tree include identifying an overall goal that an attacker would like to achieve by compromising the system security, followed by identifying one or more different ways that can assist the attacker in achieving the overall goal. From a security analyst's perspective, attacker's overall goal is a threat to the system's security. This implies that identifying attacker's goal is synonymous with identifying a security threat. However, recall that threat identification requires information about the target system for determining possible intrusion points and vulnerabilities that could potentially be exploited to attack the system (for more details, see Chapter 4).

What is shown in the Figure 5.1 is a structured, systematic approach to the attack-tree construction process. As a first step, attack-tree construction requires a clear identification of the attacker's goal, which serves as the root node of the attack tree. This is then followed by identifying subgoals and specific attack techniques that can help achieve the overall goal.

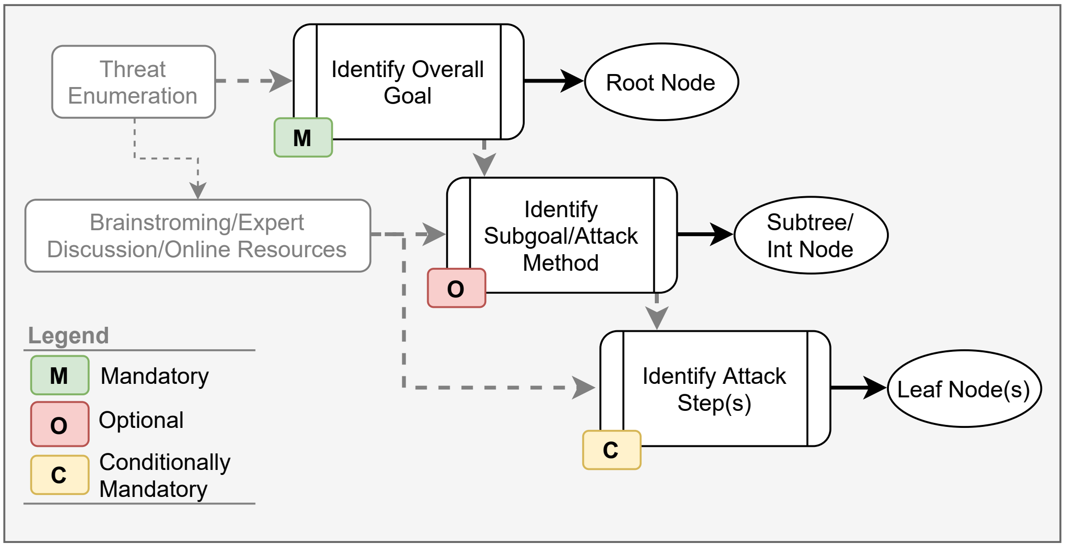


Figure 5.1: *A graphical overview of the attack tree construction approach.*

One of the threat scenarios from the threat list (resultant from enumeration process) is selected as the root node of the attack tree. The second step focuses on identifying any potential subgoals and/or attack methods, which can be accomplished using brainstorming sessions, expert discussions, and/or publicly available online resources. Note that since an attack tree must have exactly one root node and at least one attack step/leaf node, which can be the root node itself; hence it is considered mandatory to identify the root node, and conditionally at least one leaf node. On the other hand, having an intermediate node or subgoal is not necessary for all attack trees; therefore, it is considered optional. As a starting point, the threat report generated by TMT includes suggestions highlighting what kind of attack methods/techniques could be used by the attackers for each type of identified threat. This can be supplemented by brainstorming sessions that can serve as a useful tool for the identification, fine-tuning/tweaking of subgoals (intermediate nodes) and action steps (leaf nodes), wherein an overall goal is decomposed into intermediate goals and concrete actions in an iterative fashion. Finally, in addition to the expert knowledge and experience of the security team, publicly available online/open-source resources can facilitate the process of discovering modern, state-of-art attack techniques, methods, and tools used by malicious entities for materializing different types of threats, which can greatly help with attack-tree refinement.

5.2 Adaptive Cruise Control System: Examples of Constructing Attack Trees

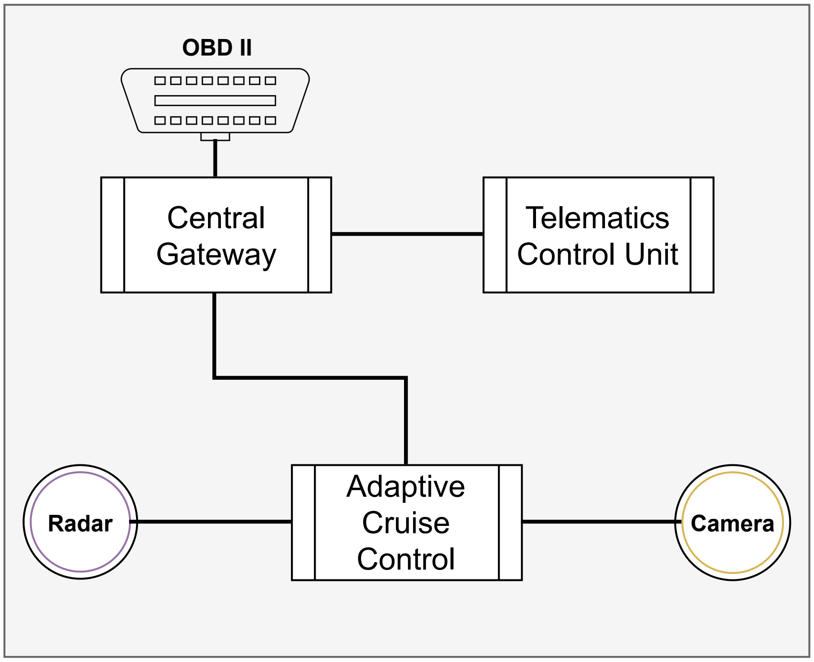


Figure 5.2: *Simplified graphical overview of some of the in-vehicle networked components including the central gateway unit, telematics control unit, adaptive cruise control ECU accompanied by radar and camera sensors.*

In what follows, we demonstrate the application of the approach by using an example involving the Adaptive Cruise Control (ACC) system. As a first step of the testing approach (that is, System Decomposition) the diagram in Figure 5.2 is produced by gathering information from various publicly available sources, providing an overview of different components and their relationships. It is imperative to note that we have deliberately included a limited number of ECUs/components in order to reduce unnecessary complexity arising from irrelevant technical details. Almost all modern vehicles do come with all the components shown in this diagram and more. A brief description of the adaptive cruise control system is provided below.

Adaptive cruise control, also referred to as dynamic or intelligent cruise control, maintains a set speed of the vehicle without relying on the input from the acceleration pedal, relieving the driver from some workload during the driving experience. It is also capable of adjusting the speed of the vehicle automatically depending on the traffic conditions, such as based on the speed of nearby moving vehicles. Adaptive cruise control relies on various on-board sensors (such as, radars, cameras, and/or lidar) for detecting any obstacles and vehicles moving around. The data from these sensors is

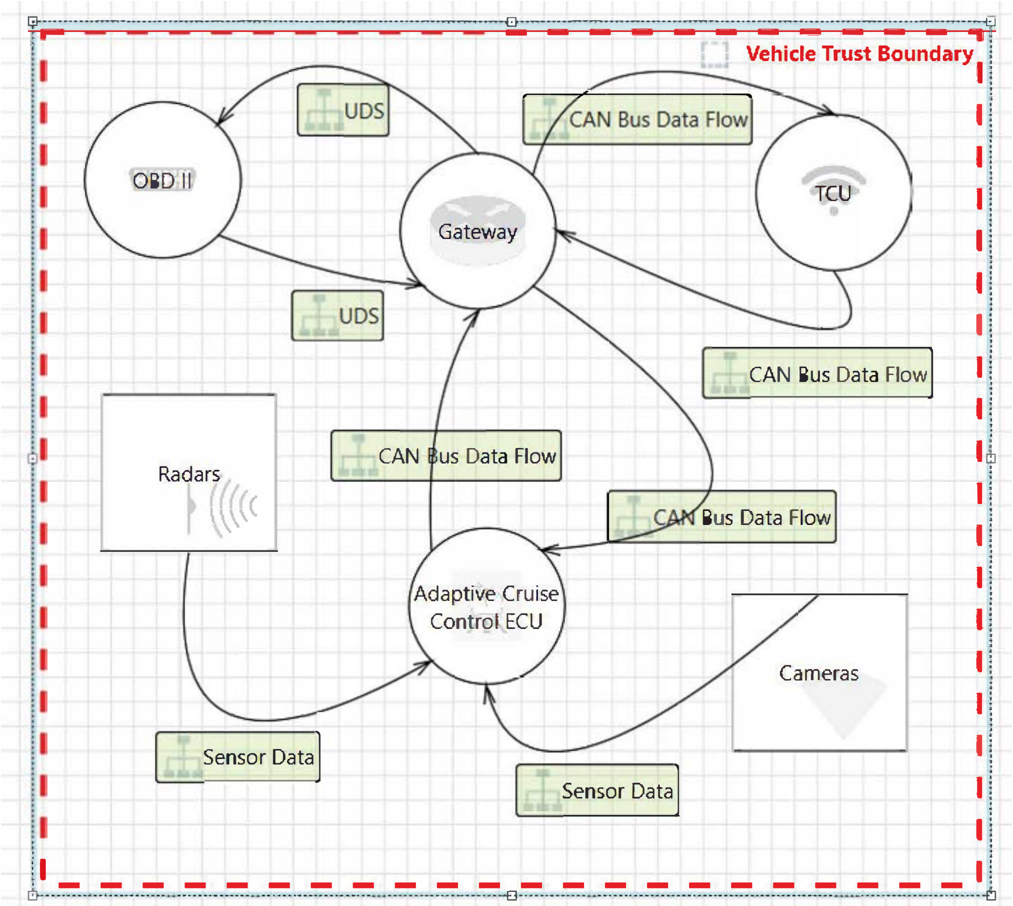


Figure 5.3: *Data flow diagram produced using the TMT tool by following the diagram presented in Figure 5.2, the same system represented using a different notation.*

consumed by the ACC ECU, which controls the throttle and braking of the vehicle.

Based on the system model shown in the Figure 5.2, data flow diagram (DFD) presented in Figure 5.3 was created using TMT. As can be noticed that the system model depicted by this DFD is almost the same except for the symbols and notation used. The threat report generated based on this model is the major output from this exercise providing the basis for constructing attack trees. Table 5.1 presents all 52 threats identified by the TMT grouped into STRIDE threat categories. Since the primary focus of this example is the ACC ECU, we therefore do not consider threats related to other ECUs/components in the list. One representative threat from each of the categories is chosen for building the attack trees. We denote the selected threats for further analysis with an asterisk symbol. As some threats may have more than one occurrence, a count value has been listed to indicate how many instances of a particular threat have been identified. For this example, we short listed three different relevant threat scenarios belonging to Spoofing, Denial of Service, and Elevation of Privilege categories. No ACC ECU related threats were identified for the Tampering, Repudiation, and Information Disclosure categories of the STRIDE model.

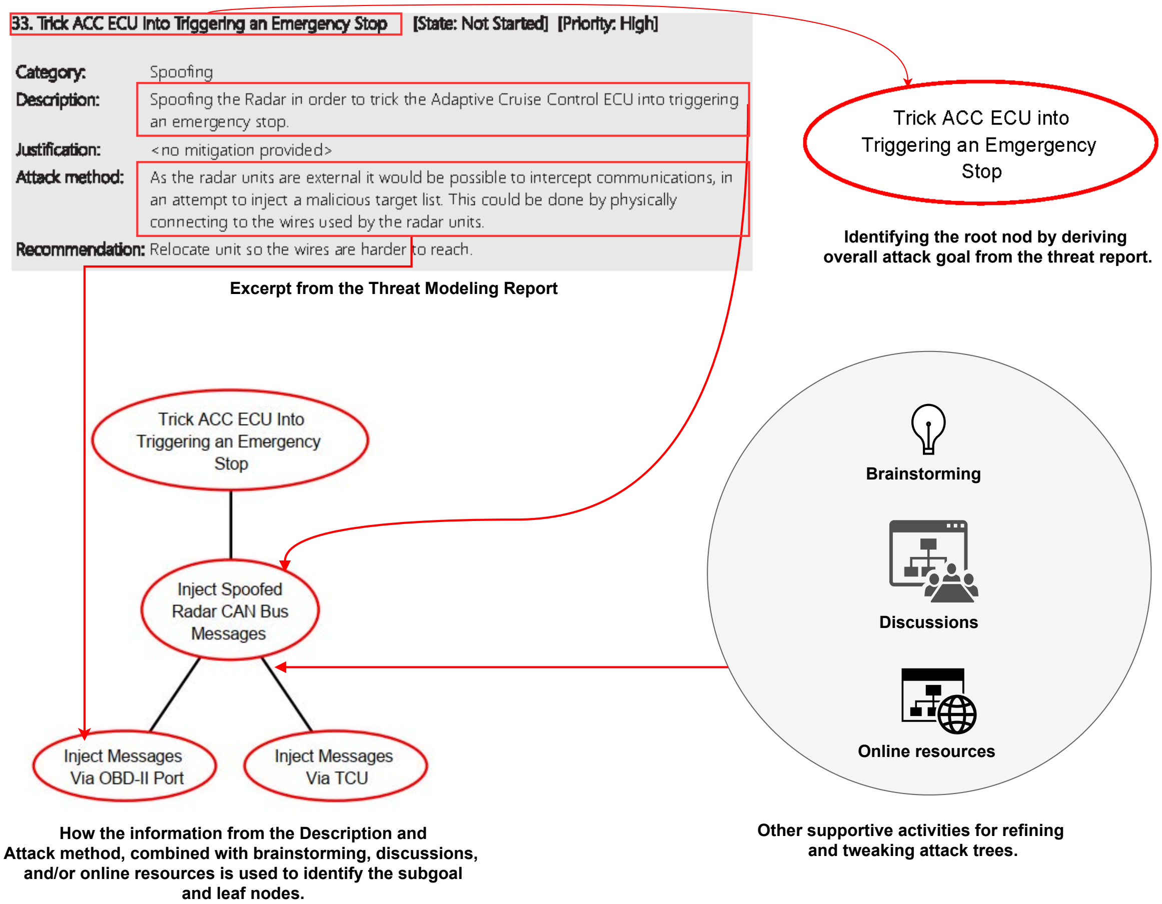
As mentioned earlier, each attack tree has exactly one root node, which represents the main goal of the adversary. One instance of each of the selected threats becomes the root node (or main goal) of the attack tree as outlined above. Now the primary goal of the attacker has been established, different methods/techniques that can potentially be used to achieve this goal are to be identified. The original report generated by TMT suggests at least one way to realize the threat. In most cases, these suggestions are relevant and serve as an excellent starting point for populating the attack tree with appropriate subgoals (child nodes) and/or actions (leaf nodes). For each threat scenario, we started with the attack method suggested by TMT followed by brainstorming and discussion sessions to determine what are the possible steps/actions that can potentially be carried out by cybercriminals. The resultant attack trees are shown in the figures 5.5 to 5.7.

The first attack tree (in Figure 5.5) represents the threat one entitled *Trick ACC ECU into Triggering an Emergency Stop* from the Table 5.1, and it belongs the the Spoofing category of the STRIDE threat classification model. The potential attack method for realizing this threat from TMT is to spoof radar messages. Figure 5.4 shows the approach in action by depicting all the steps applied to construct this attack tree. In step one, the overall goal was identified followed by step two which involved identifying the subgoal and leaf nodes by means of the information from the threat modeling report, discussions, and brainstorming. A brief analysis of the diagrams in Figures 5.2 and 5.3 enabled us to identify at least two different attack paths for accomplishing this very task of injecting spoofed radar messages. This resulted in the identification of the subgoal or an OR subtree (Inject Spoofed Radar CAN Bus Messages). Since it can be done in two different ways; hence, the subtree has been assigned with two disjunctive leaf nodes corresponding to two different attack paths: Inject Messages Via OBD-II Port and Inject Messages Via TCU.

The attack tree shown in Figure 5.7 represents the threat seven (entitled *Cause*

Table 5.1: *A summary of the 52 threat instances identified by the Threat Modeling Tool. Threats have been grouped into the STRIDE categories and a count of each threat type has been specified.*

Category	#	Threat	Count
Spoofing	1	* Trick ACC ECU Into Triggering an Emergency Stop	1
	2	Cause the Car to Perform Emergency Braking	1
Tampering	3	Modify Data Being Sent to the TCU While in Transit	1
Repudiation	–	–	–
Information Disclosure	4	Updates Could be Downloaded	6
	5	Data Flow Sniffing	8
	6	Car Could be Tracked	1
Denial of Service	7	* Cause the ACC ECU to Crash or Stop	1
	8	Cause the ACC ECU to Crash or Stop Remotely	1
	9	Flood ACC ECU With Invalid Data	1
	10	Take the ACC ECU Offline	1
	11	Cause the Gateway to Crash or Stop	3
	12	Cause the Gateway to Crash or Stop Remotely	3
	13	Flood Gateway With Invalid Data	3
	14	Take the Gateway Offline	3
	15	Cause the TCU to Crash or Stop	1
	16	Cause the TCU to Crash or Stop Remotely	1
	17	Flood TCU With Invalid Data	1
	18	Take the TCU Offline	1
	19	Cause the OBD-II to Crash or Stop	1
	20	Cause the OBD-II to Crash or Stop Remotely	1
	21	Flood OBD-II With Invalid Data	1
	22	Take the OBD-II Offline	1
Elevation of Privilege	23	Compromise the ACC ECU to Deliver Malicious Updates	1
	24	* Reflash the ACC ECU From the CAN Bus to Send Arbitrary CAN Messages	1
	25	Compromise the Gateway to Deliver Malicious Updates	1
	26	Compromise the TCU to Deliver Malicious Updates	1
	27	Reflash the TCU Firmware to Send Arbitrary CAN Messages	2
	28	Compromise the Gateway to Deliver Malicious Updates	3
	29	Compromise the OBD-II to Deliver Malicious Updates	1



33. Trick ACC ECU Into Triggering an Emergency Stop [State: Not Started] [Priority: High]

Category: Spoofing

Description: Spoofing the Radar in order to trick the Adaptive Cruise Control ECU into triggering an emergency stop.

Justification: <no mitigation provided>

Attack method: As the radar units are external it would be possible to intercept communications, in an attempt to inject a malicious target list. This could be done by physically connecting to the wires used by the radar units.

Recommendation: Relocate unit so the wires are harder to reach.

Excerpt from the Threat Modeling Report
Trick ACC ECU into Triggering an Emgergency Stop
Identifying the root nod by deriving overall attack goal from the threat report.
Trick ACC ECU Into Triggering an Emergency Stop
Inject Spoofed Radar CAN Bus Messages
Inject Messages Via OBD-II Port
Inject Messages Via TCU
Brainstorming
Discussions
Online resources
How the information from the Description and Attack method, combined with brainstorming, discussions, and/or online resources is used to identify the subgoal and leaf nodes.
Other supportive activities for refining and tweaking attack trees.

Figure 5.4: *Applying the attack-tree construction approach to build the first attack tree shown in Figure 5.5.*

the ACC ECU to Crash or Stop), and it belongs to the Denial of Service category of the STRIDE model. In contrast with the previous attack tree, this one is a SAND or sequential AND tree, with a defined ordering of the actions. TMT suggested to flood the ACC ECU with invalid data for causing it to stop or crash. In order to flood or overwhelm the target ECU with invalid messages, we must first establish a connection with the concerned ECU. Looking at the DFD diagram, we discovered a subtree with two different ways to connect to the ECU: Connect through OBD-II Port or Connect through TCU. Either of these entry points can be leveraged to flood the the ACC ECU. However, the connection must be established before we can send the messages to our target ECU; which means, correct ordering of these operations is critical. Therefore, a SAND tree is the best fit here.

Finally, the SAND attack tree in Figure 5.6 represents the threat 24 (entitled *Reflash the ACC ECU from the CAN Bus to Send Arbitrary Messages*), and it belongs to the Elevation of Privilege category of the STRIDE model. This attack tree has some similarities with the attack tree in Figure 5.7 including being a SAND attack tree, and

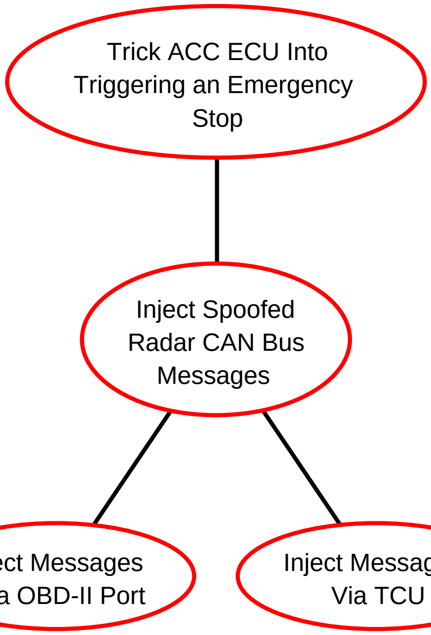

Trick ACC ECU Into Triggering an Emergency Stop
Inject Spoofed Radar CAN Bus Messages
Inject Messages Via OBD-II Port
Inject Messages Via TCU

Figure 5.5: *Attack tree depicting the threat aiming at spoofing the radar signals to trick the adaptive cruise control ECU to cause the vehicle to perform an emergency brake.*

containing an OR subtree. Since the attack paths were already known from previous two exercises, we were able to populate the subtree with those known attack paths (i.e., Connect through OBD-II Port or Connect through TCU). The remaining two steps were not difficult to identify, which include downloading the compromised firmware image into the ECU followed by reflashing the ECU with this image. This is obvious to note that similar to the previous SAND attack tree, all the steps must be performed in the specified order.

It is worth noting that the attack paths identified above do not directly provide a connection to the ACC ECU from OBD-II port or TCU, it involves the Central Gateway Unit. We used these examples to illustrate the step-by-step way of constructing the attack trees. The next chapter presents the test-case generation process, which is fully dependent on the attack trees that we construct in this particular step of the security test approach.

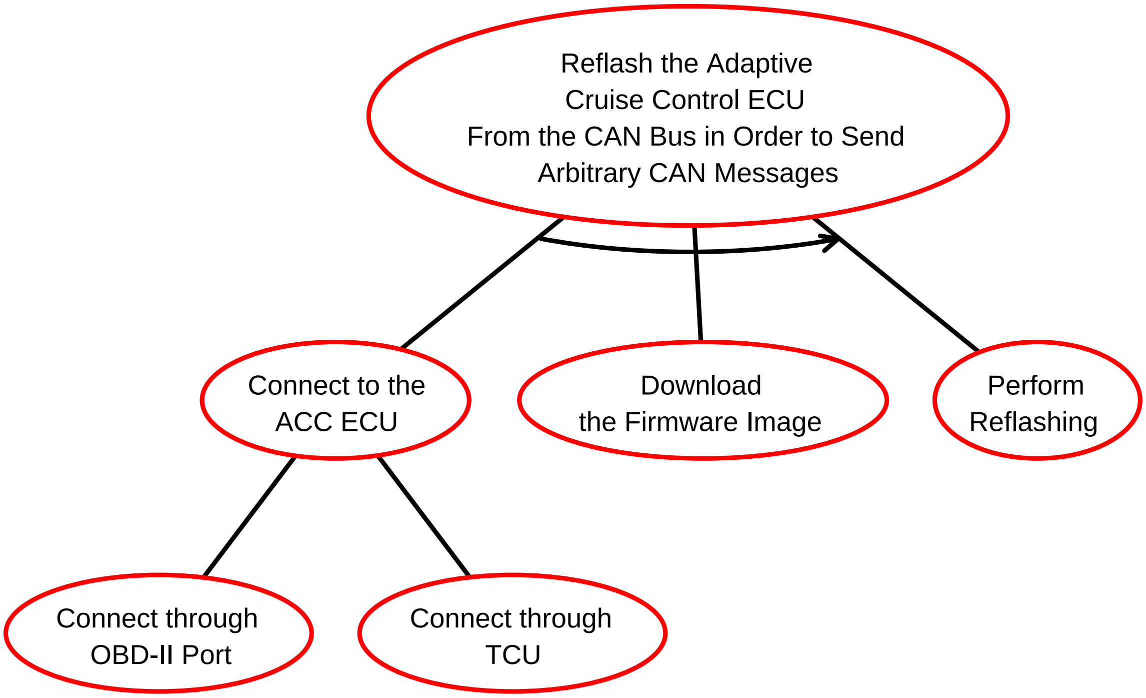


Figure 5.6: *This is another SAND attack tree with an OR subtree. As usual, after placing the threat scenario at the root node, specific attack steps have been identified by brainstorming and discussion, taking into account the suggestions from the TMT.*

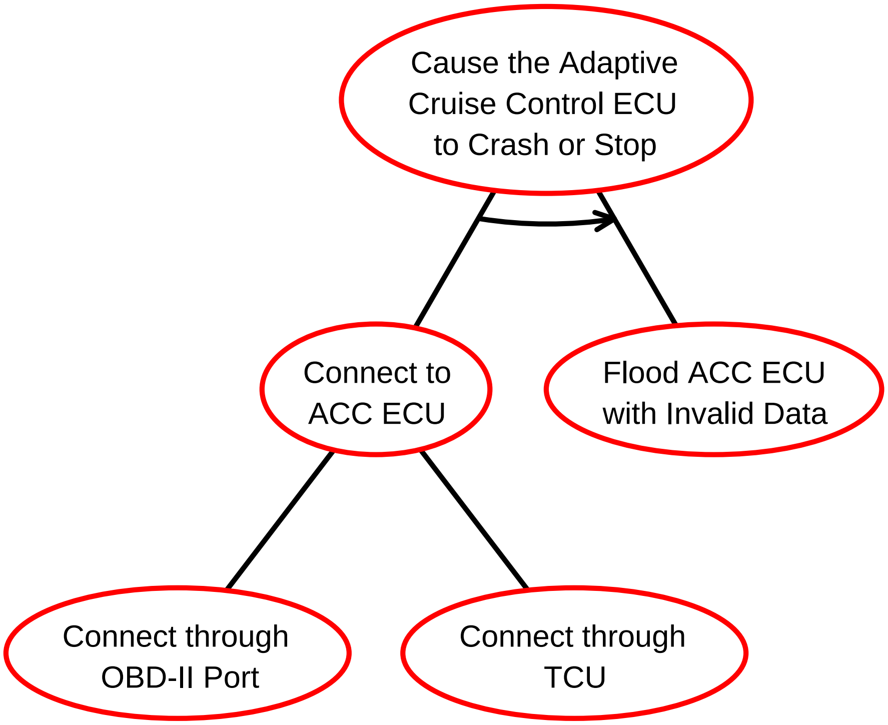

Cause the Adaptive Cruise Control ECU to Crash or Stop
Connect to ACC ECU
Flood ACC ECU with Invalid Data
Connect through OBD-II Port
Connect through TCU

Figure 5.7: *This attack tree represents the threat involving denial-of-service (DoS) attack on the adaptive cruise control ECU in order to cause it to stop working.*

5.3 Chapter summary

In this chapter, we have presented a structured, systematic approach to constructing and populating attack trees using the STRIDE threat classification model and its associated tool. Using adaptive cruise control system as an example to illustrate our approach, we have shown the step-by-step process for identifying the overall goal of the attacker followed by specific attack methods/steps that help construct/populate the attack tree. The example presented in this chapter shows all the activities from system decomposition to threat enumeration to attack tree construction.

Chapter 6

Generating Test Cases

In the last chapter, we described our systematic, step-by-step approach to constructing attack trees by using different detailed examples. This Chapter is concerned with presenting the test-case generation process, which relies on the attack trees created in previous steps. We introduce our test-case generation software tool that is capable of deriving test cases by analyzing the structure of the attack tree. After providing an overview of the test-generation approach, we explain the algorithm for the software tool followed by a formal proof to show the correctness of test case derivation approach. Finally, three different examples are used to demonstrate the approach in action.

6.1 Test Case Generation

Traditional security test-case derivation process tends to be unstructured, irreproducible, reliant on the expertise and experience of the tester, undocumented, and having no or inadequate rationales for the test design. In order to address these shortcomings or minimize their impact, model-based security testing approaches rely on the explicit model of the system-under-test for systematic (and often automated) specification, derivation, and execution of the security test cases [122, 71].

Since this study adopts a model-based security testing approach, a software tool has been designed and implemented (previously introduced in [112]) for automating the test case derivation and execution process. This tool has been completely redesigned and rewritten by eliminating its dependencies on third-party tools and libraries for the performance and efficiency improvements. The tool, written in Python programming language, has a command-line based user interface and consists of two main modules, one of which is responsible for the test case generation and the other one for executing those test cases against the system under test. The test case generator module accepts an XML-based attack tree, analyzes its structure, derives test cases based on the semantics of the input attack tree, and writes the derived test cases to a plain text file. The executor module uses that file for executing the test cases against the target system. All test cases have their corresponding test scripts stored in a separate file.

Table 6.1: *An attack tree represented in XML format.*

XML representation of the Attack Tree as displayed in Figure 2.10.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sandtree>
  <node refinement="conjunctive">
    <label>Attack Goal</label>
    <node refinement="sequential">
      <label>Subgoal 1</label>
      <node refinement="conjunctive">
        <label>Action 1</label>
        <comment>Test Script A</comment>
      </node>
      <node refinement="conjunctive">
        <label>Action 2</label>
        <comment>Test Script B</comment>
      </node>
    </node>
    <node refinement="disjunctive">
      <label>Subgoal 2</label>
      <node refinement="conjunctive">
        <label>Action 1</label>
        <comment>Test Script C</comment>
      </node>
      <node refinement="conjunctive">
        <label>Action 2</label>
        <comment>Test Script D</comment>
      </node>
    </node>
  </node>
</sandtree>
```

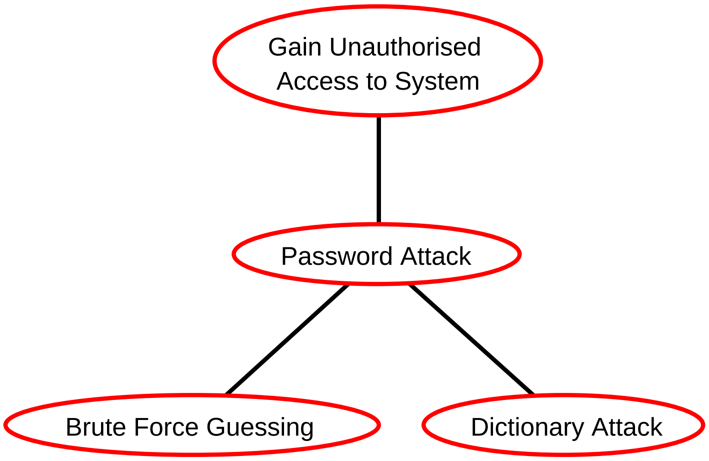


Figure 6.1: *An example attack tree with an (**OR**) subtree representing a subgoal that helps achieving the overall attack goal of the attacker.*

The test executor uses that file for loading the test scripts to be executed against the target corresponding to the security test case being executed.

Before introducing the approach in more detail, a definition of a security test case is warranted here to explain what a derived security test case entails.

Definition 6.1.1. A security test case is a finite sequence of atomic actions or events from $\mathbb{A}$ (recall from Definition 2.3.1 in Chapter 2 that $\mathbb{A}$ denotes a set of potential atomic actions of the attacker) that attempts to check if the System Under Test (SUT) can defend against the test case. The outcome of executing a security test case against the SUT can either be *PASS* or *FAIL*. A test case yields a PASS outcome if the SUT is able to thwart a cyberattack effectively, whereas failure to defend a security attack leads to a FAIL outcome.

To better illustrate the process and the resultant output (i.e., test cases), we use some simple attack trees. The first example uses a simple **OR** attack tree (shown in Figure 6.1) with one subtree having two leaf nodes depicting the primitive actions of the attacker. As evident, the overall goal of the attacker involves gaining unauthorised system access. The attacker tries to accomplish this by means of a password attack. Traversing through this attack tree results in two unique sequences of events (i.e., Brute Force Guessing and Dictionary Attack) or attacks, either of which can be used to achieve the overall attack goal. Please note that goals and subgoals are not included in the test cases, only atomic attacker actions are used to construct a security test case. Listing 6.1 shows the test cases generated from the attack tree (see Figure 6.1).

The second example uses an **AND** attack tree with two leaf nodes. It should be noted that since an AND attack tree requires both of the actions to be completed in order to achieve the overall goal, hence each test case is composed of both of the actions. However, because the order in which these actions are performed is immaterial, because

```
Test Case 1: <Brute Force Guessing>
Test Case 2: <Dictionary Attack>
```

Listing 6.1: Test Cases derived from the OR attack tree presented in Figure 6.1.

both ways lead to the same result; hence two different executions (permutations) have been resulted.

The test cases derived by analyzing the structure of this particular attack tree are shown in Listing 6.2:

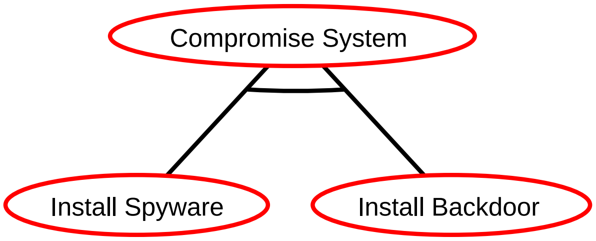


Figure 6.2: *An example (**AND**) attack tree with the overall goal of compromising the system by installing spyware (e.g. keylogger) and a backdoor for persistent unauthorised access to the system for carrying out further exploitation.*

```
Test Case 1: <Install Spyware, Install Backdoor>
Test Case 2: <Install Backdoor, Install Spyware>
```

Listing 6.2: Test Cases derived from the AND attack tree presented in Figure 6.2.

The third example involves a **SAND** subtree with two leaf nodes. Overall attack goal aims at retrieving user credentials (i.e., usernames and passwords) from the database. The attack method used by the attacker is the well-known SQL injection attack. The query is written in such a way that can fetch confidential, private information, bypassing the application access control restrictions. The test cases derived from this attack tree are shown in Listing 6.3. Since this is a SAND tree, the actions must be performed in the correct order, as shown by the direction of the arrow.

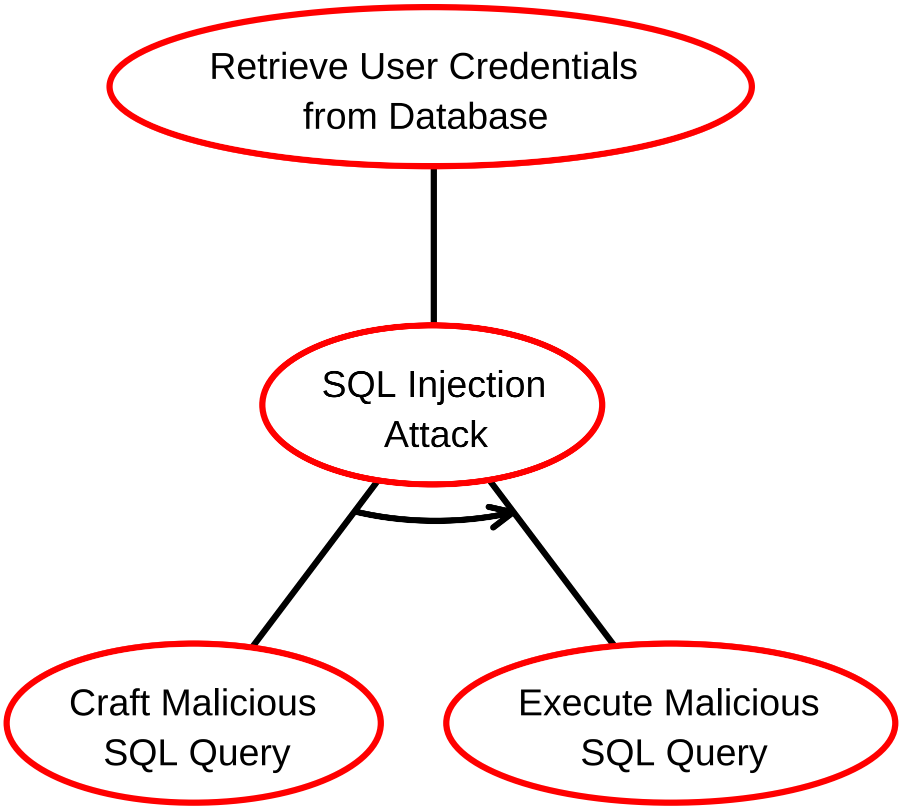

Retrieve User Credentials from Database
SQL Injection Attack
Craft Malicious SQL Query
Execute Malicious SQL Query

Figure 6.3: *An example attack tree with a (**SAND**) subtree representing a subgoal that helps achieving the overall attack goal of the attacker. The subtree in turn is composed of two leaf nodes representing the atomic attack actions.*

It is worth noting for this particular type of attack tree that both actions must be executed in the specific order shown for the overall goal to be accomplished successfully, hence two leaf nodes of this SAND tree resulted in a single security test case. The final example demonstrates four different test cases (see Listing 6.4) derived from a more complex overall **SAND** attack tree with different subtrees. The overall goal of this attack tree involves delivering a malicious update to an IoT device. The first task is preparing the malicious update to be delivered to the target device. An AND subtree depicts the two different mandatory steps/actions for preparing the update. Since the refinement type of this subtree is AND, the actions can be carried out in any order, even in parallel. The OR subtree (Compromise Cryptographic Keys for SW Signing) in turn has two leaf nodes (which are atomic actions). Only one of these actions will be suffice for compromising the keys. This combination of subtrees with different refinement types and interrelationships leads to interesting test cases, as shown in the Listing 6.4. Since, overall it is a **SAND** tree; ordering of the actions in which they are executed is very important for achieving the desired outcome.

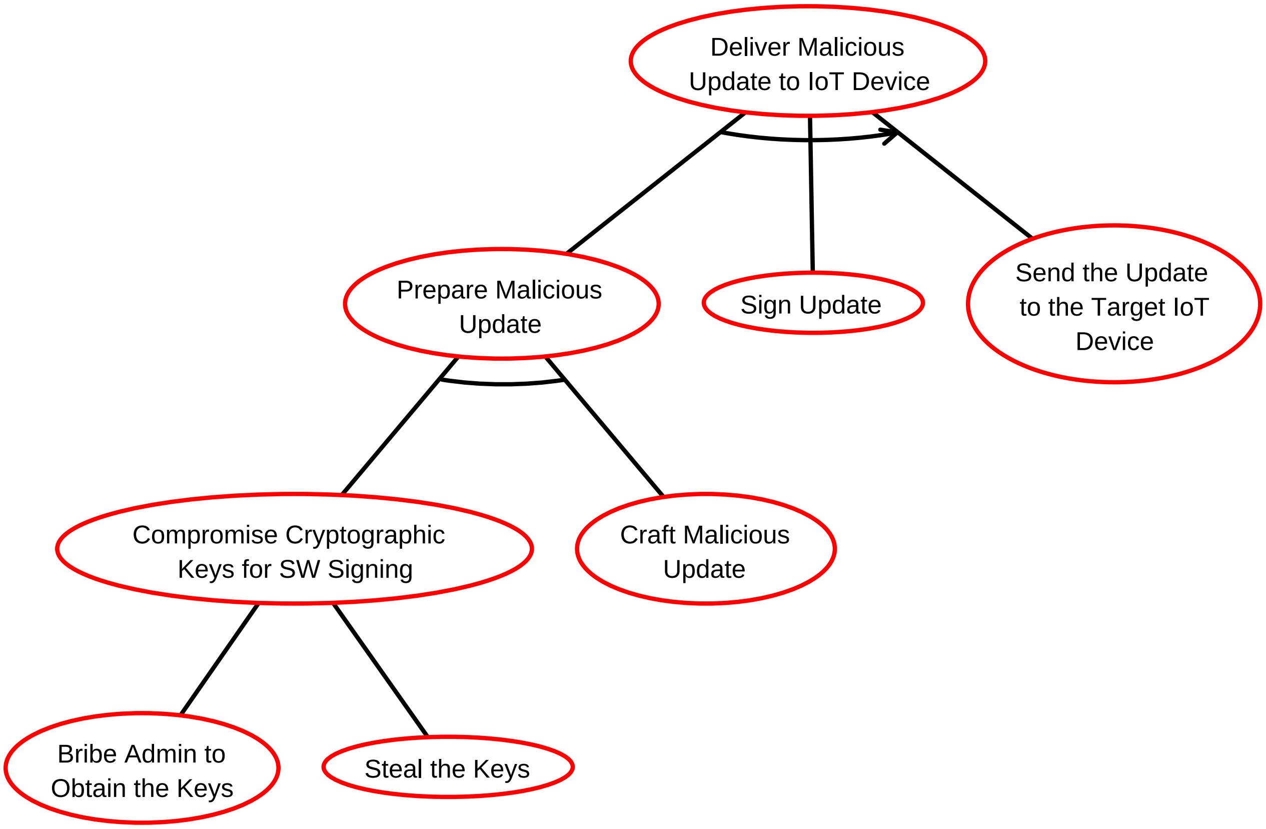

Deliver Malicious Update to IoT Device
Prepare Malicious Update
Sign Update
Send the Update to the Target IoT Device
Compromise Cryptographic Keys for SW Signing
Craft Malicious Update
Bribe Admin to Obtain the Keys
Steal the Keys

Figure 6.4: *An example overall* ***SAND*** *attack tree with an* ***OR*** *and an* ***AND*** *subtree representing different subgoals, each of which can help achieve the overall attack goal of delivering a malicious update.*

Later in this chapter we will provide more details on how the executable test cases are generated and what additional information they carry. Following section presents the algorithm we devised for generating security test cases by analyzing the structure of the attack trees.

6.2 Test Case Generation Algorithm

Algorithm 1 outlines the main logic and major steps for deriving security test cases by analyzing the structure of the attack tree. The function GenerateTestCases (for convenience, we will henceforth refer to GenerateTestCases as GTC) accepts an attack tree as input. For this purpose, the tool requires the attack tree to be in the XML representation as an example shown in the Figure 6.1. The conversion can be accomplished by using the built-in feature of the ADTool, allowing the attack tree to be exported in XML format.

As can be observed, GTC has a number of if-else blocks for determining the refine-

```
 Test Case 1:
<Craft Malicious SQL Query, Execute Malicious SQL Query>
```

Listing 6.3: Test Case derived from the SAND attack tree presented in Figure 6.3.

```
 Test Case 1:
<Bribe Admin, Craft Update, Sign Update, Send Update>
 Test Case 2:
<Steal Keys, Craft Update, Sign Update, Send Update>
 Test Case 3:
<Craft Update, Bribe Admin, Sign Update, Send Update>
 Test Case 4:
<Craft Update, Steal Keys, Sign Update, Send Update>
```

Listing 6.4: Test Cases derived from the mixed attack tree presented in Figure 6.4.

ment type of the input attack tree and processes the attack tree/node accordingly.

The first case (beginning on line three of the algorithm) determines whether the input attack tree/node is a leaf node. This is determined by establishing whether the current node has a subtree or a child node, if none of these is true, this node will be considered a leaf node and appended to the set TC. Rationale for using the set data structure here is motivated by the unique characteristics of this data structure (i.e., sets are unordered, their elements are unique with no duplicates allowed, and the elements are immutable) that make them an appropriate choice for holding test cases derived from **OR** and **AND** trees, as the preservation of ordering is only applicable and required in the sequential AND (**SAND**) trees.

The second case (from line six to line 11) is applicable where the attack tree/node is an OR tree. Since this type of attack trees is assumed to have one or more children nodes, a recursive call is made to the function GTC by supplying the current (i.e., i_{th}) child node as input, which goes through the same process and the resultant return value (that is, a test case) of this function call is added to the set TC as a subset. This process is repeated for n number of times where n is the number of children nodes for a given attack tree/subtree. Once all the children nodes have been processed, union of all the test cases (i.e., TC_0 to TC_i) is appended to the set TC. It is important to note that each leaf node of an **OR** represents one complete test case. That is, if a given **OR** has

Algorithm 1: This is the test-case generation algorithm that derives test cases by analyzing the structure and based on the the formal semantics of the attack tree. It accepts an attack tree as input, generates all possible test cases, and provides a set of concrete test cases. To view the source code implementing this algorithm, please see Appendix A.

```
Input: Attack Tree
Output: Test Cases
initialization;
Function GenerateTestCases($attackTree$):
    if $attackTree.type == leaf\ node$ then
        TC ← $attackTree$
        return TC
    else if $attackTree.type == OR\ node$ then
        foreach $child\_node_i$ in $attackTree_{(i=0,...,n)}$ do
            $TC_i \leftarrow GenerateTestCases(child\_node_i)$
        end
        $TC \leftarrow union(TC_0, ..., TC_n)$
        return TC
    else if $attackTree.type == AND\ node$ then
        foreach $child\_node_i$ in $attackTree_{(i=0,...,n)}$ do
            $TC_i \leftarrow GenerateTestCases(child\_node_i)$
        end
        $TC \leftarrow union(product(TC_0, ..., TC_n))$
        $TC \leftarrow permutations(TC)$
        return TC
    else if $rootNode.type == SAND\ node$ then
        foreach $child\_node_i$ in $rootNode_{(i=0,...,n)}$ do
            $TC_i \leftarrow GenerateTestCases(child\_node_i)$
        end
        $TC \leftarrow union(product(TC_0, ..., TC_n))$
        return TC
end
```

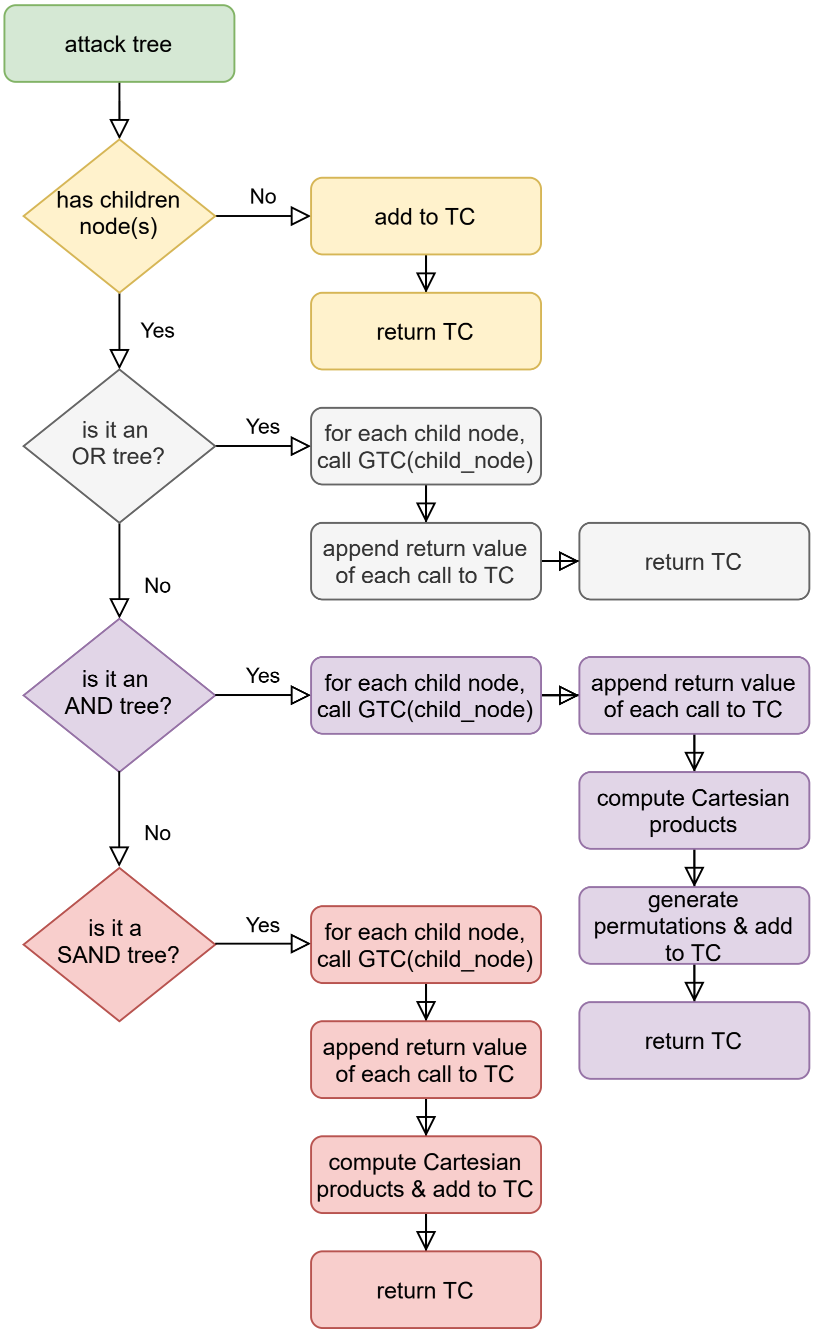

attack tree
has children node(s)
No
add to TC
return TC
Yes
is it an OR tree?
Yes
for each child node, call GTC(child_node)
append return value of each call to TC
return TC
No
is it an AND tree?
Yes
for each child node, call GTC(child_node)
append return value of each call to TC
compute Cartesian products
generate permutations & add to TC
return TC
No
is it a SAND tree?
Yes
for each child node, call GTC(child_node)
append return value of each call to TC
compute Cartesian products & add to TC
return TC

Figure 6.5: *This flowchart is a graphical representation of the algorithm 1.*

n leaf nodes, the number of derived test cases will be equal to n. However, remember that in the case of a complex attack tree with the root node being **OR**, which contains other types of (i.e., **AND** or **SAND**) attack trees as subtrees, the number of test cases will unlikely to equate the number of leaf nodes of that **OR** tree.

Starting on line 12 of the algorithm, the third case is applicable to the **AND** attack trees/nodes. Similar to the **OR** tree case explained above, children nodes are recursively processed and returned values are appended to an intermediate variable TC_i one by one. Once all the children nodes have been processed, Cartesian product of all the elements is computed followed by performing a union of all theses products, which is finally appended to the TC. An additional step here is the function permutations, which computes all the valid permutations of the test cases contained in the TC and reassigns the resultant output to the TC. The last two steps help achieve the *interleaving* of leaf nodes. This is in line with the formal semantics of the attack trees presented in [53] and described in the Section 6.3. A single test case derived from an **AND** attack tree can constitute more than one action steps. That is, multiple leaf nodes can be part of one test case. In general, the number of test cases derived from an **AND** attack tree will be $n!$ where n represents the total number of leaf nodes of a given **AND** attack tree.

The last case deals with the **SAND** attack tree/node. This case is very similar to the **AND** trees with a couple of differences. Firstly, since maintaining the sequential order of the test steps is crucial, a compatible data structure (e.g., lists) should be used that can preserve the order of the test steps. Secondly, unlike the **AND** attack trees/nodes, permutations are not generated. All children nodes are processed one-by-one by leveraging the recursion. An ordered list of elements returned is contained in the TC. A test case derived from a **SAND** attack tree is composed of all the action steps concatenated together. However, please keep in mind that a **SAND** attack tree consisting of other types of attack trees as its subtrees will yield a number of test cases based on the combined semantics of the parent and subtrees involved.

6.3 Sequence Semantics

Mantel and Probst [53] present the SAND attack-trees formal semantics by interpreting them as sequences, which is equivalent to the SP semantics of attack trees presented in [49] and described in Chapter 2. Similar to the procedure used for SP semantics in [49], the sequence semantics of a SAND tree are defined using a bottom-up approach wherein semantics of a leaf node is a sequence comprising of a single element formed by the label of the node, representing the attacker's primitive action. A set of attacks can be constructed from the semantics (depending on the refinement operator used, i.e., **OR, AND,** or **SAND**) of the subtrees of an inner node.

To begin with, we provide an introduction to the basic notions and the notation we use for presenting the sequence semantics of the SAND attack trees based on the formalisation presented in [53]. Using $\mathbb{N}$ for denoting the set of natural numbers, the notation $[m,n] = \{m, ..., n\}$ is used to denote a subset of the $\mathbb{N}$, where $m,n \in \mathbb{N} \wedge m < n$, and if $m > n$ then $[m,n] = \emptyset$ holds. Additionally, $max(N)$ denotes the unique maximal element of a nonempty, finite set $N \subseteq \mathbb{N}$. While the notation $X \rightharpoonup Y$ is used to denote the space of partial functions, $X \longrightarrow Y$ is used to denote the space of total functions from a domain X to a codomain Y. To retrieve the domain of a function, we use dom and for retrieving the codomain of a function we use $cdom$; that is, $dom(f) = dom(g) = X$, and $cdom(f) = cdom(g) =$ hold for $f : X \rightharpoonup Y$ and $g : X \longrightarrow Y$.

Furthermore, for retrieving the set of elements for which a function is defined and the set of elements that can be reached by a function, we use def and img, respectively.

In order to model infinite sequences over a set X, we use $\mathbb{N} \longrightarrow X$, and we denote the set of all such infinite sequences by $SEQ^{inf}(X)$. Additionally, in order to model finite sequences over X, we use functions $f : \mathbb{N} \rightharpoonup X$, with $def(f) = \emptyset$ or $def(f) = [0,n], \forall n \in \mathbb{N}$. The length of a finite sequence f is defined by $\#f = 0$ if $def(f) = \emptyset$, and $\#f = max(def(f)) + 1$ if $def(f) \neq \emptyset$. We denote the set of all such finite sequences by $SEQ^{fin}(X)$. Finally, to denote the set of all infinite and all finite sequences over X, we use $SEQ(X)$, which can be defined formally as: $SEQ(X) = SEQ^{inf}(X) \cup SEQ^{fin}(X)$.

For convenience, we denote an empty sequence by $\langle\rangle$; that is, $\langle\rangle : \mathbb{N} \rightharpoonup X$ with $def(\langle\rangle) = \emptyset$ and $l = \langle x_0, ..., x_n\rangle$ to denote a finite sequence with $n+1$ elements or $l : \mathbb{N} \rightharpoonup X$ with $def(l) = [0,n]$ and $l(i) = x_i \, \forall \, i \in def(l)$.

We use the following function for appending to a finite sequence:

$$\begin{gathered} \circ : ((SEQ^{fin}(X) \times SEQ^{fin}(X)) \longrightarrow SEQ^{fin}(X)) \\ \cup((SEQ^{fin}(X) \times SEQ^{inf}(X)) \longrightarrow SEQ^{inf}(X)) \end{gathered}$$

$$(l_1 \circ l_2) : i \mapsto \begin{cases} l_1(i), \ \ if \ \ i < \#l_1 \\ l_2(i - \#l_1), \ \ if \ \ i \geq \#l_1 \ and \ i < \#l_1 + \#l_2 \\ undefined, \ if \ i \geq \#l_1 + \#l_2 \end{cases}$$

and lift $\circ$ in a pointwise manner to sets of sequences by:

$$(L_1 \circ, ..., \circ L_k) = \{l_1 \circ, ..., \circ l_k \mid l_1 \in L_1 \wedge \ l_k \in L_k\}$$

A function $emb : \mathbb{N} \rightharpoonup \mathbb{N}$ is an embedding of a sequence

$$l \in SEQ(X) \text{ into } l' \in SEQ(X'), \text{ where } X \subseteq X' \text{ iff}$$

- $def(emb) = def(l)$
- $\forall n \in def(emb) : l(n) = l'(emb(n))$, and
- $\forall n \in (def(emb \backslash \{0\}) : emb(n) > (n-1)$ hold.

A sequence $l \in SEQ(X_1 \cup X_2)$ is an interleaving of $l_1 \in SEQ(X_1)$ *and* $l_2 \in SEQ(X_2)$ iff there exists an embedding emb_1 of l_1 into l and an embedding emb_2 of l_2 into l such that $def(l) = img(emb_1) \cup img(emb_2)$ *and* $img(emb_1) \cap img(emb_2) = \emptyset$ holds.

The following function is used for interleaving two sequences:

$$\sim : ((SEQ(X) \times SEQ(X) \rightarrow P(SEQ(X))$$

$(l_1 \sim l_2) = \{l \in SEQ(X) \mid l$ is an interleaving of l_1 and $l_2\}$ and lift $\sim$ in a pointwise fashion to sets of sequences by

$$(L_1 \sim L_2) = \bigcup_{l_1 \in L_1, l_2 \in L_2} l_1 \sim l_2$$

Since both $\circ$ and $\sim$ are associative operators; therefore, lifting to n-ary operators is obvious. For lifted operators, we use prefix notation as shown in the following example involving interleaving three sets L_1, L_2, L_3:

$$\sim (L_1, L_2, L_3), \text{ which is equivalent to } L_1 \sim (L_2 \sim L_3).$$

The sequence semantics of attack trees in $\mathbb{T}$ can be given by using the following function.

Definition 6.3.1. The function $[\![\cdot]\!]_{SEQ} : \mathbb{T} \rightarrow P(SEQ^{fin}(\mathbb{A}) \backslash \{\langle\rangle\})$ is defined recursively by:

1. $[\![a]\!]_{SEQ} = \{\langle a \rangle\}$,
2. $[\![\mathbf{OR}(t_1, ..., t_k)]\!]_{SEQ} = \bigcup_{i \in \{1,...,k\}} [\![t_i]\!]_{SEQ}$
   $[\![\mathbf{AND}(t_1, ..., t_k)]\!]_{SEQ} = \sim ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ})$
   $[\![\mathbf{SAND}(t_1, ..., t_k)]\!]_{SEQ} = \circ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ})$

Example 4: This example illustrates the sequence semantics of the attack tree t presented in Figure 2.13.

$$[\![t]\!]_{SEQ} = \langle a \sim c \circ d, b \sim c \circ d \rangle$$

It is imperative to note that $[\![t]\!]_{SEQ}$ is guaranteed to be a non-empty set, and since each $t \in \mathbb{T}$ has at least one sub-expression of the form a; hence, each element in $[\![t]\!]_{SEQ}$ is also guaranteed to be a non-empty sequence.

6.4 Correctness of the Test-Case Derivation Approach

In order to demonstrate the correctness of the test case derivation approach, we provide a formal proof in this section. For a detailed description of the notation and the basis for this proof, please refer to Section 6.3.

Lemma 1. $\forall t \in \mathbb{T}_{OR,AND,SAND}$, GTC(t) = $[\![t]\!]_{SEQ}$

This will be proved by induction on the structure of the t.

Base Case:
Let $t = a$; since by definition
$[\![a]\!]_{SEQ} = \{\langle a\rangle\}$,
and $GTC(t) = TC = \{\langle a\rangle\}$, as shown by lines 3 to 5 of the pseudocode in the Algorithm 1, which show that TC = $[\![t]\!]_{SEQ}$.

Inductive Step:

Case $t =$ **OR**$(t_1, ..., t_k)$:
$GTC(t_1) = TC_1 \implies TC_1 = [\![t_1]\!]_{SEQ}$ (by induction hypothesis)
...
$GTC(t_k) = TC_k \implies TC_k = [\![t_k]\!]_{SEQ}$ (by induction hypothesis).
$[\![t]\!]_{SEQ} = \bigcup_{i=1,...,k}[\![t_i]\!]_{SEQ}$ (by definition), and
$TC \leftarrow \cup(TC_0, ..., TC_n)$, as shown by lines 6 to 11 of pseudocode in the Algorithm 1, which is equivalent to the following:

$$TC = \cup(TC_0, ..., TC_n) = \cup_{i\in 1,...,k}[\![t_i]\!]_{SEQ}$$
$$\implies TC = [\![t]\!]_{SEQ}$$

Case $t =$ **AND**$(t_1, ..., t_k)$:

$GTC(t_1) = TC_1 \implies TC_1 = [\![t_1]\!]_{SEQ}$ (by induction hypothesis)
...
$GTC(t_k) = TC_k \implies TC_k = [\![t_k]\!]_{SEQ}$ (by induction hypothesis).

$[\![t]\!]_{SEQ} = \sim ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ})$ (by definition), and

$TC \leftarrow \cup(TC_1\times, ..., \times TC_k)$ and $TC \leftarrow permutations(TC)$ as shown by the lines 12 to 18 of the pseudocode in the Algorithm 1. (That is, the interleaving of $TC_1, ..., TC_k$ has been accomplished by computing the Cartesian product of all TCs (i.e., from $TC_1\ to\ TC_k$) followed by generating their permutations). The end result of this is equivalent to the following:

$$
\begin{aligned}
&TC =\sim (TC_1, ..., TC_k) =\sim ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ}) \\
&\implies TC = [\![t]\!]_{SEQ}
\end{aligned}
$$

Case $t = \mathbf{SAND}(t_1, ..., t_k)$:

$GTC(t_1) = TC_1 \implies TC_1 = [\![t_1]\!]_{SEQ}$ (by induction hypothesis)
...
$GTC(t_k) = TC_k \implies TC_k = [\![t_k]\!]_{SEQ}$ (by induction hypothesis).

$[\![t]\!]_{SEQ} = \circ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ})$ (by definition), and
$TC \leftarrow \cup(TC_1 \times, ..., \times TC_k)$, as shown by the lines 19 to 24 of the pseudocode presented in the Algorithm 1, which is equivalent to the following:

$$
\begin{aligned}
&TC = \circ(TC_1, ..., TC_k) = \circ([\![t_1]\!]_{SEQ}, ..., [\![t_k]\!]_{SEQ}) \\
&\implies TC = [\![t]\!]_{SEQ}
\end{aligned}
$$

6.5 Test Case Generation in Action

In this section we generate some test cases to showcase the effectiveness of the test-case generation approach introduced in this chapter. Examples presented in this section demonstrate the application of the test case derivation approach. We use the attack trees constructed in the Section 5.2 of Chapter 5.

6.5.1 Anatomy of an Executable Security Test Case

Before proceeding with the examples of generating test cases, it is important to understand the structure of an executable security test case and what is the meaning of each constituent part of it. Recall, we explained earlier that a security test case is composed of a sequence of one or more events or actions that an attacker can potentially perform for achieving a certain goal by compromising the system security; in addition to all the components that a standard security test case has, an executable security test case contains an additional piece of information about its associated test script, hence it is called an executable test case. In other words, an executable security test case is a sequence of pairs of events or actions from $\mathbb{A}$ and their associated test scripts. In order to differentiate with a security test case, we use the notation "[" and "]" (i.e., square brackets), instead of "⟨" and "⟩" (i.e., angle brackets) to denote a sequence. A typical example of an executable security test case is presented in Listing 6.5.

The comma-separated pair of values enclosed in the double quotes (as shown in Listing 6.5) is an example of a complete executable security test case. While the first value (extracted from the leaf node's label) denotes the description of the event or

```
Test Case 1: [("event_Leaf_Node_Label","test_script_information")]
```

Listing 6.5: Example of an executable test case containing a pair of comma-separated values enclosed in double quotes.

attack action, the second value (comes from the comment element of the XML version of the attack tree, please see Table 6.1) represents the name of the test script that gets executed when the security test case runs. To prevent run-time programming errors, all the white spaces are replaced with underscore symbols by the software tool.

All the security test cases generated by our software tool have the same format and structure as presented in this section.

6.5.2 Example Security Test Cases

Table 6.2: *This table presents the derived test cases from the attack tree shown in Figure 5.5 (see Section 5.2).*

Threat Scenario: *Trick the ACC ECU into Triggering an Emergency Stop*
Source Attack Tree: *Figure 5.5* Attack Tree Type: ***OR*** Number of Leaf Nodes: *2* Number of Test Cases: *2*

Test Case 1:
`[("event_Inject_Messages_Via_TCU","inject_tcu_message")]`
Test Case 2:
`[("event_Inject_Messages_Via_OBD-II_Port","inject_obd_message")]`

Table 6.2 shows two different test cases derived by using the **OR** attack tree displayed in Figure 5.5 in Section 5.2 as an input to the software tool. This attack tree has one **OR** subtree containing two leaf nodes. Note that our tool correctly derived the test cases by analyzing the structure of the tree. Each test case, comprising one action, is derived from the leaf node of subtree. The tool extracts label (e.g., *Inject Messages Via TCU*) from of the leaf node and uses it for the test case description by preceding it with the word *event* to indicate it is an action step. Similarly, the tool also extracts the text (e.g., *inject_tcu_message*) from the *comment* element of the XML version of the attack tree, which is usually a method name in the test script. These values are specified while constructing the attack tree in the ADTool.

Table 6.3: *This table presents the derived test cases from the **SAND** attack tree shown in Figure 5.7 (see Section 5.2).*

Threat Scenario:
Flood Cause the Adaptive Cruise Control ECU to Stop or Crash

Source Attack Tree: *Figure 5.7*
Attack Tree Type: ***SAND***
Number of Leaf Nodes: *3*
Number of Test Cases: *2*

Test Case 1:
`[("event_Connect_through_OBD-II_Port","connect_via_obd"), ("event_Flood_ACC_ECU_with_Invalid_Data","flood_acc")]`
Test Case 2:
`[("event_Connect_through_TCU","connect_via_tcu"), ("event_Flood_ACC_ECU_with_Invalid_Data","flood_acc")]`

Two different test cases derived from the **SAND** attack tree shown in the Figure 5.7 of Section 5.2 are presented in the Table 6.3. This attack tree is made up of an **OR** (with two children leaf nodes) as well as one leaf node. The subtree represents the subgoal of connecting to the ACC ECU by using one of the two different possibilities. The overall goal of this attack tree can be achieved in two different ways; hence, two test cases are derived. Since this is a **SAND** tree, the specified order of the actions/operations must be followed. Note that unlike **OR** example described above, each test case contains more than one action. This example also demonstrates that the tool successfully generated valid and correct test cases in accordance with the semantics of the attack tree.

The third example of test case derivation is presented in the Table 6.4, which summarizes two different test cases derived from the **SAND** attack tree shown in the 5.6 of Section 5.2. This attack tree is composed of two **SAND** leaf nodes and an **OR** subtree representing the subgoal *Connect to the ACC ECU*. Each leaf node of this **OR** attack tree represents an alternative to gaining access to the ACC ECU either through OBD-II port or TCU. Each test case consists of two sequential actions.

Table 6.4: *This table presents the* ***SAND*** *attack tree shown in Figure 5.6 (see Section 5.2).*

Threat Scenario: *Reflash the Adaptive Cruise Control ECU From the CAN Bus in Order to Send Arbitrary CAN Messages*
Source Attack Tree: *Figure 5.6* Attack Tree Type: ***SAND*** Number of Leaf Nodes: *4* Number of Test Cases: *2*

Test Case 1:
`[("event_Connect_through_OBD-II_Port","connect_obd"), ("event_Download_the_Firmware_Image","download_firmware"), ("event_Perform_Reflashing","reflash_acc")]`
Test Case 2:
`[("event_Connect_through_TCU","connect_tcu"), ("event_Download_the_Firmware_Image","download_firmware"), ("event_Perform_Reflashing","reflash_acc")]`

6.6 Chapter summary

In this chapter we introduced our test case derivation software tool by presenting and walking through the algorithm followed by the proof of the approach's correctness using formal methods. We showed how our tool analyzed the structure of the attack tree based on its formal semantics for deriving valid test cases. Furthermore, we provided a definition and explanation of the security test cases, supported by some appropriate generic examples. Finally, three different examples were presented to showcase the core functionality of the software tool and underpinning test case derivation approach. Each example involves generating security test cases based on a particular attack tree provided to the tool as an input. A table summarizing each test case generated is also provided showing the core elements of test case.

Chapter 7

Experimental Security Analysis of Uptane Framework

In this chapter we apply the systematic approach to the security testing of the reference implementation of the Uptane Framework and detail all the core activities. Starting with describing some key assumptions, an overview of the testing setup is provided, which is followed by the detailed description of the in-depth security analysis of the Uptane Framework by applying the testing approach. We conclude the chapter by summarizing the key findings and limitations.

We present a number of different experiments performed on various components of Uptane Framework using our systematic security testing approach. It is demonstrated how applying *System Decomposition* in the *Information Gathering* was carried out for identifying the critical assets and preparing the suitable system model for *Threat Enumeration* in *Threat Assessment* phase of the approach. Using the *Threat List* generated, a number of attack trees were constructed using the attack tree construction approach. Our software tool generated and executed the executable test cases by using these attack trees. The chosen security tests presented are derived from the threat list generated by the Threat Modeling Tool, as shown in Table 7.2, by using the system model of Uptane Framework (as shown in Figure 7.3) as a basis for the test derivation and execution process. Note that the last three threats in the table are selected from the literature (for more details, see [38]). While all the tests conducted in these experiments were successfully executed with all the associated steps/actions performed as expected, 13 out of a total of 29 security test cases failed (i.e., yielded FAIL outcome). As discussed later, while the reference implementation was able to defend various cyberattacks, the outcome of some experiments suggest that effective security controls need to be applied to the production system in order to ensure the confidentiality of the information being exchanged between ECUs and the availability of Uptane repositories and backend-server infrastructure to ensure timely and uninterrupted delivery of updates for smooth and safe operations of the vehicle.

7.1 Assumptions

Before proceeding with security testing process, we would like to highlight some important assumptions that have been made while performing the security testing.

1. First of all, most of the experimental security attacks presented in the subsequent subsections assume that attackers either have privileged access (e.g., it could be a disgruntled employee, facilitating the attacker to gain access) to the Uptane repositories or the repositories are being spoofed. We will further explain it when we describe specific experiments in the corresponding sections.
2. Some experiments rely on the assumption that the secret cryptographic keys for signing software or metadata were compromised. This could be accomplished by social engineering techniques and/or bribing an employee, for example.
3. Finally, since our primary goal has been to demonstrate what the adversaries would be able to accomplish if they could compromise the update servers, rather than showing how would they break into the system, many experiments assume that the attackers were somehow able to break into the system. Furthermore, since the production environment would have its own specific dynamics (such as type of the hardware equipment, operating systems etc.), which would be depending on the very nature of OEM's (or the vendor's) IT infrastructure, it is therefore reasonably impracticable to demonstrate or simulate the attacks (without any knowledge of the hardware/software vulnerabilities) on such environments when the details of such actual target setups are unknown.

7.2 Experimental Setup

This section details the testing environment used for security evaluation of the automotive over-the-air updates system. The testbed has been constructed using readily available, inexpensive hardware components, providing a safe, adaptable, and portable testing environment for automotive security testing. Core components of the testbed are shown in the Figure 7.2, and a description of all its hardware/software components has been summarised in Table 7.1. We use Raspberry Pi microcontrollers for simulating the Primary and Secondary ECUs, representing Uptane clients. The laptop computer hosts the Uptane repositories of the reference implementation. A standard network switch has been used to facilitate connectivity between the server and client devices.

The laptop computer hosting the Uptane repositories has Ubuntu 18.04 installed. We use the standard Raspberry Pi Desktop version of the operating system on both Raspberry computers that simulate the Uptane-compliant Primary and Secondary clients. The reference implementation of the Uptane framework was downloaded and installed on the server and client components. The reference implementation has been

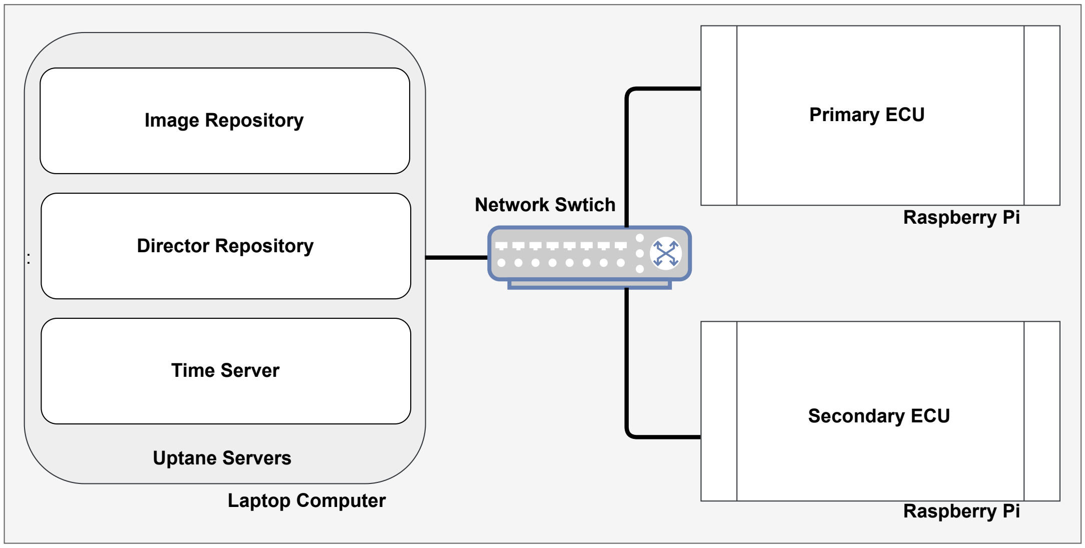


Figure 7.1: *OTA testbed schematic diagram, providing a graphical overview of the major components and communication links.*

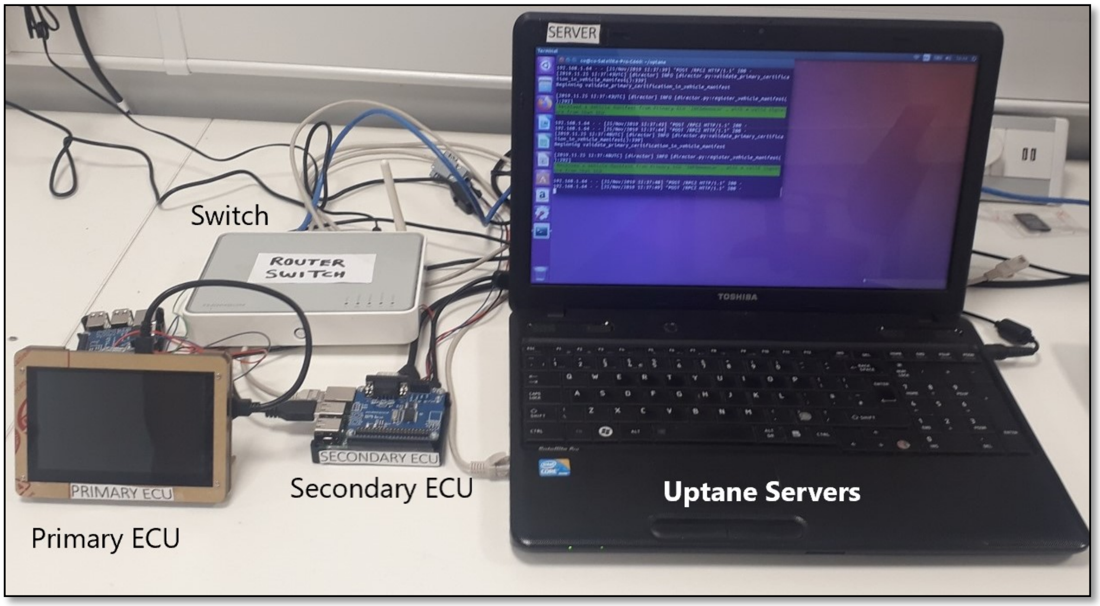


Figure 7.2: *The Testbed for OTA Updates Security Testing. Major components of the testing setup are shown in this image.*

Table 7.1: *A summary of the hardware and software components used for building the security testing environment shown in the Figure 7.2.*

Component	Machine Type	Operating System
Uptane Servers	Laptop Computer	Ubuntu 18.04
Uptane Primary ECU	Raspberry Pi Microcontroller	Raspberry Pi Desktop
Uptane Secondary ECU	Raspberry Pi Microcontroller	Raspberry Pi Desktop
Attacker Machine	Laptop Computer	Kali Linux
Communication Device	Switch	–

programmed in using Python programming language and can be downloaded from [123]. A guide providing detailed instructions on how to set up the servers and clients can also be found using the link provided above. However, remember that in order to set up the implementation on separate (distributed) physical devices, further configurations would need to be applied. Instructions for such configurations are provided in the appendix. Finally, even though over-the-air updates usually employ wireless communication technologies, this testbed rely on a wired switch, which is insignificant for the purposes of the experimentation, as using such a device does not affect the process or outcome of the security tests.

7.3 Uptane Framework: System Decomposition

Recall that in order to identify core system components and external interfaces that attackers can potentially target as entry points for compromising the security, information about the target system needs to be gathered. Ideally, both architectural and functional models of the system should be produced in order to determine the potential attack surface. Uptane Framework's reference implementation along with detailed design documentation is available on the internet, these resources provided useful starting point for our investigations. By reviewing the design documentation and other relevant information on the internet about the framework, we reproduced the architecture diagram of Uptane as shown in Figure 7.3. This detailed view was achieved by applying system decomposition approach outlined in the Section 4.2.1 of Chapter 4. This diagram shows key system components on both the backend OTA servers and within the vehicle along with associated data/communication flows.

Having identified the major entities of the OTA update system, the next step involved identifying and understanding the interactions between these entities. Based

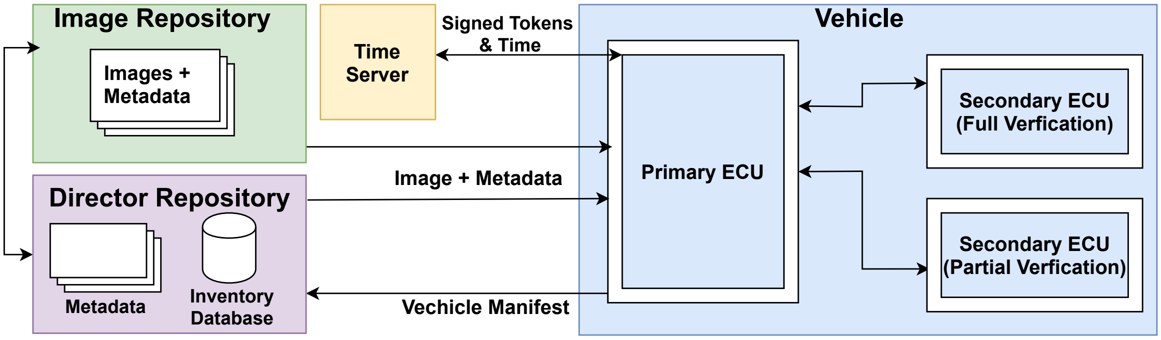


Figure 7.3: *Produced as a result of the System Decomposition process, this diagram provides a detailed architectural view of the Uptane Framework depicting the repositories, clients and the information flows.*

on the analysis of the information we gathered, a UML sequence diagram (as shown in Figure 7.4) was produced, representing various interactions between server-side and in-vehicle components. As evident, this diagram presents more meaningful insights about the key processes and the information exchange taking place between them. With this diagram, we proceed to the next activity: *Threat Enumeration*.

7.4 Uptane Framework: Threat Enumeration

While the sequence diagram in Figure 7.4 captures key system processes and interactions, it does not provide any information about the security threats. Therefore, with the help of the two diagrams (i.e., Figure 7.3 and Figure 7.4), we were able to construct the DFD, as shown in Figure 7.5.

Note that the data flow diagram (Figure 7.5 includes the major components of the Uptane OTA update system from both the server and the vehicle side. As can be observed from the DFD in Figure 7.5, both the server-side and the client-side components are surrounded by rectangles, signifying the *trust boundaries*. Three different types of data flows have been used to indicate the communication types used between different components. Communication between update server and Telematics Control Unit uses HTTP protocol, while CAN protocol has been used between in-vehicle components. The third type of data flow is command, depicting internal communication flows between server-side components (i.e., Image Repository, Director Repository, Inventory Database etc.), assuming they reside on the same physical system.

Based on this data flow diagram, the report generated by the TMT tool identifies 53 different threats (a summary is presented in the Table 7.2 and detailed threat report can be found in Appendix C) that could potentially be used to compromise the security of OTA update system in a number of different ways. As indicated earlier, the last three threats have been included from the literature to show some known attacks on the update systems. In addition to the title of the threat, Table 7.2 also shows relevant

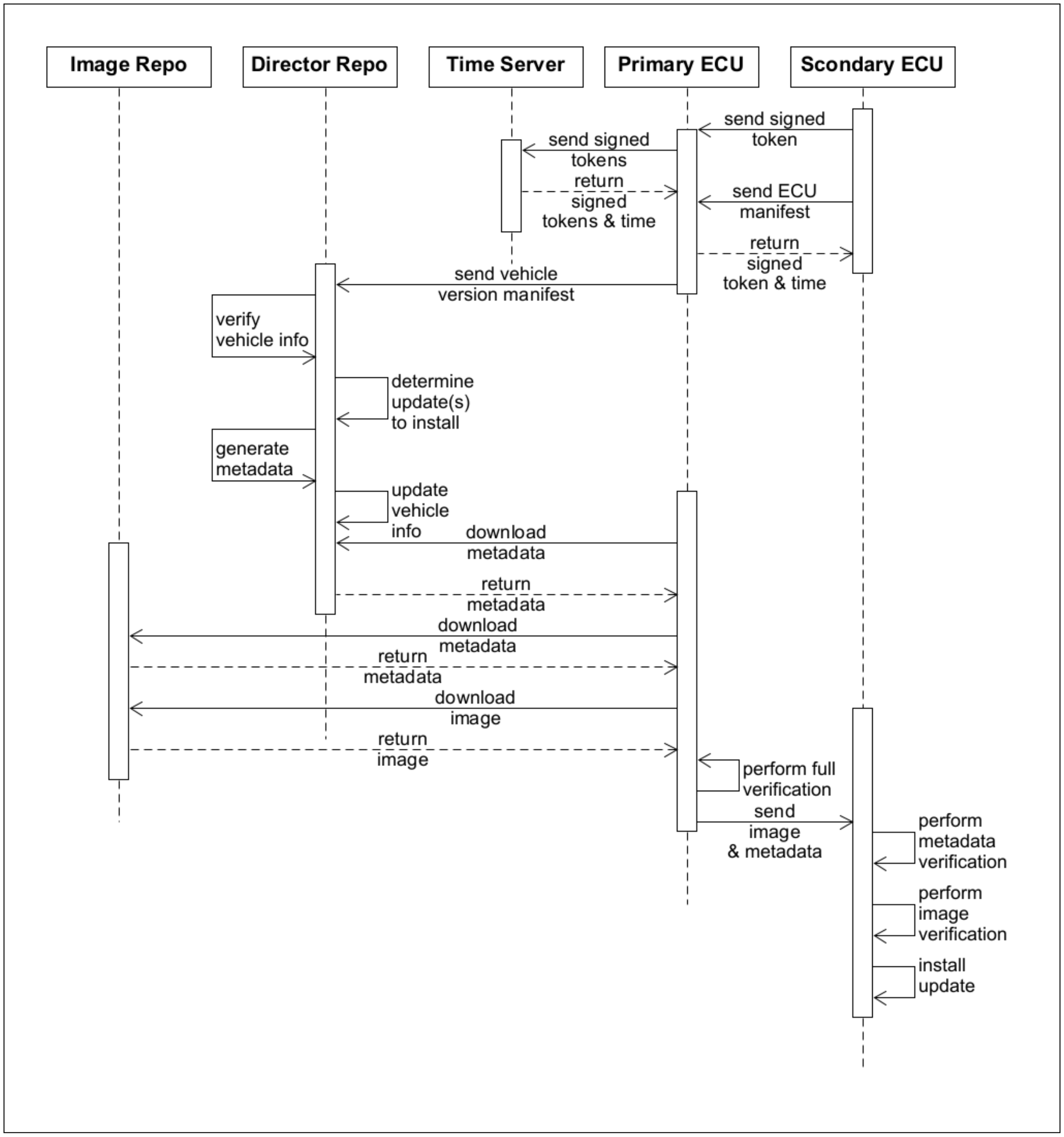


Figure 7.4: *Uptane Framework Sequence Diagram depicting Over-The-Air Update interactions between Uptane OTA Server-Side and In-Vehicle Primary and Secondary ECUs.*

STRIDE category and a count of the occurrences of each threat. For example, the first threat Data Flow Sniffing has a count value of nine, which means there are nine different data flows that are exposed sniffing attacks. Similarly, TMT identified at least seven different vulnerable points that can be used for unauthorised download of

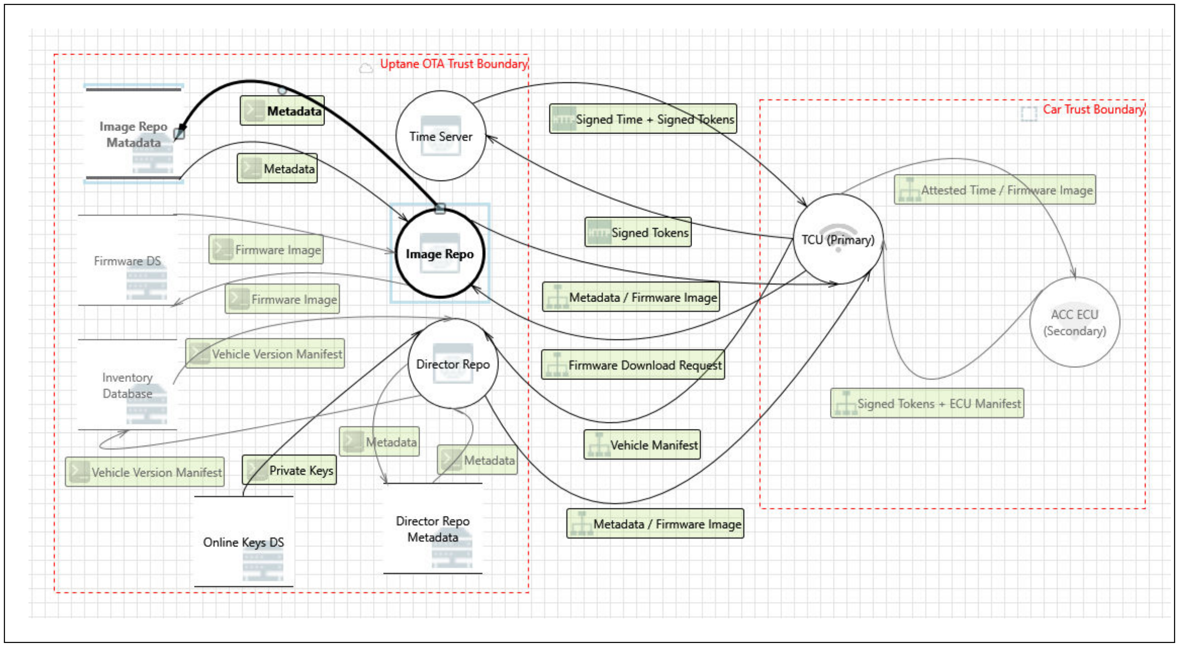


Figure 7.5: *Data Flow Diagram (created using Threat Modeling Tool and a template from NCC group) of the Uptane Framework Backend Servers (reference implementation) and in-vehicle components, showing various types of components and communication links.*

updates. The pie chart in Figure 7.6 provides an overview of the threats identified, showing the total number of threats in each category of STRIDE.

7.5 Uptane Framework: Attack-Tree Construction

We only include a subset of the threats in the experiments, as this strategy allows us to effectively demonstrate the application of systematic security testing approach with appropriate level of details. We first build attack trees for various selected threats enumerated by TMT (see Table 7.2), followed by some known threats as published in the literature (see Figure 2.1 and Table 2.1). Recall from the Section 5, the step-by-step approach for constructing the attack trees that includes identifying a goal that an attacker would like to achieve by compromising the system security, followed by identifying one or more ways that can assist in achieving the goal. That high-level goal becomes the root node of the attack tree. Also recall that each of the threats listed in Table 7.2 represents a potential high-level goal (root node). The threat report generated by TMT provides suggestions about what attack methods/techniques could be used by the attackers for each type of threat. We also used brainstorming and expert discussion sessions for identifying subgoals (intermediate nodes) and action steps (leaf nodes). Where necessary and applicable, we also made use of available online resources

Table 7.2: *This table presents a summary of the threats identified by the tool.*

Category	#	Threat	Count
Tampering	1	Modify Data Being Sent to the TCU (Primary) While in Transit	3
Repudiation	2	Director Repo Denies Writing Data	1
	3	Image Repo Denies Writing Data	1
	4	TCU (Primary) Denies Writing Data	3
	5	Time Server Denies Writing Data	1
Information Disclosure	6	* Updates Could be Downloaded	7
	7	* Data Flow Sniffing	9
	8	Car Could be Tracked	3
Denial of Service	9	* Cause the Director Repo to Crash or Stop Remotely	1
	10	* Cause Image Repo to Crash or Stop Remotely	1
	11	* Cause the TCU (Primary) to Crash or Stop Remotely	1
	12	* Cause the Time Server to Crash or Stop Remotely	1
	13	Take the Director Repo Offline	1
	14	Take the Image Repo Offline	1
	15	Take the TCU (Primary) Offline	3
	16	Take the Time Server Offline	1
	17	Flood Director Repo with Invalid Data	1
	18	Flood Image Repo with Invalid Data	1
	19	Flood TCU (Primary) with Invalid Data	3
	20	Flood Time Server with Invalid Data	1
Elevation of Privilege	21	** Compromise Director Repo in order to Send Malicious Updates	1
	22	** Compromise Image Repo in order to Send Malicious Updates	1
	23	Compromise TCU (Primary) in order to Send Malicious Updates	3
	24	Compromise Time Server in order to send Malicious Updates	1
	25	Reflash TCU (Primary) in order to Send Arbitrary CAN Messages	3
Other Known Threats	26	** Endless Data Attack	N/A
	27	* Rollback Attack	N/A
	28	* Mix and Match Attack	N/A

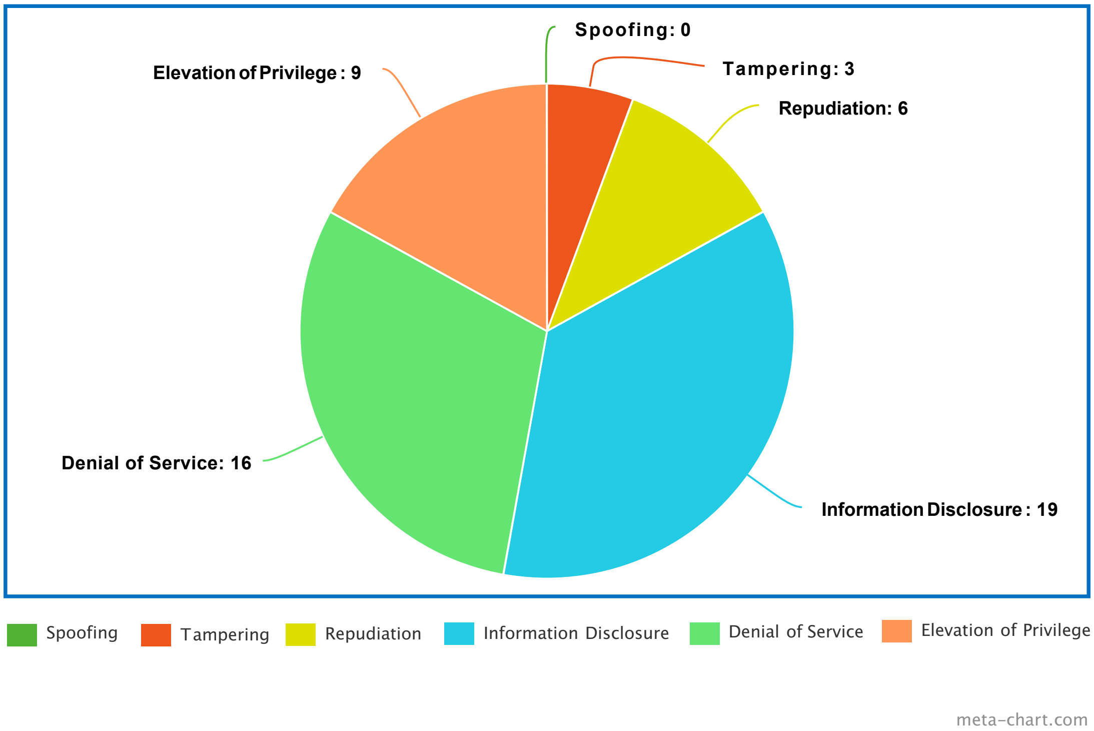


Figure 7.6: *Graphical overview of the number of identified threats to OTA update system in each category of STRIDE threat classification model.*

for the identification of relevant attack techniques used by malicious entities. For constructing each attack tree, we began by choosing each selected threat scenario as the root node of the attack tree. Following the next steps described in the attack tree construction process introduced in Chapter 5, we refined and populated each attack tree by taking into consideration the attack method suggested by TMT followed by brainstorming sessions to determine what could be the possible steps/actions that could potentially be carried out by cybercriminals. Attack trees we constructed following the steps outlined above are presented in the subsequent sections (see figures 7.7 to 7.21), each in a separate section. Once these attack trees for the selected threats had been constructed, they were combined to construct an overall attack tree for the reference implementation of the Uptane Framework as displayed in Figure 7.22.

As explained earlier, test case generation and execution are automated processes, carried out by our bespoke software tool. Attack trees are an integral and crucial part of our automated test case generation and execution process. By analyzing the structure of each of the attack trees, the tool successfully derived security test cases, generated and executed test scripts against the reference implementation, in a step-by-step manner. The execution of all the test scripts was carried out using our testbed as detailed above. Result of each experimental attack performed along with the test case details are given in a separate section of this chapter. In addition to the test scripts, each test result outlines the outcome of the test (i.e., the impact of the attack on the system) and resultant system behaviour.

In what follows, we explain how each of the attack trees was constructed.

As mentioned earlier, each attack tree has exactly one root node, which represents the main goal of the adversary. Various potential goals have been identified, as listed in the Table 7.2. One instance of each of the threats becomes the root node (or main goal) in the attack tree. Now the primary goal of the attacker has been established, different methods/techniques that can potentially be used to achieve this are to be identified. The original report generated by TMT mentions at least one way to realize the threat. In most cases, those suggestions are relevant and serve as a good starting point for populating the attack tree with appropriate subgoals (child nodes) and/or actions (leaf nodes). Brainstorming sessions also supported the identification of different attack steps/actions to realize the threats.

Threat 6 - Updates Could Be Downloaded

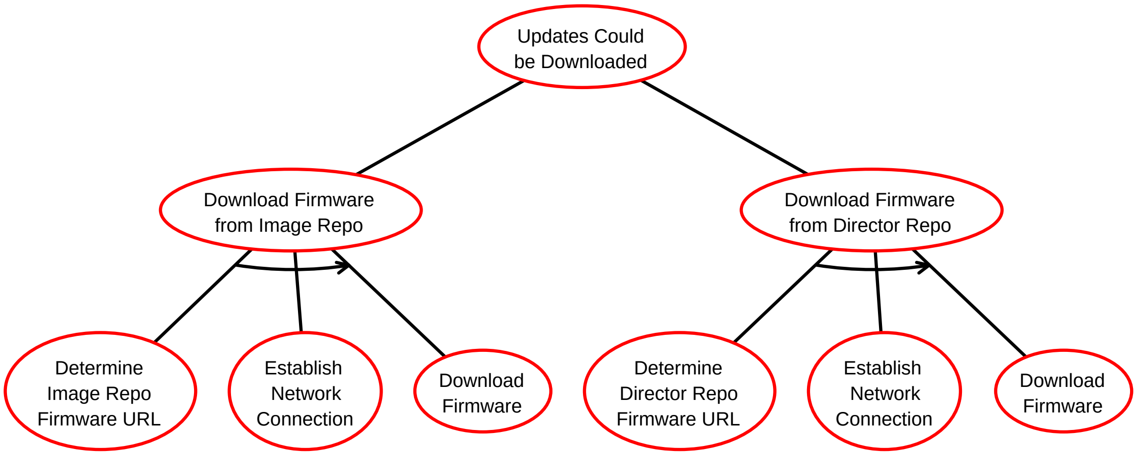

Updates Could be Downloaded
Download Firmware from Image Repo
Download Firmware from Director Repo
Determine Image Repo Firmware URL
Establish Network Connection
Download Firmware
Determine Director Repo Firmware URL
Establish Network Connection
Download Firmware

Figure 7.7: *Attack Tree - Updates Could be Downloaded: This attack tree represents Threat 6 enlisted in Table 7.2 that involves downloading firmware images from Uptane repositories without any authentication.*

The main objective of this threat is to directly download firmware images from Uptane repositories in order to steal sensitive/confidential and proprietary information (this threat belongs to the category *Read Updates*, as presented in Figure 2.1), which can later be used to craft and launch destructive attacks against connected cars by comparing the different versions of the firmware image with each other. As shown in the Figure 7.7, the root node of the attack tree has been labeled using the title of the threat 6 in the Table 7.2 (i.e., *Updates Could be Downloaded)* representing the attacker's main goal. With discussions and brainstorming, we identified at least two different ways to achieve the main goal, each of which turned into an intermediate node with its own subtree, showing two different ways (subgoals) for downloading the images from the Uptane servers: *Download Firmware from Image Repo* and *Download*

Firmware from Director Repo. However, the two steps for downloading the firmware image from either of the repositories are identical. Note that this overall OR attack tree has two SAND (recall that a sequential AND or a SAND tree is the one each action of which must be carried out in the specified order) subtrees, each depicting a possible way used by the intruder for downloading the image from the servers. Each of the subtrees has three leaf nodes representing the atomic actions of the attacker. The first action step of the subtree on the left involves determining the URL of the firmware on the Image Repo, followed by establishing a network connection with the server in the second action step. The download of the target firmware image is initiated once

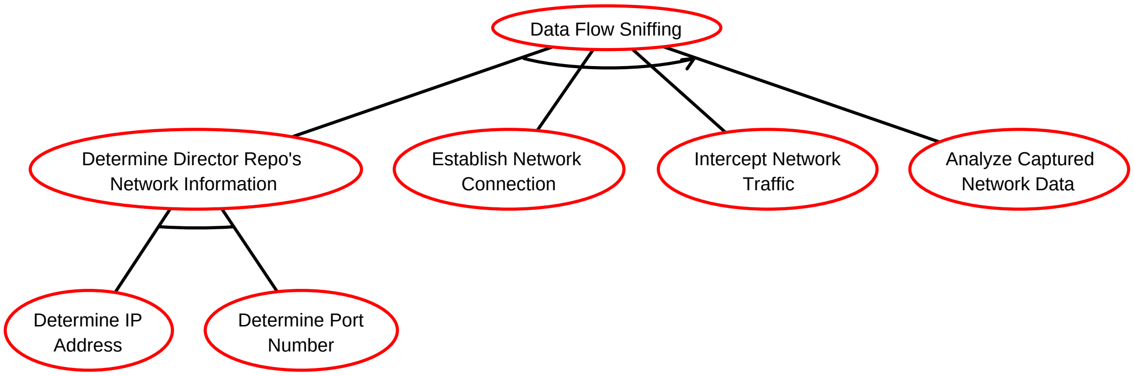

Data Flow Sniffing
Determine Director Repo's Network Information
Establish Network Connection
Intercept Network Traffic
Analyze Captured Network Data
Determine IP Address
Determine Port Number

Figure 7.8: *Attack Tree - Data Flow Sniffing: This attack tree represents Threat 7 as enlisted in Table 7.2 that involves monitoring and capturing information exchange between Uptane servers and clients.*

After assigning the title of the threat 7 to the root node of the tree, it was straightforward to identify the specific actions for materializing the threat. The **AND** subtree for determining the network information of the Director Repo has two steps: one for determining the IP address and the other for port number. Since both actions can be executed in any order; hence, **AND** conjunction was chosen as the refinement operator. In contrast, all other leaf nodes belonging to the root node must be carried out in the order shown, otherwise attack would not succeed or would not produce the desired result. This particular threat aims at intercepting network communication between Uptane repositories and the client ECUs. As can be seen in the Table 7.2, there are total nine occurrences of this threat, each corresponding to one of the data flows in the DFD in Figure 7.5. This indicates that the TMT identifies all these data flows vulnerable to this threat. We include only one instance of this threat in the testing, since the process and the end results will be identical - providing no further insights.

The **SAND** attack tree in Figure 7.8 provides a graphical overview of the threat and associated actions. Notice that *Data Flow Sniffing* represents the main goal of the attack tree. We identified one **OR** subtree and three different actions (leaf nodes) for accomplishing the main goal: the subtree has two leaf nodes which represent the actions *Determine IP Address, and Determine Port Number.* The other three leaf nodes represent the *Establish Network Connection, Intercept Network Traffic* followed by *Analyze Captured Traffic.* The first and the last actions are performed manually while other two are carried out by the tool. Note that this threat corresponds to the Eavesdrop Attack under the Read Updates category in Figure 2.1. This and the preceding threat both have the same ultimate goal of reading/downloading restricted/confidential content for malicious purposes. As mentioned earlier, such content can be used for creating more powerful and sophisticated attacks.

Threat 9 - Cause the Director Repository to Crash or Stop Remotely

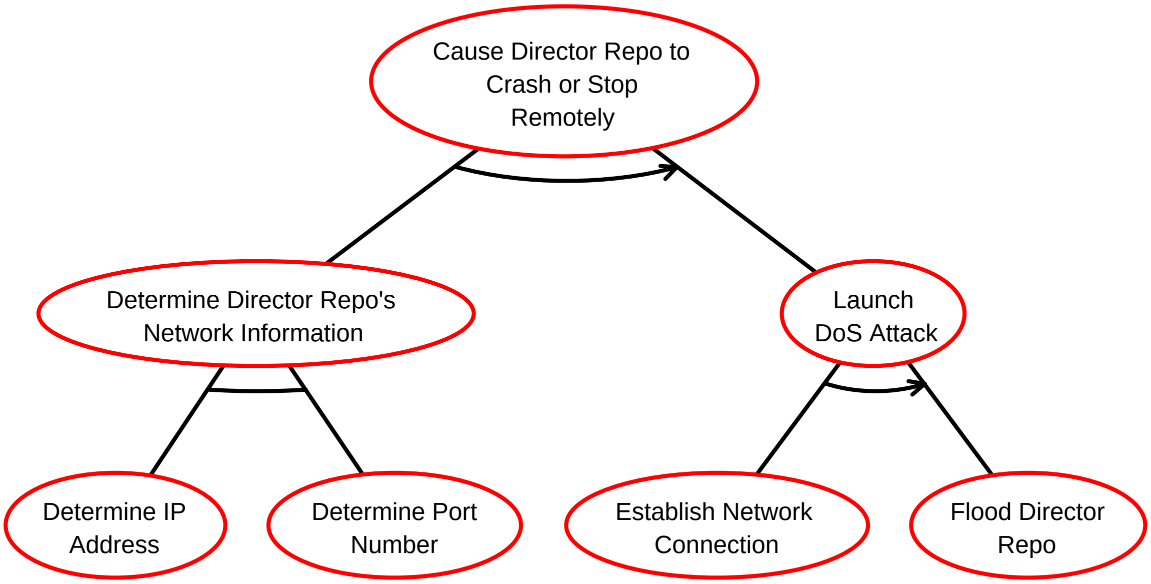

Cause Director Repo to Crash or Stop Remotely
Determine Director Repo's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Director Repo

Figure 7.9: *Attack Tree - Cause Director Repo to Crash or Stop Remotely: This attack tree represents the Threat 9 as enlisted in Table 7.2 that involves blocking the delivery of updates to the ECUs.*

There are numerous ways that adversaries can potentially employ to adversely influence the delivery and/or installation of crucial software updates. For instance, attackers can cause disruptions to the delivery of important firmware/software updates by mounting denial-of-service attacks on update servers. As shown in the Figure 2.1, denying or blocking updates is one of the strategies that hackers can adopt for stopping the removal or correction of software bugs or security loopholes in the software/firmware currently installed.

The attack tree depicted in the Figure 7.9, represents Threat 9 that involves performing a DoS attack against the Director Repository, by overwhelming it with a huge number of communication requests, causing it to stop responding to legitimate requests from clients. The main goal of the attack is to block update delivery to the ECUs in the vehicle. The **SAND** root node has one **AND** subtree and another **SAND** subtree. As explained earlier, determining network information steps do not require any particular order; therefore, an **AND** refinement makes sense here. Conversely, for the other two steps, strict order is essential thus **SAND** operator has been used.

In order to begin performing the attack, essential information regarding the target (that is, OTA update servers) is required. Hence, the first step (leaf node) involves determining the IP address and port number of the Director Repository. As this experimentation relies on the reference implementation of the Uptane Framework, we obtained this information by running the reference implementation in our local setup. In a real-world scenario, this information may need to be obtained by performing network/port scanning on the server or extracting it from an ECU (e.g., TCU) in the vehicle.

The second step, as can be seen from the attack tree in Figure 7.9, involves establishing a network connection with the Director Repository. The main purpose of this step is to determine whether the target system is available on the network and that there are no connectivity issues before executing the DoS attack. Finally the attack is launched, which sends a large number of HTTP requests to the Director Repository at port 30401.

Threat 10 - Cause Image Repository to Crash or Stop Remotely

The attack tree shown in Figure 7.10 represents the threat *Cause the Image Repo to Crash or Stop Remotely* (by mounting a DoS attack) so that it can not respond to legitimate update requests from ECUs. The root node of this **SAND** tree represents the main goal (i.e., Cause Image Repo to Crash or Stop Remotely, drawn from the threat 10 from the threat list). It is composed of two **AND** subtrees representing two subgoals that were identified easily for this and the two subsequent attack trees as they essentially accomplish the same task of bombarding the target device with a large number of invalid or malformed packets. The first subtree is concerned with determining the IP address and relevant port number. This is followed by second subtree that involves establishing the network connection and flooding the repository by sending a large number of requests to cause it to crash or become unresponsive to all client requests. Mounting a DoS attack on these server components is relatively easy, as they expose publicly accessible APIs or interfaces accepting communication requests.

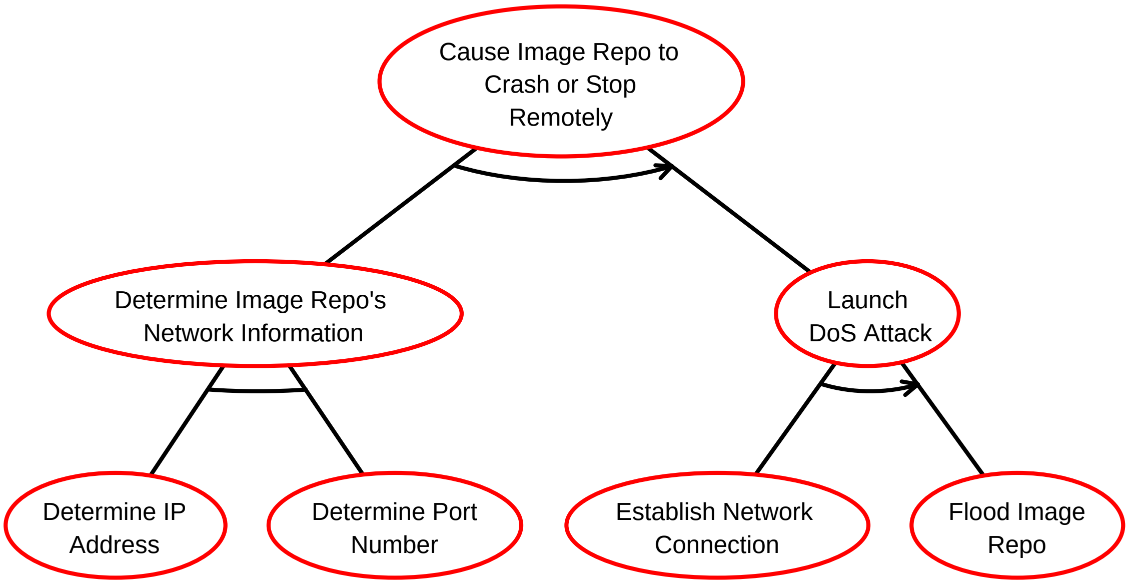

Cause Image Repo to Crash or Stop Remotely
Determine Image Repo's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Image Repo

Figure 7.10: *Attack Tree - Cause Image Repository to Crash or Stop Remotely: This attack tree represents Threat 10 enlisted in Table 7.2 that involves blocking the delivery of updates to the ECUs.*

Threat 11 - Cause the Primary ECU to Crash or Stop Remotely

As its name suggests, the main goal of this threat is to disrupt the normal operation of the Primary ECU, causing it to crash or stop working; hence, blocking the delivery of critical updates to secondary ECUs. Figure 7.11, shows the **SAND** attack tree depicting the steps for interrupting the normal operation of the Primary ECU. The root node represents the threat scenario from the Table 7.2, two **AND** subtrees were identified with each having two leaf nodes by using the attack tree construction approach. It is worth noting that this attack is initiated from a remote machine, not locally from within the host vehicle. As was the case with the preceding attacks, determining IP address/port is the precondition in order to successfully perform subsequent steps. Step two establishes whether the ECU has the internet connectivity. Lastly, the ECU is flooded with a huge number of requests, which leads to the failure of the ECU to respond to requests from clients.

Threat 12 - Cause the Time Server to Crash or Stop Remotely

This threat targets the Time Server by sending it a large number of requests causing it to exhaust and stop working properly. Unavailability or malfunctioning of this component can leave the Uptane clients vulnerable to time-specific attacks (e.g., Freeze Attack, as depicted in 2.1). The root node of the **SAND** attack tree in Figure 7.12 represents overall goal (i.e., the threat *Cause the Time Server to Crash or Stop Remotely*)

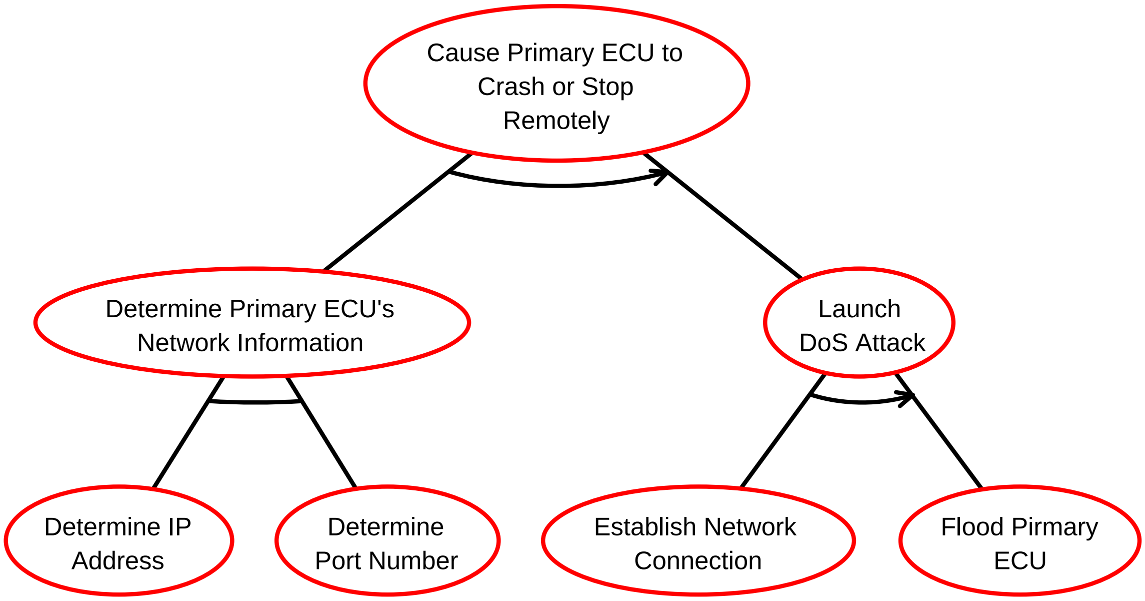

Cause Primary ECU to Crash or Stop Remotely
Determine Primary ECU's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Pirmary ECU

Figure 7.11: *Attack Tree - Cause the Primary ECU to Crash or Stop Remotely: This attack tree represents Threat 11 that involves causing the Primary ECU to crash; thus, resulting in the failure of normal update operations.*

and two **AND** subtrees that assist achieving the main goal, consisting of the action steps for launching a denial-of-service attack against the Time Server. These subtrees and corresponding leaf nodes were identified by applying the step-by-step attack tree construction approach. Since IP address along with the port number are prerequisites to the DoS attack; hence, the first subtree accomplishes this followed by the second subtree - launching the actual DoS attack against the Time Server, causing it to fail serving all legitimate requests from the clients.

Threat 21.1 - Compromise Director Repository in order to Deliver Malicious Updates (without compromised keys)

In this section, we provide the details of the threat that involves compromising the Director Repository for sending updates containing malicious contents. For the purpose of this experiment, it has been assumed that either the intruder has privileged access to Uptane servers, which allows them to read, write, and execute all types of programs on this repository, or alternatively they might be using a dedicated machine impersonating the servers.

However, in both cases they are assumed not to have access to the private keys for signing updates/metadata. With these capabilities, they attempt to send malicious updates to the clients by manipulating the firmware images on the repository. Figure

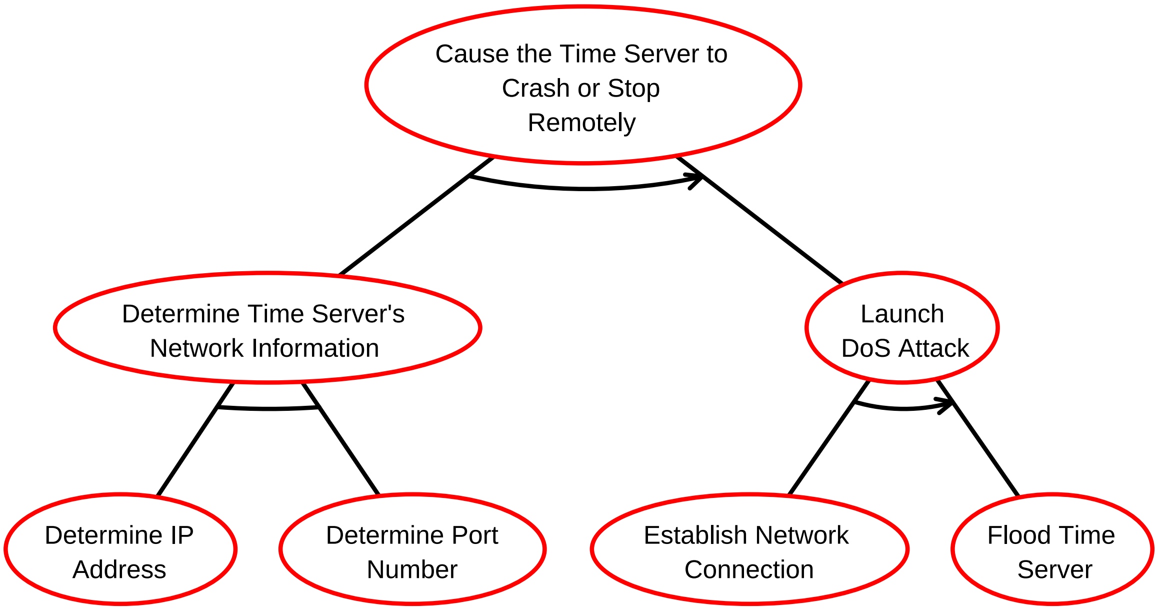

Cause the Time Server to Crash or Stop Remotely
Determine Time Server's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Time Server

Figure 7.12: *Attack Tree - Cause the Time Server to Crash or Stop Remotely: This attack tree represents Threat 12 as enlisted in Table 7.2 that involves causing the Time Server to crash; thus, affecting the distribution of timely updates to clients.*

7.13 shows the SAND attack tree with two action steps for sending malicious updates. Firstly, a valid firmware on the Image Repo is modified to add malicious data to it. Once the firmware has been modified, the firmware is added to the Director Repository by using a builtin feature of the repository. All these steps require some understanding and knowledge of the specific procedures used at the Uptane backend repository servers. This relatively simple attack tree, with only two leaf nodes, was not difficult to construct, as it involves adding some malicious contents to an existing firmware image and then adding the image to the Director repo. The reference implementation provides various methods one of which is named *add_target_to_director* that is used to add a firmware image file to the Director Repository.

Threat 21.2 - Compromise Director Repository in order to Send Malicious Updates (with compromised keys)

This is an advanced variant of the preceding attack wherein the attackers are assumed to have access to the private Directory keys for signing the metadata. Therefore, in addition to the capabilities outlined above, they can sign updates/metadata. As can be noticed from the attack tree in Figure 7.14, there is an additional step (i.e., the leaf node Generate Signed Metadata) for generating signed metadata. Therefore, in addition to adding the modified image to the repository, valid metadata is also generated. This

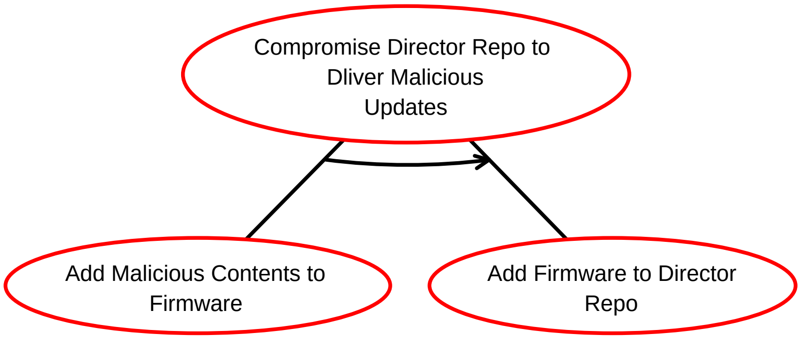

Compromise Director Repo to Dliver Malicious Updates
Add Malicious Contents to Firmware
Add Firmware to Director Repo

Figure 7.13: *Attack Tree - Compromise Director Repo in Order to Deliver Malicious Updates (without compromised keys): This attack tree represents Threat 21.1 as enlisted in Table 7.2 that involves delivering a malicious update to the target ECU.*

is accomplished by calling the method *write_to_live*. This signed metadata causes the client ECUs to believe the firmware image is trustworthy. However, since the ECU verification process verifies the metadata from both of the repositories, the update shall be rejected by the Primary due to inconsistent metadata values.

Threat 21.3 - Compromise Image and Director Repositories in order to Send Malicious Updates

The SAND attack tree shown in Figure 7.15 depicts the steps for launching the most damaging attack against the Uptane repositories, as attackers can install malware on the ECU to control it in order to take over the vehicle control. However, such an attack is highly difficult to perform, as it requires access to the private keys for Image and Director repositories, as well as administrative access to the server machines. Since Uptane standard expects the private keys of Image Repository to be stored securely offline in order to prevent key compromises, making it highly difficult for attackers to compromise the keys. Furthermore, Uptane uses key thresholds for providing protection against key compromises. Multiple keys are used for signing the metadata, which means malicious actors will have to compromise a set of keys rather than relying on a single compromised key to be able to sign the metadata.

As can be observed, the exact ordering for executing all the actions is crucial for this attack to succeed. Therefore, **SAND** trees are the natural choice in this scenario. Similar to the Director Repository, the Image Repository also has methods for adding the firmware to the repository (*add_target_to_imagerepo*) and signing the metadata (*write_to_live*). Threat 21 from the Table 7.2 becomes the root node, and as a first step, malicious contents to the image are added, which is followed by step two adding the malicious firmware to the Image Repository. Signed metadata is generated in step three on the Image repository. On the Director Repository, two remaining steps are performed: adding the malicious firmware to the Director Repository and generating

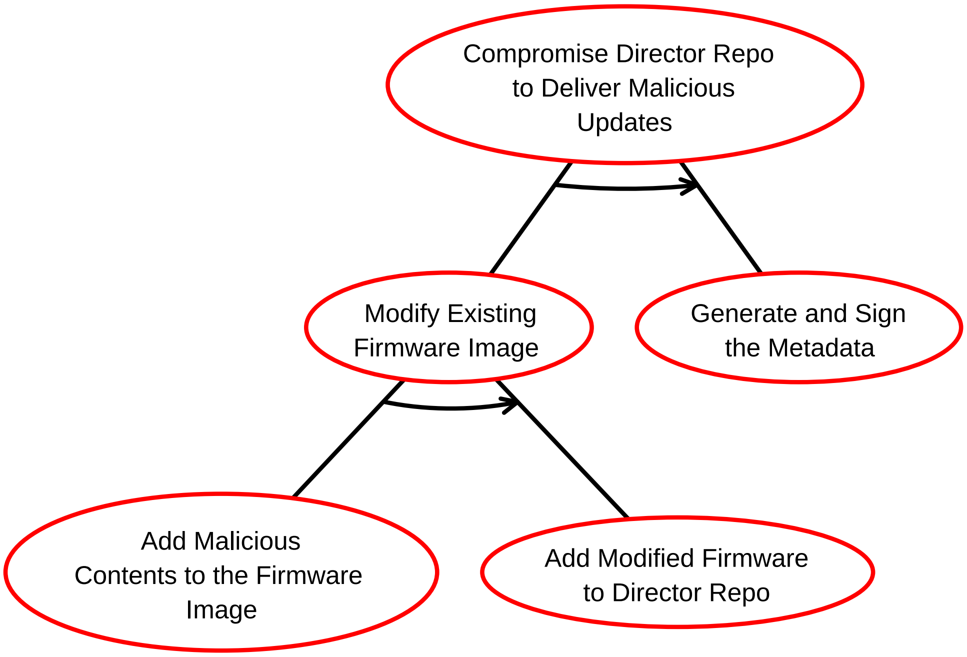

Compromise Director Repo to Deliver Malicious Updates
Modify Existing Firmware Image
Generate and Sign the Metadata
Add Malicious Contents to the Firmware Image
Add Modified Firmware to Director Repo

Figure 7.14: *Attack Tree - Compromise Director Repo in Order to Deliver Malicious Updates (with Director Repository Keys Compromised): This attack tree represents Threat 21.2 that involves delivering a malicious update to the target.*

signed metadata.

Threat 22.1 - Compromise Image Repository in order to Send Malicious Updates (without compromised keys)

In this scenario, the attackers are assumed to have access to perform certain types of operations on the Image Repository, which include making changes to firmware images stored on Image Repository and adding the modified image to the repository. Private offline keys for signing updates/metadata are, however, not available. The attacker adds malicious contents to a newly added (but not yet distributed) firmware image intended to be installed on the ECU. The target firmware image to be compromised (named *firmware.img*) is stored on both repositories.

Remember that this attack also assumes that the Director Repository has not been compromised yet. While, the adversary could add malicious contents to the firmware image and add it to the Image Repository, the attack would not succeed, as Primary ECU would detect the inconsistency in the accompanied metadata from Uptane repositories; therefore, the update would be rejected. The SAND attack tree in Figure 7.16 shows the steps for materializing this attack. The root node represents the main goal (i.e., Compromise Image Repo in order to Send Malicious Updates). First action step involves adding malicious contents to the firmware image. The firmware is then added

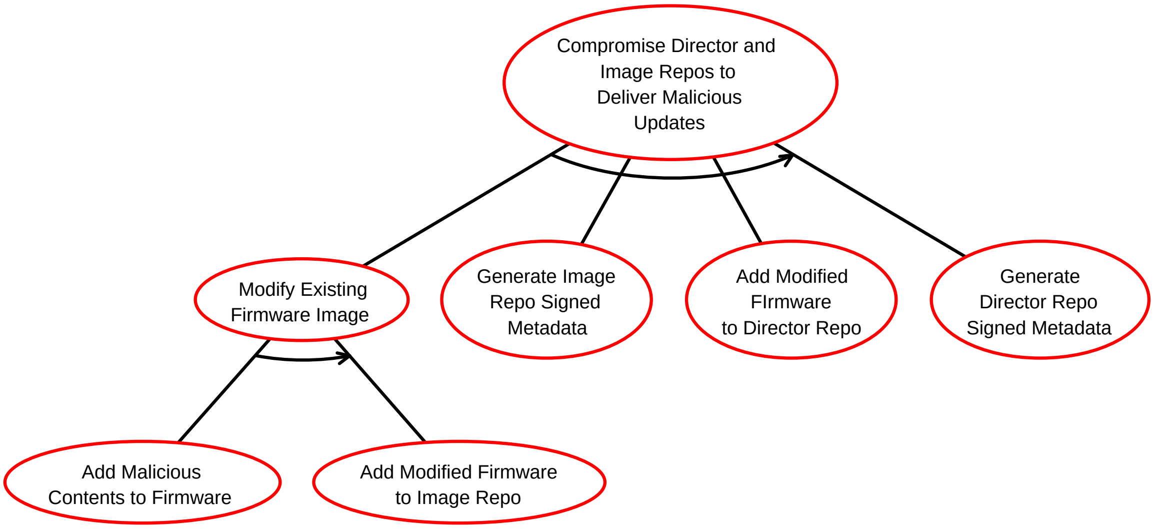

Compromise Director and Image Repos to Deliver Malicious Updates
Modify Existing Firmware Image
Generate Image Repo Signed Metadata
Add Modified FIrmware to Director Repo
Generate Director Repo Signed Metadata
Add Malicious Contents to Firmware
Add Modified Firmware to Image Repo

Figure 7.15: *Attack Tree - Compromise Image and Director Repositories: This attack tree represents Threat 21.3 as enlisted in Table 7.2 that involves delivering a malicious update to the ECU.*

to Image Repo.

Threat 22.2 - Compromise Image Repository in order to Send Malicious Updates (with compromised keys)

This attack is a variant of the attack presented in the preceding experiment. The intruder does possess the private keys (with the required threshold) in addition to the administrative access to the system. The SAND attack tree in Figure 7.17 shows the root node representing the main goal (i.e., Compromise Image Repository in order to Send Malicious Updates), with other steps (represented by leaf nodes): Add Firmware to Image Repo and Generate Signed Metadata. It is important to remember that this attack is performed independently, with the assumption that the same image has not been compromised on the Director Repository.

Threat 26.1 and Threat 26.2 - Endless Data Attack

Endless Data attack - represented by two different attack trees shown in Figures 7.18 and 7.19 each depicting a variant of the attack, which aim at delivering an update with endless stream of data, causing the ECU's storage to exhaust. There are two variants of the attack wherein the additional data is appended to the update file in the first case, and inserted in the other. This has been done to observe the behaviour of reference implementation, as according to the documentation, the Primary ECU will only see and download the amount of update data as specified in the metadata file. Appending additional data to the file does not alter the original contents of the update

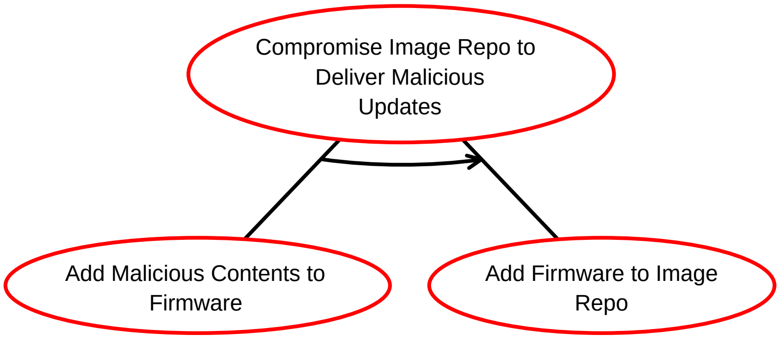

Compromise Image Repo to Deliver Malicious Updates
Add Malicious Contents to Firmware
Add Firmware to Image Repo

Figure 7.16: *Attack Tree - Compromise Image Repository in Order to Deliver Malicious Updates (without Repository Keys Compromised): This attack tree represents Threat 22.1 enlisted in Table 7.2 that involves delivering a malicious update to the ECU.*

file. On the other hand, when we insert the additional contents to the file, the original file contents might be modified. This will allow us to see how these two different scenarios are handled by the Uptane Framework. First SAND attack tree in Figure 7.18 is comprised of the following steps (represented as leaf nodes): *Append Malicious Contents to Firmware, Add Firmware to Image Repo*, and *Generate Signed Metadata.* Action steps depicted by second tree in Figure 7.19 are identical, except for the first one, which is *Insert Additional Contents in Firmware on Image Repo*. Please note that this and the following two threats, as mentioned earlier, have been included from the literature to show specific types of attack methods/techniques used by attackers to undermine the security of update systems. **SAND** tree has one direct leaf node and an **AND** subtree with two leaf nodes. It can be noted that after appending additional data to the file, the file is added to both of the repositories. While ordering of the parent tree (root node) must be ensured, two atomic actions of the **AND** can be executed in any order.

Threat 27 - Rollback Attack

This section presents the Rollback attack which involves delivering a valid but old version of the update, causing the ECU to uninstall the latest version. As can be noticed in the attack tree shown in Figure 7.20, step one and step three include deleting Timestamp metadata on both repositories. Steps two and four place an old version of the same update on repositories. This attack intends to trick the target ECU to install an old but valid firmware image, containing known vulnerabilities.

Threat 28 - Mix and Match Attack

This section presents the Mix and Match attack which delivers an update package containing incompatible versions of software components. Figure 7.21 shows the attack

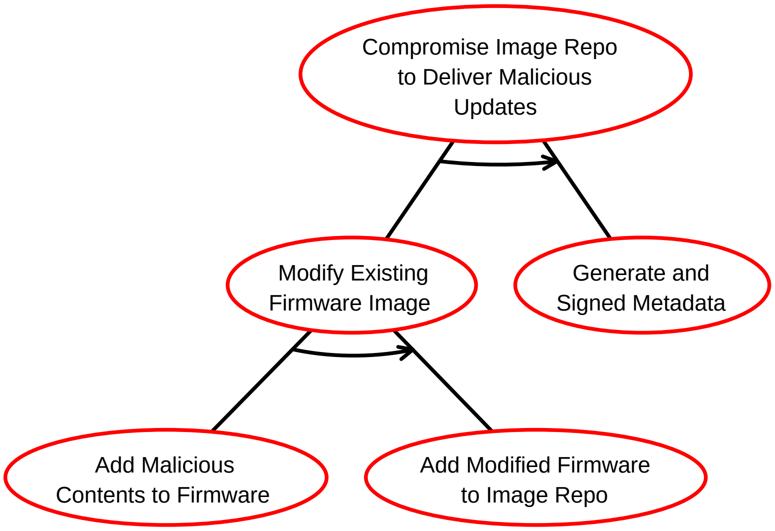

Compromise Image Repo to Deliver Malicious Updates
Modify Existing Firmware Image
Generate and Signed Metadata
Add Malicious Contents to Firmware
Add Modified Firmware to Image Repo

Figure 7.17: *Attack Tree - Compromise Image Repo to Deliver Malicious Updates: This attack tree represents the Threat 22.2 as enlisted in Table 7.2 that involves sending malicious updates to the clients by compromising the Image repository.*

tree representing various key steps performed by attackers to mount this attack. The SAND attack tree represents the main goal of the attack as root node (i.e., Mix and Match Attack) with four leaf nodes. Two **AND** subtrees were added to the parent tree with the first one deleting the currently valid snapshot metadata file from both repositories followed by the second one adding the modified snapshot to the repositories. The malicious snapshot metadata file added to the repositories show a set of valid but mutually incompatible software images to the client to be installed on different ECUs in the vehicle. If such an attack is successful, it can lead to affected vehicle operations arising from interoperability issues caused by devices containing conflicting software.

The Overall Attack Tree

We now have all the attack trees for the selected threats. They are combined into an overall attack tree for the reference implementation of the Uptane Framework. This overall tree is presented in the Figure 7.22. Subtrees have been organized into the STRIDE threat categories.

It is worth mentioning here that we only include the main goals of the trees excluding the leaf nodes/child nodes to build the overall attack tree. The overall goal of this tree becomes *Compromise Uptane Framework*, and the subtrees are different ways to accomplish this. Since this overall attack tree is an OR tree, smaller subtrees can be individually used in the experiments. Additionally, as the Uptane Framework is a distributed system, running all the tests at once is challenging. This is because when one experiment with one or more test cases is executed, it affects the system configurations

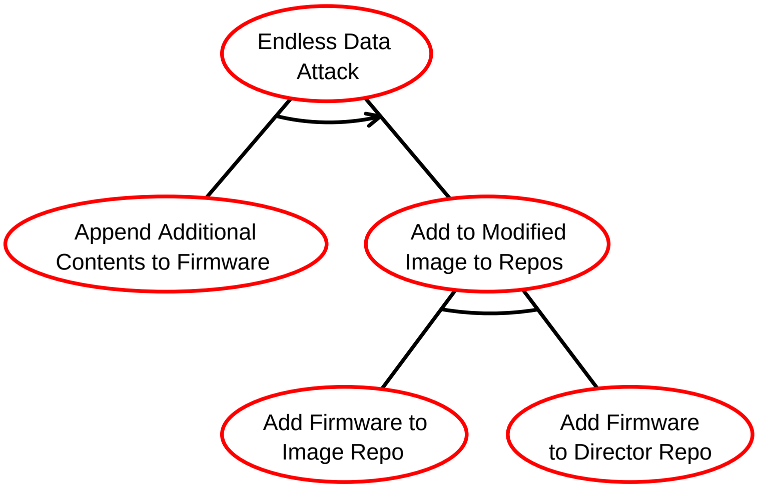

Endless Data Attack
Append Additional Contents to Firmware
Add to Modified Image to Repos
Add Firmware to Image Repo
Add Firmware to Director Repo

Figure 7.18: *Attack Tree - Endless Data Attack: This attack tree represents the Threat 26 as enlisted in Table 7.2 that involves delivering an update containing a huge amount of data in order to overwhelm the target ECU.*

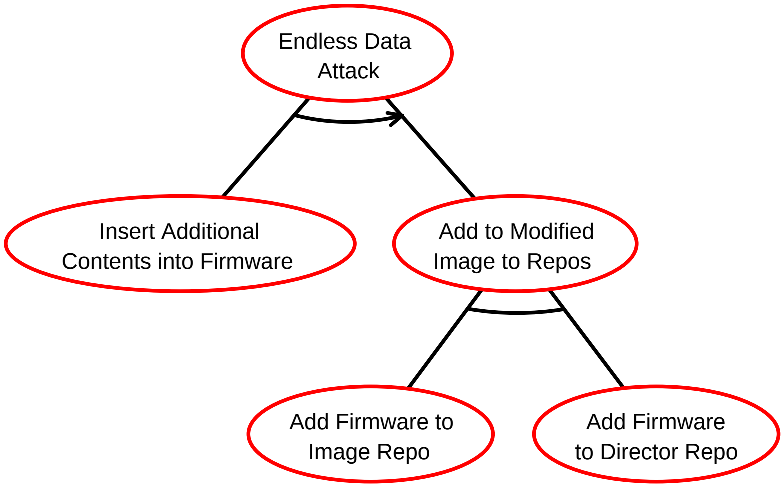

Endless Data Attack
Insert Additional Contents into Firmware
Add to Modified Image to Repos
Add Firmware to Image Repo
Add Firmware to Director Repo

Figure 7.19: *Attack Tree - Endless Data Attack (insert mode): This attack tree represents a variation of the Endless Data attack presented above that involves delivering an update containing a huge amount of data to overwhelm the ECU.*

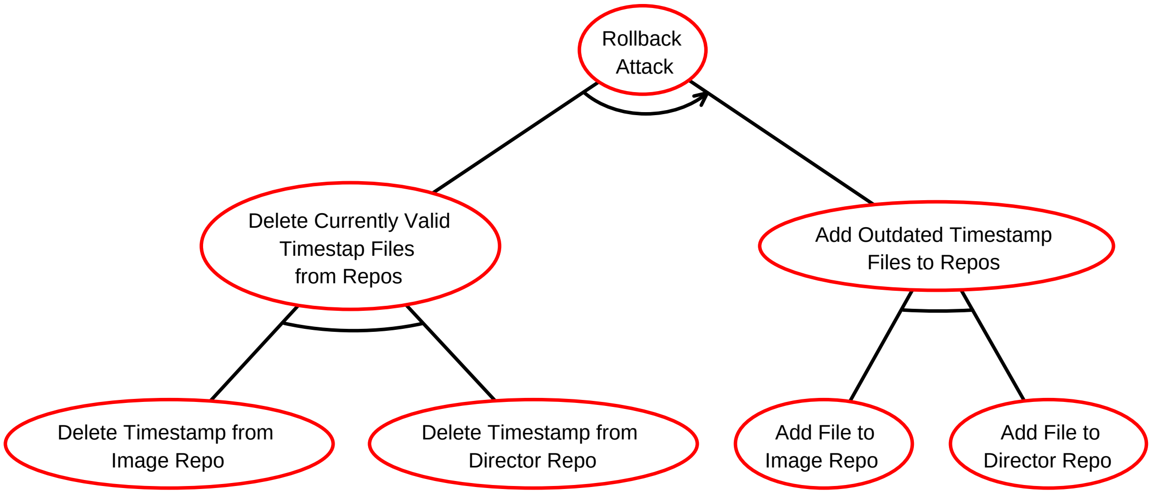

Rollback Attack
Delete Currently Valid Timestap Files from Repos
Add Outdated Timestamp Files to Repos
Delete Timestamp from Image Repo
Delete Timestamp from Director Repo
Add File to Image Repo
Add File to Director Repo

Figure 7.20: *Attack Tree - Rollback Attack: This SAND attack tree represents the Rollback Attack (Threat 27 in Table 7.2) that involves delivering an old but valid firmware image to the ECU.*

that require a system reset for the other tests to run effectively. Furthermore, some tests need human intervention/observation at the client side for the attack actions to take effect, which is not possible when all tests are executed in a single session.

7.6 Results

In this section results of the automotive OTA security-testing experimentation are presented. Each attack tree was analyzed by our bespoke software tool to derive test cases and generate appropriate test scripts to be executed against the reference implementation. The tool derived a total of 29 different test cases and generated relevant test scripts. Thirteen of these security test cases failed after successful execution of the test scripts; while a summary is provided in the Table 7.3, detailed results of these tests are presented in the following subsections individually.

In order to determine the risk level associated with each threat type, we used OWASP Risk Assessment Calculator (as described in Chapter 2). Based on the specified factor values, the tool determines an overall risk level. While the report generated by the tool shows the individual risk levels (i.e., probability and impact), the overall risk severity levels have been presented for each of the threats presented in Table 7.3.

Threat 6: Updates Could Be Downloaded

As indicated in the Table 7.4, we were able to directly download the firmware image files from both the Director and Image repositories, showing lack of any authentication/access control mechanism at the server-side to restrict the download to only legitimate

Table 7.3: *Security testing results at a glance.*

Threat ID	SUT Component	Total TCs	Failed TCs	Passed TCs	Risk Rating
6	Image & Director Repos	2	2	0	Medium
7	Image & Director Repos	2	2	0	Medium
9	Director Repo	2	2	0	Medium
10	Image Repo	2	2	0	Medium
11	Primary ECU	2	2	0	Low
12	Time Server	2	2	0	Medium
21.1	Director Repo	1	0	1	Medium
21.2	Director Repo	1	0	1	Medium
21.3	Image & Director Repos	1	1	0	Medium
22.1	Image Repo	1	0	1	Medium
22.2	Image Repo	1	0	1	Medium
26.1	Image & Director Repos	2	0	2	Low
26.2	Image & Director Repos	2	0	2	Low
27	Image & Director Repos	4	0	4	Low
28	Image & Director Repos	4	0	4	Low
	Totals:	**29**	**13**	**16**	

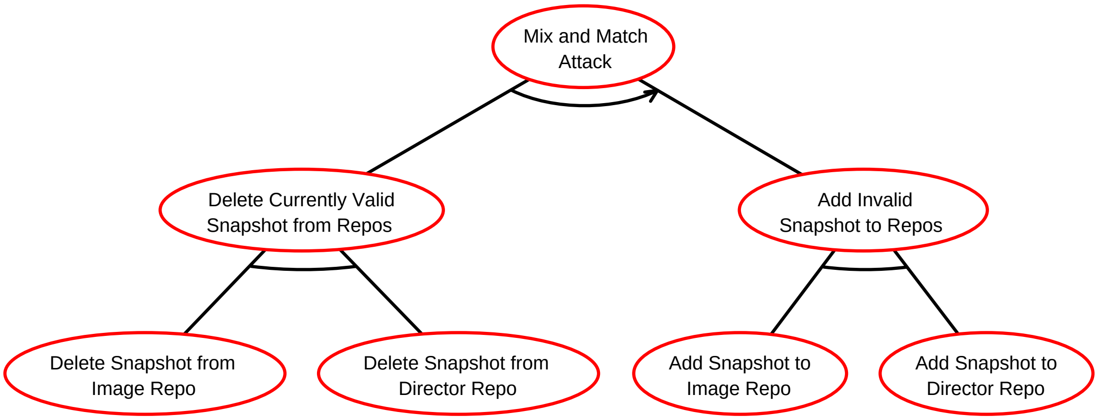

Mix and Match Attack
Delete Currently Valid Snapshot from Repos
Add Invalid Snapshot to Repos
Delete Snapshot from Image Repo
Delete Snapshot from Director Repo
Add Snapshot to Image Repo
Add Snapshot to Director Repo

Figure 7.21: *Attack Tree - Mix and Match Attack: This SAND attack tree represents one of the attacks (Threat 28 in Table 7.2) that involves delivering an update containing valid but incompatible versions of software updates.*

Table 7.4: *Summarizes the results of Threat 6 - Updates Could Be Downloaded.*

Updates Could Be Downloaded
TC_COUNT = 2 TC_1 = [("event_Determine_Image_Repo_Firmware_URL","determine_IR_url"), ("event_Establish_Network_Connection","establish_connection"), ("event_Download_Firmware","download_firmware")] Status: **FAIL**
TC_2 = [("event_Determine_Director_Repo_Firmware_URL","determine_DR_url"), ("event_Establish_Network_Connection","establish_connection"), ("event_Download_Firmware","download_firmware")] Status: **FAIL**
Test Outcome: The firmware images were successfully downloaded from both the Image Repository and Director Repository using simple HTTP calls without any authentication/authorization.

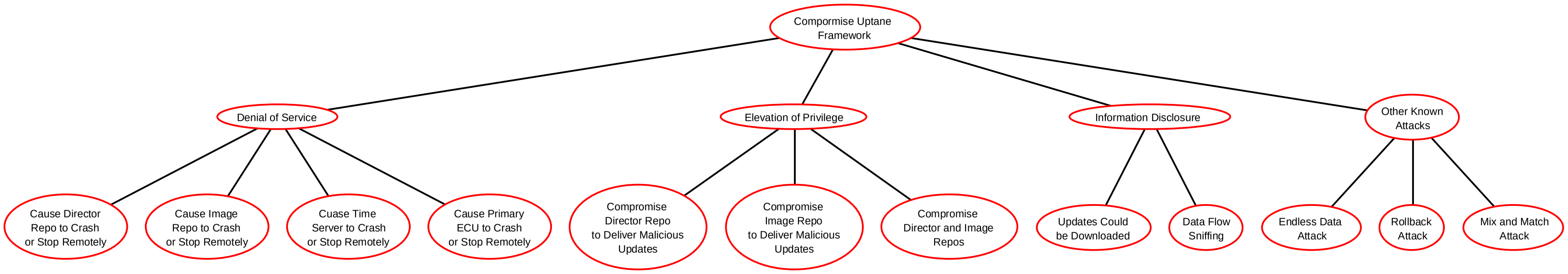

Compormise Uptane Framework
Denial of Service
Elevation of Privilege
Information Disclosure
Other Known Attacks
Cause Director Repo to Crash or Stop Remotely
Cause Image Repo to Crash or Stop Remotely
Cuase Time Server to Crash or Stop Remotely
Cause Primary ECU to Crash or Stop Remotely
Compromise Director Repo to Deliver Malicious Updates
Compromise Image Repo to Deliver Malicious Updates
Compromise Director and Image Repos
Updates Could be Downloaded
Data Flow Sniffing
Endless Data Attack
Rollback Attack
Mix and Match Attack

Figure 7.22

Attack Tree - Compromise Uptane Framework: This figure presents a comprehensive overview of all the threats (compromising Uptane Framework) included in the experimentation by combining all attack trees. Note that the attack trees have been organized into different STRIDE threat categories. Leaf nodes are nod included in this diagram.

clients. The incidence of information disclosure in this scenario may have serious ramifications, as if cybercriminals were able to reverse engineer the downloaded images to identify any vulnerabilities fixed in the current version by comparing it with an older version of the firmware, they could use that intelligence for crafting destructive attacks for compromising entire fleets of vehicles. Additionally, the reference implementation relies on the plain HTTP for communication between server and clients, as described in the following section. The risk rating for this threat has been determined to be *Medium* (see Table 7.3). Since this threat does not require any special skills (hence basic Network and programming skills was specified for Skills required factor in the *Threat Agent Factors* section), and it is easy to discover and exploit, a Medium value was calculated by the tool for the Likelihood factors. In the *Technical Impact Factors* section, the relevant factor was Loss of confidentiality, and for the Reputation damage factor in the *Business Impact Factors* section we selected Loss of goodwill, as information disclosure of sensitive update data can affect the reputation of the OEM due to lack of effective protection for ensuring confidentiality.

Threat 7: Data Flow Sniffing

Table 7.5 provides a summary of the sniffing attack, listing the steps and outlining the outcome. As can be seen in the table, information exchange between Uptane servers and clients is not encrypted. All the Remote Procedure Calls (RPCs) from the Primary and responses from the Repositories are readable. This particular interception session was also able to capture the firmware image contents. As discussed above, the wealth of information accessible to the adversaries without applying any confidentiality and privacy security controls can potentially lead to very serious consequences. As shown in the Table 7.3, the risk rating for this threat has been determined to be *Medium*, as similar to the preceding threat (i.e., Updates Could be Downloaded), the ease of exploitation and discovery along with basic skills required to accomplish this were the key factors that calculated the Medium value for Likelihood factors. As Technical and Business Impact factors under the Impact Factors were similar to those in the previous threat, hence a Medium value was calculated, resulting in an overall Medium risk rating.

Threat 9: Cause Director Repo to Crash or Stop Remotely

Table 7.6 shows the result of failed attempts from Primary ECU to receive any response from Director Repository after it was affected by the DoS attack. As Director Repository is responsible for direct communication with the Primary ECU, its unavailability will have an impact on the normal operations of the entire update process, as no further actions will succeed. In order to restore the normal operation, the servers had to be rebooted. An overall risk rating of *Medium* has been determined, as shown in the Table 7.3. Main influencing factors in the Likelihood factors area include the ease of discovery/exploitation and the relatively basic skills required along with Loss

Table 7.5: *Summarizes the results of Threat 7 - Data Flow Sniffing.*

Data Flow Sniffing

TC_COUNT = 2
TC_1 = [("event_Determine_IP_Address","determine_IP"), ("event_Determine_Port_Number","determine_port"), ("event_Establish_Network_Connection","establish_connection"), ("event_Intercept_Network_Traffic","intercept_traffic"), ("event_Analyze_Captured_Network_Data","analyze_network_data")]
Status: **FAIL**

TC_2 = [("event_Determine_Port_Number","determine_port"), ("event_Determine_IP_Address","determine_IP"), ("event_Establish_Network_Connection","establih_connection"), ("event_Intercept_Network_Traffic","intercept_traffic"), ("event_Analyze_Captured_Network_Data","analyze_network_data")] Status: **FAIL**
Test Outcome: Data exchange between Uptane repositories and Primary ECU was successfully captured. This included RPC calls, corresponding responses, and the firmware image file contents in plain text.

of Availability (i.e., Extensive primary services interrupted), Reputation damage (i.e., Loss of goodwill) in the Impact factors led to this risk score.

Threat 10: Cause Image Repository to Crash or Stop Remotely

As shown in the Table 7.7, DoS attack on Image Repository caused it to stop responding to requests from the client. However, Director Repository was not affected by the attack and continued to respond to client requests. The result shows that the Primary ECU succeeded with downloading the metadata from Director Repository. As the attack method (i.e., Denial of Service) and the target are similar, hence the same risk rating value (i.e., Medium) has been determined for this threat.

Threat 11: Cause the Primary ECU to Crash or Stop Remotely

After successful DoS attacks on Uptane repositories, we mounted a DoS attack on the Primary ECU, which succeeded in causing the Primary to stop working properly. A

Table 7.6: *Summarizes the results of Threat 9 - Cause the Director Repository to Crash or Stop Remotely.*

Cause the Director Repository to Crash or Stop Remotely

`TC_COUNT = 2`
`TC_1 = [("event_Determine_IP_Address","determine_IP"), ("event_Determine_Port_Number","determine_port"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Director_Repo","flood_dirctor")]`
Status: **FAIL**

`TC_2 = [("event_Determine_Port_Number","determine_port"), ("event_Determine_IP_Address","determine_IP"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Director_Repo","flood_director")]` Status: **FAIL**
Test Outcome: The Director Repository failed to respond to legitimate requests from the Primary ECU after successful DoS attack. The Primary ECU displayed *URLError and timed out* error messages (stating it was unable to download metadata when it attempted to request update from the Director Repository).

Low risk rating has been determined for this threat (as can be seen in the Table 7.3). The main contributing factors for a low risk rating value include the extent of damage (reputation damage is unlikely due to a single device/service in the vehicle being the target as compared to the update repositories that can affect multiple vehicle at once).

As can be seen in Table 7.8, the Secondary ECU was presented with the error messages indicating connection could not be established with the Primary ECU. Therefore, any new updates for the Secondary ECU could not be delivered. This DoS attack is less severe as opposed to the ones launched against Uptane servers, as its impact is limited to one vehicle only.

Threat 12: Cause the Time Server to Crash or Stop Remotely

The results displayed in Table 7.9 the DoS attack on the Time Server disrupted its functionality, resulting in undesirable behaviour from the Primary, as it seemed to wait forever for a response containing validated time from the Time Server. It was observed that there was no set timeout limit for preventing such situations where clients await a

Table 7.7: *Summarizes the results of Threat 10 - Cause the Image Repository to Crash or Stop Remotely.*

Cause the Image Repository to Crash or Stop Remotely
TC_COUNT = 2
TC_1 = [("event_Determine_IP_Address","determine_IP"), ("event_Determine_Port_Number","determine_port"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Image_Repo","flood_imagerepo")] Status: **FAIL**
TC_2 = [("event_Determine_Port_Number","determine_port"), ("event_Determine_IP_Address","determine_IP"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Image_Repo","flood_imagerepo")] Status: **FAIL**
Test Outcome: The Image Repository failed to respond to legitimate requests from the Primary ECU after successful DoS attack. The Primary ECU displayed *URLError and timed out* error messages when it attempted to request update from the Image Repository. As the attack method (i.e., Denial of Service) and the target are similar, hence the same risk rating value (i.e., Medium) has been determined for this threat.

response indefinitely. In order to verify whether Director Repository responds to other requests from clients, a request for registering the ECU with the Director Repository, which was successfully processed by the Director Repository, demonstrating the correct operation of all other server components/services. Similar to the DoS attacks on Image and Director repositories above, similar likelihood and impact factors chosen led to the *Medium* risk rating value for the Time Server.

Threat 21.1: Compromise Director Repository for Sending Malicious Updates

The Table 7.10 presents the outcome of the attack aiming at sending a malicious firmware image to the client. The Primary ECU did not download the compromised firmware image from Director Repository after it found a bad hash value while perform-

Table 7.8: *Summarizes the results of Threat 11 - Cause the Primary ECU (TCU) to Crash or Stop Remotely.*

Cause the Primary ECU (TCU) to Crash or Stop Remotely

TC_COUNT = 2
TC_1 = [("event_Determine_IP_Address","determine_ip"), ("event_Determine_Port_Number","determine_port"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Pirmary_ECU","flood_primary")]
Status: **FAIL**

TC_2 = [("event_Determine_Port_Number","determine_port"), ("event_Determine_IP_Address","determine_ip"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Pirmary_ECU","flood_primary")] Status: **FAIL**
Test Outcome: Following the DoS attack on Primary ECU, the Secondary ECU made several attempts to connect to the Primary ECU, the Primary ECU did not respond to any of the requests from Secondary ECU. Secondary ECU seemed to wait forever, and the process had to be manually interrupted to resume normal operation, which resulted in displaying error messages stating unsuccessful connection attempts.

ing the verification. On the other hand, it proceeded with downloading the firmware file from Image repository, because no issues were found with the metadata sent by Image Repository. The update was not presented to the Secondary by the Primary after it detected the anomaly. The risk rating for this threat has been determined to be *Medium*, as shown in the Table 7.3. While likelihood factors calculated a *Low* score (due to a high skills level required and difficulty of exploitation and discovery), the impact factors resulted in a *Medium* value.

Threat 21.2: Compromise Director Repository (with valid Keys) to Send Malicious Updates

As shown in the Table 7.11, the Primary ECU rejected to download the image from the server after it detected an anomaly in the metadata from Director and Image Repositories. The risk rating for this threat has been determined to be *Medium*, as shown in the Table 7.3. While likelihood factors calculated a *Low* score (due to a high

Table 7.9: *Summarizes the results of Threat 12 - Cause the Time Server to Crash or Stop Remotely.*

Cause the Time Server to Crash or Stop Remotely

`TC_COUNT = 2`
 	
`TC_1 = [("event_Determine_IP_Address","determine_ip"), ("event_Determine_Port_Number","determine_port"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Time_Server","flood_time_server")]`
Status: **FAIL**

`TC_2 = [("event_Determine_Port_Number","determine_port"), ("event_Determine_IP_Address","determine_ip"), ("event_Establish_Network_Connection","establish_connection"), ("event_Flood_Time_Server","flood_time_server")]` Status: **FAIL**
Test Outcome: Following the DoS attack, the Time Server failed to respond to legitimate requests for validated time from the Primary ECU. The Primary ECU begins waiting indefinitely for a valid response from the Time Server after calling the method get_time_attestation from within the method update_cycle. All the other components on the server-side were found to be responsive and operational when tested. For instance, the Director responded as usual when the Primary invoked the method clean_slate after the DoS attack on Time Server. This clearly indicates that the DoS attack successfully causes the Time Server to crash. It also indicates that there is no timeout and/or exception handling mechanisms in place that can deal with such circumstances.

level of skills level required and difficulty of exploitation and discovery), the impact factors resulted in a *Medium* value.

Unlike the previous attempt, the firmware metadata was signed with valid keys. Even though the metadata, received from both repositories, was correct; however, since inconsistent hash values were received from each of the repositories, Primary ECU discarded the update.

Table 7.10: *Summarizes the results of Threat 21.1 - Compromise Director Repo in Order to Deliver Malicious Update (without compromised keys).*

Compromise Director Repository in Order to Deliver Malicious Update (without compromised keys)

`TC_COUNT = 1`
`TC_1 = [("event_Add_Malicious_Contents_to_Firmware","add_contents"), ("event_Add_Modified_Firmware_to_Director_Repo","add_to_director")]`
Status: **PASS**

Test Outcome: Primary ECU did not download the firmware image from the Director Repository, since the hash values did not match, a *BadHashError* error occurred. In contrast, the Primary was able to successfully download firmware image from Image Repository, because the hash values were correct.

Threat 21.3: Compromise both Image and Director Repositories with Compromised Keys to Send Malicious Updates

The information presented in Table 7.12 shows the result of the most dangerous attack, wherein the attackers were able to add malicious contents to the firmware image, sign it and its associated metadata with valid keys both by Image and Director repositories. As both the update itself and associated metadata were valid and correct, the Primary proceeded with downloading the malicious image file and passing it to the Secondary for installation. The risk rating for this threat has been determined to be *Medium*, as shown in the Table 7.3. While likelihood factors calculated a *Low* score (due to a high level of skills required and difficulty of exploitation and discovery), the impact factors resulted in a *Medium* value, as in the case of successful attack, there is a potential for reputation and financial damages.

Threat 22.1: Compromise Image Repository to Send Malicious Updates

Table 7.13 summarises the results of an attack involving modifying a firmware image on the Image Repository to include malicious contents in order to send to the Primary. This attack was carried out without having access to the keys for signing the update and metadata, which could not succeed, as additional data would not be downloaded by the Primary. Moreover, as the changes were only made to the firmware at Image Repository, the verification will not be successful when the Primary ECU tries to validate the image with both Image and Director repositories. The risk rating for this

Table 7.11: *Summarizes the results of Threat 21.2 - Compromise Director Repo in Order to Deliver Malicious Update (with compromised keys)*

Compromise Director Repository in Order to Deliver Malicious Update (with compromised keys)

`TC_COUNT = 1`
`TC_1 = [("event_Add_Malicious_Contents_to_Firmware","add_contents"), ("event_Add_Modified_Firmware_to_Director_Repo","add_to_director"), ("event_Generate_and_Sign_the_Metadata","gen_director_metadata")]`
Status: **PASS**

Test Outcome: The Primary ECU refused to download the update from the Director and displayed the following error message: *Director has instructed us to download a target (/firmware1.img) that is not validated by the combination of Image + Director Repositories. That update IS BEING SKIPPED. It may be that files have changed in the last few moments on the repositories. Try again, but if this happens often, you may be connecting to an untrustworthy Director, or there may be an untrustworthy Image Repository, or the Director and Image Repository may be out of sync.*

threat has been determined to be *Medium*, as shown in the Table 7.3. While likelihood factors calculated a *Low* score (due to a high level of skills required and difficulty of exploitation and discovery), the impact factors resulted in a *Medium* value.

Threat 22.2: Compromise Image Repository (with Valid Keys) to Send Malicious Updates

The Primary ECU did not accept the update (as shown in Table 7.14, as it found an anomaly in the metadata while validating it with the Director and Image repositories. The metadata sent by the Director was different from the one sent by Image Repository, even though valid keys were used for signing the malicious update by Image repository. The risk rating for this threat has been determined to be *Medium*, as shown in the Table 7.3. While likelihood factors calculated a *Low* score (due to a high level of skills required and difficulty of exploitation and discovery), the impact factors resulted in a *Medium* value.

Table 7.12: *Summarizes the results of Threat 21.3 - Compromise Image and Director Repositories in Order to Deliver Malicious Updates (with compromised keys).*

Compromise Image and Director Repositories in Order to Deliver Malicious Updates (with compromised keys)

TC_COUNT = 1
TC_1 = [("event_Add_Malicious_Contents_to_Firmware","add_contents"), ("event_Add_Modified_Firmware_to_Image_Repo","add_to_imagerepo"), ("event_Generate_Image_Repo_Metadata","gen_imagerepo_metadata"), ("event_Add_Modified_FIrmware_to_Director_Repo","add_to_dir"), ("event_Generate_Director_Repo_Metadata","gen_dir_metadata")]
Status: **FAIL**

Test Outcome: The Primary ECU successfully downloaded and forwarded the malicious firmware image to the Secondary ECU. Both clients could not detect the presence of any malicious contents, as can be seen from the following messages: *Metadata for the following Targets has been validated by both the Director and the Image repository. They will now be downloaded:['/firmware1.img']; Successfully downloaded trustworthy 'firmware1.img' image.*

Threats 26.1 and 26.2: Endless Data Attack

Tables 7.15 and 7.16 show the results of two variations of Endless Data Attack, with the goal to inundate the Primary ECU with a large amount of data to affect its functionality. In the first scenario, we appended additional contents to a firmware image and re-added it to both Director and Image repositories.

As expected, the Primary only downloaded the original image ignoring additional appended data. The Primary reads the associated metadata and downloads the amount of update data as specified in the metadata file. We then decided to add additional data to the firmware by replacing existing contents or inserting the data in the image file at a location other than the end of file. This is done to observe the response of the Primary if this occurs. Primary ECU detected the changes made to the original firmware file by discovering anomalies in the metadata. It is worth noting that both variations of this test were carried out without generating and signing metadata and updates. A *Low* risk rating has been determined for these threats, as this attack requires special skills, privileged access and modification of update files on the update servers. Additionally,

Table 7.13: *Summarizes the results of Threat 22.1 - Compromise Image Repository in Order to Deliver Malicious Updates (without compromised keys).*

Compromise Image Repository in Order to Deliver Malicious Update (without compromised keys)

`TC_COUNT = 1`
`TC_1 = [("event_Add_Malicious_Contents_to_Firmware","add_contents"), ("event_Add_Modified_Firmware_to_Image_Repo")]`
Status: **PASS**

Test Outcome: The Primary ECU did not detect any changes made at the server to the Firmware image on Image Repository. This is because the metadata generated by both repositories was still intact; hence, Primary ECU assumed the original file was still there without any changes. Therefore, only original firmware image would be downloaded by the client, the one with valid metadata, ignoring any modifications made. This happens because the metadata contains information about the size of the firmware image file.

the impact of the attack would be limited to a single target vehicle should an attack is successful.

Threat 27: Rollback Attack

As its name implies, the objective of this attack was to cause the ECU to uninstall the newest installed version of an image and install an older one instead by replacing the timestamp on both Director and Image repositories. Table 7.17, shows the key steps (actions) and the result of the test. The Primary detected that it was sent an older version of the timestamp; therefore, it rejected it and thus the latest version of the update was not rolled back. As shown in the Table 7.3, a *Low* risk rating has been determined for this threat, mainly due to the level of difficulty, skills, window of opportunity available.

Threat 28: Mix and Match Attack

As shown in the Table 7.18, the invalid Snapshot file sent to the Primary was rejected as the accompanied metadata could not be verified. Hence, the attack could not succeed. This attack intended to include incompatible versions of the firmwares in an

Table 7.14: *Summarizes the results of Threat 22.2 - Compromise Image Repository in Order to Deliver Malicious Updates (with compromised keys).*

Compromise Image Repository in Order to Deliver Malicious Updates (with compromised keys)

`TC_COUNT = 1`
`TC_1 = [("event_Add_Malicious_Contents_to_Firmware","add_contents"), ("event_Add_Modified_Firmware_to_Image_Repo","add_to_imagerpo"), ("event_Generate_and_Sign_Metadata","gen_imagerepo_metadata")]`
Status: **PASS**

Test Outcome: The Primary did not proceed with the download and displayed the following error message: *Director has instructed us to download a target (firmware1.img) that is not validated by the combination of Image + Director Repositories. That update IS BEING SKIPPED. It may be that files have changed in the last few moments on the repositories. Try again, but if this happens often, you may be connecting to an untrustworthy Director, or there may be an untrustworthy Image Repository, or the Director and Image Repository may be out of sync.*

update package, causing interoperability issues among various ECUs having mutually incompatible software versions. As mentioned above, the inconsistent metadata allowed the Primary to establish that the update is not trustworthy, thus it refused to proceed with the download process. Uptane Framework's Snapshot metadata is an effective protection mechanism against advanced attacks involving tricking the clients to download and install software updates that are individually valid, but can lead to serious interoperability problems. As shown in the Table 7.3, the risk rating for this threat has also been determined to be *Low*, because of the difficulty of exploitation and special skills/privileged access required. A very low score for the likelihood is the key factor that resulted in the overall Low rating for this attack.

7.6.1 Key Findings

The systematic threat assessment and security testing approach we employed in this study showed promising results by revealing unmitigated threats/vulnerabilities in the reference implementation of Uptane Framework by applying a structured approach for threat identification to construct attack trees, model-based security testing for step-by-step derivation of test cases from the attack trees constructed in the preceding step,

Table 7.15: *Summarizing the results of Threats 26.1 and 26.2 - Endless Data Attack.*

Endless Data Attack (with appended contents)
`TC_COUNT = 2`
`TC_1 = [("event_Append_Additional_Contents_to_Firmware","add_contents"), ("event_Add_Firmware_to_Image_Repo","add_to_imagerepo"), ("event_Add_Firmware_to_Director_Repo","add_to_director")]` Status: **PASS**
`TC_2 = [("event_Append_Additional_Contents_to_Firmware","add_contents"), ("event_Add_Firmware_to_Director_Repo","add_to_director"), ("event_Add_Firmware_to_Image_Repo","add_to_imagerepo")]` Status: **PASS**
Test Outcome: The attack was defended by the Uptane Framework by accepting and downloading exactly the same amount of data as specified in the trusted metadata file. The appended contents were ignored by both the Primary and Secondary ECUs.

and the automation of security test-case generation and execution. As, derivation of appropriate and effective security test cases is often considered a challenging and non-trivial task, because in addition to the knowledge of potential threats, it requires a clear idea of what to test and where to start [124]. Threat modeling techniques and tools we used, allowed us to effectively identify several security threats targeting automotive OTA updates in a systematic and repeatable manner. Based on the selected subset of threats identified in the threat enumeration step, our test case generation approach was able to derive 29 different test cases providing effective coverage affecting various core components (i.e., Image repository, Director repository, Time Server, and Primary Server) of the OTA ecosystem. The range of threat types identified includes everything from tampering to elevation of privilege. The experimental cyberattacks crafted from the derived security test cases provided a variety of techniques potentially used by adversaries.

While threat enumeration using STRIDE model allowed us to identify a number of threats that could affect the security of automotive OTA update processes and procedures in a variety of ways, we also included some of the known attacks that have

Table 7.16: *Summarizing the results of Threats 26.2 - Endless Data Attack (inserting contents).*

Endless Data Attack (with overwritten or inserted contents)

`TC_COUNT = 2`
 	
`TC_1 =`
`[("event_Insert_Contents_intto_Firmware","insert_contents"), ("event_Add_Firmware_to_Image_Repo","add_to_imagerepo"), ("event_Add_Firmware_to_Director_Repo","add_to_director")]`
Status: **PASS**

`TC_2 =`
`[("event_Insert_Contents_into_Firmware","insert_contents"),`
`("event_Add_Firmware_to_Director_Repo","add_to_director"),`
`("event_Add_Firmware_to_Image_Repo","add_to_imagerepo")]`
Status: **PASS**

Test Outcome: This attack was detected and defended by the Uptane Framework; consequently, the entire update was rejected due to the inconsistent hash values. The Primary ECU refused to download the update file by showing *BadHashError* error messages.

been launched on the update systems/repositories in the past, which include: mix-and-match attack, rollback attack, and endless data attack. Experimental results of these and other attacks have shown the Uptane Framework's ability to effectively combat threats involving tampering and manipulation of updates. On the other hand, some of experimental results suggest that a production-quality, real-world implementation of the Uptane Framework would require effective measures against common threats, such as denial of service and information disclosure. For example, we showed how firmware images could be easily downloaded from Uptane servers without encountering any access-control/authentication restrictions. Additionally, the results demonstrated how the information exchange between the servers and clients could be easily intercepted. In order to ensure and maintain confidentiality of the sensitive information, state-of-the art technologies need to be applied for providing adequate protection against information disclosure threats. Similarly, the denial-of-service attacks mounted against the Uptane repositories and Primary ECU demonstrated how the timely delivery of critical updates can be hampered, affecting the availability of crucial update services.

Table 7.17: *Summarizing the results of Threat 27 - Rollback Attack.*

Rollback Attack

TC_COUNT = 4
TC_1 = [("event_Delete_Timestamp_from_Image_Repo","delete_timestamp"), ("event_Delete_Timestamp_from_Director_Repo","delete_timestamp"), ("event_Add_File_to_Image_Repo","add_timestamp"), ("event_Add_File_to_Director_Repo","add_timestamp")]
Status: **PASS**

tTC_2 = [("event_Delete_Timestamp_from_Image_Repo","delete_timestamp"), ("event_Delete_Timestamp_from_Director_Repo","delete_timestamp"), ("event_Add_File_to_Director_Repo","add_timestamp"), ("event_Add_File_to_Image_Repo","add_timestamp")] Status: **PASS**
TC_3 = [("event_Delete_Timestamp_from_Director_Repo","delete_timestamp"), ("event_Delete_Timestamp_from_Image_Repo","delete_timestamp"), ("event_Add_File_to_Image_Repo","add_timestamp"), ("event_Add_File_to_Director_Repo","add_timestamp")] Status: **PASS**
TC_4 = [("event_Delete_Timestamp_from_Director_Repo","delete_timestamp"), ("event_Delete_Timestamp_from_Image_Repo","delete_timestamp"), ("event_Add_File_to_Director_Repo","add_timestamp"), ("event_Add_File_to_Image_Repo","add_timestamp")] Status: **PASS**
Test Outcome: The Primary ECU refused to proceed with the update process with the following error message: *The Director has instructed us to download a Timestamp that is older than the currently trusted version. This instruction has been rejected. As the Primary has rejected the update, the Secondary was not presented with the update by the Primary ECU.*

Table 7.18: *Summarizing the results of Threat 28 - Mix and Match Attack.*

Results: Mix and Match Attack
`TC_COUNT = 4`
`TC_1 = [("event_Delete_Snapshot_from_Image_Repo","delete_snapshot_ir"), ("event_Delete_Snapshot_from_Director_Repo","delete_snapshot_dr"), ("event_Add_Snapshot_to_Image_Repo","add_snapshot_ir"), ("event_Add_Snapshot_to_Director_Repo","add_snapshot_dr")]` Status: **PASS**
`TC_2 = [("event_Delete_Snapshot_from_Image_Repo","delete_snapshot_ir"), ("event_Delete_Snapshot_from_Director_Repo","delete_snapshot_dr"), ("event_Add_Snapshot_to_Director_Repo","add_snapshot_director"), ("event_Add_Snapshot_to_Image_Repo","add_snapshot_ir")]` Status: **PASS**
`TC_3 = [("event_Delete_Snapshot_from_Director_Repo","delete_snapshot_dr"), ("event_Delete_Snapshot_from_Image_Repo","delete_snapshot_ir"), ("event_Add_Snapshot_to_Image_Repo","add_snapshot_ir"), ("event_Add_Snapshot_to_Director_Repo","add_snapshot_dr")]` Status: **PASS**
`TC_4 = [("event_Delete_Snapshot_from_Director_Repo","delete_snapshot_dr"), ("event_Delete_Snapshot_from_Image_Repo","delete_snapshot_ir"), ("event_Add_Snapshot_to_Director_Repo","add_snapshot_drr"), ("event_Add_Snapshot_to_Image_Repo","add_snapshot_ir")]` Status: **PASS**
Test Outcome: The Primary ECU refused to proceed with the update process by showing *BadHashError* message.

Failed Security Test Cases As mentioned earlier, about fifty percent of (13 out of 29) security test cases did not succeed in this experimental security analysis of the reference implementation, which involve denial-of-service, elevation of privilege, and information disclosure attacks against the Uptane repositories and the Primary ECU. While it can be argued that since this particular implementation has not been developed for meeting the production-level quality, the findings from the experimental results still serve the useful purpose of showing serious issues that must be taken into account before they are discovered by malicious actors in the real world, leading to serious ramifications. The first two attacks (i.e., Updates Could be Downloaded and Data Flow Sniffing) have shown the repositories' susceptibility of becoming easy targets of eavesdropping and Man-In-The-Middle (MITM) attacks. In particular, we found that the network data captured being exchanged between the repositories and the Primary contained sensitive information (such as, vehicle manifest, contents of the remote procedure calls in plain text) that can be highly useful for the criminals to launch more complex and damaging attacks in the future. Hence, appropriate security controls must be applied to protect the confidentiality of the information, both in transit and while stored on a device.

Denial-of-service attacks mounted on the repositories demonstrated their fragility to collapse within a few seconds of being bombarded with excessive number of illegitimate requests. Since this type of attacks are very common and are relatively easy to carry out, OEMs and vendors must address this issue by deploying effective countermeasures to fail such malicious attempts and ensure timely delivery of important updates.

Successful experimental attacks involving elevation of privilege threats showed the potential impact of these attacks on the system after being compromised. We learned that compromising only one repository (Image or Director) could not influence the update itself or the procedures, as the metadata verification at the Primary (or Secondary) side will certainly fail due to incorrect/inconsistent sets of metadata from each of the repositories. It should also be noted that while we were able to compromise both the Director and Image repositories to deliver malicious updates to the target ECUs, practically such attacks require the attacker to compromise the secret cryptographic keys of both repositories, which is considerably difficult if not impossible to accomplish, as the keys for the Image Repository are kept offline securely. Hence, based on the effectiveness of this defensive countermeasure, we believe the framework provides adequate protection against such attacks unless the attacker is able to compromise all the keys.

Passed Security Test Cases Various attacks launched against the Uptane did not succeed proving the effectiveness of its protection mechanisms. Different types of metadata have been introduced to ensure protection against certain common attacks performed against update systems. For instance, mix-and-match and attacks are thwarted by using Snapshot metadata; similarly, freeze update, rollback and replay attacks are handled through the Timestamp metadata. Targets metadata is used to prove the authenticity of the firmware images. Moreover, the framework is also equipped with effective mechanisms to ensure protection against endless data attack and slow-retrieval attack. Thus, our findings from these experimental attacks suggest that tampering and replay attacks (as well as many other attacks) are well defended by the system.

To conclude, while the Uptane Framework offers solutions to various major threats to the update process by introducing effective mechanisms, our experimental results show the reference implementation is vulnerable to eavesdropping and denial-of-service attacks. Exploitation of these vulnerabilities in the production environment can cause serious disruptions to the distribution of important updates to connected vehicles. OEMs and other relevant stakeholders must take into consideration such threats and apply necessary security controls (e.g., authentication and access control) for maximum protection.

7.6.2 Limitations

While the approach we have used enabled us to systematically reveal various types of threats and subsequently derive security test cases, not all of those threats could be included in the experimentation for various legitimate reasons.

1. First of all, the main focus of this research work was on the threats that could potentially compromise the OTA procedures on the backend server-side (i.e., Image Repo, Director Repo, Time Server, etc.), rather than targeting and exploiting vehicle-side vulnerabilities/threats (e.g., Reflash the TCU (Primary) Firmware in Order to Send Arbitrary CAN Messages), as they have been extensively explored in prior studies [89, 125, 7]. Additionally, vehicle-side security compromises often require physical access to the vehicle, which was not conducive to the automated testing process.

2. Second, we did not take the threats (six different threats were identified by the tool) from the Repudiation category of STRIDE model into consideration for testing, as these threats independently are not capable of causing any harm/disruption to the Update procedures or system. That is, threats involving a component or a human user, either at server side or in the vehicle. denies performing a certain action, does not have any effect on the update process.

3. Third, the threat (having three instances) *Car Could be Tracked* has not been included in the experimentation, since it is not directly relevant/exclusive to OTA update system as well as it does not have any direct impact on the delivery or installation of the updates. Moreover, such threats are usually associated with the vehicle's external connectivity by using telematics, for instance.

4. Finally, we excluded those threats from the experimentation that have very similar attack steps, affects, and/or results to the ones investigated. For instance, *Take the Director Repo Offline, Flood Director Repo with Invalid Data*, and *Cause*

the Director Repo to Crash or Stop Remotely have similar attack method (i.e., DoS attack) and the same effect: that is, causing the Director Repository to become unresponsive to legitimate request from clients. This is simply because representative examples presented provide sufficient details and insights about various aspects of the threats.

Some attack trees we constructed could be further elaborated by including more attack steps; however, since our primary goal has been to demonstrate what the attacker would be able to do if they succeeded in compromising the update system, we intentionally did not focus on the tactics and methods that attackers can potentially use for breaking into the OTA servers, in order to prevent complexities affecting the automated test-case generation/execution. Moreover, since our investigations relied on automated test-case generation and execution process, social engineering techniques (e.g., stealing offline cryptographic keys for signing updates and metadata on Uptane Image Repository or the credentials of a system administrator) and other manual steps were not feasible to be included in the attack trees and test scripts.

Although, Threat Modeling Tool identified various threats in different categories of STRIDE, (as can be seen in Table 7.2 and 7.6) it reported no Spoofing threats to the Uptane repositories and clients. From a technical standpoint, it is perfectly possible that the attackers may use spoofing as one of the strategies to compromise the update procedures. In fact, several of our experiments rely on the assumption that the Uptane repositories were being spoofed or impersonated by malicious actors.

7.7 Chapter summary

This chapter presented the experimental security analysis of the Uptane Framework by applying the systematic security approach. All the key activities and associated steps described with appropriate details demonstarting the processes of information gathering through to attack tree construction and test-case generation and execution. We showed how we modelled the Uptane system to produce both a static and dynamic representation of the system, which enabled us to craft the data flow diagram in the threat enumeration process for identifying all the threats. Next, the identified threats were used to build 15 different attack trees, each corresponding to a particular threat in the threat modeling list, demonstrating our systematic approach to attack tree construction. For each of the attack trees, one or more corresponding test cases were generated by the software tool and executed against the reference implementation of the Uptane Framework. Finally, results of all the experiments along with key findings and limitations are highlighted.

Chapter 8

Countermeasures

This chapter presents some countermeasures against the experimental information disclosure and denial of service attacks on Uptane repositories and clients that succeeded (i.e., the ones with a ”FAIL” status) in the previous chapter. These experimental attacks were able to affect the security of the reference implementation, compromising the confidentiality and availability of the system. Based on high-level recommendations/guidelines from UNECE WP.29, appropriate mitigations have been discussed, highlighting their strengths and weaknesses. Attack-defence trees have been included to provide graphical representation of the attacks and relevant countermeasures/mitigations. While the suggested countermeasures/mitigations, best practices presented in this chapter are first steps towards effective protection, they should not be considered ultimate or comprehensive solutions.

8.1 Protection against Information Disclosure

In this section countermeasures against two different threats violating the confidentiality of the updates are presented. First, a discussion on how to protect the sensitive updates from unauthorised access is provided followed by a solution for ensuring security against interception of the sensitive communication/data.

8.1.1 How to Stop Unauthorised Update Downloads

Four attacks were successful in compromising the confidentiality of the updates, resulting in the disclosure of sensitive, proprietary updates information. As indicated earlier, while leakage of such information itself does not directly affect the operations/-functionality of the system, it can subsequently can be leveraged by the adversary to craft destructive, more sophisticated attacks.

Recall that this threat involved downloading the firmware images from the Uptane repositories without having to go through/compromise any security mechanisms (see Table 7.4 in Chapter 7). The reference implementation of Uptane Framework does not currently implement any security controls (e.g., authentication/access control protections) to prevent unauthorised access/download of the firmware images. Hence, anyone with access to the IP address/URL/port numbers of the repository servers will be easily able to download the update files from the server.

The new regulations (see Table C1 in Annex 5 of the UNECE WP.29 [2]) concerning mitigations for protecting backend systems recommends the introduction/implementation of appropriate security controls to minimise unauthorised access to the backend systems, for examples.

Obviously this is merely a high-level suggestion without any details about the specific solutions that can effectively be deployed to provide protection against such threats. One effective approach to address this would be restricting downloading capability to authenticated devices/clients only; this can practically be achieved by introducing authentication/authorization mechanisms, wherein a client device must be able to present valid credentials (e.g., a unique identifier/serial number along with an associated cryptographic key or digital certificate) before it should be allowed by the Uptane repositories to download/access the requested resource. This is shown graphically by the attack defence tree in the Figure 8.1, with the countermeasures included. This mitigation would need implementation of the authentication/access control functionality at both Uptane backend servers (i.e., Director repository and Image repository) and compliant clients in the vehicle.

Established best practices and guidelines (e.g., using adequate key lengths, appropriate encryption algorithms etc.) should be followed to protect against brute force attacks.

There are a number of other key factors that must be considered thoroughly, as security of the cryptographic materials stored on the device and key management would be additional challenges to deal with. Importantly, compromised keys can easily thwart such protective measures. Additionally, as ECUs (i.e., client devices) are embedded systems, they typically have limited computing and storage capabilities than required for cryptographic systems. This means more powerful ECUs (ideally with access to hardware security modules or secure hardware extension) would be needed for the security of cryptographic materials.

8.1.2 Preventing Data Flow Sniffing

Eavesdropping or interception of the sensitive information is another successful attack that demonstrated (see Table 7.5 in Chapter 7) how plain-text information exchange between Uptane repositories (i.e., Image repository and Director repository) and clients can be intercepted by the attackers that may facilitate them in their future efforts to compromise the security of the system. The intercepted information includes remote procedure calls, update data, vehicle identification number (VIN), and other similar pieces of data that can potentially be used by the attackers to craft and launch more advanced attacks on the system.

High-level mitigation relevant to this threat is outlined in the Table B1 of Annex 5

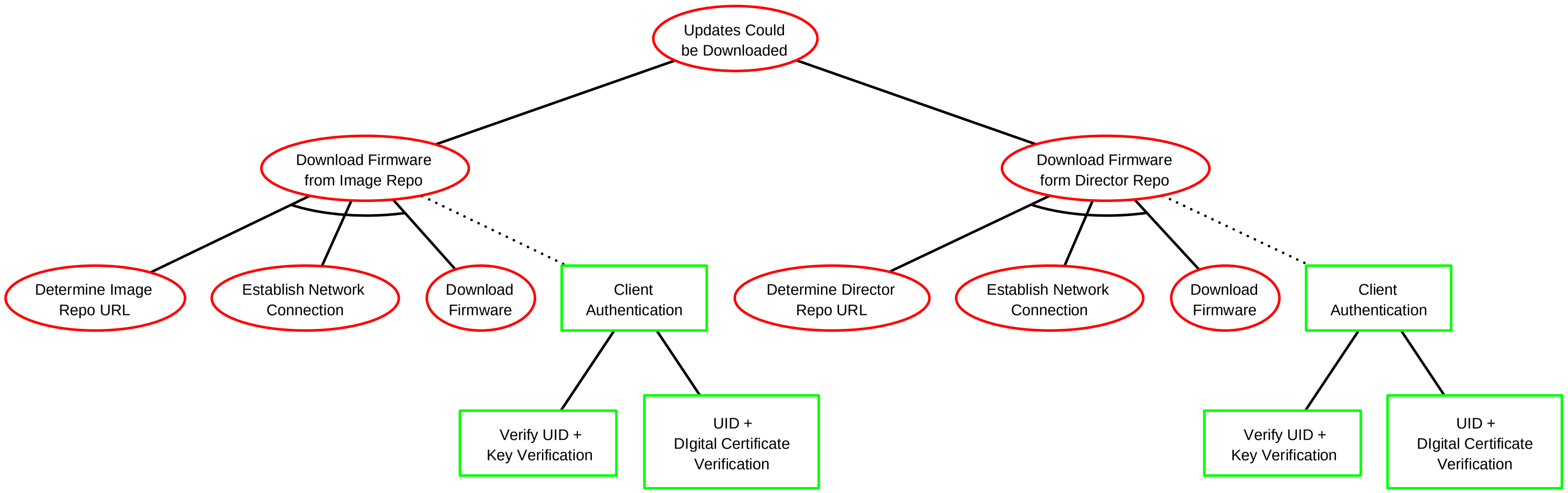

Updates Could be Downloaded
Download Firmware from Image Repo
Download Firmware form Director Repo
Determine Image Repo URL
Establish Network Connection
Download Firmware
Client Authentication
Determine Director Repo URL
Establish Network Connection
Download Firmware
Client Authentication
Verify UID + Key Verification
UID + DIgital Certificate Verification
Verify UID + Key Verification
UID + DIgital Certificate Verification

Figure 8.1: *Attack defence tree depicting the countermeasure to mitigate the threat.*

of the UNECE WP.29 ([2]), which suggests that the protection of the confidential data transmitted to/from the vehicle shall be ensured. In order to guarantee the confidentiality of such sensitive information (or the communication channel), employing widely used technologies, such as Transport Layer Security (TLS) can be one of the effective solutions. TLS is a cryptographic protocol that uses encryption to ensure privacy and integrity of the data exchange between communication computing devices. However, earlier versions of the TLS are known to have various vulnerabilities; therefore, TLS 1.3 is highly recommended, as it has significant improvements overcoming shortcomings of the previous versions. All the public data flows (e.g., publicly exposed APIs by Director and Image repositories) should be protected by this mechanism.

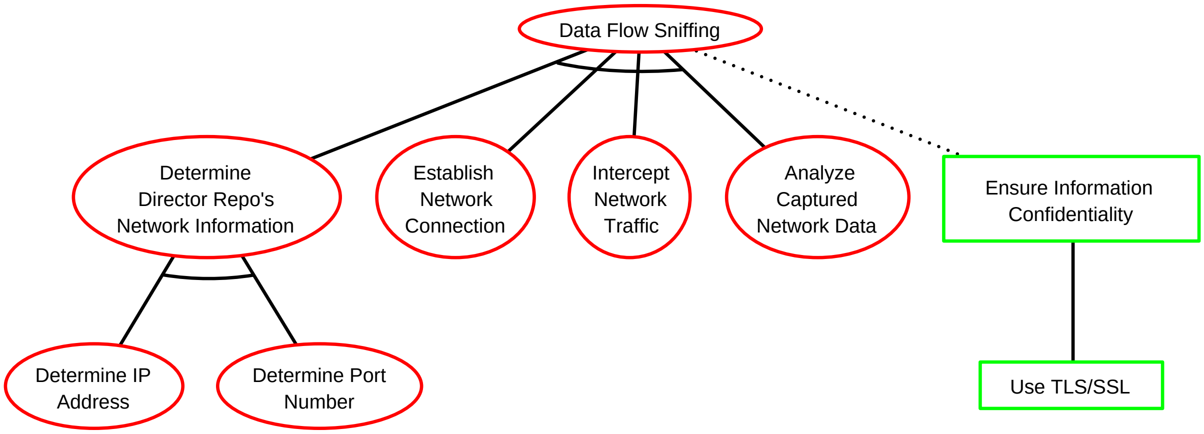

Data Flow Sniffing
Determine Director Repo's Network Information
Establish Network Connection
Intercept Network Traffic
Analyze Captured Network Data
Ensure Information Confidentiality
Determine IP Address
Determine Port Number
Use TLS/SSL

Figure 8.2: *Attack defence tree depicting the countermeasure for ensuring confidentiality of the information.*

8.2 Strategies for Dealing with Denial of Service Attacks

Denial of service attacks pose a serious threat to the system security by causing the system to stop serving the clients as a result of various kinds of resource depletion. More than half of the successful threats involved DoS attacks on different components both at server side and at the vehicle end. This section presents some of the common approaches to dealing with denial of service attacks on both ends of the OTA update ecosystem.

8.2.1 Mitigations against DoS Attacks on OTA Backend Servers

As explained later in the section, a multi-layered approach would be required to secure the Uptane backend systems against DoS attacks. UNECE WP.29 (see Table B2 in [2]) recommends effective preventive and recovery measures against system outage events caused by malicious DoS attacks in order to ensure continuous availability/delivery of critical services. However, WP.29 does not suggest specific strategies, techniques that should be used. Since there are numerous types of DoS attacks targeting different aspects of the system (and are categorised into different groups such as application level, network level, data level, and operating system level) [126]; hence, there is no single, universal strategy, technique, or approach that can be effectively leveraged to coping with this threat. It is worth noting here that (as noted in [127]) DoS attacks can either adopt a resource corruption/destruction strategy or a resource exhaustion approach, the experimental DoS attacks in this thesis have employed the latter approach. In particular, we have used brute-force based DoS attacks which involve overwhelming the victim system by flooding it with fake network messages (more details on this can be found in [127]). Application-level vulnerabilities that could be exploited to cause the Uptane repositories/components to crash (e.g., due to no/poor input validation or buffer overflow), should be fixed early in the development phase. In what follows, we outline some common strategies (as outlined in [128] and depicted in the attack defence trees presented in Figures figures 8.3 to 8.5) that can be adopted for protection against the brute-force DoS attacks performed against the Uptane repositories and client in this thesis.

Intrusion Detection and Prevention Systems Since the Uptane repositories (i.e., Image repository, Director repository, and Time Server) expose public interfaces, an intrusion detection and/or prevention system can be the first line of defence against DoS (and other types of attacks) from the external world. The process of monitoring the events taking place in a computer system or network environment for signs of malicious activity (e.g., large number of connection attempts) is commonly referred to as intrusion detection. While an intrusion detection system automates the process of intrusion detection, an intrusion protection system is additionally capable of intervening to protect the system from possible incidents [129]. In order to take appropriate actions against DoS and other types of attacks and to limit the extent of damage they can cause, automated intrusion detection systems can be an effective approach. Intrusion prevention systems can respond to potential threats/attacks by blocking, limiting, and/or filtering the network traffic as required [130] to secure Uptane repositories.

Mitigations provided by Internet and cloud service providers In many cases, service providers (with varying capabilities) can assist in dealing with resource exhaustion (i.e., DoS attacks) by offering mitigating measures that can ensure security of Uptane repositories against DoS attacks. For instance, packet scrubbing (a packet filtering service) can be used to filter packets that seemingly intended to cause a DoS attack. Some mitigations deal with attacks at higher levels of the network. Some service providers may also implement basic packet filtering in order to prevent network firewalls being overwhelmed by inbound access requests. It is worth noting that detecting unusual activity may be a challenging and non-trivial task in situations wherein

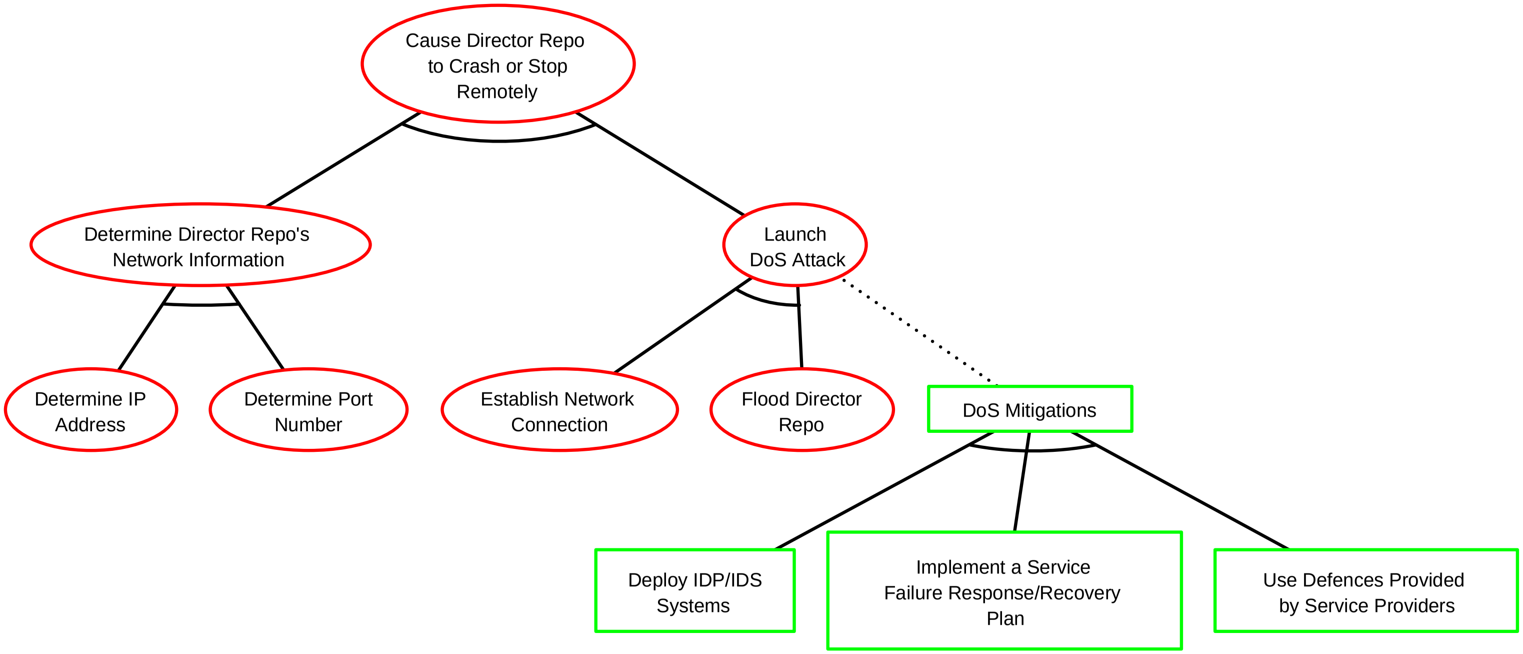

Cause Director Repo to Crash or Stop Remotely
Determine Director Repo's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Director Repo
DoS Mitigations
Deploy IDP/IDS Systems
Implement a Service Failure Response/Recovery Plan
Use Defences Provided by Service Providers

Figure 8.3: *Attack defence tree depicting the countermeasures for ensuring the availability of critical services provided by Director Repository.*

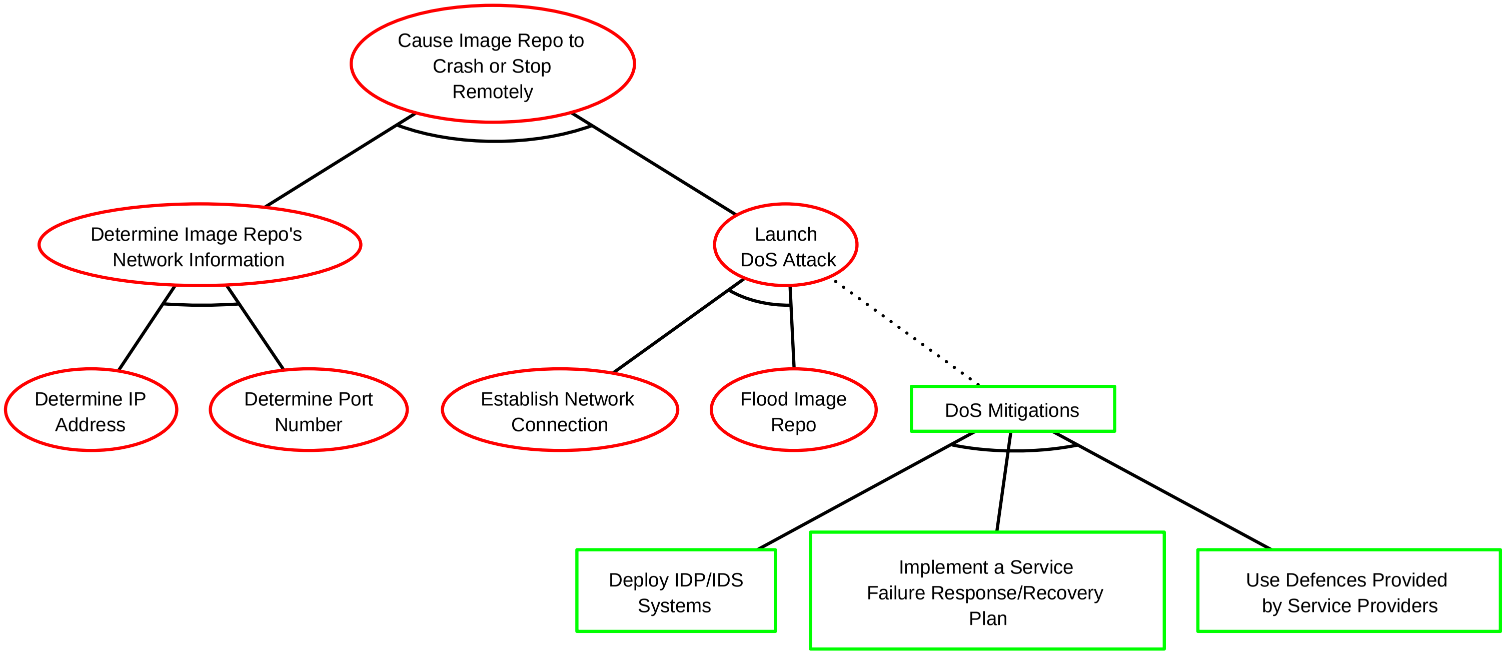

Cause Image Repo to Crash or Stop Remotely
Determine Image Repo's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Image Repo
DoS Mitigations
Deploy IDP/IDS Systems
Implement a Service Failure Response/Recovery Plan
Use Defences Provided by Service Providers

Figure 8.4: *Attack defence tree depicting the countermeasures for ensuring the availability of critical services provided by Image Repository.*

encrypted protocols (such as TLS or HTTPS) are being used. Other important considerations include the following:

- Will the service provider automatically enable the DoS mitigations, whether they will inform you when doing so?
- Time required for restoring/re-enabling the service if interrupted or shutdown,
- Time required to enable the additional mitigations.

Finally, availability of the Uptane services may be improved by using multiple service providers. This can be useful in providing uninterrupted delivery of essential services (by responding to requests from legitimate Uptane clients) when one service provider is under attack. However, this introduces additional cost and management complexities. Moreover, this solution may not be effective if the second provider is the reseller of the first one.

Implementing a service failure response and recovery plan In order to combat DoS attacks against Uptane repositories effectively, it is vital to define a robust response plan, specifying all necessary actions to be taken along with specific responsibilities of each team member should an attack is detected. National Cyber Security Centre (NCSC) recommends the following key elements of a DoS response plan [131], as summarized in the Table 8.1.

8.2.2 Mitigations against DoS Attacks on In-Vehicle Components

Even though DoS attacks on PTA backend servers and in-vehicle components along with the relevant countermeasures may have commonalities, there are important differences, requiring special considerations. Importantly, in contrast to standard ICT systems, in-vehicle components (IVCs) and in-vehicle networks (IVNs) have computing and storage constraints along with strict real-time requirements due to their very safety-critical nature, involving cyber-physical components.

As can be seen from the Figure 8.6, IDS and IPS systems have been included as recommended countermeasures against DoS attacks on the Primary ECU. However, IDS and IPS systems for resource-constrained IVCs and IVNs must take into account the special consideration and limitations of such embedded systems. There are various types of IDS systems with different characteristics, strengths, and limitations. Below is a brief overview of two broad categories of intrusion detection systems:

Anomaly-based IDS Anomaly-based intrusion detection systems use a combination of machine-learning, statistical, physical finger-printing, and rule-based methods for detecting attacks. During the training phase, a learning model is developed that helps

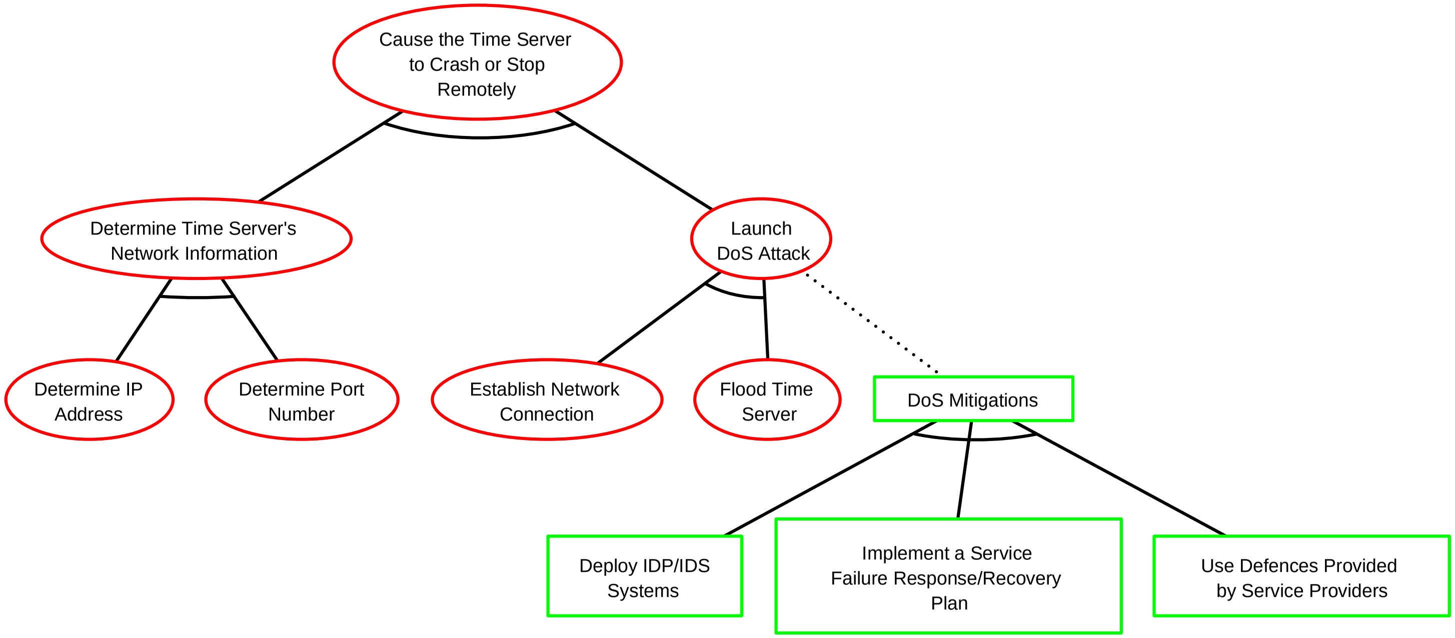

Cause the Time Server to Crash or Stop Remotely
Determine Time Server's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Time Server
DoS Mitigations
Deploy IDP/IDS Systems
Implement a Service Failure Response/Recovery Plan
Use Defences Provided by Service Providers

Figure 8.5: *Attack defence tree depicting the countermeasures for ensuring the availability of services provided by Time Server.*

Table 8.1: *Service failure response and recovery plan.*

Approach	**Description**
Grace degradation	• Access should be prioritized based on its source; for instance, restricting access to IP addresses from a certain country/region • Compute/database-intensive dynamic content generation should be disabled • Only authenticated users should be allowed to access dynamically generated contents
Tackling changing attack tactics and recurring attacks	Attackers may attempt to overwhelm the system by using different attack methods after learning more about the mitigations in place. Resources should be wisely and efficiently utilised in order to deal with multiple, repeating attacks attempts effectively.
Retaining admin access	A different network/subnet should be used to keep management access to deal with situations when under a DoS attack. Public DNS zones that are likely to be a target of the DoS attacks should not be relied upon for this purpose.
Ensuring a scalable fall-back plan	Service(s) should be able to rapidly scale in order to effectively deal with surges in sessions taking place concurrently. While it is easy to achieve horizontal scaling with native cloud applications, some re-engineering efforts may be required for pre-existing, non-cloud applications. Virtualization could be useful for achieving automated scaling in privately owned data centres, assuming adequate hardware capacity exists to accommodate this capability.

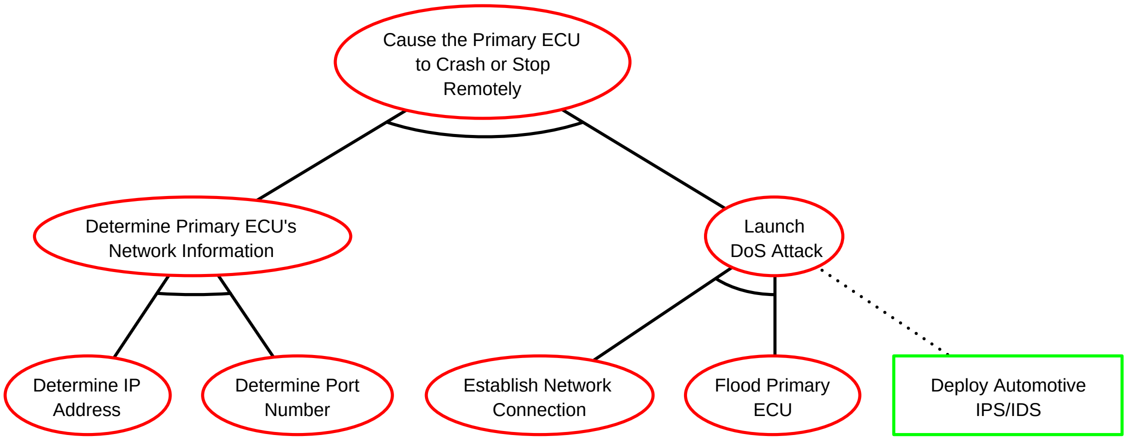

Cause the Primary ECU to Crash or Stop Remotely
Determine Primary ECU's Network Information
Launch DoS Attack
Determine IP Address
Determine Port Number
Establish Network Connection
Flood Primary ECU
Deploy Automotive IPS/IDS

Figure 8.6: *Attack defence tree depicting the recommended countermeasure for ensuring the availability of functionality provided by Primary ECU.*

Automotive security architectures typically include intrusion detection/prevention systems and firewalls to provide protection against cyberattacks and intrusions. Furthermore, network segmentation is also a widely used approach that splits CAN networks into multiple subnetworks to limit the proliferation of cyberattacks by placing safety-critical components (such as braking and steering) on a separate subnetwork.

The Primary ECU (or OTA master) in this study has been assumed to be a telematics control unit (TCU), which is not a safety-critical system; while a remote DoS attack on this system can disrupt the update process for the ECUs in the affected car, the DoS attack would not have an impact on the Uptane repositories and other vehicles. Such an attack is unlikely to be a lucrative venture for the adversary due to its limited scope and damage capability.

8.3 Chapter Summary

In line with the guidelines from UNECE WP.29, this chapter presented countermeasures for securing Uptane repositories and clients against information disclosure and denial of service attacks. For protection against information disclosure threats, mechanisms such as authentication, access control and appropriate encryption (e.g., TLS) have been suggested. The resources stored on Image and Director repositories should be restricted to authenticated clients only, which can be enforced using digital certificates and unique identifiers of the ECUs. Publicly exposed API interfaces by Director and Image repositories should implement TLS to ensure the confidentiality of the information exchange between the repositories and client ECUs. DoS attacks on the Uptane repository can be defended by deploying intrusion detection and prevention systems, implementing a comprehensive service failure response and recovery plan, and using mitigations offered by service providers. Even though intrusion prevention/detection systems can be effective mitigations at the vehicle end, being resource-constrained devices,IVCs need additional special considerations. While the suggested countermeasures/mitigations in this chapter are believed to be a good starting point, they do not represent comprehensive solutions.

Chapter 9

Conclusion and Future Work

This concluding chapter summarizes the entire study by highlighting key points, including the application of the systematic, model-based security testing approach, demonstrated by an in-depth security analysis of the Uptane reference implementation, the resultant major contributions and findings, as well as the directions for future work. In particular, we employed a MBST approach, it primarily concentrates on the specification and derivation of security test cases by using explicit models of the system under test.

9.1 Summary of the Study

The Uptane framework is being adopted by many major OEMs for the delivery of all types of software and firmware updates to the in-vehicle components found in the connected cars. These modern cars host sophisticated computing systems running software applications with millions of lines of code, requiring frequent and regular updates for functional enhancements, maintenance, and security fixes. Thus, remotely-delivered updates and associated procedures must be secure, as malicious or compromised updates can undermine the security and safety of the vehicle and its occupants. Most importantly, taking into consideration its potential future widespread adoption (affecting millions of cars), in-depth security analysis of this solution is crucial. In order to provide a comprehensive security testing/analysis, this study has showcased the application of a MBST, systematic threat assessment and security testing approach for automotive OTA update system (using the reference implementation of the Uptane Framework).

Our testing approach included systematic threat enumeration of major threats to the OTA update system by using the standard threat classification system STRIDE and associated tool called Threat Modeling Tool. Chapter 4, presents a complete description of the approach involving systematic asset identification (i.e., system decomposition) and producing a suitable representation of the system (i.e., system description) for the identification of associated security threats (i.e., threat enumeration).

Each of the identified threats forms the basis for constructing an attack tree. Since the quality of the generated test cases is largely determined by the quality of approach used (which is attack-tree based threat modeling in our case), a well-defined and structured approach could make a big difference in constructing effective attack trees, chapter 5 is dedicated to explain and demonstrate the step-by-step attack-tree construction approach with appropriate relevant examples. This approach starts with an identified threat scenario (which represents the root node of the attack tree) followed by populating the attack tree by identifying subgoals and the specific actions (depicted by leaf nodes). Information provided by the threat modeling report along with brainstorming, and discussions, the attack trees went through a well-defined refinement process.

Moreover, we presented a powerful software tool for automated test-case generation and execution, which is capable of deriving test cases by analyzing attack tree's structure and underpinning formal semantics. That is, depending on the refinement connective used (i.e., $\mathbf{OR}, \mathbf{AND}, \mathbf{SAND}$), and the complexity of the attack tree, the tool generates valid and correct test cases. In order to prove the validity of our test-case derivation approach, we provided formal proof.

In-depth security analysis and system testing of the Uptane Framework carried out in the previous chapter, demonstrating the validity of our approach by enumerating various threats by examining the system model, constructing attack trees from the results from threat enumeration, deriving effective security cases by analyzing the structure of the constructed attack trees, and finally running those test cases against the implementation. Detailed results and findings of the security evaluation presented in the previous chapter show the effectiveness of our approach by revealing unmitigated threats and vulnerabilities of the system. The experimental security attacks crafted from the attack trees, help us evaluate the security controls and mechanisms built into the framework. The findings from the experimental results of this study show that while in general the Uptane is an effective solution providing protection against numerous security threats, the reference implementation is vulnerable to information disclosure and denial-of-service threats, which must be given serious consideration in the production environment to protect the updates from cyberattacks.

9.2 Summary of the contributions of this research

This doctoral research makes several scientific and technical contributions, as summarized below:

Contribution 1 A systematic threat analysis approach for constructing attack trees (introduced in Chapter 4). This approach provides a structured, step-by-step method for constructing attack trees. Core components, assets of the system under test are identified in the *System Decomposition* step, resulting in an appropriate graphical representation of the system to be used for threat analysis in the next step. The system model is transformed into a suitable data flow model for identifying security threats in the *Threat Enumeration* step. Combined with the threat report produced in the preceding step, we show practical steps for constructing attack trees in a methodical manner. To validate the approach, we first provided a number of examples demonstrating the application of the approach in general, followed by applying it to the reference implementation of the Uptane framework. As a result of the *System Decomposition*, A UML sequence diagram of the Uptane was constructed documenting and exhibiting the behavioral aspects of the system, which was then transformed into a suitable data flow model for examining the system for identifying security threats (applying *Threat Enumeration* approach). Consequently, 52 different types of threats were identified targeting both the Uptane repositories and clients. A subset of these threats were selected from the list of threats and 15 attack trees were constructed, by applying the step-by-step approach for constructing attack trees introduced in Chapter 5.

Contribution 2 A test-derivation approach using model-based security testing approach based on attack trees (introduced in Chapter 6). We introduced this powerful approach to deriving security test cases by analyzing the structure of attack trees. We demonstrated how different refinements (i.e., disjunctive, conjunctive, and sequential conjunctive) along with different node types can assist with deriving valid security test cases. We exemplified the approach with many examples showing how the security test cases are derived using the structured approach. Additionally, we also described and explained the structure of a typical security test case derived from the attack tree. As explained below, we automate this process by implementing a software tool that generates executable security test cases by analyzing the structure of the attack tree.

An in-depth, experimental security analysis of the Uptane Framework by applying the systematic threat assessment and security testing approach (presented in Chapter 7). We applied our approach for the in-depth security analysis of the reference implementation of the Uptane Framework, involving comprehensive threat enumeration, attack-tree construction, thorough test-case generation and execution. Fifteen different experiments constituting 29 different derived test cases from 15 attack trees, helped us carry out an extensive security analysis of the Uptane framework by mounting a range of security attacks on the Director Repository, Image Repository, Time Server, and Primary ECU. Thirteen of these test cases failed out of a total 29 test cases generated and executed. Key findings and insights from these experimental attacks have been summarized providing threat- and target component-specific as well as system-wide impact of these attacks. Based on these findings, we have highlighted the strengths and weaknesses of the Uptane Framework as an OTA update delivery solution in the automotive domain. In addition to demonstrating the effectiveness and validity of our methodical testing approach, by means of this full security analysis, we discovered that the reference implementation requires effective countermeasures for ensuring the updates are protected against threats, such as disruptions to the updates and information theft in a production environment.

Contribution 4 The automation of the test case generation and execution by implementing a special-purpose software tool (presented in Chapter 6). This powerful software tool is capable of generating and executing security test cases by performing intelligent analysis of the attack-tree structure. As mentioned earlier, 29 different test cases were successfully generated and executed by the tool by processing 15 attack trees, showing the correct functioning, efficiency and effectiveness of the tool. The algorithm that we devised and used to implement the tool has been detailed in Chapter 6. Additionally, the source code has also been made available, which can be found in Appendix A.

Contribution 5 A comprehensive survey of the testbeds and testing approaches, providing a critical analysis of the testing environments and methods in the automotive cybersecurity testing (presented in Chapter 3). The testbed used for the experimental security analysis of the Uptane Framework in this study was designed by using the insightful findings from the extensive survey of a number of testing setups proposed in the relevant literature. We critically reviewed and analyzed each of the investigated testbeds by evaluating the capabilities and features offered by each of these solutions. A comparative analysis of these testing environments based on the important characteristics/factors was also carried out for determining the key considerations that must be taken into account for ensuring sound and effective design of the setup. In particular, the comparison focused on the adaptability, portability, cost, fidelity, as well as safety considerations. Moreover, the supported communication technologies, attack surfaces/vectors, and attack types reported for each of the environment have been highlighted. We also reviewed and described some of the most widely used testing methods in the domain, which include automotive vulnerability scanning, fuzz testing, and penetration testing. The wealth of information included in this comprehensive survey provides both theoretical background knowledge and practical advice for the factors and key considerations that can be highly handy in the decision-making process for designing and building safe, cost-effective, and adaptable testing environment, as to the best of our knowledge, currently there exists no such study.

9.3 Beneficiaries of this Research

The work presented in this thesis is useful for various stakeholders including both from the academia and the industry, ranging from the new entrants in the field to the experienced researchers and practitioners. In what follows, we highlight how this research work can be useful for different individuals and groups who are related to or interested in the automotive cyber security testing.

9.3.1 Research Community in Automotive Cybersecurity

This research work benefits the automotive cybersecurity testing researchers in a number of ways:

- The overall systematic threat assessment and security approach introduced in this study can be used by researchers to investigate and evaluate the security of various automotive systems in an organized and structured way; additionally, the approach can be adapted to a variety of scenarios and configurations.

- In particular, the methodical approach for constructing attack trees and the related test-case generation approach can help derive test cases relatively efficiently, saving time and manual effort, assisting the researchers with their security testing efforts.

- The software tool can be adapted and used for generating security test cases and executing them in the security assessment of different automotive systems. The tool can be extended in a number of ways to further enhance the quality of the overall testing experience and results obtained. We outline some of the possible extensions in the future work section later in this chapter.

- The comprehensive survey on automotive cybersecurity testbeds and testing methods combines the major innovative solutions proposed in the literature that can help the researchers with identifying the most effective solution for meeting particular requirements of the security evaluations. Furthermore, the survey can be used as a reference to design and set up a lab-based testing environment to conduct security testing of automotive systems/components in a safe and cost-effective manner.

- Finally, the thorough security analysis of the Uptane Framework can assist with understanding the key security threats/vulnerabilities, requirements and unique challenges of the OTA updates in the context of connected vehicles; particularly, the findings from the experimental security testing can be used for designing more effective, secure and efficient techniques and solutions for protecting the updates and other system components from cyberattacks and threats.

9.3.2 Stakeholders form the Automotive Industry

Some of the relevant parties from the industry benefiting from this research are identified below:

- OEMs seeking to adopt and implement OTA technology for the delivery of updates can find the information in this research useful. Specifically, comprehensive overview of the Uptane Framework along with the findings from its security analysis will help them make informed decision whether this particular solution meets their requirements and if it is technically and operationally feasible for them to adopt this technology.

- The systematic security testing approach, the software tool, and the testbed can be employed by the engineering and consulting companies, such as EDAG and HORIBA MIRA, to conduct security assessments of the vehicle systems.
- Finally, individual security practitioners working independently (e.g., freelancers) can also be the beneficiaries of the particular processes, tools, and techniques introduced in this thesis.

9.4 Future Work

While the work presented in this thesis has introduced several approaches techniques and tools for carrying out the systematic security testing of the automotive OTA updates (that are not strictly restricted to such systems, it can equally be effective for other automotive systems), we believe these are just the first steps towards a long journey. The presented work can be extended in various ways, some of which are outlined here.

- Firstly, we aim to carry out further evaluation of our systematic security approach (and its integrated attack-tree construction and test-case generation processes) by applying it to other similar automotive OTA solutions. Moreover, we look forward to applying our approach to other automotive systems other than OTA.
- Secondly, we plan to automate the attack-tree construction process by implementing a software tool. The tool would be capable of constructing the attack trees from the threat modeling report generated in the *Threat Enumeration* step of our approach, which is currently handled manually. This can be accomplished by implementing the template-based attack tree construction method proposed in [133].
- Finally, in order to enhance the usability and hence the productivity, we would like to implement a graphical user interface (GUI) for our test-case generation and execution software tool implemented in this study. Currently, the software tool offers a command-line user interface, which is effective and appropriate for the purpose; however, a graphical user interface will support the class of users that prefers and finds the GUI environments more convenient and productive.

9.5 Chapter summary

This final chapter provided a summary of the entire study followed by an overview of the key contributions. Stakeholders who are likely to benefit from this research from the academia and industry are identified, highlighting the usefulness for each group of beneficiaries. Finally, an overview of the future work is outlined.

References

[1] R. Shaw and B. Jackman, “An introduction to flexray as an industrial network,” in *2008 IEEE International Symposium on Industrial Electronics.* IEEE, 2008, pp. 1849–1854.

[2] Proposal for a new UN Regulation on uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system. http://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-2020-079-Revised.pdf, [Last accessed: 10-06-2020].

[3] J. Mössinger, “Software in automotive systems,” *IEEE software*, vol. 27, no. 2, pp. 92–94, 2010.

[4] D. J. Coe, J. Kulick, A. Milenkovic, and L. Etzkorn, “Virtualized in-situ software update verification: Verification of over-the-air automotive software updates,” *IEEE Vehicular Technology Magazine*, 2019.

[5] J. P. Trovao, “An overview of automotive electronics [automotive electronics],” *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 130–137, 2019.

[6] “GM’s total recall cost: $4.1 billion,” https://money.cnn.com/2015/02/04/news/companies/gm-earnings-recall-costs/index.html, [Last accessed: 20-06-2020].

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

[8] K. Munro, “Deconstructing flame: the limitations of traditional defences,” *Computer Fraud & Security*, vol. 2012, no. 10, pp. 8–11, 2012.

[9] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, “A look in the mirror: Attacks on package managers,” in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 565–574.

[10] P. Ruissen and R. Vloothuis, "Insecurities within automatic update systems v1. 16," 2007.

[11] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy, "A security analysis of an in-vehicle infotainment and app platform," in *10th {USENIX} Workshop on Offensive Technologies ({WOOT} 16)*, 2016.

[12] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[13] S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, and T. Engel, "A car hacking experiment: When connectivity meets vulnerability," in *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6.

[14] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, p. 94, 2014.

[15] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4. San Francisco, 2011, pp. 447–462.

[16] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[17] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical dos attacks on embedded networks in commercial vehicles," in *International Conference on Information Systems Security*. Springer, 2016, pp. 23–42.

[18] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent transportation systems*, vol. 16, no. 2, pp. 546–556, 2014.

[19] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2008, pp. 235–248.

[20] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.

[21] O. Avatefipour and H. Malik, "State-of-the-art survey on in-vehicle network communication (can-bus) security and vulnerabilities," *arXiv preprint arXiv:1802.01725*, 2018.

[22] J. Howden, L. Maglaras, and M. A. Ferrag, "The security aspects of automotive over-the-air updates," *International Journal of Cyber Warfare and Terrorism (IJCWT)*, vol. 10, no. 2, pp. 64–81, 2020.

[23] N. Dejon, D. Caputo, L. Verderame, A. Armando, and A. Merlo, "Automated security analysis of iot software updates," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2019, pp. 223–239.

[24] S. Halder, A. Ghosal, and M. Conti, "Secure over-the-air software updates in connected vehicles: A survey," *Computer Networks*, p. 107343, 2020.

[25] M. Khurram, H. Kumar, A. Chandak, V. Sarwade, N. Arora, and T. Quach, "Enhancing connected car adoption: Security and over the air update framework," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 2016, pp. 194–198.

[26] M. Cheah, S. A. Shaikh, J. Bryans, and P. Wooderson, "Building an automotive security assurance case using systematic security evaluations," vol. Computers & Security. 77, pp. 360–379.

[27] "Know the term: Sota/fota," https://www.business.att.com/learn/tech-advice/know-the-term--sota-fota.html, [Last accessed: 20-06-2020].

[28] T. Chowdhury, E. Lesiuta, K. Rikley, C.-W. Lin, E. Kang, B. Kim, S. Shiraishi, M. Lawford, and A. Wassyng, "Safe and secure automotive over-the-air updates," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 172–187.

[29] H. Dakroub and R. Cadena, "Analysis of software update in connected vehicles," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 7, no. 2014-01-0256, pp. 411–417, 2014.

[30] "How automakers will save $35 billion by 2022," https://fortune.com/2015/09/04/ihs-auto-software, [Last accessed: 15-06-2020].

[31] "Software-over-the-air (sota): An automotive accelerator," https://www.bearingpoint.com/files/BEI008-07-ICL_SOTA-Software-over-the-air.pdf, [Last accessed: 25-06-2020].

[32] "Hackers trick thousands into downloading dangerous 'google chrome update'," https://www.forbes.com/sites/daveywinder/2020/03/26/warning-hackers-trick-thousands-into-downloading-dangerous-google-chrome-update, [Last accessed: 21-06-2020].

[33] "Hackers hijacked asus software updates to install backdoors on thousands of computers," https://www.vice.com/en/article/pan9wn/hackers-hijacked-

asus-software-updates-to-install-backdoors-on-thousands-of-computers, [Last accessed: 21-06-2020].

[34] "Extortionist continues to scan for exposed git creds," https://www.itnews.com.au/news/extortionist-continues-to-scan-for-exposed-git-creds-525178, [Last accessed: 25-06-2020].

[35] "Attackers sign malware using crypto certificate stolen from opera software," https://arstechnica.com/information-technology/2013/06/attackers-sign-malware-using-crypto-certificate-stolen-from-opera-software, [Last accessed: 25-06-2020].

[36] "Debian investigation report after server compromises," https://www.debian.org/News/2003/20031202, [Last accessed: 25-06-2020].

[37] "Cybersecurity in automotive: Mastering the challenge," https://www.gsaglobal.org/wp-content/uploads/2020/03/Cybersecurity-in-automotive-Mastering-the-challenge.pdf, [Last accessed:06-12-2019].

[38] T. K. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing software updates for automobiles," *14th ESCAR Europe*, 2016.

[39] Uptane Alliance, "Ieee-isto 6100.1.0.0 uptane standard for design and implementation," n.d, https://uptane.github.io/papers/ieee-isto-6100.1.0.0.uptane-standard.html, [Last accessed: 06-12-2019].

[40] S. Halder, A. Ghosal, and M. Conti, "Secure ota software updates in connected vehicles: A survey," *arXiv preprint arXiv:1904.00685*, 2019.

[41] K. Mansour, W. Farag, and M. ElHelw, "Airodiag: A sophisticated tool that diagnoses and updates vehicles software over air," in *2012 IEEE International Electric Vehicle Conference*. IEEE, 2012, pp. 1–7.

[42] K. Mayilsamy, N. Ramachandran, and V. S. Raj, "An integrated approach for data security in vehicle diagnostics over internet protocol and software update over the air," *Computers & Electrical Engineering*, vol. 71, pp. 578–593, 2018.

[43] M. Steger, A. Dorri, S. S. Kanhere, K. Römer, R. Jurdak, and M. Karner, "Secure wireless automotive software updates using blockchains: A proof of concept," in *Advanced Microsystems for Automotive Applications 2017*. Springer, 2018, pp. 137–149.

[44] S. C. HPL, "Introduction to the controller area network (can)," *Application Report SLOA101*, pp. 1–17, 2002.

[45] A. Hafeez, H. Malik, O. Avatefipour, P. R. Rongali, and S. Zehra, "Comparative study of can-bus and flexray protocols for in-vehicle communication," SAE Tech. Paper, Tech. Rep., 2017.

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

[47] M. H. Eiza and Q. Ni, "Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 45–51, 2017.

[48] B. Schneier, "Modeling security threats," *Dr. Dobb's journal*, vol. 24, no. 12, 1999.

[49] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, "Attack trees with sequential conjunction," in *IFIP International Information Security and Privacy Conference.* Springer, 2015, pp. 339–353.

[50] S. Mauw and M. Oostdijk, "Foundations of attack trees." Springer Berlin Heidelberg, vol. 3935, pp. 186–198.

[51] A. Bossuat and B. Kordy, "Evil twins: handling repetitions in attack–defense trees," in *International Workshop on Graphical Models for Security.* Springer, 2017, pp. 17–37.

[52] M. Audinot, S. Pinchinat, and B. Kordy, "Is my attack tree correct?" in *European Symposium on Research in Computer Security.* Springer, 2017, pp. 83–102.

[53] H. Mantel and C. W. Probst, "On the meaning and purpose of attack trees," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF).* IEEE, 2019, pp. 184–18 415.

[54] J. Bryans, H. N. Nguyen, and S. A. Shaikh, "Attack defense trees with sequential conjunction," in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE).* IEEE, 2019, pp. 247–252.

[55] A. Shostack, *Threat modeling: Designing for security.* John Wiley & Sons, 2014.

[56] Z. Ma and C. Schmittner, "Threat modeling for automotive security analysis," *Advanced Science and Technology Letters*, vol. 139, pp. 333–339, 2016.

[57] F. Den Braber, I. Hogganvik, M. S. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps—a guided tour to the coras method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[58] T. UcedaVelez and M. M. Morana, *Risk centric threat modeling.* Wiley Online Library, 2015.

[59] Y. Chen, B. Boehm, and L. Sheppard, "Value driven security threat modeling based on attack path analysis," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07).* IEEE, 2007, pp. 280a–280a.

[60] A. Shostack, "Experiences threat modeling at microsoft." *MODSEC@ MoDELS*, vol. 2008, 2008.

[61] I. Williams and X. Yuan, "Evaluating the effectiveness of microsoft threat modeling tool," in *Proceedings of the 2015 information security curriculum development conference*, 2015, pp. 1–6.

[62] A. Karahasanovic, P. Kleberger, and M. Almgren, "Adapting threat modeling methods for the automotive industry," in *Proceedings of the 15th ESCAR Conference*, 2017, pp. 1–10.

[63] J. S. Park, D. Kim, S. Hong, H. Lee, and E. Myeong, "Case study for defining security goals and requirements for automotive security parts using threat modeling," SAE Technical Paper, Tech. Rep., 2018.

[64] S. NIST, "800-30 revision 1," *Guide for Conducting Risk Assessments*, 2012.

[65] J. R. Nurse, S. Creese, and D. De Roure, "Security risk assessment in internet of things systems," *IT Professional*, vol. 19, no. 5, pp. 20–26, 2017.

[66] OWASP, "OWASP Risk Assessment Calculator ," https://www.security-net.biz/files/owaspriskcalc.html, [Last accessed: 05-25-2021].

[67] J. Williams, "Owasp risk rating methodology," https://owasp.org/www-community/OWASP_Risk_Rating_Methodology, [Last accessed: 10-06-2021].

[68] A. Y. Putra, "Introduction implementation of OWASP Risk Rating Management ," https://slideplayer.com/slide/12574283/, [Last accessed: 05-25-2021].

[69] "Test cases and test suites," https://www.ibm.com/support/knowledgecenter/SSYMRC_7.0.1/com.ibm.rational.test.qm.doc/topics/c_testcase_overview.html [Accessed: 10-12-2019].

[70] I. Schieferdecker, J. Grossmann, and M. Schneider, "Model-based security testing," *arXiv preprint arXiv:1202.6118*, 2012.

[71] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, "Model-based security testing: a taxonomy and systematic classification," *Software Testing, Verification and Reliability*, vol. 26, no. 2, pp. 119–148, 2016.

[72] E. d. Santos, A. Simpson, and D. Schoop, "A formal model to facilitate security testing in modern automotive systems," *arXiv preprint arXiv:1805.05520*, 2018.

[73] A. Wasicek, P. Derler, and E. A. Lee, "Aspect-oriented modeling of attacks in automotive cyber-physical systems," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[74] T. Elrad, M. Aksit, G. Kiczales, K. Lieberherr, and H. Ossher, "Discussing aspects of aop," *Communications of the ACM*, vol. 44, no. 10, pp. 33–38, 2001.

[75] C. Chavez and C. Lucena, "A metamodel for aspect-oriented modeling," in *Workshop on Aspect-Oriented Modeling with UML (AOSD-2002)*, 2002.

[76] P. H. Nguyen, S. Ali, and T. Yue, "Model-based security engineering for cyber-physical systems: A systematic mapping study," *Information and Software Technology*, vol. 83, pp. 116–135, 2017.

[77] C. Smith, *The car hacker's handbook: a guide for the penetration tester*. no starch press, 2016.

[78] R. Buttigieg, M. Farrugia, and C. Meli, "Security issues in controller area networks in automobiles," in *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. IEEE, 2017, pp. 93–98.

[79] C. Riggs, C.-E. Rigaud, R. Beard, T. Douglas, and K. Elish, "A survey on connected vehicles vulnerabilities and countermeasures," *Journal of Traffic and Logistics Eng. Vol*, vol. 6, no. 1, 2018.

[80] S. Rizvi, J. Willet, D. Perino, S. Marasco, and C. Condo, "A threat to vehicular cyber security and the urgency for correction," *Procedia computer science*, vol. 114, pp. 100–105, 2017.

[81] S.-H. Chen and C.-H. R. Lin, "Evaluation of dos attacks on vehicle can bus system," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Springer, 2018, pp. 308–314.

[82] D. Klinedinst and C. King, "On board diagnostics: Risks and vulnerabilities of the connected vehicle," *Software Engineering Institute-Carnegie Mellon University*, vol. 10, 2016.

[83] M. Bozdal, M. Samie, and I. Jennions, "A survey on can bus protocol: Attacks, challenges, and potential solutions," in *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE, 2018, pp. 201–205.

[84] A. Bretting and M. Ha, "Vehicle control unit security using open source autosar," Master's thesis, 2015.

[85] H. K. Kalutarage, M. O. Al-Kadri, M. Cheah, and G. Madzudzo, "Context-aware anomaly detector for monitoring cyber attacks on automotive can bus," in *ACM Computer Science in Cars Symposium*, 2019, pp. 1–8.

[86] H.-Y. Kim, Y.-H. Choi, and T.-M. Chung, "Rees: Malicious software detection framework for meego-in vehicle infotainment," in *2012 14th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2012, pp. 434–438.

[87] T. Lin and L. Chen, "Common attacks against car infotainment systems," 2019.

[88] D. K. Nilsson and U. E. Larson, "Simulated attacks on can buses: vehicle virus," in *IASTED International conference on communication systems and networks (AsiaCSN)*, 2008, pp. 66–72.

[89] W. Yan, "A two-year survey on security challenges in automotive threat landscape," in *2015 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 2015, pp. 185–189.

[90] M. Rumez, J. Lin, T. Fuchß, R. Kriesten, and E. Sax, "Anomaly detection for automotive diagnostic applications based on n-grams," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 1423–1429.

[91] C. Miller and C. Valasek, "Car hacking: for poories," Technical report, IOActive Report, Tech. Rep., 2015.

[92] P. Borazjani, C. Everett, and D. McCoy, "Octane: An extensible open source car security testbed," in *Proceedings of the Embedded Security in Cars Conference*, 2014.

[93] J. Daily, R. Gamble, S. Moffitt, C. Raines, P. Harris, J. Miran, I. Ray, S. Mukherjee, H. Shirazi, and J. Johnson, "Towards a cyber assurance testbed for heavy vehicle electronic controls," *SAE International Journal of Commercial Vehicles*, vol. 9, no. 2016-01-8142, pp. 339–349, 2016.

[94] D. S. Fowler, M. Cheah, S. A. Shaikh, and J. Bryans, "Towards a testbed for automotive cybersecurity," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2017, pp. 540–541.

[95] X. Zheng, L. Pan, H. Chen, R. Di Pietro, and L. Batten, "A testbed for security analysis of modern vehicle systems," in *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 1090–1095.

[96] T. Toyama, T. Yoshida, H. Oguma, and T. Matsumoto, "Pasta: Portable automotive security testbed with adaptability," *Black Hat Europ, Tech. Rep.*, 2018.

[97] P. S. Oruganti, M. Appel, and Q. Ahmed, "Hardware-in-loop based automotive embedded systems cybersecurity evaluation testbed," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, 2019, pp. 41–44.

[98] C. E. Everett and D. McCoy, "Octane (open car testbed and network experiments): Bringing cyber-physical security research to researchers and students," in *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*, 2013.

[99] R. E. Haas and D. P. Möller, "Automotive connectivity, cyber attack scenarios and automotive cyber security," in *2017 IEEE International Conference on Electro Information Technology (EIT).* IEEE, 2017, pp. 635–639.

[100] J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey," *Cybersecurity*, vol. 1, no. 1, p. 6, 2018.

[101] P. Patki, A. Gotkhindikar, and S. Mane, "Intelligent fuzz testing framework for finding hidden vulnerabilities in automotive environment," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA).* IEEE, 2018, pp. 1–4.

[102] S. Bayer, T. Kreuzinger, D. Oka, and M. Wolf, "Successful security tests using fuzzing and hil test systems," 2016.

[103] D. S. Fowler, J. Bryans, S. A. Shaikh, and P. Wooderson, "Fuzz testing for automotive cyber-security," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W).* IEEE, 2018, pp. 239–246.

[104] D. S. Fowler, J. Bryans, M. Cheah, P. Wooderson, and S. A. Shaikh, "A method for constructing automotive cybersecurity tests, a can fuzz testing example," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C).* IEEE, 2019, pp. 1–8.

[105] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, "Automotive security testing—the digital crash test," in *Energy Consumption and Autonomous Driving.* Springer, 2016, pp. 13–22.

[106] E. F. M. Josephlal and S. Adepu, "Vulnerability analysis of an automotive infotainment system's wifi capability," in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE).* IEEE, 2019, pp. 241–246.

[107] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, “Technical guide to information security testing and assessment,” *NIST Special Publication*, vol. 800, no. 115, pp. 2–25, 2008.

[108] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, “Security crash test-practical security evaluations of automotive onboard it components,” *Automotive-Safety & Security 2014*, 2015.

[109] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, “Enhancement of automotive penetration testing with threat analyses results,” *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 1, no. 11-01-02-0005, pp. 91–112, 2018.

[110] PTES, “Penetration testing and execution standard,” 2014, http://www.pentest-standard.org/index.php/Main_Page, [Last accessed: 06-12-2019].

[111] M. Cheah, S. A. Shaikh, O. Haas, and A. Ruddle, “Towards a systematic security evaluation of the automotive bluetooth interface,” *Vehicular Communications*, vol. 9, pp. 8–18, 2017.

[112] S. Mahmood, A. Fouillade, H. N. Nguyen, and S. A. Shaikh, “A model-based security testing approach for automotive over-the-air updates,” in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 6–13.

[113] A. Vasenev, F. Stahl, H. Hamazaryan, Z. Ma, L. Shan, J. Kemmerich, and C. Loiseaux, “Practical security and privacy threat analysis in the automotive domain: Long term support scenario for over-the-air updates,” 2019.

[114] S. Winsen, “Threat modelling for future vehicles: on identifying and analysing threats for future autonomous and connected vehicles,” Master’s thesis, University of Twente, 2017.

[115] S. J. V. C. S. E. Committee *et al.*, “Cybersecurity guidebook for cyber-physical vehicle systems,” *SAE International*, 2016.

[116] S. Marisetty, D. Srivastava, and J. A. Hoffmann, “An architecture for in-vehicle infotainment systems,” 2010.

[117] N. Group, “NCC Group Template for the Microsoft Threat Modeling Tool 2016 for Automotive Security ,” https://github.com/nccgroup/The_Automotive_Threat_Modeling_Template, [Last accessed: 10-06-2019].

[118] A. Schaad and T. Reski, “” open weakness and vulnerability modeler”(ovvl)–an updated approach to threat modeling.”

[119] M. Cheah, H. N. Nguyen, J. Bryans, and S. A. Shaikh, "Formalising systematic security evaluations using attack trees for automotive applications." Springer International Publishing, vol. 10741, pp. 113–129.

[120] H. OMOTUNDE, R. Ibrahim, and M. Ahmed, "An optimized attack tree model for security test case planning and generation," *Journal of Theoretical and Applied Information Technology*, vol. 96, no. 17, pp. 5635–5649, 2018.

[121] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, "Security test generation using threat trees," in *2009 ICSE Workshop on automation of software test.* IEEE, 2009, pp. 62–69.

[122] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software testing, verification and reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[123] Uptane Alliance, "Uptane reference implementation code," https://uptane.github.io/papers/ieee-isto-6100.1.0.0.uptane-standard.html, [Last accessed: 06-12-2019].

[124] Y. Liu and I. Traore, "Systematic security analysis for service-oriented software architectures," in *IEEE International Conference on e-Business Engineering (ICEBE'07).* IEEE, 2007, pp. 612–621.

[125] P. Carsten, T. R. Andel, M. Yampolskiy, and J. T. McDonald, "In-vehicle networks: Attacks, vulnerabilities, and proposed solutions," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–8.

[126] A. Prakash, M. Satish, T. S. S. Bhargav, and N. Bhalaji, "Detection and mitigation of denial of service attacks using stratified architecture," *Procedia Computer Science*, vol. 87, pp. 275–280, 2016.

[127] J. Smith, "Denial of service: prevention, modelling and detection," Ph.D. dissertation, Queensland University of Technology, 2007.

[128] ENISA, "ENISA (2018) ENISA Threat Landscape Report 2018 ," https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018, [Last accessed: 06-04-2021].

[129] K. Scarfone and P. Mell, "Intrusion detection and prevention systems," in *Handbook of Information and Communication Security.* Springer, 2010, pp. 177–192.

[130] A. Householder, A. Manion, L. Pesante, G. M. Weaver, and R. Thomas, "Managing the threat of denial-of-service attacks," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2001.

[131] NCSC, "Denial of Service (DoS): Preparing for DoS Attacks ," https://www.actionfraud.police.uk/cms/wp-content/uploads/2020/02/ddos2.pdf, [Last accessed: 06-05-2021].

[132] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and countermeasures for in-vehicle networks," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–37, 2021.

[133] J. Bryans, L. S. Liew, H. N. Nguyen, G. Sabaliauskaite, S. Shaikh, and F. Zhou, "A template-based method for the generation of attack trees," in *IFIP International Conference on Information Security Theory and Practice.* Springer, 2019, pp. 155–165.

Appendices

Appendix A

Test Case Generator: Source code

The source code presented below implements the test case generation algorithm presented in Section 6.2 of Chapter 6.

```python

import xml.etree.ElementTree as ET
from itertools import combinations as comb
from itertools import permutations as permutation
from itertools import product

class AttackTree:

    def __init__(
            self,
            label,
            comment,
            node_type,
            children,
    ):
        self.label = label
        self.comment = comment
        self.node_type = node_type
        self.children = children
    def toString(self):
        lbl = '_'.join(self.label.text.split())
        if(self.comment is not None):
            return '("event_' + str(lbl) + '","' \
               + str(self.comment.text) + '")'
        else:
            comment = 'None'
            return '("event_' + str(lbl) + '","' \
                   + comment+ '")'

OR_Node = 'disjunctive'
AND_Node = 'conjunctive'
SAND_Node = 'sequential'

```

```python

# Method for deriving test cases
def parse_attack_tree(attackTree):

    label = attackTree.find('label')
    comment = attackTree.find('comment')
    node_type = attackTree.get('refinement')
    children = []

    for node in attackTree.findall('node'):
        children.append(parse_attack_tree(node))

    return AttackTree(label, comment, node_type, children)

   # Method for generating test scripts
def gen_test_case_script(at):

    if(not at.children):
        Leaf_Nodes = set()
        Leaf_Nodes.add((at.toString(),))
        return Leaf_Nodes

    elif (at.node_type==OR_Node):
        OR_Nodes = set()
        for child in at.children:
            OR_Node_Temp = gen_test_case_script(child)
            OR_Nodes = OR_Nodes.union(OR_Node_Temp)
        return OR_Nodes

    elif(at.node_type==AND_Node):
        AND_Nodes = list()
        for child in at.children:
            AND_Nodes.append(gen_test_case_script(child))
        Combined_AND_Nodes = combine_and(AND_Nodes)
        return Combined_AND_Nodes

    elif (at.node_type == SAND_Node):
        SAND_Nodes = list()
        for child in at.children:
            SAND_Nodes.append(gen_test_case_script(child))
        Combined_SAND_Nodes = combine_sand(SAND_Nodes)
        return Combined_SAND_Nodes

def gen_test_case(tcs):
    Test_Cases = list()
    for counter, seq in enumerate(tcs):
        Test_Cases.append('TC_'+str(counter+1)+' = ['+', '.join(seq)+' ]')
    return Test_Cases


```

```python

def combine_sand(SAND_tcs):

    return ([sum(seq, ()) for seq in product(*SAND_tcs)])

def combine_and(AND_tcs):
    Test_Cases = list()

    terms = ([sum(seq, ()) for seq in product(*AND_tcs)])

    for i in terms:
       Test_Cases += ([seq for seq in permutation(i)])
    return Test_Cases

# Write test scripts output to file
def gen_output_file(tree):

    test_cases  = gen_test_case_script(tree)

    test_case_scrips = gen_test_case(test_cases)

    output_script = 'TC_COUNT = ' + str(len(test_case_scrips)) + '\n'

    output_script += '\n'.join(test_case_scrips)

    test_cases = ['TC_'+ str(count+1) for count in range(len(test_case_scrips))]

    output_script += '\nTest_Cases = [' + ', '.join(test_cases) + ']'

    output_file_name =  "output.txt"

    f = open(output_file_name, 'w')

    f.write(output_script)

    f.close()
###------------- End of Program ---------------- ###
```

Link to Complete Source Code/Documentation

Please use this link to access complete source code for the Test Case Generator/Executor, Uptane Reference Implementation as well as instructions setting up and running the testing environment.
https://tinyurl.com/zdhn56pm

Appendix B

XML Source Code for Attack Trees

This appendix includes the XML source code for all the attack trees used in the experiments in Chapter 7 for deriving test cases as well as the test scripts executed against the Uptane reference implementation.

Threat 6: Updates Could be Downloaded

```

<?xml version=’1.0’?>
<sandtree>
  <node refinement="disjunctive">
    <label>Updates Could
be Downloaded</label>
    <node refinement="sequential">
      <label>Download Firmware
from Image Repo</label>
      <node refinement="conjunctive">
        <label>Determine Image
Repo Firmware
URL</label>
        <comment>determine_url</comment>
      </node>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Download
Firmware</label>
        <comment>download_firmware</comment>
      </node>
    </node>
    <node refinement="sequential">
      <label>Download Firmware
from Director Repo</label>
```

```
      <node refinement="conjunctive">
        <label>Determine Director
Repo Firmware
URL</label>
        <comment>determine_url</comment>
      </node>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Download
Firmware</label>
        <comment>download_firmware</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 7: Data Flow Sniffing - attack tree source code

```
<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Data Flow Sniffing</label>
    <node refinement="conjunctive">
      <label>Determine Director Repo&apos;s
Network Information</label>
      <node refinement="conjunctive">
        <label>Determine IP
Address</label>
        <comment>determine_IP</comment>
      </node>
      <node refinement="conjunctive">
        <label>Determine Port
Number</label>
        <comment>determine_port</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Establish Network
Connection</label>
      <comment>establih_connection</comment>
    </node>
    <node refinement="sequential">
      <label>Intercept Network
Traffic</label>
      <comment>intercept_traffic</comment>
    </node>
    <node refinement="sequential">
```

```
      <label>Analyze Captured
Network Data</label>
      <comment>analyze_network_data</comment>
    </node>
  </node>
</sandtree>
```

Threat 9: Cause Director Repo to Crash or Stop Remotely

```

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Cause Director Repo to
Crash or Stop
Remotely</label>
    <node refinement="conjunctive">
      <label>Determine Director Repo&apos;s
Network Information</label>
      <node refinement="conjunctive">
        <label>Determine IP
Address</label>
        <comment>determine_IP</comment>
      </node>
      <node refinement="conjunctive">
        <label>Determine Port
Number</label>
        <comment>determine_port</comment>
      </node>
    </node>
    <node refinement="sequential">
      <label>Launch
DoS Attack</label>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Flood Director
Repo</label>
        <comment>flood_dirctor</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 10: Cause Image Repo to Crash or Stop Remotely

```

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Cause Image Repo to
Crash or Stop
Remotely</label>
    <node refinement="conjunctive">
      <label>Determine Image Repo&apos;s
Network Information</label>
      <node refinement="conjunctive">
        <label>Determine IP
Address</label>
        <comment>determine_IP</comment>
      </node>
      <node refinement="conjunctive">
        <label>Determine Port
Number</label>
        <comment>determine_port</comment>
      </node>
    </node>
    <node refinement="sequential">
      <label>Launch
DoS Attack</label>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Flood Image
Repo</label>
        <comment>flood_imagerepo</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Theat 11: Cause Primary ECU to Crash or Stop Remotely

```

<?xml version=’1.0’?>
<sandtree>
  <node refinement="sequential">
    <label>Cause Primary ECU to
Crash or Stop
Remotely</label>
    <node refinement="conjunctive">
      <label>Determine Primary ECU&apos;s
Network Information</label>
      <node refinement="conjunctive">
        <label>Determine IP
Address</label>
        <comment>determine_ip</comment>
      </node>
      <node refinement="conjunctive">
        <label>Determine
Port Number</label>
        <comment>determine_port</comment>
      </node>
    </node>
    <node refinement="sequential">
      <label>Launch
DoS Attack</label>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Flood Pirmary
ECU</label>
        <comment>flood_primary</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 12: Cause Time Server to Crash or Stop Remotely

```
<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Cause the Time Server to
Crash or Stop
Remotely</label>
    <node refinement="conjunctive">
      <label>Determine Time Server&apos;s
Network Information</label>
      <node refinement="conjunctive">
        <label>Determine IP
Address</label>
        <comment>determine_ip</comment>
      </node>
      <node refinement="conjunctive">
        <label>Determine Port
Number</label>
        <comment>determine_port</comment>
      </node>
    </node>
    <node refinement="sequential">
      <label>Launch
DoS Attack</label>
      <node refinement="conjunctive">
        <label>Establish Network
Connection</label>
        <comment>establish_connection</comment>
      </node>
      <node refinement="conjunctive">
        <label>Flood Time
Server</label>
        <comment>flood_time_server</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 21: Compromise Director Repo to Deliver Malicious Updates

```xml
<?xml version=’1.0’?>
<sandtree>
  <node refinement="sequential">
    <label>Compromise Director Repo
to Deliver Malicious
Updates</label>
    <node refinement="sequential">
      <label>Modify Existing
Firmware Image</label>
      <node refinement="conjunctive">
        <label>Add Malicious
Contents to the Firmware
Image</label>
        <comment>add_contents</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Modified Firmware
to Director Repo</label>
        <comment>add_to_director</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Generate and Sign
the Metadata</label>
      <comment>sign_director</comment>
    </node>
  </node>
</sandtree>
```

Threat 22: Compromise Image Repo to Deliver Malicious Updates

```xml

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Compromise Image Repo
to Deliver Malicious
Updates</label>
    <node refinement="sequential">
      <label>Modify Existing
Firmware Image</label>
      <node refinement="conjunctive">
        <label>Add Malicious
Contents to Firmware</label>
        <comment>add_contents</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Modified Firmware
to Image Repo</label>
        <comment>add_to_imagerpo</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Generated and
Signed Metadata</label>
      <comment>sign_imagerepo</comment>
    </node>
  </node>
</sandtree>
```

Threat 22.2 Compromise Image and Director Repos to Deliver Malicious Updates

```

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Compromise Director and
Image Repos to
Deliver Malicious
Updates</label>
    <node refinement="sequential">
      <label>Modify Existing
Firmware Image</label>
      <node refinement="conjunctive">
        <label>Add Malicious
Contents to Firmware</label>
        <comment>add_contents</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Modified Firmware
to Image Repo</label>
        <comment>add_to_imagerepo</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Generate Image
Repo Signed
Metadata</label>
      <comment>sign_imagerepo</comment>
    </node>
    <node refinement="sequential">
      <label>Add Modified
FIrmware
to Director Repo</label>
      <comment>add_dir</comment>
    </node>
    <node refinement="sequential">
      <label>Generate
Director Repo
Signed Metadata</label>
      <comment>sign_dir</comment>
    </node>
  </node>
</sandtree>
```

Threat 26: Endless Data Attack (insert data)

```xml

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Endless Data
Attack</label>
    <node refinement="conjunctive">
      <label>Insert Additional
Contents into Firmware</label>
      <comment>add_contents</comment>
    </node>
    <node refinement="conjunctive">
      <label>Add to Modified
Image to Repos</label>
      <node refinement="conjunctive">
        <label>Add Firmware to
Image Repo</label>
        <comment>add_imagerepo</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Firmware
to Director Repo</label>
        <comment>add_director</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 26.2: Endless Data Attack (append data)

```xml

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Endless Data
Attack</label>
    <node refinement="conjunctive">
      <label>Append Additional
Contents to Firmware</label>
      <comment>add_contents</comment>
    </node>
    <node refinement="conjunctive">
      <label>Add to Modified
Image to Repos</label>
      <node refinement="conjunctive">
        <label>Add Firmware to
Image Repo</label>
        <comment>add_imagerepo</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Firmware
to Director Repo</label>
        <comment>add_director</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 28: Mix-and-Match Attack

```

<?xml version='1.0'?>
<sandtree>
  <node refinement="sequential">
    <label>Mix and Match
Attack</label>
    <node refinement="conjunctive">
      <label>Delete Currently Valid
Snapshot from Repos</label>
      <node refinement="conjunctive">
        <label>Delete Snapshot from
Image Repo</label>
        <comment>delete_snapshot_imagerpo</comment>
      </node>
      <node refinement="conjunctive">
        <label>Delete Snapshot from
Director Repo</label>
        <comment>delete_snapshot_director</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Add Invalid
Snapshot to Repos</label>
      <node refinement="conjunctive">
        <label>Add Snapshot to
Image Repo</label>
        <comment>add_snapshot_imagerpo</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add Snapshot to
Director Repo</label>
        <comment>add_snapshot_director</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Threat 27: Rollback Attack

```

<?xml version=’1.0’?>
<sandtree>
  <node refinement="sequential">
    <label>Rollback
Attack</label>
    <node refinement="conjunctive">
      <label>Delete Currently Valid
Timestap Files
from Repos</label>
      <node refinement="conjunctive">
        <label>Delete Timestamp from
Image Repo</label>
        <comment>delete_timestamp</comment>
      </node>
      <node refinement="conjunctive">
        <label>Delete Timestamp from
Director Repo</label>
        <comment>delete_timestamp</comment>
      </node>
    </node>
    <node refinement="conjunctive">
      <label>Add Outdated Timestamp
Files to Repos</label>
      <node refinement="conjunctive">
        <label>Add File to
Image Repo</label>
        <comment>add_timestamp</comment>
      </node>
      <node refinement="conjunctive">
        <label>Add File to
Director Repo</label>
        <comment>add_timestamp</comment>
      </node>
    </node>
  </node>
</sandtree>
```

Appendix C

Threat Modeling Reports

Following are three excerpted threat modeling reports generated using MS Threat Modeling Tool, as part of the *Threat Enumeration* process. They are represented in the exact form and presentation as output by the tool. These reports were used to construct attack trees and consequently the test cases. The first report is based on the Uptane Framework, whereas the other two are related to adaptive cruise control and infotainment system.

C.1 Uptane Framework Threat Modelling Report

This first threat modeling report details all the threats that are summarized in Section 7.4 of Chapter 7.

Threat Modeling Report

Created on 12/09/2020 16:42:00

Threat Model Name:

Owner:

Reviewer:

Contributors:

Description:

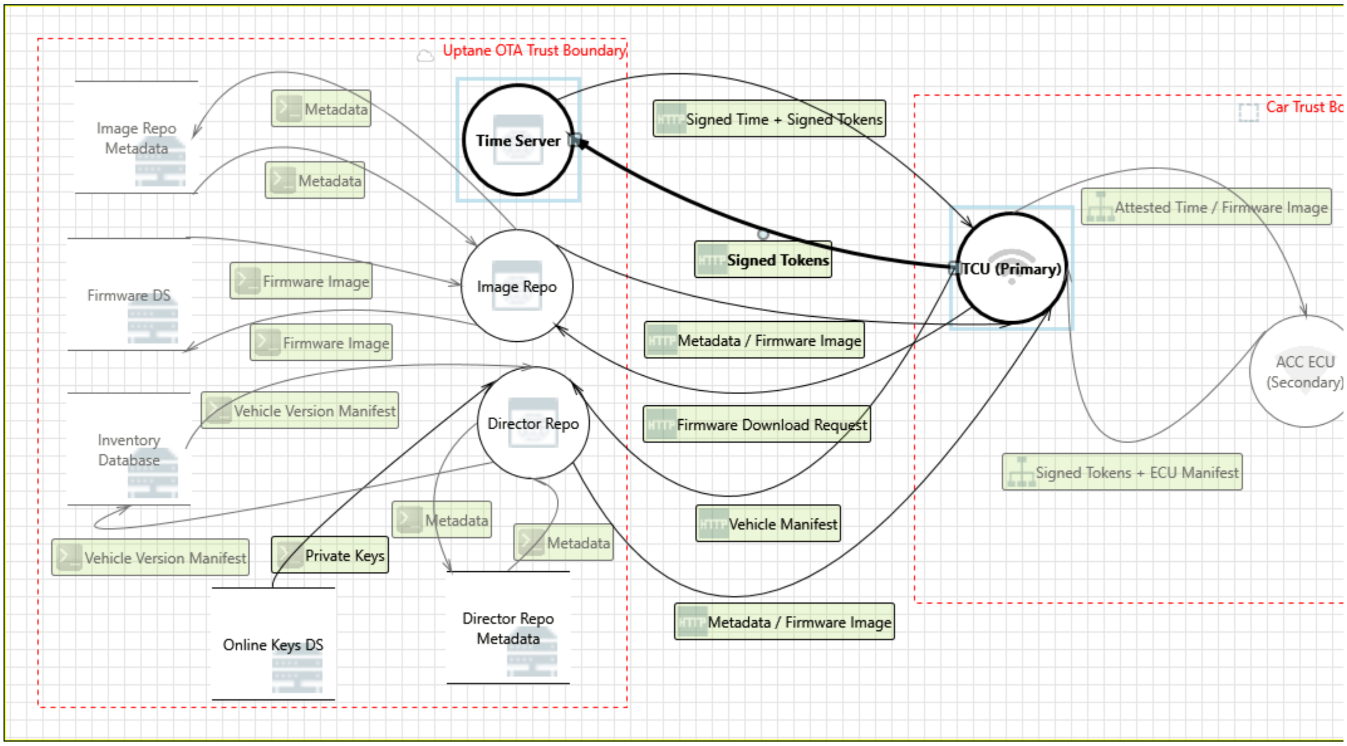
Assumptions:

External Dependencies:

Threat Model Summary:

Not Started	53
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	53
Total Migrated	0

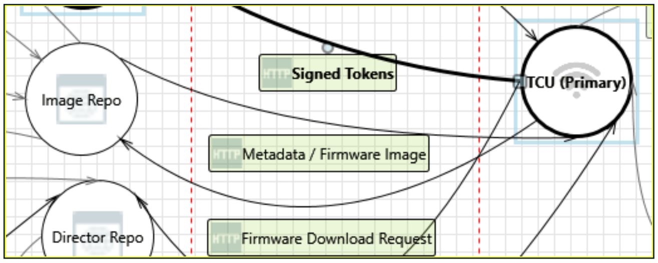
Diagram: Uptane OTA Reference Implementation



Uptane OTA Reference Implementation Diagram Summary:

Not Started	53
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	53
Total Migrated	0

Interaction: Firmware Download Request



1. Compromise the Image Repo in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to exploit the Image Repo.
Justification: <no mitigation provided>
Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation: Ensure that the server is kept up to date and perform regular security testing.

2. Take the Image Repo Offline [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Image Repo.
Justification: <no mitigation provided>
Attack method: Perform an network attack and case resource exhaustion.
Recommendation: Have a number of Image Repo delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

3. Flood Image Repo With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Image Repo by flooding with invalid data.
Justification: <no mitigation provided>
Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.
Recommendation: Rely on additional sensors in the event of one is unavailable.

4. Cause the Image Repo to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Image Repo that crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Justification: <no mitigation provided>
Attack method: Flooding Image Repo with invalid messages or data.
Recommendation: Implement data validation and shutdown communications channel to Image Repo if flooding is detected. Potentially enter in a special safety mode if Image Repo is unavalaible.

5. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Information disclosure by downloading the updates from Image Repo.
Justification: <no mitigation provided>
Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

6. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification: <no mitigation provided>
Attack method: Man-in-the-middle using attack using hardware or software.
Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

7. Image Repo Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation

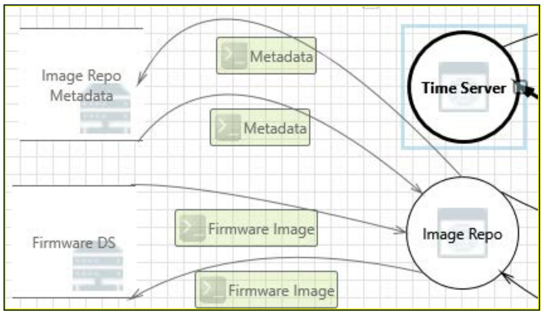
Description: Image Repo claims that it did not write data received from an entity on the other side of the trust boundary.

Justification: <no mitigation provided>

Attack method: An attacker is able to write data on Image Repo.

Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

Interaction: Metadata



8. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure

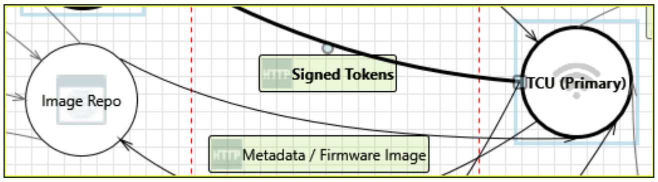
Description: Information disclosure by downloading the updates from Image Repo Matadata.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

Interaction: Metadata / Firmware Image



9. Reflash the TCU (Primary) Firmware in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the TCU (Primary) firmware.

Justification: <no mitigation provided>

Attack method: Physically connect to the target is [TCU] and attempt to reflash the chip.

Recommendation: All firmware should be encrypted and signed to prevent modification. There should be a secure boot process to prevent any invalid firmware from booted.

10. Compromise the TCU (Primary) in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the TCU (Primary).

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

11. Take the TCU (Primary) Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary).

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of TCU (Primary) delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

12. Flood TCU (Primary) With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary) by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

13. Cause the TCU (Primary) to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary) that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU (Primary) with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU (Primary) if flooding is detected. Potentially enter in a special safety mode if TCU (Primary) is unavalaible.

14. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from TCU (Primary).

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

15. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

16. Car Could be Tracked [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by performing a MITM attack in order to track a car.

Justification: <no mitigation provided>

Attack method: Downgrade or false base station attack.

Recommendation: Encrypt communications so that passive interception cannot identify specific vehicles. Rolling/unique identifiers that change over time, to prevent tracking.

17. TCU (Primary) Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation

Description: TCU (Primary) claims that it did not write data received from an entity on the other side of the trust boundary.

Justification: <no mitigation provided>

Attack method: An attacker is able to write data on TCU (Primary).

Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

18. Modify Data Being Sent to the TCU (Primary) While in Transit [State: Not Started] [Priority: High]

Category: Tampering

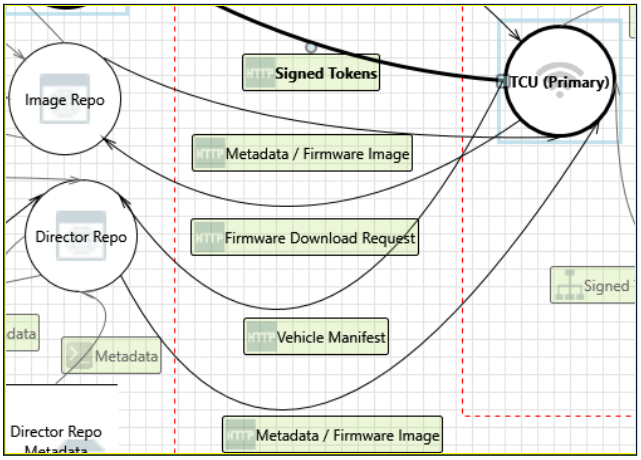
Description: Tamper with data in trasit sent to the TCU (Primary).

Justification: <no mitigation provided>

Attack method: MITM on the TCU for example a 3g to 2g downgrade attack, or false base station attack.

Recommendation: Disable 2G communications, only 3G and 4G should be allowed. Use a secure communication channel between the car and the server.

Interaction: Metadata / Firmware Image



19. Reflash the TCU (Primary) Firmware in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to reflash the TCU (Primary) firmware.
Justification: <no mitigation provided>
Attack method: Physically connect to the target is [TCU] and attempt to reflash the chip.
Recommendation: All firmware should be encrypted and signed to prevent modification. There should be a secure boot process to prevent any invalid firmware from booted.

20. Compromise the TCU (Primary) in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to exploit the TCU (Primary).
Justification: <no mitigation provided>
Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation: Ensure that the server is kept up to date and perform regular security testing.

21. Take the TCU (Primary) Offline [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on TCU (Primary).
Justification: <no mitigation provided>
Attack method: Perform an network attack and case resource exhaustion.
Recommendation: Have a number of TCU (Primary) delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

22. Flood TCU (Primary) With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on TCU (Primary) by flooding with invalid data.
Justification: <no mitigation provided>
Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.
Recommendation: Rely on additional sensors in the event of one is unavailable.

23. Cause the TCU (Primary) to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on TCU (Primary) that crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Justification: <no mitigation provided>
Attack method: Flooding TCU (Primary) with invalid messages or data.
Recommendation: Implement data validation and shutdown communications channel to TCU (Primary) if flooding is detected. Potentially enter in a special safety mode if TCU (Primary) is unavalaible.

24. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Information disclosure by downloading the updates from TCU (Primary).

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

25. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

26. Car Could be Tracked [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by performing a MITM attack in order to track a car.

Justification: <no mitigation provided>

Attack method: Downgrade or false base station attack.

Recommendation: Encrypt communications so that passive interception cannot identify specific vehicles. Rolling/unique identifiers that change over time, to prevent tracking.

27. TCU (Primary) Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation

Description: TCU (Primary) claims that it did not write data received from an entity on the other side of the trust boundary.

Justification: <no mitigation provided>

Attack method: An attacker is able to write data on TCU (Primary).

Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

28. Modify Data Being Sent to the TCU (Primary) While in Transit [State: Not Started] [Priority: High]

Category: Tampering

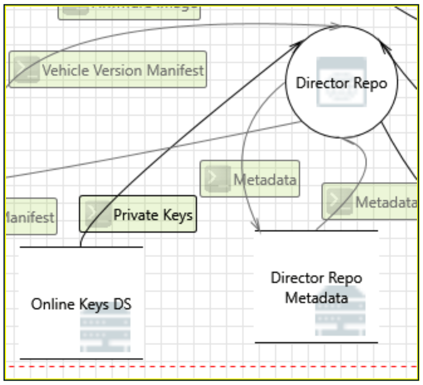
Description: Tamper with data in trasit sent to the TCU (Primary).

Justification: <no mitigation provided>

Attack method: MITM on the TCU for example a 3g to 2g downgrade attack, or false base station attack.

Recommendation: Disable 2G communications, only 3G and 4G should be allowed. Use a secure communication channel between the car and the server.

Interaction: Private Keys



29. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

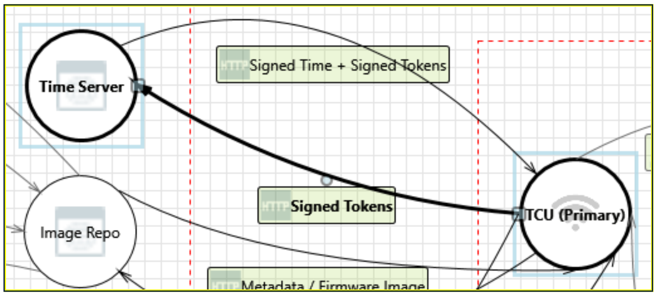
Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Interaction: Signed Time + Signed Tokens



30. Reflash the TCU (Primary) Firmware in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the TCU (Primary) firmware.

Justification: <no mitigation provided>

Attack method: Physically connect to the target is [TCU] and attempt to reflash the chip.

Recommendation: All firmware should be encrypted and signed to prevent modification. There should be a secure boot process to prevent any invalid firmware from booted.

31. Compromise the TCU (Primary) in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the TCU (Primary).

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

32. Take the TCU (Primary) Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary).

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of TCU (Primary) delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

33. Flood TCU (Primary) With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary) by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

34. Cause the TCU (Primary) to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU (Primary) that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU (Primary) with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU (Primary) if flooding is detected. Potentially enter in a special safety mode if TCU (Primary) is unavalaible.

35. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from TCU (Primary).

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

36. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

37. Car Could be Tracked [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by performing a MITM attack in order to track a car.

Justification: <no mitigation provided>

Attack method: Downgrade or false base station attack.

Recommendation: Encrypt communications so that passive interception cannot identify specific vehicles. Rolling/unique identifiers that change over time, to prevent tracking.

38. TCU (Primary) Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation

Description: TCU (Primary) claims that it did not write data received from an entity on the other side of the trust boundary.

Justification: <no mitigation provided>

Attack method: An attacker is able to write data on TCU (Primary).

Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

39. Modify Data Being Sent to the TCU (Primary) While in Transit [State: Not Started] [Priority: High]

Category: Tampering

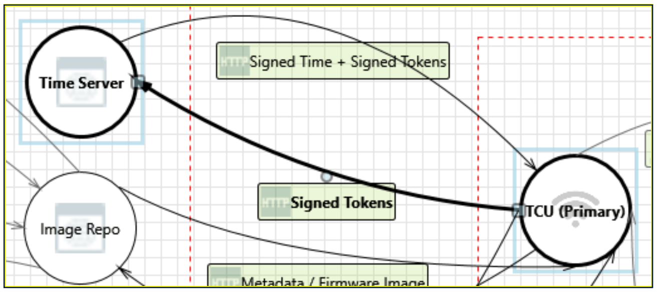
Description: Tamper with data in trasit sent to the TCU (Primary).

Justification: <no mitigation provided>

Attack method: MITM on the TCU for example a 3g to 2g downgrade attack, or false base station attack.

Recommendation: Disable 2G communications, only 3G and 4G should be allowed. Use a secure communication channel between the car and the server.

Interaction: Signed Tokens



40. Compromise the Time Server in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the Time Server.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

41. Take the Time Server Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Time Server.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Time Server delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

42. Flood Time Server With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Time Server by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

43. Cause the Time Server to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Time Server that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Time Server with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Time Server if flooding is detected. Potentially enter in a special safety mode if Time Server is unavalaible.

44. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from Time Server.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

45. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

46. Time Server Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation

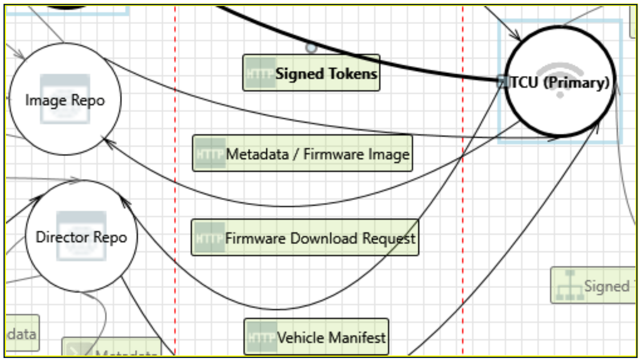
Description: Time Server claims that it did not write data received from an entity on the other side of the trust boundary.

Justification: <no mitigation provided>

Attack method: An attacker is able to write data on Time Server.

Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

Interaction: Vehicle Manifest



47. Compromise the Director Repo in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to exploit the Director Repo.
Justification: <no mitigation provided>
Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation: Ensure that the server is kept up to date and perform regular security testing.

48. Take the Director Repo Offline [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Director Repo.
Justification: <no mitigation provided>
Attack method: Perform an network attack and case resource exhaustion.
Recommendation: Have a number of Director Repo delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

49. Flood Director Repo With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Director Repo by flooding with invalid data.
Justification: <no mitigation provided>
Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.
Recommendation: Rely on additional sensors in the event of one is unavailable.

50. Cause the Director Repo to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on Director Repo that crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Justification: <no mitigation provided>
Attack method: Flooding Director Repo with invalid messages or data.
Recommendation: Implement data validation and shutdown communications channel to Director Repo if flooding is detected. Potentially enter in a special safety mode if Director Repo is unavalaible.

51. Updates Could Be Downloaded [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Information disclosure by downloading the updates from Director Repo.
Justification: <no mitigation provided>
Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

52. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification: <no mitigation provided>
Attack method: Man-in-the-middle using attack using hardware or software.
Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

53. Director Repo Denies Writing Data [State: Not Started] [Priority: High]

Category: Repudiation
Description: Director Repo claims that it did not write data received from an entity on the other side of the trust boundary.
Justification: <no mitigation provided>
Attack method: An attacker is able to write data on Director Repo.
Recommendation: Consider using logging or auditing to record the source, time, and summary of the received data.

C.2 Adaptive Cruise Control Threat Modeling Report

This threat modeling report presents the details of all the threats summarized in Section 5.2 of Chapter 5.

Threat Modeling Report

Created on 20/12/2020 22:09:15

Threat Model Name:

Owner:

Reviewer:

Contributors:

Description:

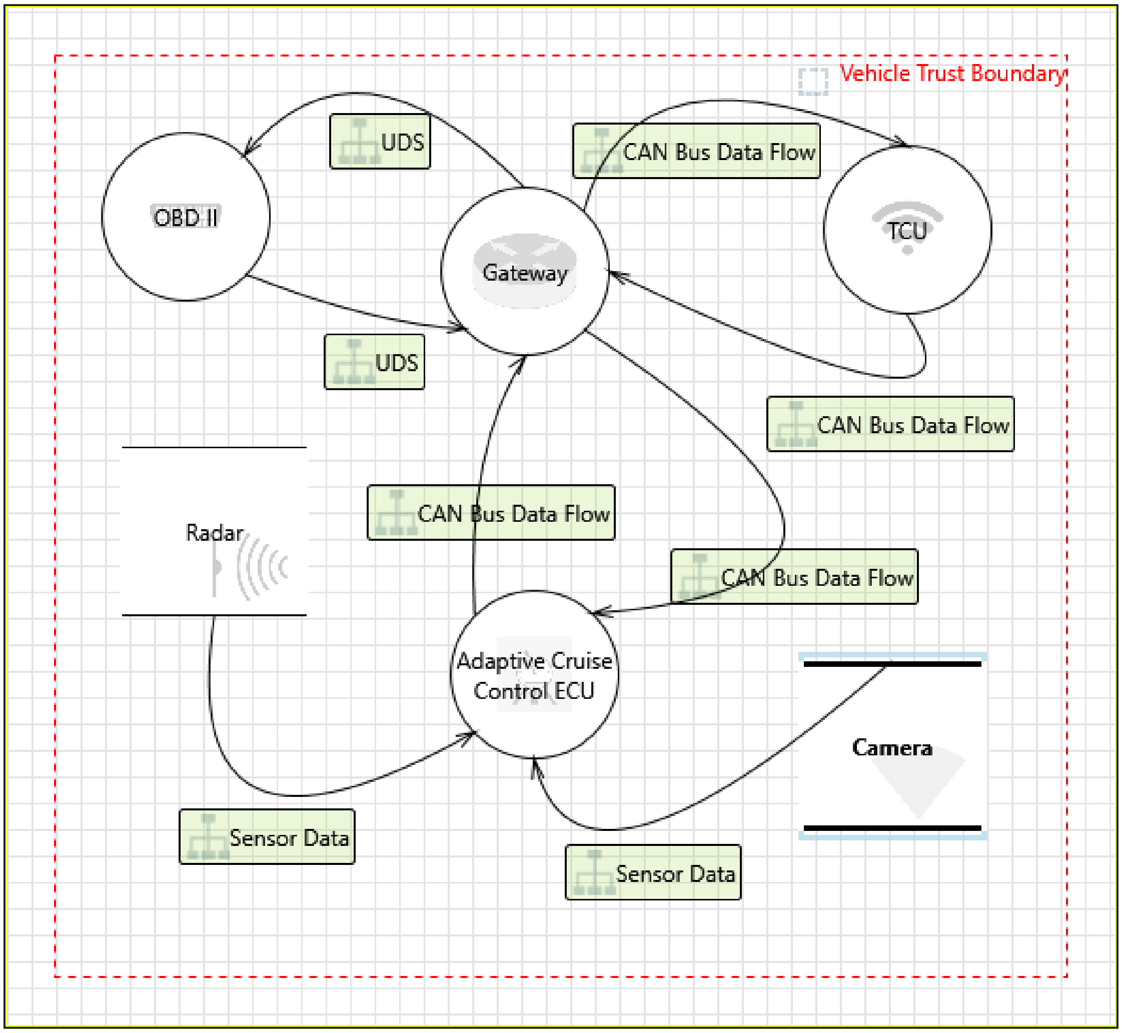
Assumptions:

External Dependencies:

Threat Model Summary:

Not Started	52
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	52
Total Migrated	0

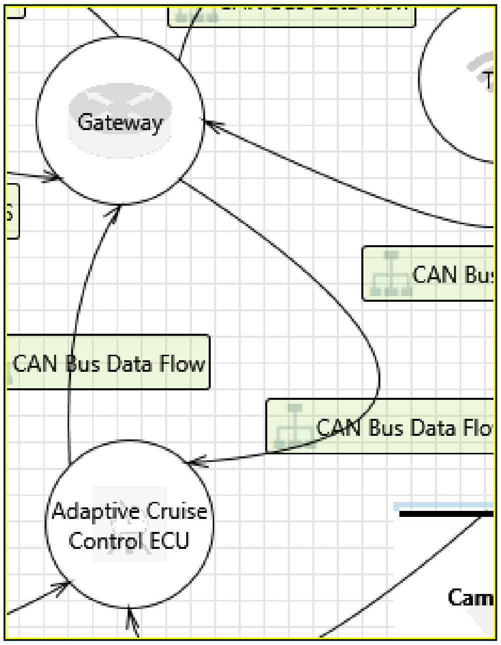
Diagram: Adapative Cruise Control



Adapative Cruise Control Diagram Summary:

Not Started	52
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	52
Total Migrated	0

Interaction: CAN Bus Data Flow



1. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.
Justification for Status Change:	

2. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Information disclosure by downloading the updates from Adaptive Cruise Control ECU.
Justification:	<no mitigation provided>
Attack method:	Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation:	Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.
Justification for Status Change:	

3. Cause the Adaptive Cruise Control ECU to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Adaptive Cruise Control ECU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Adaptive Cruise Control ECU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Adaptive Cruise Control ECU if flooding is detected. Potentially enter in a special safety mode if Adaptive Cruise Control ECU is unavalaible.

Justification for Status Change:

4. Cause the Adaptive Cruise Control ECU to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Adaptive Cruise Control ECU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Adaptive Cruise Control ECU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Adaptive Cruise Control ECU if flooding is detected. Potentially enter in a special safety mode if Adaptive Cruise Control ECU is unavalaible.

Justification for Status Change:

5. Flood Adaptive Cruise Control ECU With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Adaptive Cruise Control ECU by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

6. Take the Adaptive Cruise Control ECU Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Adaptive Cruise Control ECU.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Adaptive Cruise Control ECU delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

Justification for Status Change:

7. Compromise the Adaptive Cruise Control ECU in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the Adaptive Cruise Control ECU.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Justification for Status Change:

8. Reflash the Adaptive Cruise Control ECU From the CAN Bus in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the Adaptive Cruise Control ECU.

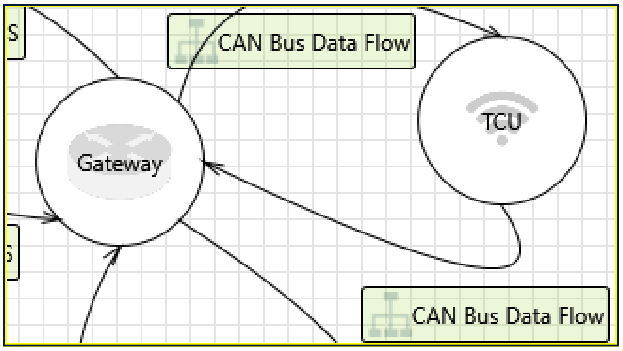
Justification: <no mitigation provided>

Attack method: As the radar units are external it would be possible to connect into the private CAN network and use diagnostic CAN messages to reflash the ACC ECU.

Recommendation: Ensure that diagnostic messages are ignored when received from the private CAN. Additionally all firmware should be signed and validated to prevent malicious updates from being flashes to the ECU.

Justification for Status Change:

Interaction: CAN Bus Data Flow



9. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Justification for Status Change:

10. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from Gateway.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

Justification for Status Change:

11. Cause the Gateway to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Gateway with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Gateway if flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

12. Cause the Gateway to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Gateway with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Gateway if flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

13. Flood Gateway With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

14. Take the Gateway Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Gateway delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

Justification for Status Change:

15. Compromise the Gateway in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the Gateway.

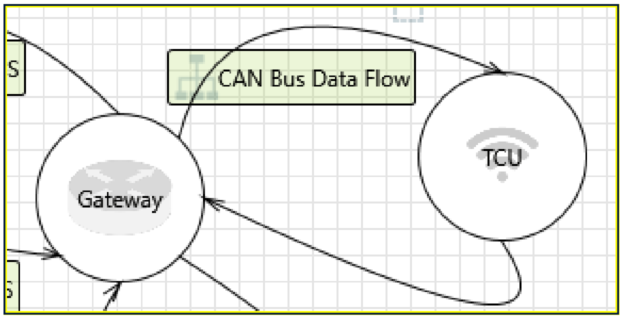
Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Justification for Status Change:

Interaction: CAN Bus Data Flow



16. Modify Data Being Sent to the TCU While in Transit [State: Not Started] [Priority: High]

Category:	Tampering
Description:	Tamper with data in trasit sent to the TCU.
Justification:	<no mitigation provided>
Attack method:	MITM on the TCU for example a 3g to 2g downgrade attack, or false base station attack.
Recommendation:	Disable 2G communications, only 3G and 4G should be allowed. Use a secure communication channel between the car and the server.
Justification for Status Change:	

17. Car Could be Tracked [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Information disclosure by performing a MITM attack in order to track a car.
Justification:	<no mitigation provided>
Attack method:	Downgrade or false base station attack.
Recommendation:	Encrypt communications so that passive interception cannot identify specific vehicles. Rolling/unique identifiers that change over time, to prevent tracking.
Justification for Status Change:	

18. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Justification for Status Change:

19. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from TCU.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

Justification for Status Change:

20. Cause the TCU to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU if flooding is detected. Potentially enter in a special safety mode if TCU is unavalaible.

Justification for Status Change:

21. Cause the TCU to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU if flooding is detected. Potentially enter in a special safety mode if TCU is unavalaible.

Justification for Status Change:

22. Flood TCU With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

23. Take the TCU Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of TCU delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

Justification for Status Change:

24. Compromise the TCU in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the TCU.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Justification for Status Change:

25. Reflash the TCU Firmware in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the TCU firmware.

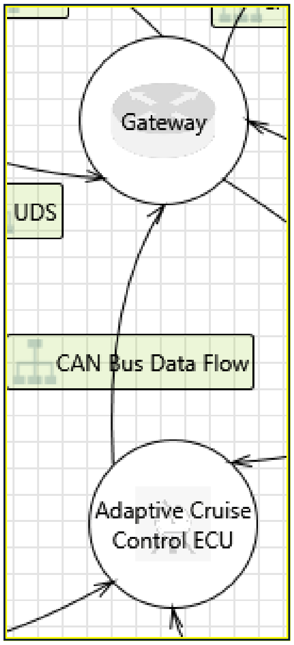
Justification: <no mitigation provided>

Attack method: Physically connect to the target is [TCU] and attempt to reflash the chip.

Recommendation: All firmware should be encrypted and signed to prevent modification. There should be a secure boot process to prevent any invalid firmware from booted.

Justification for Status Change:

Interaction: CAN Bus Data Flow



26. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.
Justification for Status Change:	

27. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Information disclosure by downloading the updates from Gateway.
Justification:	<no mitigation provided>
Attack method:	Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation:	Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

Justification for Status Change:

28. Cause the Gateway to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Gateway with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Gateway if flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

29. Cause the Gateway to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Gateway with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Gateway if flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

30. Flood Gateway With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

31. Take the Gateway Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Gateway delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

Justification for Status Change:

32. Compromise the Gateway in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the Gateway.

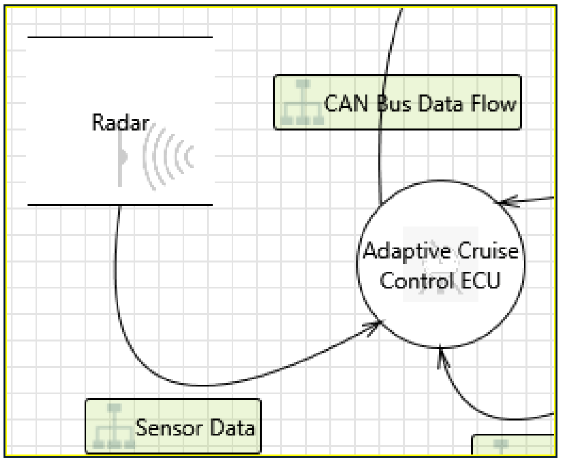
Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Justification for Status Change:

Interaction: Sensor Data



33. Trick ACC ECU Into Triggering an Emergency Stop [State: Not Started] [Priority: High]

Category: Spoofing

Description: Spoofing the Radar in order to trick the Adaptive Cruise Control ECU into triggering an emergency stop.

Justification: <no mitigation provided>

Attack method: As the radar units are external it would be possible to intercept communications, in an attempt to inject a malicious target list. This could be done by physically connecting to the wires used by the radar units.

Recommendation: Relocate unit so the wires are harder to reach.

Justification for Status Change:

34. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Justification for Status Change:

35. Reflash the ACC ECU From the CAN Bus in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the Adaptive Cruise Control ECU.

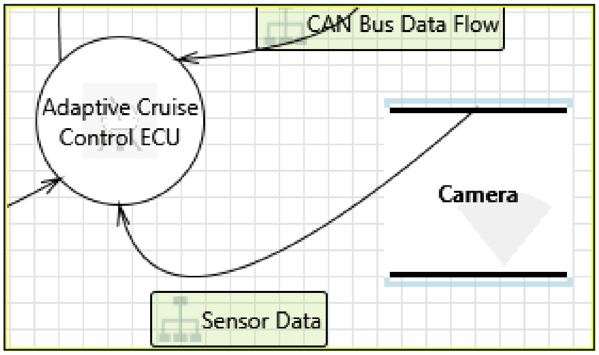
Justification: <no mitigation provided>

Attack method: As the radar units are external it would be possible to connect into the private CAN network and use diagnostic CAN messages to reflash the ACC ECU.

Recommendation: Ensure that diagnostic messages are ignored when received from the private CAN. Additionally all firmware should be signed and validated to prevent malicious updates from being flashes to the ECU.

Justification for Status Change:

Interaction: Sensor Data



36. Cause the Car to Perform Emergency Braking [State: Not Started] [Priority: High]

Category: Spoofing

Description: Spoofing front Camera Cameras and Sensors data in order to trigger emergency braking.

Justification: <no mitigation provided>

Attack method: Spoof front radar data in order to cause the car to emergency break. The front sensor module connects directly to the ACC ECU therefore it is not protected by the gateway module, allowing for messages to be spoofed.

Recommendation: Implement a gateway module so that the the Cameras and Sensors modules can only send messages relating to themselves.

Justification for Status Change:

37. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Justification for Status Change:

38. Reflash the ACC ECU From the CAN Bus in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to reflash the Adaptive Cruise Control ECU.

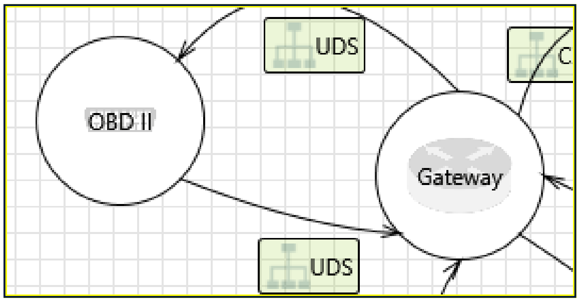
Justification: <no mitigation provided>

Attack method: As the radar units are external it would be possible to connect into the private CAN network and use diagnostic CAN messages to reflash the ACC ECU.

Recommendation: Ensure that diagnostic messages are ignored when received from the private CAN. Additionally all firmware should be signed and validated to prevent malicious updates from being flashes to the ECU.

Justification for Status Change:

Interaction: UDS



39. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.
Justification for Status Change:	

40. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Information disclosure by downloading the updates from Gateway.
Justification:	<no mitigation provided>
Attack method:	Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation:	Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.
Justification for Status Change:	

41. Cause the Gateway to Crash or Stop [State: Not Started] [Priority: High]

Category:	Denial of Service
Description:	DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Justification:	<no mitigation provided>
Attack method:	Flooding Gateway with invalid messages or data.
Recommendation:	Implement data validation and shutdown communications channel to Gateway if

flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

42. Cause the Gateway to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Gateway with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Gateway if flooding is detected. Potentially enter in a special safety mode if Gateway is unavalaible.

Justification for Status Change:

43. Flood Gateway With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

44. Take the Gateway Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Gateway.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Gateway delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

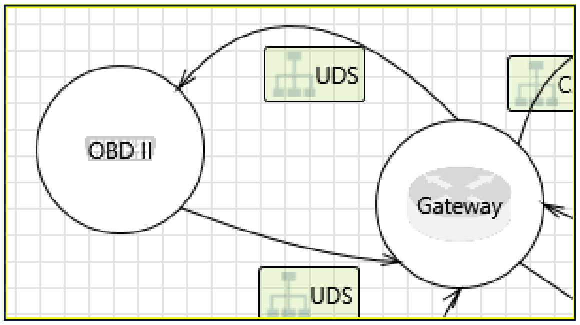
Justification for Status Change:

45. Compromise the Gateway in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description:	Elevation of privileges in order to exploit the Gateway.
Justification:	<no mitigation provided>
Attack method:	Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation:	Ensure that the server is kept up to date and perform regular security testing.
Justification for Status Change:	

Interaction: UDS



46. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.
Justification for Status Change:	

47. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Information disclosure by downloading the updates from OBD II.
Justification:	<no mitigation provided>
Attack method:	Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.
Recommendation:	Ensure that connections to the delivery server are authenticated and encrypted and

access should be limited to only the required files.

Justification for Status Change:

48. Cause the OBD II to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on OBD II that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding OBD II with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to OBD II if flooding is detected. Potentially enter in a special safety mode if OBD II is unavalaible.

Justification for Status Change:

49. Cause the OBD II to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on OBD II that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding OBD II with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to OBD II if flooding is detected. Potentially enter in a special safety mode if OBD II is unavalaible.

Justification for Status Change:

50. Flood OBD II With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on OBD II by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

Justification for Status Change:

51. Take the OBD II Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on OBD II.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of OBD II delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

Justification for Status Change:

52. Compromise the OBD II in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the OBD II.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Justification for Status Change:

C.3 In-Vehicle Infotainment System Threat Modeling Report

The detailed threat modeling report presented below provides full descriptions of the threats presented in Section 4.3.2 of Chapter 4.

Threat Modeling Report

Created on 26/12/2020 00:20:58

Threat Model Name:

Owner:

Reviewer:

Contributors:

Description:

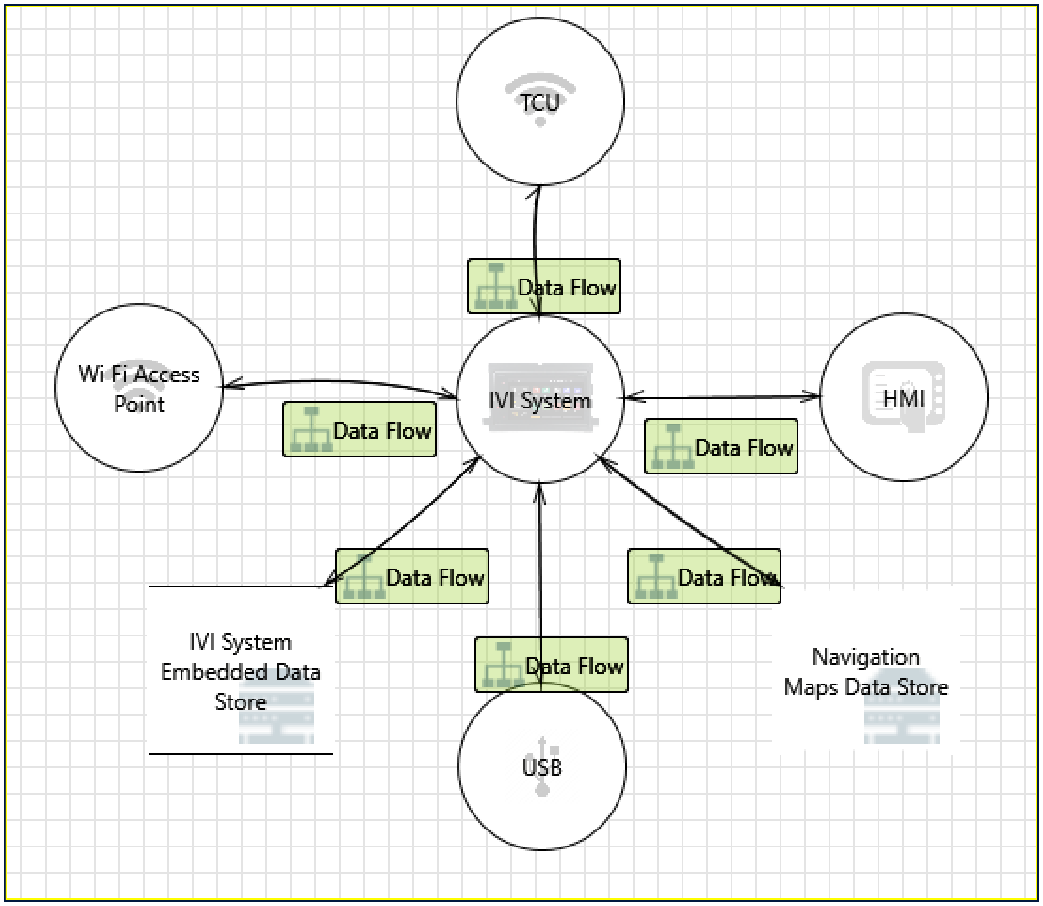
Assumptions:

External Dependencies:

Threat Model Summary:

Not Started	86
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	86
Total Migrated	0

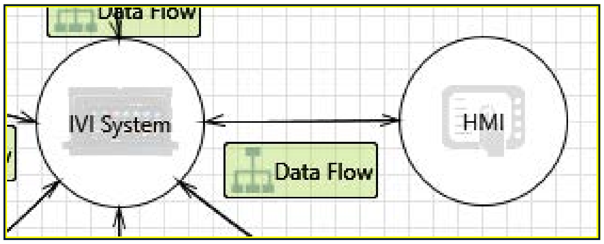
Diagram: In-Vehicle Infotainment



In-Vehicle Infotainment Diagram Summary:

Not Started	86
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	86
Total Migrated	0

Interaction: Data Flow



1. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category:	Elevation of Privilege
Description:	An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop

further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

2. Compromise the IVI System in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

3. Take the IVI System Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of IVI System delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

4. Flood IVI System With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

5. Cause the IVI System to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

6. Cause the IVI System to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

7. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from IVI System.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

8. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

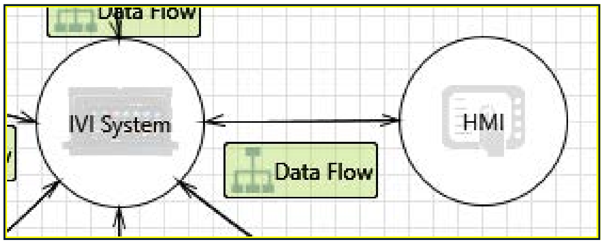
Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Interaction: Data Flow



9. Compromise the HMI in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the HMI.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

10. Take the HMI Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on HMI.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of HMI delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

11. Prevent ADAS Information Being Displayed [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on the HMI that prevents Advanced Driver Assistance Systems (ADAS) data being visualised.

Justification: <no mitigation provided>

Attack method: Flood CAN bus with invalid messages.

Recommendation: Have a direct connection from the ADAS ECU to the HMI interface.

12. Flood HMI With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on HMI by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

13. Cause the HMI to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on HMI that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding HMI with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to HMI if flooding is detected. Potentially enter in a special safety mode if HMI is unavalaible.

14. Cause the HMI to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on HMI that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding HMI with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to HMI if flooding is detected. Potentially enter in a special safety mode if HMI is unavalaible.

15. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from HMI.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

16. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

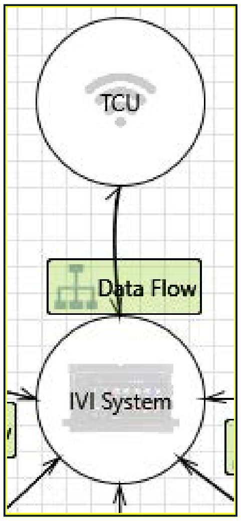
Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Interaction: Data Flow



17. Reflash the TCU Firmware in Order to Send Arbitrary CAN Messages [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to reflash the TCU firmware.
Justification: <no mitigation provided>
Attack method: Physically connect to the target is [TCU] and attempt to reflash the chip.
Recommendation: All firmware should be encrypted and signed to prevent modification. There should be a secure boot process to prevent any invalid firmware from booted.

18. Compromise the TCU in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege
Description: Elevation of privileges in order to exploit the TCU.
Justification: <no mitigation provided>
Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation: Ensure that the server is kept up to date and perform regular security testing.

19. Take the TCU Offline [State: Not Started] [Priority: High]

Category: Denial of Service
Description: DoS on TCU.
Justification: <no mitigation provided>
Attack method: Perform an network attack and case resource exhaustion.
Recommendation: Have a number of TCU delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

20. Flood TCU With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

21. Cause the TCU to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU if flooding is detected. Potentially enter in a special safety mode if TCU is unavalaible.

22. Cause the TCU to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on TCU that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding TCU with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to TCU if flooding is detected. Potentially enter in a special safety mode if TCU is unavalaible.

23. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from TCU.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

24. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations.

In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

25. Car Could be Tracked [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by performing a MITM attack in order to track a car.

Justification: <no mitigation provided>

Attack method: Downgrade or false base station attack.

Recommendation: Encrypt communications so that passive interception cannot identify specific vehicles. Rolling/unique identifiers that change over time, to prevent tracking.

26. Modify Data Being Sent to the TCU While in Transit [State: Not Started] [Priority: High]

Category: Tampering

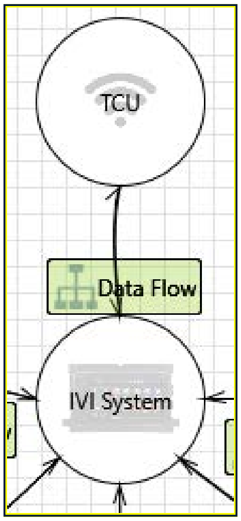
Description: Tamper with data in trasit sent to the TCU.

Justification: <no mitigation provided>

Attack method: MITM on the TCU for example a 3g to 2g downgrade attack, or false base station attack.

Recommendation: Disable 2G communications, only 3G and 4G should be allowed. Use a secure communication channel between the car and the server.

Interaction: Data Flow



27. Pretend to Be the TCU in Order to Exploit the IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: If data from the server is not sufficiently validated an attacker could pretend to be the TCU in order to deliver a malicious update to the IVI System.

Recommendation: Ensure that connections to the TCU are authenticated and encrypted and access should be limited to only the required files. All firmware should be encrypted and signed to prevent modification.

28. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

29. Compromise the IVI System in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

30. Take the IVI System Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of IVI System delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

31. Flood IVI System With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

32. Cause the IVI System to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

33. Cause the IVI System to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

34. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from IVI System.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

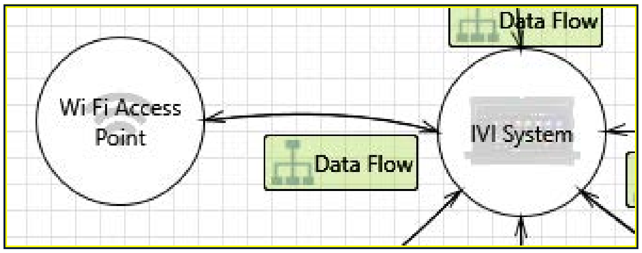
35. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations.

	In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

Interaction: Data Flow



36. Remote Attack Against Vehicle over the Internet [State: Not Started] [Priority: High]

Category:	Spoofing
Description:	Attacker tricks the vehicle infotainment system to connect to a malicious Wi-Fi hotspot. Attacker chains together vulnerabilities discovered in previous scenarios and is able to authenticate to the vehicle's own Wi-Fi hotspot. Attacker gains full administrative control of the infotainment system and can extract PII (Personally Identifiable Information) from the address book database, which would constitute a data breach under existing Data Protection regulations and the GDPR (General Data Protection Regulation) legislation. The attacker can also disable vehicle safety functions, which potentially impacts driver and vehicle occupant safety. In addition, malware such as ransomware could be installed
Justification:	<no mitigation provided>
Attack method:	An attacker spoofs a Wi-Fi hotspot to which the car has previously been connected and then exploits multiple vulnerabilities to eventually connect to the vehicle's own Wi-Fi hotspot and gain full administrative access to the IVI System. This attack is performed over a wireless network and therefore, does not require physical access to the car.
Recommendation:	Ensure that the wireless network to which the car connects are protected by strong authentication mechanisms and using a strong pre-shared key for authentication. This would prevenet the car to connect to the malicious hotspot and prevent chaining a number of vulnerabilities together in order to gain remote access to IVI System.

37. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of

the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

38. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from IVI System.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

39. Cause the IVI System to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

40. Cause the IVI System to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

41. Flood IVI System With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

42. Take the IVI System Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of IVI System delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

43. Compromise the IVI System in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

44. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

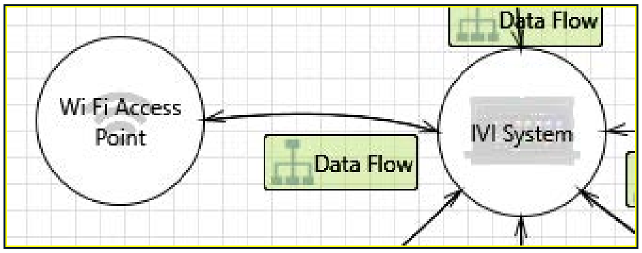
Description: An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

Interaction: Data Flow



45. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

46. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from Wi Fi Access Point.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

47. Cause the Wi Fi Access Point to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Wi Fi Access Point that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Wi Fi Access Point with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Wi Fi Access Point if flooding is detected. Potentially enter in a special safety mode if Wi Fi Access Point is unavalaible.

48. Cause the Wi Fi Access Point to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Wi Fi Access Point that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Wi Fi Access Point with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Wi Fi Access Point if flooding is detected. Potentially enter in a special safety mode if Wi Fi Access Point is unavalaible.

49. Flood Wi Fi Access Point With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Wi Fi Access Point by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

50. Take the Wi Fi Access Point Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Wi Fi Access Point.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Wi Fi Access Point delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

51. Compromise the Wi Fi Access Point in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

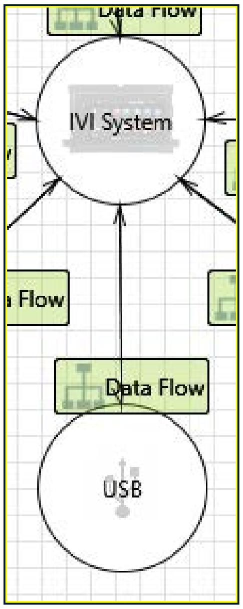
Description: Elevation of privileges in order to exploit the Wi Fi Access Point.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Interaction: Data Flow



52. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

53. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from USB.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

54. Cause the USB to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on USB that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method:	Flooding USB with invalid messages or data.
Recommendation:	Implement data validation and shutdown communications channel to USB if flooding is detected. Potentially enter in a special safety mode if USB is unavalaible.

55. Cause the USB to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category:	Denial of Service
Description:	DoS on USB that crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Justification:	<no mitigation provided>
Attack method:	Flooding USB with invalid messages or data.
Recommendation:	Implement data validation and shutdown communications channel to USB if flooding is detected. Potentially enter in a special safety mode if USB is unavalaible.

56. Flood USB With Invalid Data [State: Not Started] [Priority: High]

Category:	Denial of Service
Description:	DoS on USB by flooding with invalid data.
Justification:	<no mitigation provided>
Attack method:	Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.
Recommendation:	Rely on additional sensors in the event of one is unavailable.

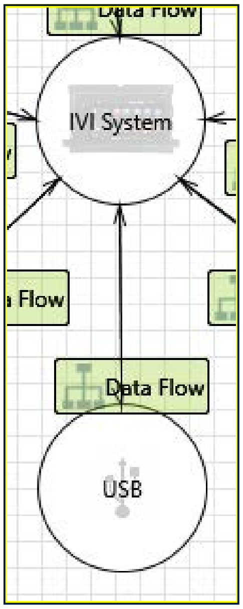
57. Take the USB Offline [State: Not Started] [Priority: High]

Category:	Denial of Service
Description:	DoS on USB.
Justification:	<no mitigation provided>
Attack method:	Perform an network attack and case resource exhaustion.
Recommendation:	Have a number of USB delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

58. Compromise the USB in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category:	Elevation of Privilege
Description:	Elevation of privileges in order to exploit the USB.
Justification:	<no mitigation provided>
Attack method:	Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation:	Ensure that the server is kept up to date and perform regular security testing.

Interaction: Data Flow



59. Gaining Network Level Access to IVI System Components [State: Not Started] [Priority: High]

Category:	Tampering
Description:	An attacker can scan for available newtork services on the IVI System. Attacker will find login services available but will not know the correct credentials. Attacker identifies that the IVI System system actually comprises two separate, connected components. Attacker discovers a "backdoor" service that appears to provide some kind of interactive access that does not require a password. Attacker discovers another service that can be used to remotely control the IVI System and retrieve sensitive information. Attacker discovers how to leverage the "backdoor" service to gain full administrative control of the IVI System, retrieve valid admin login credentials and extract PII (Personally Identifiable Information) from the address book database, which would constitute a data breach under existing Data Protection regulations and GDPR (General Data Protection Regulation) legislation. The attacker can also disable vehicle safety functions, which potentially impacts driver and vehicle occupant safety. In addition, malware such as ransomware could be installed.
Justification:	<no mitigation provided>
Attack method:	An attacker purchases a USB network adaptor and connects their laptop to the infotainment system via the USB port within the vehicle. This potentially results in an attacker with physical access to the USB port e.g. valet parking attendant, being able to gain full administrative control of the IVI System.
Recommendation:	Infrastructure assessment of the available network interfaces and services access via USB port.

60. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations.

In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

61. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from IVI System.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

62. Cause the IVI System to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

63. Cause the IVI System to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System if flooding is detected. Potentially enter in a special safety mode if IVI System is unavalaible.

64. Flood IVI System With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

65. Take the IVI System Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of IVI System delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

66. Compromise the IVI System in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the IVI System.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

67. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

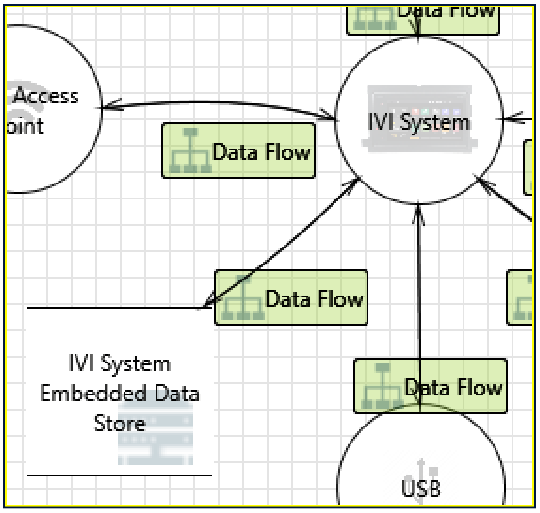
Description: An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

Interaction: Data Flow



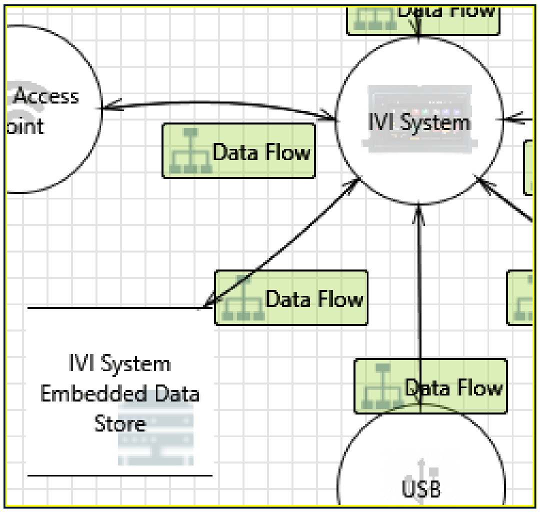
68. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

69. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category:	Elevation of Privilege
Description:	An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.
Justification:	<no mitigation provided>
Attack method:	An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.
Recommendation:	Hardware security technical assessment of the IVI System.

Interaction: Data Flow



70. Potential SQL Injection Vulnerability for IVI System Embedded Data Store [State: Not Started] [Priority: High]

Category: Tampering

Description: SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parameterized data can be manipulated by a skilled and determined attacker.

Justification: <no mitigation provided>

Attack method: An attacker uses both manual and automated tools to inject SQL statements within IVI System Embedded Data Store.

Recommendation: Ensure that user supplied input cannot be included in the SQL statements which are executed against the database. In general, dynamic SQL should not be used within the application.

71. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

72. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from IVI System Embedded Data Store.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

73. Cause the IVI System Embedded Data Store to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System Embedded Data Store that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System Embedded Data Store with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System Embedded Data Store if flooding is detected. Potentially enter in a special safety mode if IVI System Embedded Data Store is unavalaible.

74. Cause the IVI System Embedded Data Store to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System Embedded Data Store that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding IVI System Embedded Data Store with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to IVI System Embedded Data Store if flooding is detected. Potentially enter in a special safety mode if IVI System Embedded Data Store is unavalaible.

75. Flood IVI System Embedded Data Store With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on IVI System Embedded Data Store by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

76. Take the IVI System Embedded Data Store Offline [State: Not Started] [Priority: High]

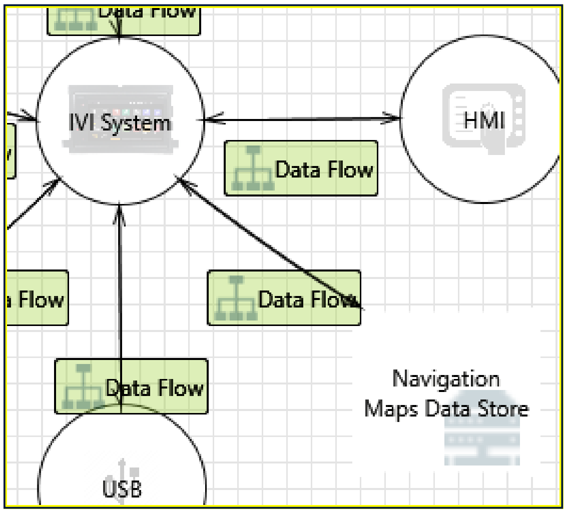
Category: Denial of Service

Description:	DoS on IVI System Embedded Data Store.
Justification:	<no mitigation provided>
Attack method:	Perform an network attack and case resource exhaustion.
Recommendation:	Have a number of IVI System Embedded Data Store delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

77. Compromise the IVI System Embedded Data Store in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category:	Elevation of Privilege
Description:	Elevation of privileges in order to exploit the IVI System Embedded Data Store.
Justification:	<no mitigation provided>
Attack method:	Network based vulnerabilities, through outdated software or configuration weaknesses.
Recommendation:	Ensure that the server is kept up to date and perform regular security testing.

Interaction: Data Flow



78. Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.
Justification:	<no mitigation provided>
Attack method:	Man-in-the-middle using attack using hardware or software.
Recommendation:	Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

79. Hardware Teardown and Reverse Engineering on IVI System [State: Not Started] [Priority: High]

Category: Elevation of Privilege

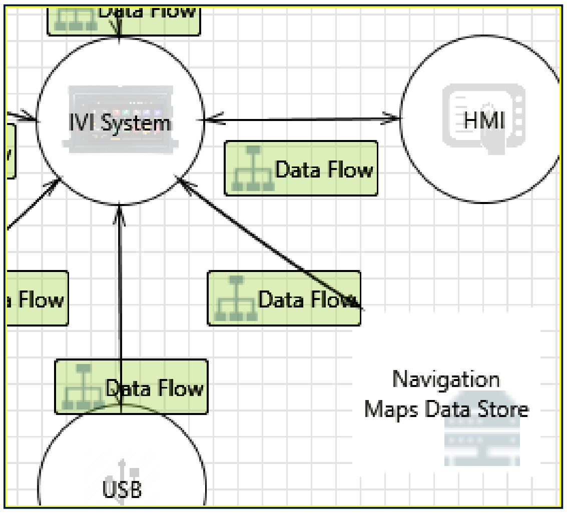
Description: An attacker gains access to all the files on the In-Vehicle Infotainment system. In addtion, the attacker can extract sensitive data e.g. login credentials and develop further attacks. Morever, it can reverse engineer files and find vulnerabilities over time.

Justification: <no mitigation provided>

Attack method: An attacker purchases IVI System from an online auction site, dismantles the unit, removes the memory chips, extracts their content and analyses the software.

Recommendation: Hardware security technical assessment of the IVI System.

Interaction: Data Flow



80. Data Flow Sniffing [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Data flowing across [Generic Data Flow] may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. In general, as a way to compromise both integrity and availability. Severity might change depending on the attacker's access vector.

Justification: <no mitigation provided>

Attack method: Man-in-the-middle using attack using hardware or software.

Recommendation: Consider encrypting the data flow. For web traffic this can be achieved by using HTTPS.

81. Updates Could Be Downloaded From a Web Server Resulting in Potentially Sensitive Information Being Disclosed [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Information disclosure by downloading the updates from Navigation Maps Data Store.

Justification: <no mitigation provided>

Attack method: Reverse engineer the head unit firmware to find information about the update server and download software update files which may contain sensitive information.

Recommendation: Ensure that connections to the delivery server are authenticated and encrypted and access should be limited to only the required files.

82. Cause the Maps Data Store to Crash or Stop [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Navigation Maps Data Store that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Maps Data Store with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Maps Data Store if flooding is detected. Potentially enter in a special safety mode if Maps Data Store is unavalaible.

83. Cause the Maps Data Store to Crash or Stop Remotely [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Navigation Maps Data Store that crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: <no mitigation provided>

Attack method: Flooding Maps Data Store with invalid messages or data.

Recommendation: Implement data validation and shutdown communications channel to Maps Data Store if flooding is detected. Potentially enter in a special safety mode if Maps Data Store is unavalaible.

84. Flood Maps Data Store With Invalid Data [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Navigation Maps Data Store by flooding with invalid data.

Justification: <no mitigation provided>

Attack method: Either physically by clipping onto the sensor wires and inject valid data or with external input e.g. a bright torch.

Recommendation: Rely on additional sensors in the event of one is unavailable.

85. Take the Maps Data Store Offline [State: Not Started] [Priority: High]

Category: Denial of Service

Description: DoS on Navigation Maps Data Store.

Justification: <no mitigation provided>

Attack method: Perform an network attack and case resource exhaustion.

Recommendation: Have a number of Maps Data Store delivery servers across a broad geographic radius, in the event of one server failing the system should continue unhindered.

86. Compromise the Maps Data Store in Order to Deliver Malicious Updates [State: Not Started] [Priority: High]

Category: Elevation of Privilege

Description: Elevation of privileges in order to exploit the Navigation Maps Data Store.

Justification: <no mitigation provided>

Attack method: Network based vulnerabilities, through outdated software or configuration weaknesses.

Recommendation: Ensure that the server is kept up to date and perform regular security testing.

Appendix D

Ethics Documentation