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

Mahmood, S., Nguyen, H. N. & Shaikh, S

DOI 10.1016/j.vehcom.2022.100468
ISSN 2214-2096

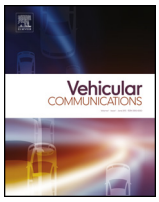
Publisher: Elsevier

Vehicular Communications 35 (2022) 100468

Contents lists available at ScienceDirect

Vehicular Communications

www.elsevier.com/locate/vehcom



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

Shahid Mahmood [a,b], Hoang Nga Nguyen [a], Siraj Ahmed Shaikh [a,*]

[a] Systems Security Group, Centre for Future Transport and Cities (CFTC), Coventry University, CV1 5FB, United Kingdom
[b] Intelligent Mobility and Software, FEV United Kingdom

ARTICLE INFO

Article history:
Received 4 October 2020
Received in revised form 5 January 2022
Accepted 13 March 2022
Available online 18 March 2022

Keywords:
Automotive over-the-air
Automotive OTA
Model-based security testing
Threat modeling
Automotive cybersecurity
Uptane

ABSTRACT

Modern cars host numerous special-purpose computing and connectivity devices facilitating the correct functioning of various in-vehicle systems. These devices host complex software systems with over 100-million lines of code, requiring regular and timely updates for functional and security improvements. Addressing the shortcomings of the legacy update system, over-the-air (OTA) software update system has emerged as an efficient, cost-effective, and convenient solution for delivering updates to automobiles remotely. While OTA offers several benefits, it introduces new security challenges requiring immediate attention, as attackers can abuse these update systems 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 primarily 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 study closes this gap by presenting an in-depth security evaluation of Uptane framework, by employing a structured threat analysis approach to constructing attack trees and applying a model-based security testing approach for generating effective security test cases. We implement a software tool that generates the security test cases by analysing the structure of the attack trees and ultimately executing those test cases against the target system. We carry out several experimental attacks on the Uptane reference implementation. While many of the experimental results showed that the reference implementation is secure against different threats and cyberattacks, some findings suggest that the implementation is vulnerable to the denial-of-service and eavesdropping attacks.

1. Introduction

Modern vehicles are equipped with numerous sophisticated computing (i.e., 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 [60,18]. 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 of the vehicle [85].

For the deployment of these updates, vehicles had traditionally been required to visit a service centre or dealership where a mechanic had to install the updates. This old mechanism for updating in-vehicle software is not only expensive and inefficient for car-makers, it is inconvenient for the customers as well. For example, General Motors had to spend $4.1 billion on vehicle recalls in 2014 while their total net income for that particular year was $4 billion [40]. Another interesting related example is the work from Koscher et al. [47], 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 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.

* Corresponding author.
E-mail addresses: mahmo136@coventry.ac.uk (S. Mahmood), ac1222@coventry.ac.uk (H.N. Nguyen), aa8135@coventry.ac.uk (S.A. Shaikh).

https://doi.org/10.1016/j.vehcom.2022.100468
2214-2096/

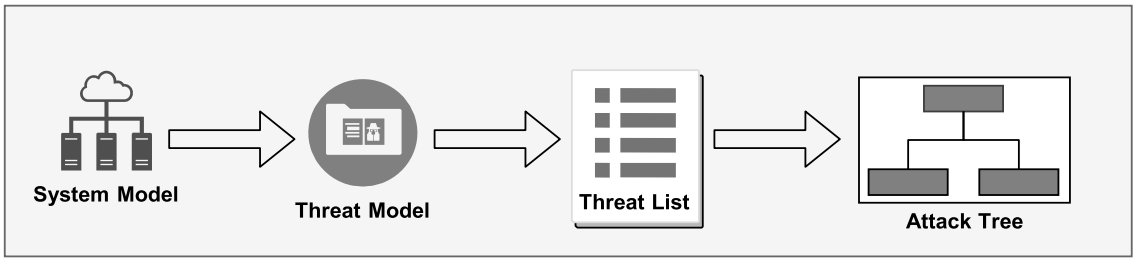


Fig. 1. Contribution 1: Overview of the threat analysis approach leading to the construction of an attack tree systematically.

1.1. Motivation, novelty and contributions

Undoubtedly, OTA offers several benefits, but it also introduces new security challenges that must be considered seriously, as attackers can compromise the software update system [62,9,71] to compromise the vehicle security.

Unlike other devices that receive OTA updates, security of automotive updates is a huge concern, because a malicious or corrupt update can never be as devastating as it can be in the case of a connected car, which could potentially lead to loss of human life [48,36]. While protection of the OTA updates is vital for any IoT device, update security in connected cars should be even a greater concern [24]. OEMs must adopt effective security measures for the detection and mitigation/prevention of any potential security breaches in OTA updates. This means security testing of these systems is crucial for evaluating security measures in place and what further actions are required for strengthening the protection against potential cyberattacks. While there are numerous existing studies [47,57,68,42,58,15,59,61,67,39,49,3] that have extensively investigated cybersecurity testing of automotive systems exploring challenges, testing approaches, testing environments, security threats, vulnerabilities, relevant solutions and defensive mechanisms, security testing of automotive OTA updates has not been investigated adequately. In order to ensure these update systems are secure and have adequate protection in place against various cyberattacks and threats, thorough security analysis and evaluations must be performed. To the best of our knowledge, there is currently no study that employs a disciplined security testing approach for evaluating the security posture of automotive software over-the-air update systems. We attempt to close this gap by conducting systematic threat assessment and security testing of the automotive OTA updates by using a combination of model-based security testing techniques and penetration testing.

This research contributes to:

Contribution 1: A systematic threat analysis approach for constructing attack trees

Demonstrated by an in-depth analysis of OTA/Uptane, 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 Fig. 1).

Contribution 2: A model-based security testing approach based on attack tree to derive security test cases

Our model-based security testing approach uses attack trees to derive effective security test cases by analysing the structure of these attack trees. In order to validate the approach, we demonstrate a number of cyberattacks using the generated security test cases.

Contribution 3: An in-depth experimental security analysis of the reference implementation of the Uptane Framework

A comprehensive security analysis of the Uptane Framework's reference implementation, including 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.

Contribution 4: 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, which is capable of deriving appropriate 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.

1.2. Structure of the paper

The rest of this paper is organised as follows. In Section 2, we present background information by providing an overview of the automotive cybersecurity testing, an introduction of the Uptane Framework along with related solutions, and threat modeling, including STRIDE classification model and attack trees. Section 3 gives an overview of the related work, highlighting some of the existing model-based and systematic security testing studies with a particular focus on automotive domain. We introduce our systematic threat assessment and security testing approach in Section 4, detailing its different phases, associated steps, tools and techniques to support different activities. Experiments, demonstrating the application of the testing approach are presented in Section 5, providing details of various security tests conducted on the reference implementation of the Uptane Framework along with the test results. A discussion on the approach, experiments and the results is presented in Section 6. Finally, Section 7 concludes this paper.

2. Background

This section provides background information pertaining to security testing of automotive systems.

2.1. Automotive security testing

While the aim of software testing is to detect bugs, to validate if it satisfies specified requirements, and to verify it is fit for purpose, software security testing is concerned with the verification of the system whether it meets specified security requirements (i.e., information confidentiality, privacy, integrity, availability, authenticity, access control, misuse prevention, etc.) [27]. Several software security approaches have been proposed, including formal methods that use formal specification language and rely on mathematical model of the software; fault injection-based security testing is another approach that focuses on the application interaction points, network interfaces, user input, file systems, and environment variables; risk-based security testing involves finding high-risk vulnerabilities in the software as early as possible [83].

However, as opposed to software security testing in general, automotive security testing is perhaps more important due to the nature of catastrophic consequences involving loss of human life in some cases. Modern automobiles are exposed to numerous cybersecurity threats due to their built-in powerful computing and communication capabilities. Identifying vulnerabilities and security flaws in the communication and other onboard technologies in connected cars is critical, as cybercriminals can exploit those weaknesses for gaining access to the safety-critical systems of the vehicle [19].

Most cars today host many computing devices, known as Electronic Control Units (ECUs). Each ECU has specific responsibilities, and they may need to communicate with each other and with the external world for successful completion of their tasks. For local communication, they rely on one or more of the in-vehicle communication networks, such as Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, and Media-Oriented Systems Transport (MOST). Each type of network has been designed to support applications with different needs. For example, while LIN is mostly used for low-speed applications, applications requiring high-speed data-transfers use MOST [35]. Legally mandated Onboard Diagnostic (OBD) ports in the modern vehicles are used for ECU firmware updates, repairing and inspections of the vehicle. They are also 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 [82].

There are several entry points that attackers can take advantage of for breaking into a vehicle's internal system, which have been extensively explored and presented by previous studies. For example, [8,39,70] explore CAN exploitation, [65] reports attacks leveraging OBD port, and security issues related to in-vehicle infotainment are presented in [45]. Over-the-air software update systems for automobiles can also be targeted by hackers in several different ways, as described in [48] for compromising the security and safety of the connected vehicles.

While automotive OTA offers numerous benefits (e.g., seamless delivery of software updates remotely), presence of security flaws and vulnerabilities in such systems can be exploited by adversaries to undermine the security of connected cars. For example, attackers can compromise the repositories that host the software updates, as described by Kuppusamy et al. in [48]. Various testing methods (for example, [31,14,74,12,29,6,5,38,63,13]) and testing environments (e.g., [26,84,97,96,30]) have been proposed for the security testing of automotive systems. These testbeds and techniques have been designed primarily for discovering security flaws in vehicular networks (e.g., CAN, MOST, LIN, etc.), ECUs, and infotainment systems. Cybersecurity testing of the automotive OTA software update systems has not been considered by these works.

2.2. Automotive over the air (OTA) updates: an overview

The phrase *over the air* (*OTA*) *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 [37]. 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 [17]. In the automotive industry, OEMs are increasingly embracing the OTA technology for delivering updates to embedded devices in the connected vehicle [20]. Car manufacturers are predicted to save $35 billion in year 2022 by relying on OTA technology for sending updates remotely [46].

2.2.1. Use cases

Some of the major use cases for automotive updates are outlined below [72]:

- **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.

2.2.2. Vulnerabilities and threats

Software update systems have been attacked and compromised by cybercriminals (for example, [92,95,73,32,21]), 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 [7]. A revised draft proposal for these regulations has recently been published by UNECE [2], which identifies major threats to automotive update procedures along with relevant vulnerabilities and attack methods.

Safety standards, such as ISO 26262 and IEC 61508 for road vehicles provide recommendations for the interaction between the safety and security. Automotive security engineering standards, including SAE J3061 and ISO-SAE 21434, include recommendations for threat and risk assessments (i.e., threat analysis and risk assessment or TARA) in order to establish if there are cybersecurity threats that can affect the safety. In particular, ISO-SAE 21434 defines a structured approach for ensuring cybersecurity engineering of in-vehicle systems, reducing the likelihood of security attacks. While not directly concerned with the automotive systems, ISO 27000 series and IEC 62443 standards may be relevant for the automotive production and back-end systems [77].

A more comprehensive threat model has been presented in [48] that identifies various security threats to the OTA system that can potentially be used by malicious entities for compromising the security of connected vehicles. Each threat has one or more types of related attacks, ranging from reading the contents of an update to gaining the vehicle control. Attackers can have one or more attack goals, as described in [48] and [87]. 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
- 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

2.3. 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 [48]; 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.

2.3.1. AiroDiag: an OTA diagnostics and updates system

Mansour et al. [54] propose AiroDiag - an over-the-air system - for performing automotive diagnostics and software updates. 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 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. 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.

2.3.2. 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. [56]. The authors use the customised RSA cryptographic 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 techniques. However, a major limitation of their proposed method is the performance when it comes to encryption/decryption time.

2.3.3. OTA update security using blockchain techniques

For effective security protection of OTA updates for connected vehicles, Steger et al. [81] 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.

The solutions described above focus on the confidentiality and integrity of the update contents by using cryptography and Hash functions. 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.

2.4. The uptane framework

Uptane [87], 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 Fig. 2, Uptane Framework has three core components: the Image repository, the Director repository, and the Time Server. A brief description of each is presented below:

2.4.1. 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 metadata stored on the repository as a protective measure against any attempt from attackers to tamper with this metadata.

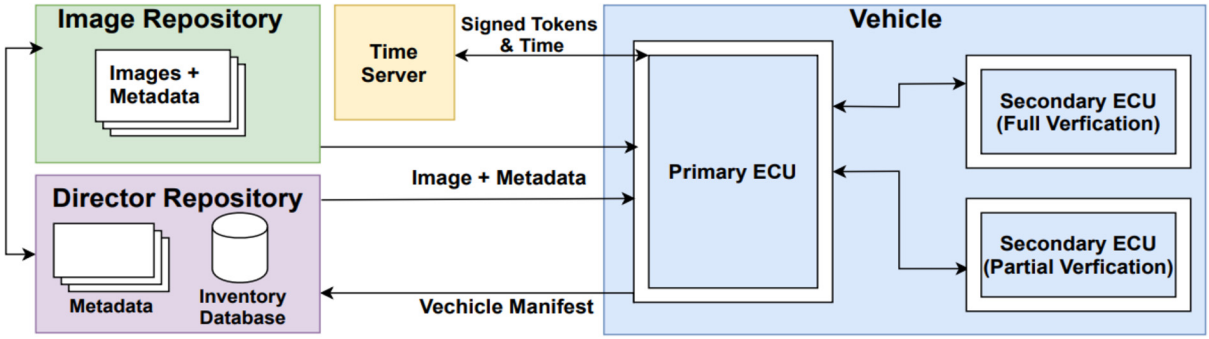


Fig. 2. An overview of the Uptane Framework, illustrating the interconnections and flow of information among the Time Server, Image repository, Director repository, Primary ECU and Secondary ECU.

2.4.2. 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.

2.4.3. 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 built-in 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.

2.4.4. 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.

2.4.5. Roles

Uptane Framework relies on different roles, each responsible for signing different type of metadata as explained below: *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. *Snapshot Role* is concerned with signing metadata indicating the images released by the repository at the same time. *Targets Roles* has the responsibility of signing metadata (e.g., cryptographic hashes and file size etc.) that verifies the image. *Timestamp Role* is used to sign metadata to indicate the availability of new metadata or images on the repository. More comprehensive introduction of the Framework can be found in [87]. In what follows we present an overview of the threat modeling approaches.

2.5. Threat modeling

Threat modeling is a security analysis technique that helps identify risks by using abstractions [80]. 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 aids in modeling adversaries in order to understand their motivations, capabilities, tactics, techniques, and procedures [52]. Several threat modeling methods and techniques have been proposed, some of which are briefly described below:

2.5.1. CORAS

CORAS [22] - based on Australian Risk Management Standard AS/NZS 4360:2004 - is a threat modeling and specification language, comprising 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.

2.5.2. PASTA

PASTA or Process for Attack Simulation and Threat Analysis [86] 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 and 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.

2.5.3. T-MAP

T-MAP [16] 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.

2.5.4. 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 [79], as summarised in Table 1. 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, it is an extension of the familiar and widely used CIA triad (Confidentiality, Integrity, Availability). Finally, STRIDE has an associated threat-modeling tool that we leverage in this study for threat enumeration, which contributes to our attack-tree construction process.

Table 1
An overview of the STRIDE model summarising 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	Nonrepudiation
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

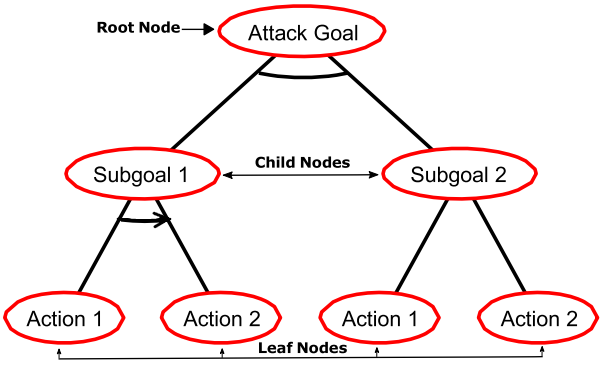


Fig. 3. Example Attack Tree: This example attack tree depicts a root, two child, and four different leaf nodes. Annotations indicate the type of role that is assumed by each of the node types.

2.5.5. Threat modeling tool

Microsoft's Threat Modeling Tool (TMT), which was originally developed for supporting its well-known Security Development Lifecycle (SDL) process for a systematic identification of various security threats during the initial phases of the software development lifecycle [91]. TMT, as its name implies, allows security analysts to model the target system by employing one of the standard notations, known as data flow diagrams (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 [52,44,66,17].

2.5.6. Attack trees

Attack trees are used to represent attacks against a system in a tree structure, with root node as the attack goal and leaf nodes as different ways to achieve that goal [78]. They can help in the identification of various potential threats to a system from the perspective of an attacker. Being a structured approach, attack trees enable systematic security evaluation by focusing on threats and associated actions that can be performed by malicious actors for mounting attacks on the target system. An attack tree (as shown in Fig. 3) contains a goal (the root of the tree), a set of sub-goals, structured using the operators conjunction (**AND**) and disjunction (**OR**), and leaf nodes, which represent atomic attacker actions. The **AND** nodes are complete when all child nodes are carried out and the **OR** nodes are complete when at least one child node is complete.

Extensions have been proposed using **Sequential AND** (or **SAND**) [43]. We follow the formalisation of attack trees given in [43,55]. If $\mathbb{A}$ is the set of possible atomic attacker actions, the elements of the attack tree $\mathbb{T}$ are $\mathbb{A} \cup \{\textbf{OR}, \textbf{AND}, \textbf{SAND}\}$, and an attack tree is generated by the following grammar, where $a \in \mathbb{A}$:

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

Attack tree semantics have been defined by interpreting the attack tree as a set of series-parallel (SP) graphs [43].

3. Related work

In this section we provide an overview of the related studies focusing on penetration and model-based security testing of automobiles.

3.1. Model-based-security testing

Model-based security testing 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 [1]) in a systematic and efficient manner [76]. MBST primarily uses models to verify if the target system meets its security requirements (as specified in Section 2.1 above) [28]. 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. [74] 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 analyse 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 modeled. 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. [90] present aspect-oriented programming as a powerful technique for security evaluation of cyberphysical systems, especially focusing on safety-critical elements in automotive control systems. Aspect-oriented modeling (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.) [25]. Aspect-oriented modeling is used to express crosscutting concerns at a higher level of abstraction by means of modeling elements [11]. The technique presented by [64] 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

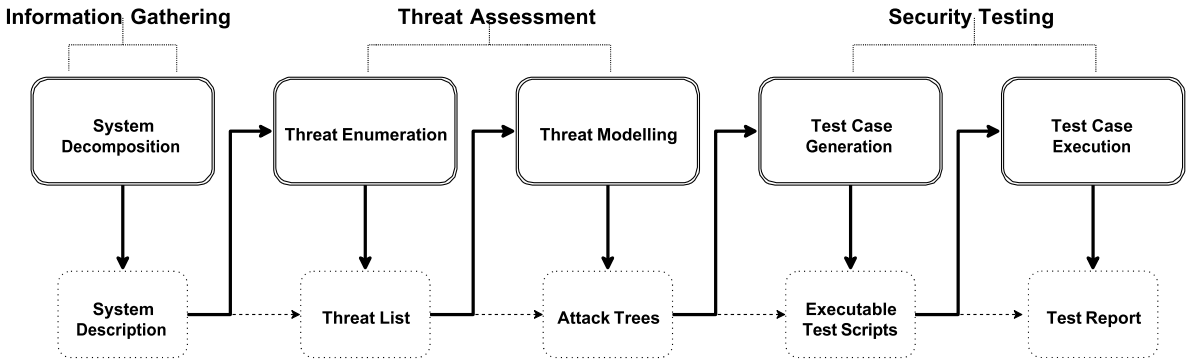


Fig. 4. An overview of the threat assessment and security testing approach used for automotive OTA update system, showing key stages, inputs and outputs of each phase.

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 modeled 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 study, on the other hand, uses threat models for automated derivation and execution of security test cases by combining the MBST with penetration testing.

3.2. 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.

Durrwang et al. [23] 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.

A Framework for systematic security testing of automotive Bluetooth interfaces is proposed by Cheah et al. [14], 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.

4. Threat assessment and security testing approach

Our work diverges from the approaches discussed above in many ways: the approach we use carries out systematic security testing of automotive OTA updates. We use a structured threat enumeration approach along with threat trees for systematic derivation of executable security test cases by using the custom software tool capable of automatic test-case derivation and execution.

In this section, we present details of our testing approach (initially introduced in [53])[1] that incorporates a software tool for generating and executing test cases automatically.

The testing approach we employ in this study is inspired by the Penetration Testing and Execution Standard (PTES) [69] and some of the ideas presented in [14,89,44,93]. A key feature of the PTES testing methodology is the use of threat modeling techniques. 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 automotive OTA update system. A graphical overview of our approach is presented in Fig. 4. 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 and interactions. Description of the system is the output from this phase, which is then used by the next phase *Threat Assessment* for examining the system in order to determine what potential threats/vulnerabilities can be leveraged by cybercriminals for compromising system security. Based on the information received from the preceding phase, the step *Threat Enumeration* generates a *Threat List*, which in turn helps *Threat Modeling* step in the same phase. Since the threat list only provides high-level descriptions of the potential threats, specific attack actions/steps can be identified by creating *Attack Trees*. Next steps in the following phase (i.e., *Security Testing*) use these attack trees for deriving test cases, preparing test scripts, and finally executing them against target system. Our prototype software tool analyses 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. We describe all the phases/steps with more details in following subsections.

4.1. Information gathering

Effective testing necessitates a very good understanding and technical details of the system under test. However, in most cases,

[1] The source code for the test-case generation/execution tool and Uptane reference implementation along with a guide for setting up hardware/software environment can be downloaded from https://tinyurl.com/rydjmqa.

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 needs to be gathered from other sources, including publicly available documentation, technical manuals, and often by directly observing (and, if practicable, sometimes reverse engineering) the system.

System Decomposition - As stated above, one of the key elements and prerequisites to effective security testing is the knowledge and understanding of the target system, this step is, therefore, concerned with finding and collecting as much information about the system as possible. In particular, identification of core hardware and software system components, any interfaces, interrelationships among these components, communication technologies, protocols, and key processes can be highly useful. All the information gathered in this step should help build a model of the system. The system model should depict both architectural and behavioural aspects of the system. To achieve this, we decide to rely on the standard and widely used modeling technique: Unified Modeling Language (UML). An alternative to this is SysML. With the system model created in this phase, we are now able to proceed with the next most important step: identification of the threats.

4.2. Threat assessment

This phase is comprised of two major steps: threat enumeration and threat modeling. Threat enumeration involves identifying any potential threats to the system by using the widely utilised, standard approach STRIDE supported by an associated threat modeling tool, which is capable of enumerating various types of threats in a structured way.

4.2.1. Threat enumeration

As we indicated earlier, for structured threat identification, we employ Microsoft's Threat Modeling Tool. The dynamic system model produced in the Systems Decomposition step of Information Gathering phase (as shown in Fig. 4), serves as an input for drawing the DFD of the system. Once the DFD drawing 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 system STRIDE.

4.2.2. Threat modeling

While the threat report generated using the TMT enumerates all identified threats that can potentially be leveraged by the adversary to compromise the system security, it is often limited to a high-level description of the threat, providing no information about the specific steps/actions performed for compromising the system security [75]. 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 methods used and actions performed by an attacker. Attack tree construction process 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.

4.3. Security testing

This phase is concerned with deriving the test cases and associated test scripts for executing them in order to evaluate the security protection effectiveness. We automate the test-case generation and execution process by developing a software tool. Details of this tool are presented in the following subsections.

Algorithm 1: A test-case generation algorithm that derives test cases by analysing the structure and based on 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.

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

4.3.1. Test case generation and execution

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 [88,28].

Since this study adopts a model-based security testing approach, a software tool has been designed and implemented (previously introduced in [53]) 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, analyses 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. 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.

4.3.2. The algorithm

Algorithm 1 outlines the main logic and major steps for deriving security test cases by analysing the structure of the attack tree. The function GenerateTestCases (for convenience, we will hence-

Table 2
An attack tree represented in XML format.

XML representation of the Attack Tree as displayed in Fig. 3.

```
<?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>
```

forth 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 Table 2. 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 refinement 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 tree 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 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 these 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. 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.

Table 3
A list of various assumptions that have been made for various security test cases.

Assumption	Justification
A1: Attackers either had privileged access to the Uptane repositories or the repositories were being spoofed.	There are numerous techniques used by adversaries for gaining privileged access. For example, it could be a disgruntled employee (i.e., insider threat), facilitating the attacker to gain access (by providing admin credentials); it can also be achieved by social engineering techniques (for instance, phishing attacks); exploitation of vulnerabilities arising from configuration flaws can also lead to privilege escalation (such as default/blank admin or root passwords). An extensive coverage of various privilege escalation attack vectors/techniques (including dictionary attacks, Pass-the-Hash, credential stuffing, password spraying, etc.) is presented in [33].
A2: The secret cryptographic keys for signing software or metadata were compromised.	This could be accomplished by social engineering techniques, such as *pretexting* (see [34,41,51,4] for more information) and/or bribing an employee, for example.
A3: Attackers were able to break into the system.	Firstly, since the primary goal of this study 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. 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 service provider'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.

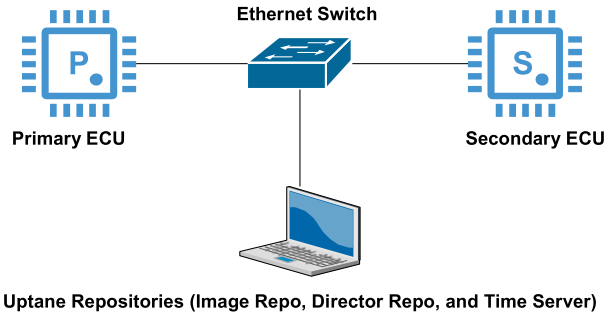


Fig. 5. OTA testbed schematic, providing a graphical overview of the major components and communication links.

5. Experiments and results

In this section, experimentation and associated results of the automotive OTA security testing are presented. Additionally, as shown in Table 3, we have listed some important assumptions that have been made while performing the security testing. For each assumption, appropriate justifications have also been provided. We indicate in the relevant subsequent sections (see Tables 9 to 13) where one or more of these assumptions are applicable.

Each attack tree was analysed by the 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 15 different test cases and generated relevant test scripts. Seven of these security test cases failed after successful execution of the test scripts. While the reference implementation was able to defend different attacks, some attacks suggest that effective security controls need to be applied to the production system in order to protect both the information exchange between ECUs and Uptane repositories and backend-server infrastructure to ensure timely and uninterrupted delivery of updates.

Experimental Setup

This section details the testing environment (see Fig. 5) 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 setup for automotive security testing. Core components of the testbed are shown in the Fig. 6, and a description of all its hardware/software components has been summarised in Table 4. 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.

5.1. Information gathering

In this first phase (consisting of the System Decomposition step) of the process, information about the Uptane Framework is gathered, as detailed below.

5.1.1. System decomposition

Recall that to identify core system components/external interfaces that attackers can potentially target as intrusion points, 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 this information about the framework on the internet, we reproduced the architecture diagram of Uptane as shown in Fig. 2.

Having identified the major entities of the OTA update system, the next step is to identify and understand the interactions between these entities. Based on the analysis of the information gathered, a UML sequence diagram (as shown in Fig. 7) was produced, representing a variety of interactions between server-side and in-vehicle components. As can be clearly noticed, this diagram presents more meaningful insights about the key processes and the information exchange taking place between them. This leads to the next steps in the test process, that is, threat assessment.

5.2. Threat assessment

Threat assessment (as depicted in Fig. 4) has two steps, how each step is performed against Uptane Framework xplained in the following subsections separately.

5.2.1. Threat enumeration

While the sequence diagram in Fig. 7 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., Fig. 2 and Fig. 7), the DFD shown in Fig. 8 was constructed. Note that the data flow diagram (Fig. 8) 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 Fig. 8, both the server-side and the client-side components are surrounded by rectangles, signifying the *trust boundaries*. Three different types of data flows

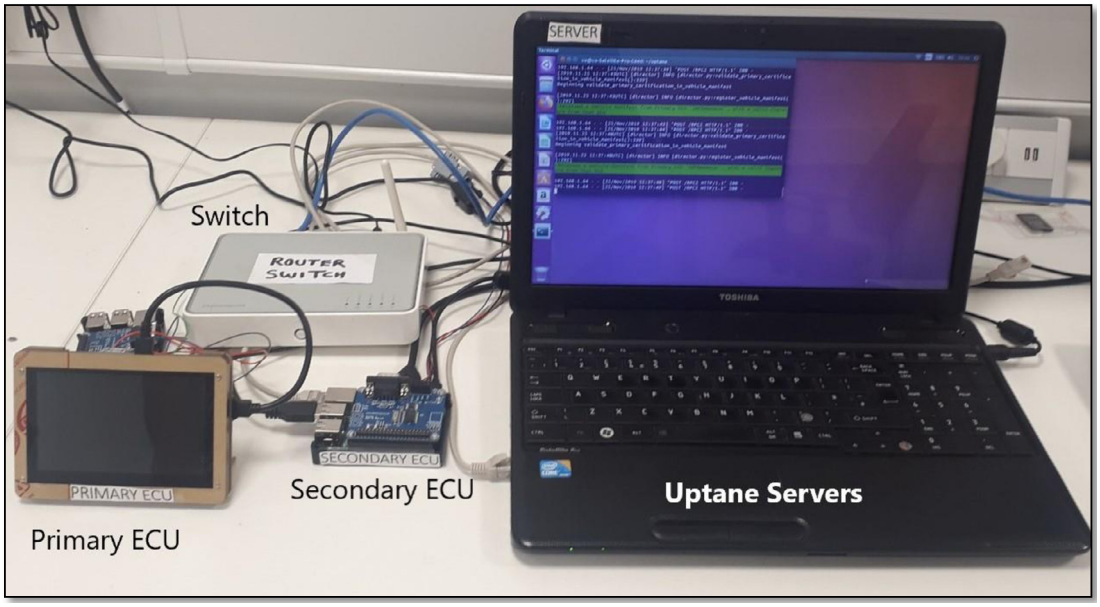


Fig. 6. The Testbed for OTA Updates Security Testing.

Table 4
A summary of the hardware and software components used for building the security testing environment shown in the Fig. 6.

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	–

have been used to indicate the communication types used between different components. Communication between update server and Telematics Control Unit (TCU) uses HTTP protocol, while CAN protocol has been used between in-vehicle components. The third type of data flow is command, depicting communication flows between server-side components (i.e., Image repository, Director repository, Inventory Database etc.), assuming they reside on the same physical system.

The report generated by the TMT tool identifies 53 different types of threats (a summary is presented in the Table 5) that could potentially be used to compromise the security of OTA update system in a number of different ways. In addition to the title of the threat, Table 5 also shows relevant 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 can potentially be sniffed by attackers. Similarly, TMT identified at least seven different vulnerable points that can be used for unauthorised download of updates. The pie chart in Fig. 9 provides an overview of the threats identified, showing the total number of threats in each category of STRIDE.

5.2.2. Threat modeling

Potential threats identified by the TMT in preceding step, do not provide a complete picture of how the attacker would actually materialize those threats. Therefore, in order to identify different ways the attackers can employ, attack trees are constructed, as detailed below.

5.2.3. Attack-tree construction

This subsection deals with the process of constructing attack trees based on the threats identified in prior steps. Only a subset of the threats is included in the experiments, as this strategy facilitates the effective demonstration of the application of systematic security testing approach with appropriate level of details.

The attack trees for various selected threats enumerated by TMT (see Table 5) are constructed, followed by some known threats as published in the literature. Recall that 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 5 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. Brainstorming along with expert discussion sessions were held for identifying subgoals (intermediate nodes) and action steps (leaf nodes). Where necessary and applicable, online resources for the identification of relevant attack techniques used by malicious entities were also consulted. For constructing each attack tree, each selected threat scenario was assigned as the root node of the attack tree. Each attack tree was refined and populated 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. The attack trees constructed following the steps outlined above are presented in the subsequent section (see Figs. 10 to 12). As can be noticed that only three of the total of 15 attack trees are presented in the article, whereas a summary of the other relevant threats have been provided. 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 Fig. 13.

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 analysing 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 the testbed as detailed earlier. Results of each experimental attack performed along with the test case details are given in a separate section later in the article. In addition to the test scripts, each test result outlines the

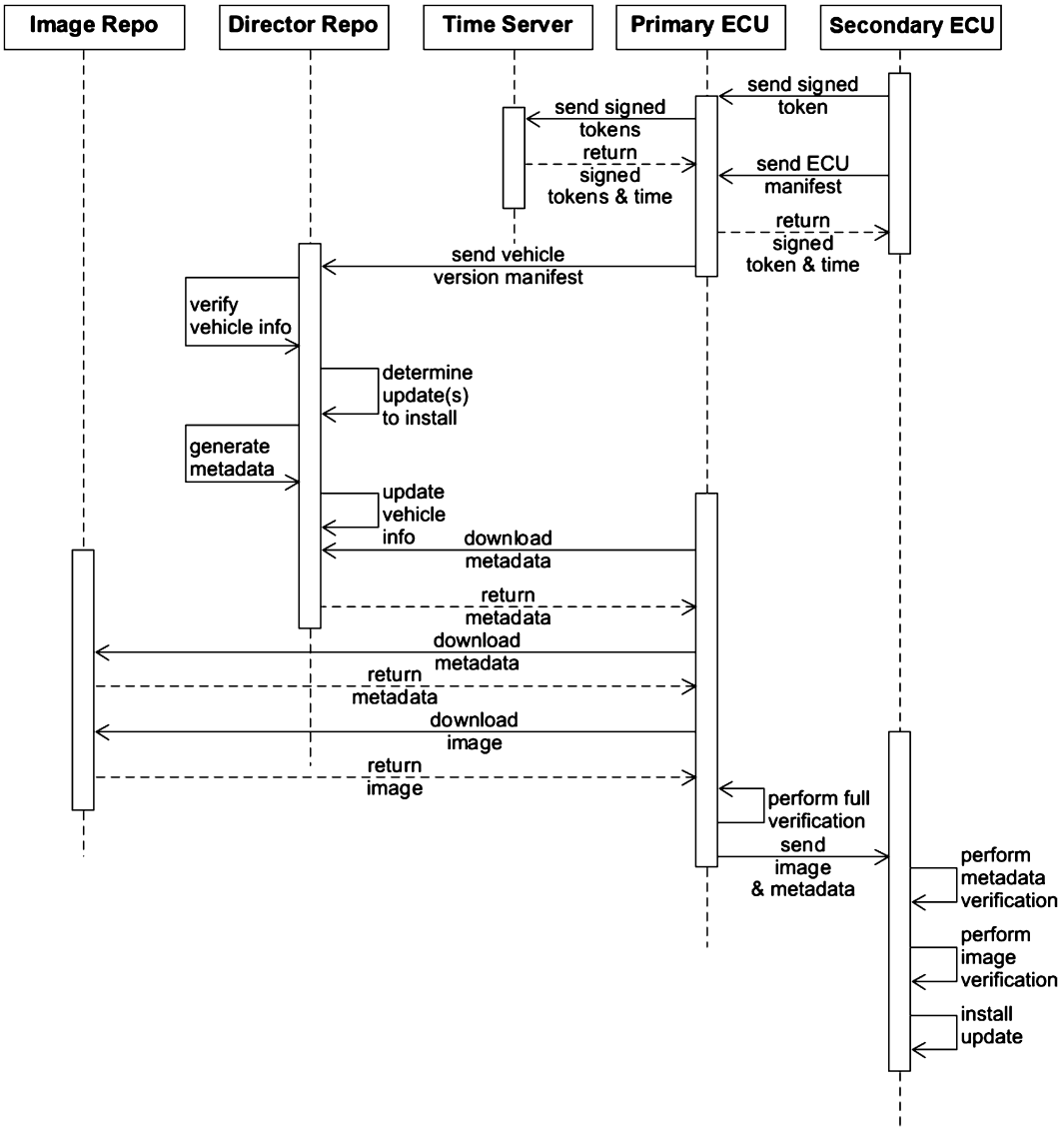


Fig. 7. Uptane Framework Sequence Diagram depicting Over-The-Air Update interactions between Uptane OTA Server- Side and In-Vehicle Primary and Secondary ECUs.

outcome of the test (i.e., the impact of the attack on the system) and resultant system behaviour. In what follows, the process for constructing the attack trees is outlined.

Threat 6 - Updates Could Be Downloaded

The main objective of this attack is to download firmware images from Uptane repositories directly in order to read sensitive/confidential and proprietary information (this threat belongs to the category Read Updates), which can later be used to craft and launch destructive attacks against connected cars. As shown in the Fig. 10, in addition to the main goal, the attack tree shows two different ways (subgoals) for downloading the images from the Uptane servers: either downloading them from Image repository or from Director repository.

However, the two steps for downloading the firmware image from either of the repositories are identical. This 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 order as depicted by the arrow) subtrees, each depicting a possible way used by the intruder for downloading the image from the servers.

Threat 7 - Data Flow Sniffing

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 5, there are total nine occurrences of this threat, each corresponding to one of the data flows in the DFD in Fig. 8. This indicates that the TMT identifies all these data flows vulnerable to this threat. Only one instance of this threat has been included in the testing, since the process and the end results will be identical - providing no further insights. The **SAND** attack tree in Fig. 11 provides a graphical overview of the threat and associated actions. Notice that *Data Flow Sniffing* represents the main goal of the attack tree, with four different actions (leaf nodes) identified for accomplishing the main goal: 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. This and the preceding threats both have the

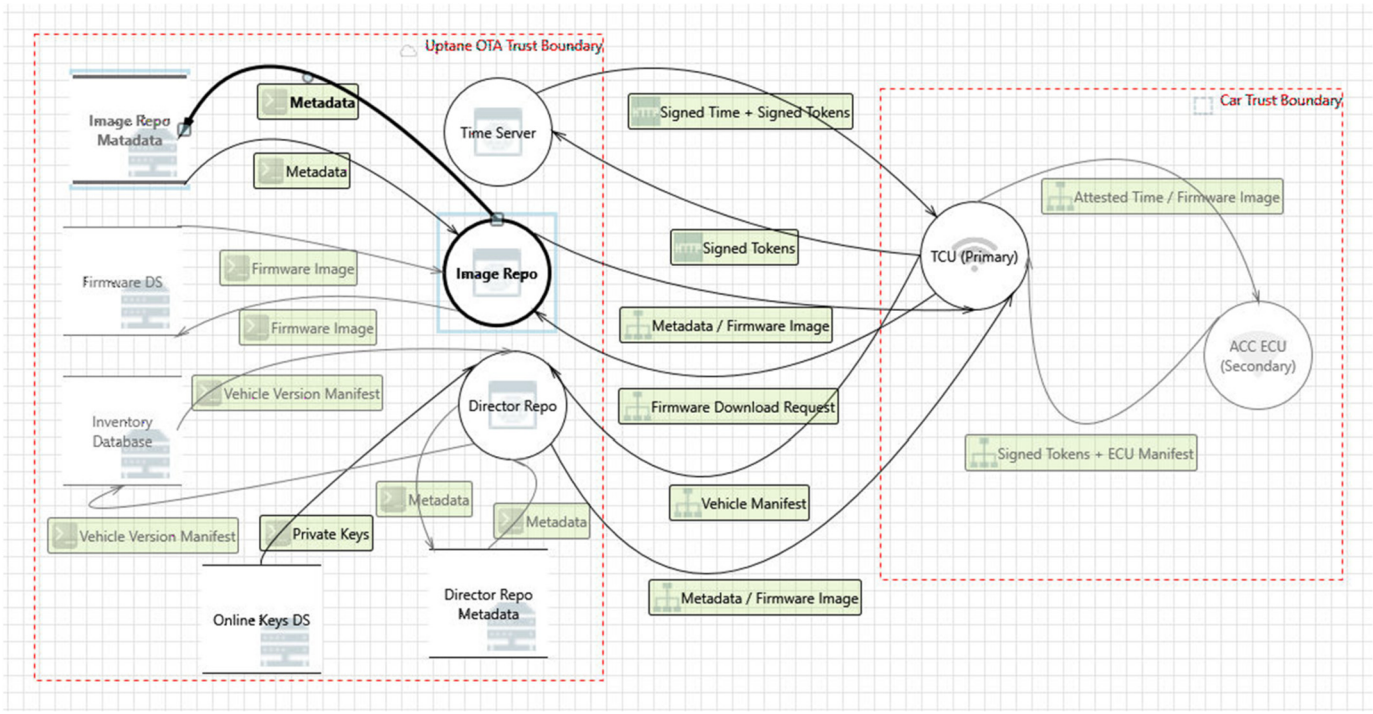


Fig. 8. Data Flow Diagram (created using Threat Modeling Tool and a template from NCC Group) of the Uptane Framework Backend Servers (reference implementation) and in-vehicle components.

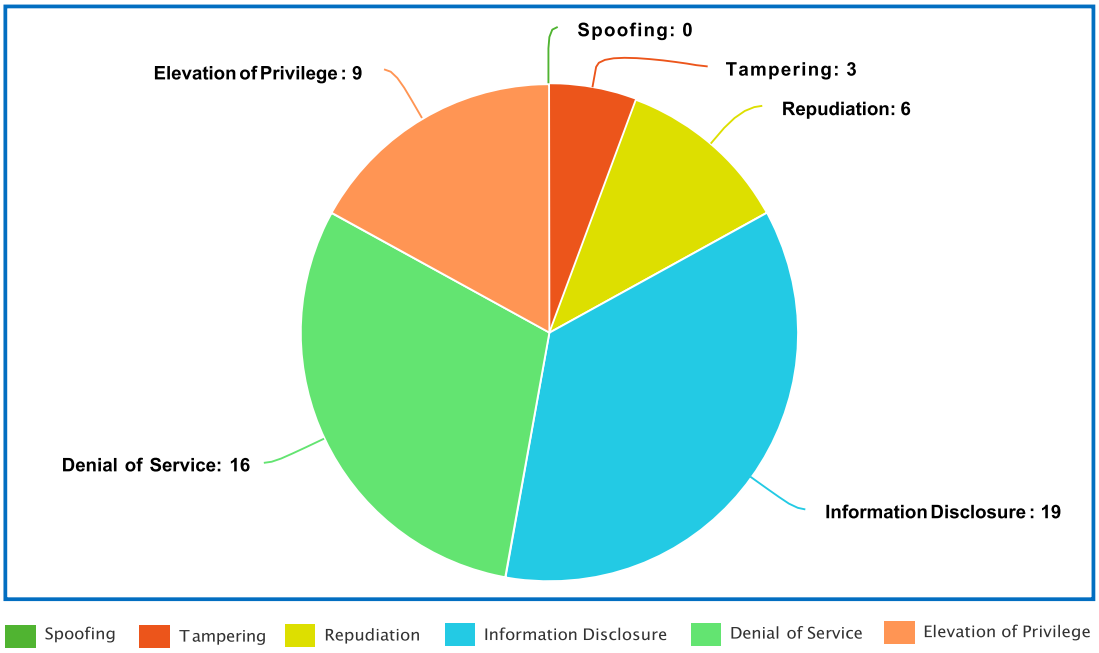


Fig. 9. Graphical overview of the number of identified threats to OTA update system in each category of STRIDE threat classification model.

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

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. 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 Fig. 12, 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.

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 ad-

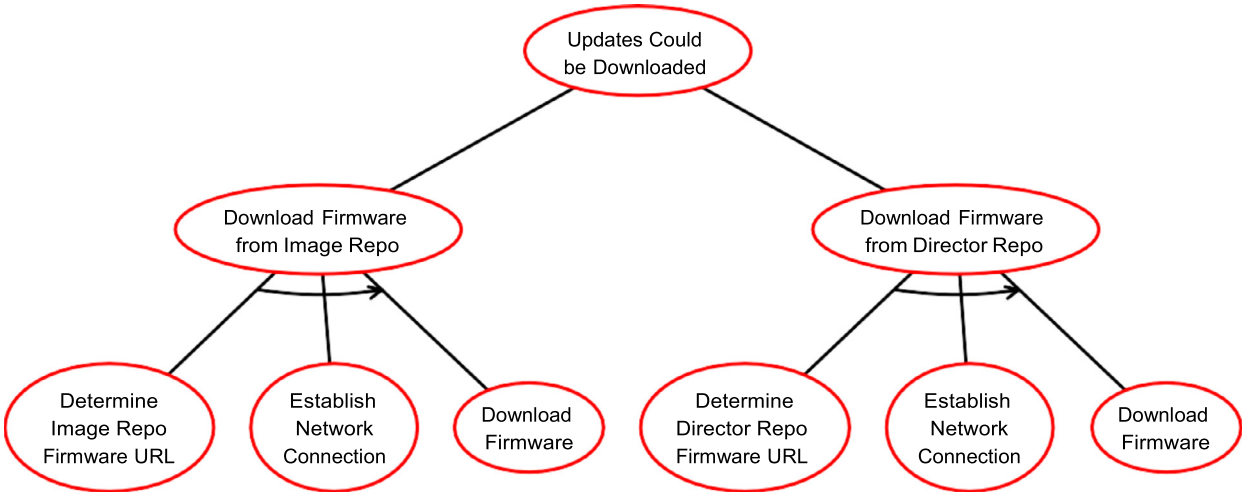


Fig. 10. Attack Tree - Updates Could be Downloaded: This attack tree represents Threat 6 enlisted in Table 5 that involves downloading firmware images from Uptane repositories without any authentication.

Table 5
A summary of the threats identified by the tool. Threats marked with * have been included in the experimentation, whereas ** indicates that more than one variants of the threat have been considered.

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 Attacks	26	**Endless Data Attack	N/A
	27	*Rollback Attack	N/A
	28	*Mix and Match Attack	N/A

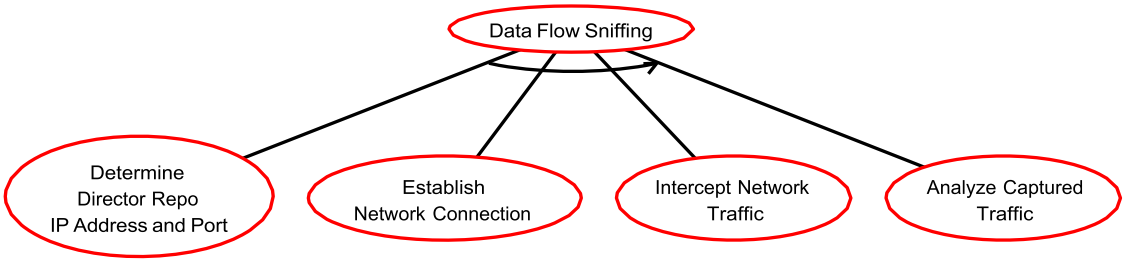


Fig. 11. Attack Tree - Data Flow Sniffing: This attack tree represents Threat 7 as enlisted in Table 5 that involves monitoring and capturing information exchange between Uptane servers and clients.

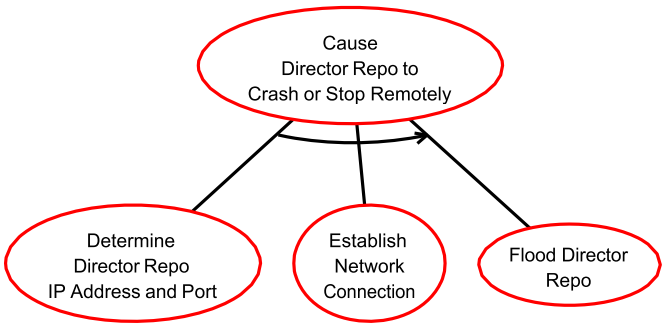


Fig. 12. Attack Tree - Cause Director Repo to Crash or Stop Remotely: This attack tree represents the Threat 9 as enlisted in Table 5 that involves blocking the delivery of updates to the ECUs.

dress and port number of the Director repository. As this experimentation relies on the reference implementation of the Uptane Framework, this information was obtained by running the reference implementation in the 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 Fig. 12, 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.

In a similar way, attack trees involving DoS attacks on Image Repo (Threat 10), Primary ECU (Threat 11), Time Server (Threat 12) as well as others involving compromising Uptane repositories/components (i.e., Threats 21.1 - 3, 22.1 - 2, 26.1 - 2, 27, and 28) were constructed (however, in order to conserve space, those attack trees have not been presented in the article) for the experimental attacks.

The Overall Attack Tree

Once all the attack trees for the selected threats were constructed, they were combined into an overall attack tree for the reference implementation of the Uptane Framework. This overall tree is presented in the Fig. 13. Subtrees have been organised into the STRIDE threat categories.

It is worth mentioning here that only the main goals of the trees have been included, 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 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.

5.3. Results

In this section, results of the automotive OTA security-testing experimentation are presented. Each attack tree was analysed by the 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 15 different test cases and generated relevant test scripts. Seven of these security test cases

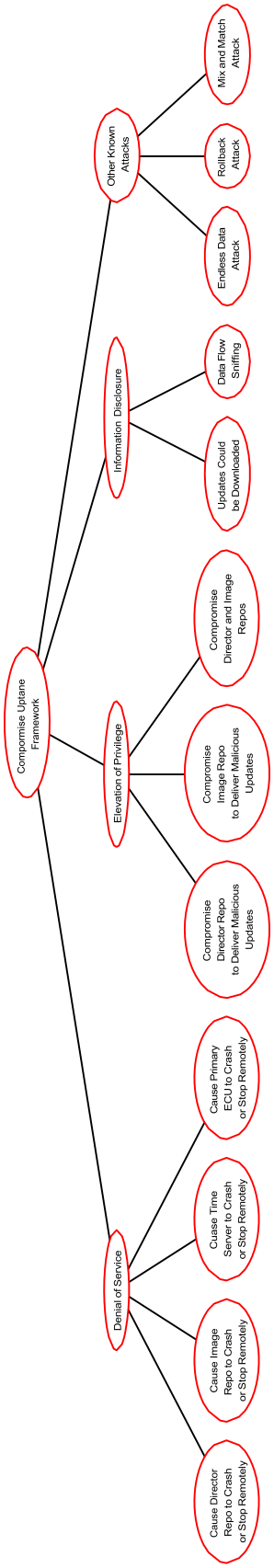


Fig. 13. Attack Tree - Compromise Uptane Framework: This figure presents a comprehensive overview of all the threats (compromising Uptane Framework) included in the experimentation by combining all attack trees. Note that the attack trees have been organised into different STRIDE threat categories. Leaf nodes are nod included in this diagram.

Table 6
The results of the analysis of Threats 6 and 7.

Threat 6 - Updates Could Be Downloaded	Threat 7 - Data Flow Sniffing
Applicable Assumptions: None.	*Applicable Assumptions: None.*
Actions: (Subgoal: Download Firmware from Image Repo) 1. Determine Firmware Image URL on Image Repo 2. Establish Network Connection 3. Download Firmware (Subgoal: Download Firmware from Director Repo) 1. Determine Firmware Image URL on Director Repo 2. Establish Network Connection 3. Download Firmware	Actions: 1. Determine Director Repo IP Address and Port 2. Establish Network Connection 3. Intercept Network Traffic Between Uptane Servers and Clients 4. Analyse Captured Network Traffic
Test Outcome: The firmware images were successfully downloaded from both the Image repository and Director repository using simple HTTP calls without any authentication/authorization. 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. Status: FAIL

Table 7
The results of the analysis of Threats 9 and 10.

Threat 9 - Cause the Director Repository to Crash or Stop Remotely	Threat 10 - Cause Image Repository to Crash or Stop Remotely
Applicable Assumptions: None.	*Applicable Assumptions: None.*
Actions: 1. Determine the IP Address and Port Number of the Director Repo 2. Establish Network Connection 3. Flood Director Repo	Actions: 1. Determine the IP Address and Port Number of the Image Repo 2. Establish Network Connection 3. Flood Image Repo
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). 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. Status: FAIL

failed after successful execution of the test scripts; detailed results of these tests are presented in the following subsections.

Threat 6: Updates Could Be Downloaded

As indicated in the Table 6 (left column), 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 clients.

Threat 7: Data Flow Sniffing

Table 6 (right column) 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.

Threat 9: Cause Director Repo to Crash or Stop Remotely

Table 7 (left column) 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 functioning of the entire update process, as no further actions will succeed. In order to restore the normal operation, the servers had to be rebooted.

Threat 10: Cause Image Repository to Crash or Stop Remotely

As shown in the Table 7 (right column), 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.

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. As can be seen in Table 8 (left column), 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 8 (right column) 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 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.

Threat 21.1: Compromise Director Repository for Sending Malicious Updates

The Table 9 (left column) presents the outcome of the attack aiming at sending a malicious firmware image to the client. The

Table 8
The results of the analysis of Threats 11 and 12.

Threat 11 - Cause the Primary ECU (TCU) to Crash or Stop Remotely	Threat 12 - Cause the Time Server to Crash or Stop Remotely
Applicable Assumptions: None.	*Applicable Assumptions: None.*
Actions: 1. Determine the IP Address and Port Number of the Primary ECU 2. Establish Network Connection 3. Flood Primary ECU	Actions: 1. Determine the IP Address and Port Number of the Time Server 2. Establish Network Connection 3. Flood Time Server
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.	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.
Status: FAIL	Status: FAIL

Table 9
The results of the analysis of Threats 21.1 and 21.2.

Threat 21.1 - Compromise Director Repository in Order to Deliver Malicious Update (without compromised keys)	Threat 21.2 - Compromise Director Repository in Order to Deliver Malicious Update (with compromised keys)
Applicable Assumptions: A1 and A3 - See Table 3	*Applicable Assumptions: A1, A2, and A3 - See Table 3*
Actions: 1. Add Malicious Contents to the Firmware Image 2. Add Firmware Image to Director Repo	Actions: 1. Add Malicious Contents to the Firmware Image 2. Add Firmware Image to Director Repo 3. Generate Signed Metadata for the Firmware
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.	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.*
Status: PASS	Status: PASS

Primary ECU did not download the compromised firmware image from Director repository after it found a bad hash value while performing 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.

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

As shown in the Table 9 (right column), the Primary ECU rejected to download the image from the server after it detected an anomaly in the metadata from Director and Image repositories. Unlike the previous attempt, the firmware metadata was signed with valid keys. Even though the metadata, received from both repositories, were correct; however, since inconsistent hash values were received from each of the repositories, Primary ECU discarded the update.

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

The information presented in Table 10 (left column) 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.

Threat 22.1: Compromise Image Repository to Send Malicious Updates

Table 10 (right column) 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.

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 11 (left column), 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.

Threats 26.1 and 26.2: Endless Data Attack

Table 11 (right column) and Table 12 (column left) show the results of two variations of Endless Data Attack, with the goal to inundate the Primary ECU with large amount of data to affect its functionality. In the first scenario, we appended additional

Table 10
The results of the analysis of Threats 21.3 and 22.1.

Threat 21.3 - Compromise Image and Director Repositories in Order to Deliver Malicious Updates (with compromised keys)	Threat 22.1 - Compromise Image Repository in Order to Deliver Malicious Update (without compromised keys)
Applicable Assumptions: A1, A2, and A3 - See Table 3.	*Applicable Assumptions: A1 and A3 - See Table 3.*
Actions: 1. Add Malicious Contents to the Firmware Image 2. Add Firmware Image to Image Repo 3. Generate Signed Image Repo Metadata 4. Add Firmware Image to Director Repo 5. Generate Signed Director Repo Metadata	Actions: 1. Add Malicious Contents to the Firmware Image 2. Add Firmware Image to Director Repo
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.* Status: FAIL	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. Status: PASS

Table 11
The results of the analysis of Threats 22.2 and 26.1.

Threat 22.2 - Compromise Image Repository in Order to Deliver Malicious Updates (with compromised keys)	Threat 26.1 - Endless Data Attack (with appended contents)
Applicable Assumptions: A1, A2, and A3 - See Table 3.	*Applicable Assumptions: A1 and A3 - See Table 3.*
Actions: 1. Add Malicious Contents to the Firmware Image 2. Add Firmware Image to Image Repo 3. Generate Signed Metadata for Firmware Image	Actions: 1. Append Additional Contents to Firmware on Image Repo 2. Add Image to Image Repo 3. Add Firmware to Director Repo
Test Outcome: The Primary did not proceed with the download and showed 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.* 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. Status: PASS

Table 12
The results of the analysis of Threats 26.2 and 27.

Threat 26.2 - Endless Data Attack (with overwritten or inserted contents)	Threat 27 - Rollback Attack
Applicable Assumptions: A1 and A3 - See Table 3.	*Applicable Assumptions: A1 and A3 - See Table 3.*
Actions: 1. Locate and Open the Target Image File 2. Insert Additional Data into the File 3. Save the Changes and Close the Target File	Actions: 1. Delete the Currently Trusted Timestamp on the Image Repo 2. Place the Outdated Timestamp on the Image Repo 3. Delete the Currently Trusted Timestamp on the Director Repo 4. Place the Outdated Timestamp on the Director Repo
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 *Bad-HashError* error messages. 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.* Status: PASS

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/updates.

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 12 (column right), summarises the key steps (actions) and the result of the test. The Primary detected that it was sent an older version of the timestamp; there-

Table 13
The results of the analysis of Threat 28 - Mix and Match Attack.

Results: Mix and Match Attack
Applicable Assumptions: A1 and A3 - See Table 3.
Actions: 1. Delete Currently Valid Firmware File on the Image Repo 2. Add an Outdated Version of the Firmware to the Image Repo 3. Delete Currently Valid Firmware File on the Director Repo 4. Add an Outdated Version of the Firmware to Director Repo
Test Outcome: The Primary ECU refused to proceed with the update process by showing *BadHashError* message. Status: PASS

fore, it rejected it and thus the latest version of the update was not rolled back.

Threat 28: Mix and Match Attack

As shown in the Table 13, the older firmware image sent to the Primary was rejected as the accompanied metadata did not contain the correct attributes for this image. Hence, the attack could not be succeeded. This attack intended to install an incompatible version of the firmware in an update package containing other images as well.

6. Discussion

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, step-by-step derivation of test cases, and the automation of security test-case generation and execution. As, derivation of appropriate and effective security test cases is often considered a challenging task, because in addition to the knowledge of potential threats, it requires a clear idea of what to test and where to start [50]. Threat modeling techniques and tools we used allowed us to identify several security threats targeting automotive OTA updates in a systematic and repeatable manner.

6.1. Key findings

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 been performed on the update systems/repositories in the past, which include: mix-and-match attack, rollback attack, and endless data attack. Experimental results of these attacks have shown that the Uptane Framework has strong mechanisms to effectively combat threats involving tampering of updates.

On the other hand, some of our testing 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 going through 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.

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.

6.2. Limitations

While the approach we have used enabled us to systematically identify various threats and derive security test cases, we did not include all those threats in the experimentation for different reasons. First of all, we mainly concentrated on the threats compromising the OTA update procedures at the server-side, rather than considering the vehicle-side exploits (e.g., Reflash the TCU (Primary) Firmware in Order to Send Arbitrary CAN Messages), as they have been extensively explored in prior studies [94,10,47]. Second, we did not consider threats from the Repudiation category of STRIDE model, as these threats independently are not capable of causing any harm/disruption to the Update procedures or system. Third, the threat *Car Could be Tracked* has not been included in the experimentation, since it is not directly relevant/exclusive to OTA update system, and it does not impact the delivery or installation of the updates. 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.

Some attack trees we constructed could be further elaborated to include 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. 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 5 and 9) it reported no Spoofing threats to the Uptane repositories and clients. It is perfectly possible that the attackers may use spoofing as one of the strategies to compromise the updates. In fact, in many of our experiments we assumed that the attackers were spoofing/impersonating Uptane servers.

7. Conclusion

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 systematic threat assessment and model-based 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. 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 this study), a well-defined and structured approach could make a big difference in constructing effective attack trees and security test cases.

Core scientific and technical contributions of this work are outlined below:

1. A systematic threat analysis approach for constructing attack trees;
2. A test-derivation approach using model-based security testing approach based on attack trees;
3. An in-depth, experimental security analysis of the Uptane Framework by applying the systematic threat assessment and security testing approach;
4. The automation of the test case generation and execution by implementing a special-purpose software tool. This powerful software tool is capable of generating and executing security test cases by performing intelligent analysis of the attack-tree structure.

In-depth security analysis and system testing of the Uptane Framework carried out in this study, demonstrating the validity of our approach by enumerating various threats by examining the system model, constructing attack trees from the results of threat enumeration, deriving effective security cases by analysing 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 preceding section show the effectiveness of our approach by revealing unmitigated threats and vulnerabilities of the system. The experimental security attacks crafted from the attack trees, helped 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.

For future work, we aim to automate the attack-tree construction process, which is currently a manual step in our approach. We also plan to evaluate the testing approach on other automotive OTA update systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] 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 December 2019).

[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, UNECE, Jun 2020, http://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-2020-079-Revised.pdf.

[3] Omid Avatefipour, Hafiz Malik, State-of-the-art survey on in-vehicle network communication (can-bus) security and vulnerabilities, arXiv preprint, arXiv: 1802.01725, 2018.

[4] Filipe Breda, Hugo Barbosa, Telmo Morais, Social engineering and cyber security, in: International Technology, Education and Development Conference, vol. 3, 2017, pp. 106–108.

[5] Jeremy Bryans, Hoang Nga Nguyen, Siraj Shaikh, Attack defense trees with sequential conjunction, in: 2019 IEEE HASE, 2019, pp. 247–252.

[6] Jeremy Bryans, Lin Shen Liew, Hoang Nga Nguyen, Giedre Sabaliauskaite, Siraj Shaikh, Fengjun 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.

[7] Ondrej Burkacky, Johannes Deichmann, Benjamin Klein, Klaus Pototzky, Gundbert Scherf, Cybersecurity in automotive: Mastering the challenge, GSA, Mar 2020, https://www.gsaglobal.org/wp-content/uploads/2020/03/Cybersecurity-in-automotive-Mastering-the-challenge.pdf.

[8] Robert Buttigieg, Mario Farrugia, Clyde 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.

[9] Justin Cappos, Justin Samuel, Scott Baker, John 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] Paul Carsten, Todd R. Andel, Mark Yampolskiy, Jeffrey 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.

[11] Christina Chavez, Carlos Lucena, A metamodel for aspect-oriented modeling, in: Workshop on Aspect-Oriented Modeling with UML (AOSD-2002), 2002.

[12] Madeline Cheah, Hoang Nga Nguyen, Jeremy Bryans, Siraj A. Shaikh, Formalising Systematic Security Evaluations Using Attack Trees for Automotive Applications, vol. 10741, Springer International Publishing, 2017, pp. 113–129.

[13] Madeline Cheah, Siraj A. Shaikh, Jeremy Bryans, Paul Wooderson, Building an automotive security assurance case using systematic security evaluations, Comput. Secur. 77 (2018) 360–379, https://doi.org/10.1016/j.cose.2018.04.008.

[14] Madeline Cheah, Siraj A. Shaikh, Olivier Haas, Alastair Ruddle, Towards a systematic security evaluation of the automotive bluetooth interface, Veh. Commun. 9 (2017) 8–18.

[15] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces, in: In USENIX Security Symposium, vol. 4, San Francisco, 2011, pp. 447–462.

[16] Yue Chen, Barry Boehm, Luke 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, p. 280a.

[17] Thomas Chowdhury, Eric Lesiuta, Kerianne Rikley, Chung-Wei Lin, Eunsuk Kang, BaekGyu Kim, Shinichi Shiraishi, Mark Lawford, Alan Wassyng, Safe and secure automotive over-the-air updates, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2018, pp. 172–187.

[18] David James Coe, Jeffrey Kulick, Aleksandar Milenkovic, Letha Etzkorn, Virtualized in-situ software update verification: verification of over-the-air automotive software updates, IEEE Veh. Technol. Mag. (2019).

[19] Ankush Dadwal, Hironori Washizaki, Yoshiaki Fukazawa, Takahiro Iida, Masashi Mizoguchi, Kentaro Yoshimura, Prioritization in automotive software testing: systematic literature review, in: QuASoQ@APSEC, 2018, pp. 52–58.

[20] Husein Dakroub, Robert Cadena, Analysis of software update in connected vehicles, SAE Int. J. Passeng. Cars, Electron. Electr. Syst. 7 (2014-01-0256) (2014) 411–417.

[21] Debian, Debian investigation report after server compromises, Debian, Dec 2003, https://www.debian.org/News/2003/20031202.

[22] Folker Den Braber, Ida Hogganvik, M. Soldal Lund, Ketik Stølen, Fredrik Vraalsen, Model-based security analysis in seven steps—a guided tour to the coras method, BT Technol. J. 25 (1) (2007) 101–117.

[23] Jürgen Dürrwang, Johannes Braun, Marcel Rumez, Reiner Kriesten, Alexander Pretschner, Enhancement of automotive penetration testing with threat analyses results, SAE Int. J. Transp. Cybersecur. Priv. 1 (11-01-02-0005) (2018) 91–112.

[24] Panagiotis Efstathiadis, Anna Karanika, Nestoras Chouliaras, Leandros Maglaras, Ioanna Kantzavelou, Smart cars and over-the-air updates, in: Cybersecurity Issues in Emerging Technologies, CRC Press, 2021, pp. 137–152.

[25] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, Harold Ossher, Discussing aspects of aop, Commun. ACM 44 (10) (2001) 33–38.

[26] Christopher E. Everett, Damon McCoy, Octane (open car testbed and network experiments): Bringing cyber-physical security research to researchers and students, Presented as part of the 6th Workshop on Cyber Security Experimentation and Test, 2013.

[27] Michael Felderer, Matthias Büchler, Martin Johns, Achim D. Brucker, Ruth Breu, Alexander Pretschner, Security Testing: A Survey, Advances in Computers, vol. 101, Elsevier, 2016, pp. 1–51.

[28] Michael Felderer, Philipp Zech, Ruth Breu, Matthias Büchler, Alexander Pretschner, Model-based security testing: a taxonomy and systematic classification, Softw. Test. Verif. Reliab. 26 (2) (2016) 119–148.
[29] Daniel S. Fowler, Jeremy Bryans, Madeline Cheah, Paul Wooderson, Siraj A. Shaikh, A method for constructing automotive cybersecurity tests, a CAN fuzz testing example, in: 2019 IEEE QRS-C, 2019, pp. 1–8.
[30] Daniel S. Fowler, Madeline Cheah, Siraj Ahmed Shaikh, Jeremy Bryans, Towards a testbed for automotive cybersecurity, in: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2017, pp. 540–541.
[31] Daniel S. Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, Paul 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.
[32] Dan Goodin, Attackers sign malware using crypto certificate stolen from opera software, Ars Technica, Jun 2013, https://arstechnica.com/information-technology/2013/06/attackers-sign-malware-using-crypto-certificate-stolen-from-opera-software/.
[33] Morey J. Haber, Brad Hibbert, Privileged attack vectors: building effective cyber-defense strategies to protect organizations, Apress (2017).
[34] Christopher Hadnagy, Social Engineering: The Science of Human Hacking, John Wiley & Sons, 2018.
[35] Azeem Hafeez, Hafiz Malik, Omid Avatefipour, Prudhvi Raj Rongali, Shan Zehra, Comparative study of can-bus and flexray protocols for in-vehicle communication, Technical report, SAE Tech. Paper, 2017.
[36] Subir Halder, Amrita Ghosal, Mauro Conti, Secure over-the-air software updates in connected vehicles: a survey, Comput. Netw. 178 (2020) 107343.
[37] Tina Hampton, Know the term: sota/fota, AT&T Business, https://www.business.att.com/learn/tech-advice/know-the-term-sota-fota.html.
[38] John Heneghan, Siraj Ahmed Shaikh, Jeremy Bryans, Madeline Cheah, Paul Wooderson, Enabling security checking of automotive ECUs with formal CSP models, in: 2019 IFIP DSN-W, IEEE, 2019, pp. 90–97.
[39] Tobias Hoppe, Stefan Kiltz, Jana 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.
[40] Chris Isidore, Gm's total recall cost: $4.1 billion, 2020, https://money.cnn.com/2015/02/04/news/companies/gm-earnings-recall-costs/index.html.
[41] Koteswara Ivaturi, Lech Janczewski, A taxonomy for social engineering attacks, in: International Conference on Information Resources Management, Centre for Information Technology, Organizations, and People, 2011, pp. 1–12.
[42] Sasan Jafarnejad, Lara Codeca, Walter Bronzi, Raphael Frank, Thomas Engel, A car hacking experiment: when connectivity meets vulnerability, in: 2015 IEEE Globecom Workshops (GC Wkshps), IEEE, 2015, pp. 1–6.
[43] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović, Rolando Trujillo-Rasua, Attack Trees with Sequential Conjunction, vol. 455, Springer International Publishing, 2015, pp. 339–353.
[44] Adi Karahasanovic, Pierre Kleberger, Magnus Almgren, Adapting threat modeling methods for the automotive industry, in: Proceedings of the 15th ESCAR Conference, 2017, pp. 1–10.
[45] Ho-Yeon Kim, Young-Hyun Choi, Tai-Myoung 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.
[46] Kirsten Korosec, How automakers will save $35 billion by 2022, Fortune, Sep 2015, https://fortune.com/2015/09/04/ihs-auto-software/.
[47] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al., Experimental security analysis of a modern automobile, in: 2010 IEEE Symposium on Security and Privacy, IEEE, 2010, pp. 447–462.
[48] Trishank Karthik Kuppusamy, Akan Brown, Sebastien Awwad, Damon McCoy, Russ Bielawski, Cameron Mott, Sam Lauzon, André Weimerskirch, Justin Cappos, Uptane: securing software updates for automobiles, in: 14th ESCAR Europe, 2016.
[49] Jiajia Liu, Shubin Zhang, Wen Sun, Yongpeng Shi, In-vehicle network attacks and countermeasures: challenges and future directions, IEEE Netw. 31 (5) (2017) 50–58.
[50] Yanguo Liu, Issa Traore, Systematic security analysis for service-oriented software architectures, in: IEEE International Conference on e-Business Engineering (ICEBE'07), IEEE, 2007, pp. 612–621.
[51] Xin Luo, Richard Brody, Alessandro Seazzu, Stephen Burd, Social engineering: the neglected human factor for information security management, Inf. Res. Manag. J. 24 (3) (2011) 1–8.
[52] Zhendong Ma, Christoph Schmittner, Threat modeling for automotive security analysis, Adv. Sci. Technol. Lett. 139 (2016) 333–339.
[53] Shahid Mahmood, Alexy Fouillade, Hoang Nga Nguyen, Siraj 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.
[54] Karim Mansour, Wael Farag, Mohamed 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.
[55] Sjouke Mauw, Martijn Oostdijk, Foundations of Attack Trees, vol. 3935, Springer Berlin Heidelberg, 2005, pp. 186–198.
[56] Kathiresh Mayilsamy, Neelaveni Ramachandran, Vismitha Sunder Raj, An integrated approach for data security in vehicle diagnostics over Internet protocol and software update over the air, Comput. Electr. Eng. 71 (2018) 578–593.
[57] Sahar Mazloom, Mohammad Rezaeirad, Aaron Hunter, Damon McCoy, A security analysis of an in-vehicle infotainment and app platform, in: 10th USENIX Workshop on Offensive Technologies (WOOT 16), 2016.
[58] Charlie Miller, Chris Valasek, A survey of remote automotive attack surfaces, Black Hat USA 2014 (2014) 94.
[59] Charlie Miller, Chris Valasek, Remote exploitation of an unaltered passenger vehicle, Black Hat USA 2015 (2015) 91.
[60] Jürgen Mössinger, Software in automotive systems, IEEE Softw. 27 (2) (2010) 92–94.
[61] Subhojeet Mukherjee, Hossein Shirazi, Indrakshi Ray, Jeremy Daily, Rose Gamble, Practical dos attacks on embedded networks in commercial vehicles, in: International Conference on Information Systems Security, Springer, 2016, pp. 23–42.
[62] Kate Munro, Deconstructing flame: the limitations of traditional defences, Comput. Fraud Secur. 2012 (10) (2012) 8–11.
[63] Hoang Nga Nguyen, Siamak Tavakoli, Siraj Ahmed Shaikh, Oliver Maynard, Developing a QRNG ECU for automotive security: experience of testing in the real-world, in: 2019 IEEE ICSTW, 2019, pp. 61–68.
[64] Phu H. Nguyen, Shaukat Ali, Tao Yue, Model-based security engineering for cyber-physical systems: a systematic mapping study, Inf. Softw. Technol. 83 (2017) 116–135.
[65] Dennis K. Nilsson, Ulf E. Larson, Simulated attacks on can buses: vehicle virus, in: IASTED International Conference on Communication Systems and Networks (AsiaCSN), 2008, pp. 66–72.
[66] Jin Seo Park, Daehyun Kim, Seokmin Hong, Hyunjung Lee, EuiJung Myeong, Case study for defining security goals and requirements for automotive security parts using threat modeling, Technical report, SAE Technical Paper, 2018.
[67] Jonathan Petit, Steven E. Shladover, Potential cyberattacks on automated vehicles, IEEE Trans. Intell. Transp. Syst. 16 (2) (2014) 546–556.
[68] Jonathan Petit, Bas Stottelaar, Michael Feiri, Frank Kargl, Remote attacks on automated vehicles sensors: experiments on camera and lidar, Black Hat Europe 11 (2015) 2015.
[69] PTES, Penetration testing and execution standard, 2014, http://www.pentest-standard.org/index.php/Main_Page. (Accessed 6 December 2019).
[70] Caleb Riggs, Carl-Edwin Rigaud, Robert Beard, Tanner Douglas, Karim Elish, A survey on connected vehicles vulnerabilities and countermeasures, J. Traffic Logist. Eng. 6 (1) (2018).
[71] P. Ruissen, R. Vloothuis, Insecurities Within Automatic Update Systems v1.16, 2007.
[72] Joe Rumer, Mark Burnett, Software-over-the-air (sota): an automotive accelerator, BearingPoint Institute, 2017, https://www.bearingpoint.com/files/BEI008-07-ICL_SOTA-Software-over-the-air.pdf?download=0&itemId=473686.
[73] Juha Saarinen, Extortionist continues to scan for exposed git creds, IT News, May 2019, https://www.itnews.com.au/news/extortionist-continues-to-scan-for-exposed-git-creds-525178.
[74] Eduardo dos Santos, Andrew Simpson, Dominik Schoop, A formal model to facilitate security testing in modern automotive systems, arXiv preprint, arXiv:1805.05520, 2018.
[75] Andreas Schaad, Tobias Reski, "Open Weakness and Vulnerability Modeler" (OVVL): an updated approach to threat modeling, in: Proceedings of the 16th International Joint Conference on E-Business and Telecommunications - SECRYPT, 2019, pp. 417–424.
[76] Ina Schieferdecker, Juergen Grossmann, Martin Schneider, Model-based security testing, arXiv preprint, arXiv:1202.6118, 2012.
[77] Christoph Schmittner, Georg Macher, Automotive cybersecurity standards-relation and overview, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2019, pp. 153–165.
[78] Bruce Schneier, Secrets and Lies: Digital Security in a Networked World, John Wiley & Sons, 2015.
[79] Adam Shostack, Experiences threat modeling at Microsoft, MODSEC@MoDELS 2008 (2008).
[80] Adam Shostack, Threat Modeling: Designing for Security, John Wiley & Sons, 2014.
[81] Marco Steger, Ali Dorri, Salil S. Kanhere, Kay Römer, Raja Jurdak, Michael Karner, Secure wireless automotive software updates using blockchains: a proof of concept, in: Advanced Microsystems for Automotive Applications 2017, Springer, 2018, pp. 137–149.
[82] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, Youssef 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.
[83] Gu Tian-yang, Shi Yin-Sheng, Fang You-yuan, Research on software security testing, World Acad. Sci., Eng. Technol. 70 (2010) 647–651.

[84] Tsuyoshi Toyama, Takuya Yoshida, Hisashi Oguma, Tsutomu Matsumoto, PASTA: portable automotive security testbed with adaptability, Blackhat Europe, London, 2018.
[85] Joao P. Trovao, An overview of automotive electronics [automotive electronics], IEEE Veh. Technol. Mag. 14 (3) (2019) 130–137.
[86] Tony UcedaVelez, Marco M. Morana, Risk Centric Threat Modeling, Wiley Online Library, 2015.
[87] 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. (Accessed 6 December 2019).
[88] Mark Utting, Alexander Pretschner, Bruno Legeard, A taxonomy of model-based testing approaches, Softw. Test. Verif. Reliab. 22 (5) (2012) 297–312.
[89] Alexandr Vasenev, Florian Stahl, Hayk Hamazaryan, Zhendong Ma, Lijun Shan, Joerg Kemmerich, Claire Loiseaux, Practical security and privacy threat analysis in the automotive domain: Long term support scenario for over-the-air updates, 2019.
[90] Armin Wasicek, Patricia Derler, Edward 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.
[91] Imano Williams, Xiaohong Yuan, Evaluating the effectiveness of Microsoft threat modeling tool, in: Proceedings of the 2015 Information Security Curriculum Development Conference, 2015, pp. 1–6.
[92] Davey Winder, Hackers trick thousands into downloading dangerous 'Google chrome update', Mar 2020, https://www.forbes.com/sites/daveywinder/2020/03/26/warning-hackers-trick-thousands-into-downloading-dangerous-google-chrome-update/.
[93] Stijn Winsen, Threat modelling for future vehicles: on identifying and analysing threats for future autonomous and connected vehicles, Master's thesis, University of Twente, 2017.
[94] Wei 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.
[95] Kim Zetter, Mar 2019, https://www.vice.com/en/article/pan9wn/hackers-hijacked-asus-software-updates-to-install-backdoors-on-thousands-of-computers.
[96] Xi Zheng, Lei Pan, Hongxu Chen, Rick Di Pietro, Lynn Batten, A testbed for security analysis of modern vehicle systems, in: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, 2017, pp. 1090–1095.
[97] Mohd Azrin Mohd Zulkefli, Pratik Mukherjee, Zongxuan Sun, Jianfeng Zheng, Henry X. Liu, Peter Huang, Hardware-in-the-loop testbed for evaluating connected vehicle applications, Transp. Res., Part C, Emerg. Technol. 78 (2017) 50–62.