



Ain Shams University
Faculty of Computer & Information Sciences
General Program



SecOTA: Universal Secure Flash-Over-The-Air Framework for Embedded and IoT Systems

This documentation is submitted as required for
the degree of Bachelor's in General Program, Computer Systems Department
Computer and Information Sciences
Ain Shams University

By

<i>Omar Hesham Ali Elshopky</i>	<i>Ahmed Hossam Elden Eltaher</i>
<i>Mahmoud Ahmed Nabil Mohamed</i>	<i>Eslam Ashraf Ali Gomaa</i>
<i>Enas Ahmed Abdelazem Rizk</i>	<i>Rana Anwar Ibrahim Abd Elkahlik</i>

Supervisors

Dr. Karim Emara
Computer Systems Department
Faculty of Computer and Information Sciences,
Ain Shams University

T.A. Fatma Elwasify
Computer Systems Department
Faculty of Computer and Information Sciences,
Ain Shams University

Cairo, June 2024

Acknowledgments

All praise and thanks to ALLAH, who bestowed upon us the ability, experience, and support to complete this work. We hope this work will be accepted from us and prove beneficial for future research and products.

We are deeply grateful to those who helped us through the long hard nights, our parents, families, and friends, whose unwavering support sustained us throughout our years of study. We hope to reciprocate their kindness.

Our sincerest gratitude goes to our supervisors, Dr. Karim Emara and T.A. Fatma Elwasify, whose patience, knowledge, and experience were invaluable throughout our thesis. Thank you for the countless hours spent in meetings, reviewing work, and brainstorming.

We also extend our heartfelt thanks to our mentor, Eng. Mark Attia, who generously gave more than his work hours to share his professional experience and insights from top-tier FOTA-related projects, which made this work as professional as it is.

Finally, we would like to thank everyone who offered us support and encouragement.

Abstract

This thesis addresses the lack of secure, easy-to-deploy on-premise flash over the air (FOTA) frameworks that consider physical security, providing companies with an efficient way to stop the financial losses caused by recalls, due to software failures, emerging security threats, or rapid development bugs, and save time developing in-house FOTA solutions. Our project develops a universal and secure FOTA comprehensive framework for embedded and IoT systems, implementing the Uptane standard, which is based on The Update Framework (TUF). TUF is primarily used for software updates by companies like Microsoft, Google, and Amazon, while Uptane adapts it to the environment and capabilities of embedded and IoT devices. The development was conducted based on the ISO 21434 standard, ensuring adherence to automotive cybersecurity guidelines. Our work introduces a specialized Threat Analysis and Risk Assessment (TARA) focusing on client-side attacks, often overlooked in favor of backend attacks. Using both the STRIDE methodology and the MITRE ATT&CK framework for identifying threats and tactics, techniques, and procedures (TTPs), we identified and mitigated risks related to physical and client app attacks on the Electronic Control Unit (ECU). Mitigations include updating deprecated cryptographic algorithms to the latest standards, network Intrusion Prevention Systems (IPS), and process manipulation countermeasures. Our framework, SecOTA, is designed with security by design principles and employs a microservice architecture for compromise resilience. It is ready to deploy universally across a wide range of boards, ensuring a robust and adaptable solution for secure FOTA in embedded and IoT systems.

تتناول هذه الأطروحة نقص الأطر الأمنية وسهولة النشر لتحديث البرامج عبر الهواء (التحديث عن بعد) "FOTA"، التي تأخذ في اعتبارها الأمن السيبراني للأجهزة، وتوفر للشركات طريقة فعالة لتوفير الوقت اللازم لتطوير حلول برمجية تعطي الأجهزة قابلية التحديث عن بعد، وتوقف نزيف الأموال الناتج عن استدعاء المنتجات المتضررة بسبب فشل برمجي، أو ظهور تهديدات أمنية جديدة، أو الأخطاء الناتجة عن التسرع في التطوير لاكتساح السوق في أسرع وقت. يطور مشروعنا إطاراً أمنياً متكاملاً يضيف ميزات التحديث عن بعد "FOTA" للأنظمة الأجهزة المدمجة وأجهزة إنترنت الأشياء، باعتماد معيار *Uptane* المستند إلى إطار تحديث البرمجيات *TUF*. يستخدم *TUF* بشكل أساسي لتحديث البرامج من قبل شركات عالمية مثل *Microsoft*, *Google*, و *Amazon*, بينما يكيف *Uptane* الإطار ليتناسب مع بيئة وقدرات أجهزة الأنظمة المدمجة. تم العمل وفقاً لمعيار "ISO 21434"، مما يضمن الامتثال لإرشادات الأمن السيبراني للمركبات. يقدم عملنا تحليلاً متخصصاً للتهديدات وتقييم المخاطر "TARA"، واضعاً وحدة التحكم الإلكترونية كمركز التحليل ومركزاً على الهجمات من جهة العميل، التي غالباً ما تُغفل لصالح الهجمات من جهة الخوادم. باستخدام منهجيات *STRIDE* وإطار *MITRE ATT&CK* لتحديد التكتيكات والتقنيات والإجراءات التي يمكن أن تهدد النظام، ومن خلاله استطعنا تحديد وتقليل المخاطر المتعلقة بالهجمات الفيزيائية وتطبيقات العميل على وحدة التحكم الإلكترونية، بما في ذلك تحديث خوارزميات التشفير المهجورة إلى أحدث المعايير، وتنفيذ أنظمة منع التسلل الشبكية، وتدابير مكافحة تلاعب العمليات. تم تصميم إطارنا بمبادئ الأمان السيبراني منذ مرحلة التصميم، ويعتمد على معمارية الخوادم المصغرة لمرونة أكبر في مجابهة التهديدات والهجمات. إنه جاهز للنشر وقابل للعمل على نطاق واسع عبر مجموعة واسعة من اللوحات المدمجة، مما يضمن حلاً قوياً وقابلاً للتكيف لمعظم أجهزة الأنظمة المدمجة وإنترنت الأشياء.

Table of Contents

Acknowledgments	ii
Abstract	iii
List of Figures	vii
List of Tables.....	viii
List of Abbreviations.....	ix
Chapter 1: Introduction	1
1.1 Problem Definition	1
1.2 Motivation	2
1.3 Objectives	3
1.4 Methodology.....	4
1.5 Time Plan.....	6
1.6 Thesis Outline.....	7
Chapter 2: Literature Review	10
2.1 Theoretical Background	10
2.1.1 Flash Over-The-Air Updates.....	10
2.1.2 A/B Partition Update Mechanism.....	11
2.1.3 ISO 21434 Automotive Cybersecurity Standards.....	13
2.1.4 Threat Analysis and Risk Assessment	14
2.1.5 STRIDE Methodology	15
2.1.6 MITRE ATT&CK Framework	16
2.1.7 Common Vulnerability Scoring System	17
2.1.8 Microservice Architecture	18
2.1.9 Online and Offline Keys Management	19
2.1.10 Intrusion Prevention System	20
2.1.11 Process Manipulation.....	21
2.1.12 Secure Booting.....	22
2.2 Previous Studies and Works.....	23
2.2.1 The Update Framework (TUF)	23
2.2.2 Uptane	23
2.2.3 Comparative Analysis and Critique.....	24
Chapter 3: System Architecture	25

3.1	System Characteristics	25
3.2	System Architecture	27
3.2.1	Director Server	27
3.2.2	Image Server	28
3.2.3	Time Server	29
3.2.4	Targets	29
3.2.5	Inter-server Communication	30
Chapter 4: System Implementation and Deployment		31
4.1	Software Tools Used	31
4.2	Setup Configuration	33
4.2.1	Primary ECU	33
4.2.2	Secondary ECU	34
4.2.3	Servers Infrastructure	34
4.3	Experimental and Results	37
4.3.1	User Experience (UX) Research	37
4.3.2	User Interface (UI) Design	39
4.3.3	Threat Analysis and Risk Assessment (TARA) Results	41
4.3.4	Deployment Diagram	47
Chapter 5: Application Run		49
5.1	Deploy Server Infrastructure	49
5.2	Integrate target with application	51
5.3	Devices Management	52
Chapter 6: Conclusion and Future Work		57
6.1	Conclusion	57
6.2	Future Work	58
6.2.1	Enhanced Anomaly Detection and Threat Intelligence Integration	58
6.2.2	Expansion of Hardware Platform Compatibility	58
6.2.3	Standardization and Certification	59
6.2.4	Continuous Improvement and User Feedback Integration	59
6.2.5	Exploration of Blockchain Technology Integration	59
6.2.6	Ethical Considerations and Privacy Protection	60
References		61

List of Figures

Figure 1.1: Project gantt chart.....	6
Figure 2.1: Flash Over-The-Air Procedures	11
Figure 2.2: Bootloader state machine	12
Figure 2.3: Overview of ISO 21434.....	13
Figure 2.4: The TARA procedure based on ISO/SAE 21434.....	15
Figure 2.5: Microservice architecture	18
Figure 2.6: Uptane architecture.....	24
Figure 3.1: Framework characteristics.....	25
Figure 3.2: System architecture	27
Figure 4.1: UX research for image server features	38
Figure 4.2: UX research snippet of the campaign features	38
Figure 4.3: UX research snippet for the director server features.....	39
Figure 4.4: Campaign list UI screen	40
Figure 4.5: Campaign creation UI screens.....	40
Figure 4.6: TARA Item definition	41
Figure 4.7: System deployment diagram	48
Figure 5.1: Deploy Kubernetes cluster	49
Figure 5.2: Deployed pods	50
Figure 5.3: Deployed pods 2	50
Figure 5.4: Move SecOTA package to the buildroot project.....	51
Figure 5.5: Open buildroot config file with nano	51
Figure 5.6: Add SecOTA package to the menu	51
Figure 5.7: Choose SecOTA package in the menu	51
Figure 5.8: List device screen	52
Figure 5.9: Create a new device.....	52
Figure 5.10: Successfully created a device	53
Figure 5.11: List ECUs screen	53
Figure 5.12: Step 1 of creating a new ECU	54
Figure 5.13: Step 2 of creating a new ECU	54
Figure 5.14: Step 3 of creating a new ECU	55
Figure 5.15: Successfully created an ECU.....	55
Figure 5.16: Software updates list.....	56
Figure 5.17: Campaigns List	56

List of Tables

Table 2.1: Vulnerabilities' categories in STRIDE Methodology	16
Table 2.2: CVSS v3 rating	18
Table 4.1: Vulnerabilities severity definition	41
Table 4.2: Deprecated OpenSSL vulnerability card	42
Table 4.3: Process manipulation vulnerability card.....	43
Table 4.4: open ports and insecure connections vulnerability card.....	44
Table 4.5: drop-request attack card.....	45
Table 4.6: slow retrieval attack card	45
Table 4.7: mix-and-match attack card.....	45
Table 4.8: Freeze attack card	45
Table 4.9: partial bundle installation attack card	45
Table 4.10: rollback attack card	46
Table 4.11: Mixed-bundles attack card.....	46
Table 4.12: endless data attack card.....	46

List of Abbreviations

FOTA:	Flash Over The Air
IoT:	Internet of Things
TUF:	The Update Framework
ISO:	International Organization for Standardization
TARA:	Threat Analysis and Risk Assessment
STRIDE:	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege
MITRE ATT&CK:	MITRE Adversarial Tactics, Techniques, and Common Knowledge
TTPs:	Tactics, Techniques, and Procedures
ECU:	Electronic Control Unit
IPS:	Intrusion Prevention Systems
CVSS:	Common Vulnerability Scoring System
HSM:	Hardware Security Module
UFW:	Uncomplicated Firewall
NTP	Network Time Protocol
NTS	Network Time Security
K8S	Kubernetes

Chapter 1:

Introduction

1.1 Problem Definition

In today's fast-paced business environment, the prioritization of rapid development often results in the swift creation of critical features within tight schedules. This accelerated pace typically leaves little room for thorough testing and comprehensive security checks. Consequently, after a product's release, organizations commonly face two primary challenges: the need to add features that were initially discussed but not included, and the necessity to identify and rectify post-deployment bugs or security vulnerabilities.

The limitations of wired updates exacerbate these challenges. This method typically relies on highly skilled personnel, often engineers, to oversee the upgrade process or requires the recall and retrieval of distributed products for in-house updates. Both approaches are time-intensive and come with substantial costs.

A more efficient solution is employing flash over the air (FOTA) updates, which eliminate the need for expert engineers or physical connections. However, existing FOTA options are often not fully prepared for use or may not be compatible with the specific hardware in use, necessitating additional development time and potentially raising security concerns. Furthermore, many current FOTA frameworks do not adequately address physical security or client-side attack vectors, leaving embedded and IoT systems vulnerable. This thesis develops a universal and secure on-premises FOTA framework to mitigate these issues, ensuring robust, adaptable, and secure updates for embedded and IoT systems.

1.2 Motivation

The rapidly evolving business landscape places immense pressure on organizations to deliver products and services quickly to remain competitive. While rapid development enables swift feature deployment, it often leads to significant post-release challenges. Two primary concerns emerge:

1. **Feature Enhancement and Bug Resolution:** Accelerated development timelines can result in the exclusion of critical features initially planned, necessitating post-deployment enhancements. Additionally, bugs and security vulnerabilities often surface after release, requiring timely fixes.
2. **Overcoming Wired Update Limitations:** Traditional methods of software and firmware updates typically require expert personnel, such as engineers, to oversee the upgrade process, or they may necessitate product recalls for in-house updates. These approaches are time-intensive and incur substantial costs. Furthermore, they do not adequately address physical security threats, leaving devices vulnerable to tampering during the update process.

The motivation for this project stems from the need for a more efficient, cost-effective, and physically secure solution to these challenges. Flash over-the-air (FOTA) updates provide a promising alternative by eliminating the need for expert personnel and physical connections. However, current FOTA options are often not fully prepared for use or compatible with specific hardware, requiring further development to enhance their readiness and security. Moreover, many existing frameworks do not adequately address physical security or client-side attack vectors.

This project aims to bridge these gaps by offering a ready-to-deploy and secure FOTA framework that integrates seamlessly with diverse hardware and software environments while prioritizing security. Our framework, SecOTA, adheres to industry standards in security, such as ISO 21434, and privacy regulations. By implementing the Uptane standard, which is based on The Update Framework (TUF) and adapted for embedded and IoT devices, our solution ensures robust, adaptable, and secure updates that protect against physical tampering.

1.3 Objectives

The objective of the "Universal Secure Flash-Over-The-Air Framework for Embedded and IoT Systems" project is to achieve the following specific goals:

1. **Develop a Secure and Ready-to-Deploy FOTA Framework:** Create a robust Firmware Over-The-Air (FOTA) framework that ensures the security and integrity of software and firmware updates, making it ready for deployment in a variety of embedded and IoT device systems.
2. **Support Multiple Architectures and Boards:** Design the framework to be compatible with a broad range of architectures and boards commonly used in embedded systems, offering versatility to cater to diverse hardware requirements.
3. **Compliance with Industry Standards:** Adhere to established industry standards for software update security, including ISO 21434. Incorporate best practices and security measures to protect against unauthorized tampering, data breaches, and physical security threats.
4. **User Acceptance and GDPR Compliance:** Develop a user-friendly web interface that allows Original Equipment Manufacturers (OEMs) to manage and accept updates. Ensure compliance with GDPR privacy regulations, including user acceptance tracking for incoming updates.
5. **Efficient Device Management:** Enable device monitoring, bulk updates, and individual updates through the web interface, simplifying the management of device versions and firmware updates.
6. **Enhanced Security:** Focus on enhancing the security features of the framework to address potential threats and vulnerabilities associated with Over-The-Air updates. This includes updating deprecated cryptographic algorithms, implementing network Intrusion Prevention Systems (IPS), and deploying process manipulation countermeasures to ensure the integrity and confidentiality of updates during transmission.
7. **Documentation and User Guidance:** Create comprehensive documentation that encompasses the codebase, configuration options, use cases, and deployment procedures to facilitate easy adoption and understanding of the framework.

By achieving these objectives, the project aims to provide a comprehensive and secure solution for device updates, addressing the challenges associated with rapid development and ensuring the efficient management of software and firmware updates in the contemporary business landscape.

1.4 Methodology

The methodology for this project is structured to ensure a thorough and secure development of the SecOTA framework. The process is divided into several key phases, each comprising specific tasks and deliverables. The overall approach can be summarized as follows:

[1] Research and Analysis

- Conduct an extensive review of existing FOTA standards, including Uptane and TUF, as well as the latest implementations and research in the field.
- Perform a competitive analysis of top FOTA providers to understand the landscape and identify key features and best practices.

[2] System Design and Architecture

- Determine specifications and select the appropriate architecture for the SecOTA framework.
- Develop a system map outlining features and functionalities based on insights from the UX walkthrough and competitive analysis.

[3] Parallel Development Processes

Backend Server and UI Frontend Development

- Implement microservice-based backend servers in alignment with the Uptane standard.
- Design the user interface (UI) and implement the frontend based on the designed UI, ensuring seamless integration with the backend APIs.
- Tasks include API interface development for the Director Server and Images Server, customizable GUI creation for both servers and integrating these components.

Security-Enhanced Client Development

- Conduct a Threat Analysis and Risk Assessment (TARA) following ISO 21434 standards, which include:
 - i. Item Asset Identification
 - ii. Threat Scenario Identification
 - iii. Impact Rating
 - iv. Attack Path Analysis
 - v. Attack Feasibility Rating
 - vi. Risk Determination
 - vii. Risk Treatment Decision
- Define cybersecurity goals based on TARA outcomes and translate these into security concepts to mitigate identified risks.
- Implement the security concepts in the client, focusing on enhancing protection against physical and client-side attacks.

[4] Integration and Testing

- Connect the backend APIs with the security-enhanced client to ensure seamless operation and communication.
- Conduct comprehensive system and integration testing to validate the functionality, security, and compatibility of the entire framework.

[5] Deployment and Documentation

- Utilize Docker and Kubernetes for packaging and deploying the microservice architecture, facilitating ease of deployment and scalability.
- Create a package and meta-layer for the client application to simplify integration with embedded Linux projects.
- Develop thorough documentation covering the codebase, configuration options, use cases, and deployment procedures to support easy adoption and understanding.

By following these structured phases and employing rigorous scientific methods, the project aims to deliver a secure, efficient, and universally deployable FOTA framework that addresses the critical challenges of rapid development, physical security, and comprehensive device management in the embedded and IoT ecosystem.

1.5 Time Plan

The SecOTA framework's development unfolds across five (5) phases, summarized as follows:

I. Phase 1

In this initial phase, the framework's specifications and constraints are established. The optimal architecture, informed by recent research and threat modeling results, is selected, and the codebase structure is prepared for further development.

II. Phase 2

This stage focuses on the development of essential bootloader functionalities. Simultaneously, necessary modifications are made to Uptane to prepare it for seamless interfacing and integration with the bootloader.

III. Phase 3

Phase 3 encompasses the development of features aligned with GDPR privacy regulations, including user acceptance of updates and the creation of web interfaces for this purpose.

IV. Phase 4

During this phase, integration between the backend systems and the bootloader is accomplished, along with the incorporation of additional functionality that can enhance the bootloader's capabilities.

V. Phase 5

The final phase involves comprehensive testing and the creation of detailed documentation for the framework.

The following is the initial Gantt chart that provides a precise schedule and work plan for each functionality within the framework.

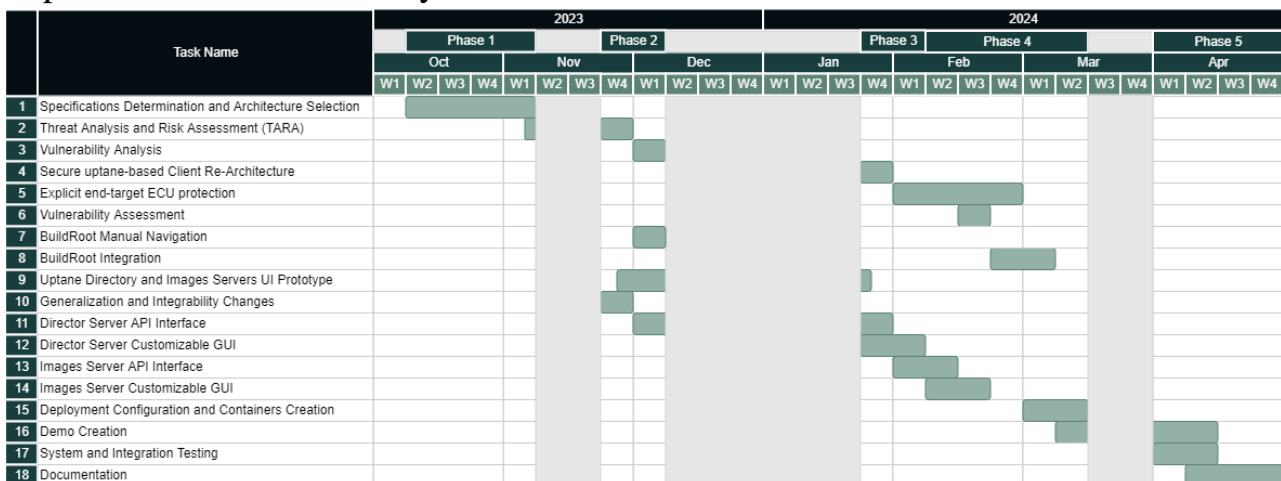


Figure 1.1: Project gantt chart

1.6 Thesis Outline

This thesis follows a structured outline encompassing various sections that delve into the development and implementation of the framework. Each chapter is designed to explore specific aspects crucial to understanding and deploying the framework effectively.

Acknowledgment

This section expresses gratitude to individuals and institutions whose contributions supported the completion of the thesis. It acknowledges the guidance, support, and resources provided throughout the research and writing process.

Abstract

The abstract succinctly summarizes the thesis, encompassing its problem statement, methodology, key findings, and conclusions. It provides a high-level overview that allows readers to grasp the essence of the research without delving into the full document.

List of Figures

The list of figures includes all graphical representations used in the thesis, accompanied by their respective page numbers. This section aids readers in locating specific visual data and illustrations referenced within the text.

List of Tables

Comprising all tabular data presented in the thesis along with their corresponding page references, the list of tables facilitates easy navigation and retrieval of specific data points and numerical findings essential to the study.

List of Abbreviations

The list of abbreviations provides definitions for all acronyms and abbreviations utilized throughout the thesis. This section enhances readability by ensuring that specialized terms and shorthand are clearly explained for readers' comprehension.

Chapter 1: Introduction

The introduction provides a comprehensive background of the study, detailing the context, scope, and significance of the research topic. It articulates the specific problem statement addressed, outlines the objectives guiding the study, discusses the broader implications of the research, and previews the organizational structure of the thesis to guide readers through its content.

Chapter 2: Literature Review

The literature review offers an in-depth examination of existing knowledge and research related to FOTA technologies. It includes an analysis of current FOTA frameworks, an overview of prominent security standards such as TUF and Uptane, and a discussion on relevant regulatory guidelines like ISO 21434 and GDPR. This chapter identifies gaps in current solutions and highlights opportunities for further research in enhancing FOTA security and efficiency.

Chapter 3: System Architecture

The system architecture chapter outlines the overarching design and structure of the developed framework. It discusses the methodology used to determine specifications and select architecture, emphasizes the importance of TARA in mitigating security threats, and details the development processes for backend and frontend systems. The chapter also covers the integration, and communication strategies, and the tools and technologies employed to realize a robust and secure FOTA solution.

Chapter 4: System Implementation and Deployment

This chapter delves into the detailed design and implementation of both backend systems and security-enhanced client components within the FOTA framework. It explains the integration of these components, describes the user interface design, and ensures compliance with regulatory standards such as GDPR and ISO 21434. The chapter aims to provide a comprehensive understanding of how the theoretical framework discussed in previous chapters is translated into a practical, deployable solution.

Chapter 5: Application Run

Chapter 5 focuses on the practical aspects of deploying and running the developed FOTA framework. It outlines step-by-step procedures for deploying the system, discusses user acceptance and feedback received during deployment, and evaluates the framework's performance and security effectiveness. This chapter provides empirical evidence of the framework's utility and addresses any identified challenges or improvements needed for future iterations.

Chapter 6: Conclusion and Future Work

The conclusion chapter summarizes the key findings and contributions of the thesis, highlighting advancements made in FOTA technology and cybersecurity. It acknowledges the limitations encountered during the research process and offers recommendations for future research directions. This chapter aims to consolidate the thesis's impact on the field while providing a roadmap for further exploration and development in secure FOTA frameworks.

References

The references section provides a comprehensive list of all sources cited throughout the thesis, ensuring academic integrity and enabling readers to access the primary and secondary materials that informed the research and development of the SecOTA framework.

Chapter 2:

Literature Review

2.1 Theoretical Background

This section provides the foundational concepts essential to the development of the SecOTA framework. It covers key principles such as Flash Over-The-Air (FOTA) updates and the A/B partition update mechanism. Additionally, it explores ISO 21434 automotive cybersecurity standards, microservice architecture, IPS (Intrusion Prevention Systems), process manipulation, secure booting, and management of offline and online keys. These theories are critical for understanding the implementation of the solution provided by the thesis.

2.1.1 Flash Over-The-Air Updates

Flash Over-The-Air (FOTA) updates are a critical technology for remotely updating the firmware of embedded and IoT devices. This method allows devices to receive software updates wirelessly, eliminating the need for physical access to the device. FOTA updates are essential for maintaining device security, fixing bugs, and adding new features without significant downtime or manual intervention. This seamless updating process ensures that devices remain up-to-date with the latest security patches and feature enhancements, thus extending their operational lifespan and reliability.

The FOTA process involves several key steps to ensure a secure and successful update. First, the server initiates the update by preparing and securely transmitting the update package to the target device. This package includes the new firmware, metadata, and necessary cryptographic signatures. Upon receiving the package,

the device verifies its integrity and authenticity using cryptographic methods. After successful verification, the device installs the new firmware, maintains a backup of the current version, reboots, and performs validation checks. If the new firmware passes these checks, the update is confirmed, completing the FOTA process.

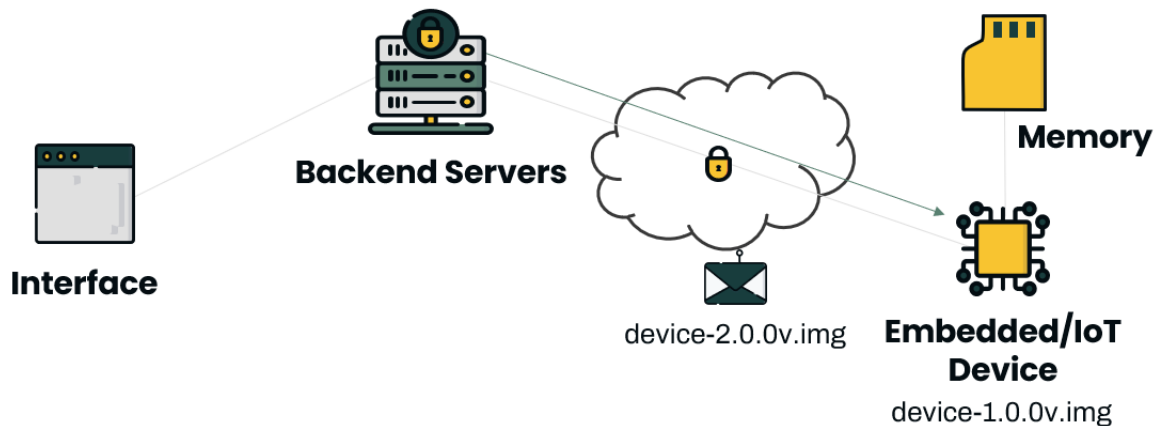


Figure 2.1: Flash Over-The-Air Procedures

The implementation of FOTA updates significantly enhances the management and maintenance of embedded and IoT systems. By enabling remote updates, FOTA reduces the need for manual intervention, thereby saving time and resources. It ensures that devices can quickly receive critical security patches and feature enhancements, improving overall system security and functionality. Furthermore, FOTA supports the scalability of IoT deployments, allowing updates to be rolled out to large numbers of devices efficiently and consistently. This capability is crucial for maintaining the security and performance of IoT networks as they grow and evolve.

2.1.2 A/B Partition Update Mechanism

The A/B partition update mechanism is a robust method used in firmware updates to ensure that devices can recover from update failures without losing functionality. This technique involves maintaining two separate partitions on the device, labeled A and B. At any given time, one partition is active and running the current firmware, while the other partition remains inactive and is used for updates. This approach ensures that the device can always revert to a known good state, enhancing reliability and resilience.

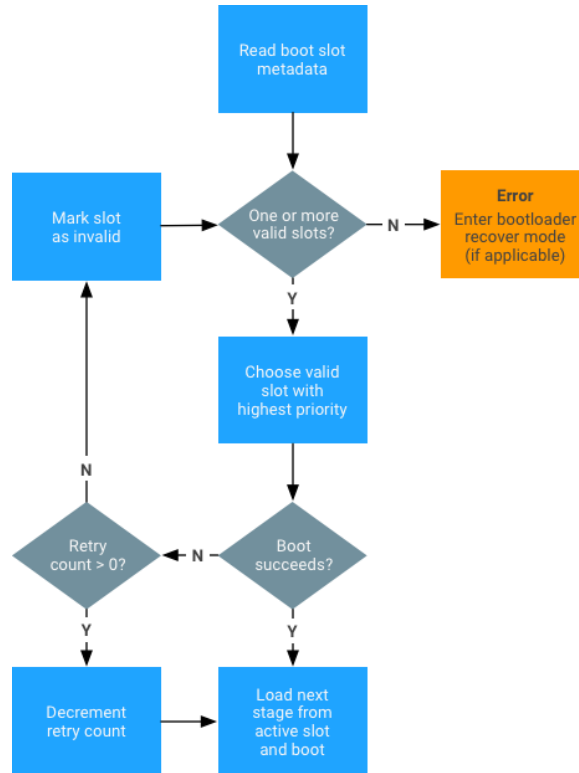


Figure 2.2: Bootloader state machine

The A/B partition update process begins with the preparation of the new firmware, which is transmitted to the device and installed on the inactive partition. During this stage, the device continues to operate using the current firmware on the active partition, ensuring uninterrupted functionality. Once the installation is complete, the device performs a verification process to ensure the integrity and authenticity of the new firmware. After successful verification, the device switches to the updated partition upon the next reboot. If any issues arise during the update or if the new firmware fails, the device can revert to the previous partition, ensuring continuous operation.

The implementation of the A/B partition update mechanism significantly enhances the reliability and resilience of firmware updates in embedded and IoT systems. By maintaining two separate partitions, the device ensures seamless updates without downtime or the risk of rendering the device inoperable due to update failures. This method is particularly beneficial for critical systems where maintaining continuous operation is essential. Additionally, the A/B partitioning approach simplifies the update process for end-users and administrators, providing a robust fallback mechanism that increases overall system stability and security.

2.1.3 ISO 21434 Automotive Cybersecurity Standards

ISO 21434 is a comprehensive standard developed to address the growing cybersecurity challenges in the automotive industry. It provides guidelines and requirements for managing cybersecurity risks throughout the lifecycle of automotive systems, from conceptual design to decommissioning. This standard aims to ensure that cybersecurity is integrated into every phase of vehicle development, enhancing the overall security and safety of automotive systems.

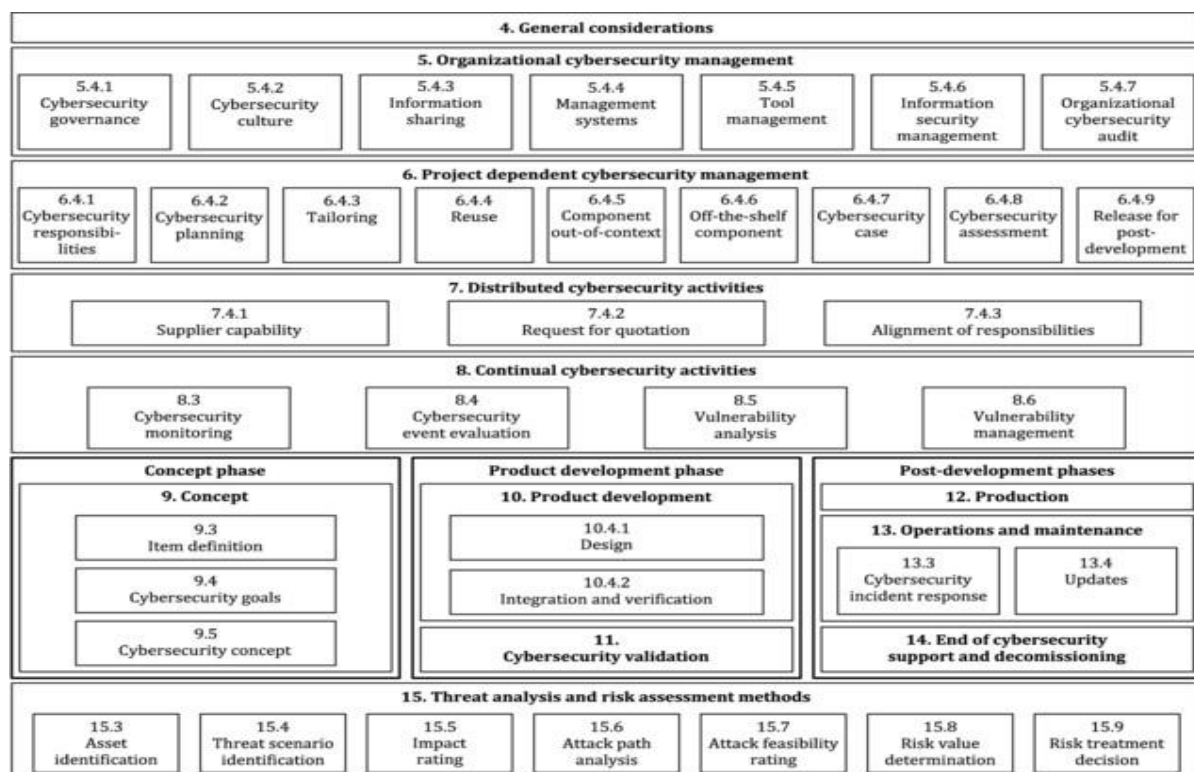


Figure 2.3: Overview of ISO 21434

ISO 21434 emphasizes the importance of conducting thorough risk assessments to identify potential threats and vulnerabilities. This involves analyzing the potential impact of security breaches and implementing appropriate countermeasures to mitigate risks. The standard advocates for incorporating security considerations early in the design phase, ensuring that security is built into the system architecture. It includes defining security requirements, designing secure components, and implementing security controls. Additionally, ISO 21434 outlines processes for verifying and validating security measures, which include testing for vulnerabilities and conducting security audits.

Implementing ISO 21434 Automotive Cybersecurity Standards significantly enhances the security and resilience of automotive systems. By integrating cybersecurity into every phase of the vehicle lifecycle, organizations can better protect against potential threats and vulnerabilities. The standard's emphasis on risk management, security by design, and continuous improvement ensures that automotive systems are robust and capable of withstanding cybersecurity challenges. This proactive approach not only safeguards the vehicle's functionality and safety but also builds consumer trust and confidence in the security of automotive systems.

2.1.4 Threat Analysis and Risk Assessment

Threat Analysis and Risk Assessment (TARA) is a systematic approach used to identify, analyze, and mitigate potential threats and risks to a system. In the context of embedded and IoT systems, TARA plays a crucial role in ensuring the security and resilience of these systems against various types of attacks. The primary goal of TARA is to prioritize security efforts based on the potential impact and likelihood of threats, thereby optimizing resource allocation for effective risk management.

TARA begins with identifying potential threats that could exploit vulnerabilities in the system. This includes understanding the attack surface and recognizing various threat vectors using tools like STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) and the MITRE ATT&CK framework. Once threats are identified, the next step involves assessing the risks associated with each threat and evaluating both the likelihood of occurrence and the potential impact on the system. This assessment is crucial for understanding the risk landscape and guiding the selection of appropriate mitigation strategies.

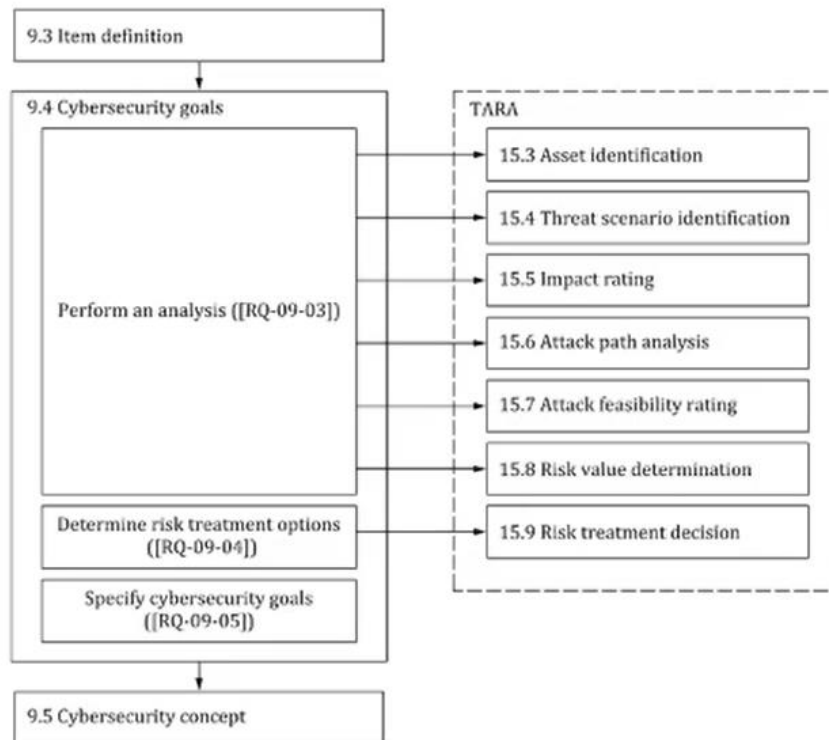


Figure 2.4: The TARA procedure based on ISO/SAE 21434

Implementing TARA significantly enhances the security posture of embedded and IoT systems by systematically addressing identified risks. By prioritizing threats and vulnerabilities based on their potential impact and likelihood, organizations can allocate resources more effectively to mitigate the most critical risks first. TARA provides a structured framework for proactive risk management, ensuring that security measures are continuously monitored and updated to address evolving threats. This approach not only protects the system from current threats but also prepares it to adapt to future challenges, thereby enhancing long-term security and resilience.

2.1.5 STRIDE Methodology

The STRIDE methodology is a threat modeling framework developed by Microsoft to systematically identify and categorize potential threats to software and systems. It provides a structured approach to analyzing security vulnerabilities based on six threat categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. This

method is widely adopted for its effectiveness in understanding and mitigating security risks across various domains, including embedded and IoT systems.

Table 2.1: Vulnerabilities' categories in STRIDE Methodology

Threat	Desired Security Property
S poofing	Authentication
T ampering	Integrity
R epudiation	Non-repudiation
I nformation Disclosure	Confidentiality
D enial of service	Availability
E levation of Privilege	Authorization

The STRIDE methodology begins by categorizing threats into specific types, each representing a distinct security concern. For example, Spoofing involves attackers impersonating legitimate users or entities to gain unauthorized access. Tampering focuses on unauthorized modification of data or code, while Repudiation addresses the ability of attackers to deny their actions, complicating the forensic investigation. Information Disclosure concerns unauthorized access to sensitive data, and Denial of Service aims to disrupt system availability. Elevation of Privilege involves attackers gaining higher levels of access than intended.

Implementing the STRIDE methodology significantly enhances the security posture of embedded and IoT systems by systematically addressing identified threats. By categorizing threats into well-defined categories and implementing mitigation strategies tailored to each category, organizations can prioritize security measures effectively. This approach ensures comprehensive protection against a wide range of security risks, strengthening the resilience of the system against both current and emerging threats.

2.1.6 MITRE ATT&CK Framework

The MITRE ATT&CK Framework is a comprehensive knowledge base developed by MITRE Corporation, encompassing adversary tactics, techniques, and procedures (TTPs) observed in real-world cyber-attacks. It provides a structured approach to understanding and categorizing adversary behaviors across different stages of an attack lifecycle. The framework is widely utilized in

cybersecurity to strengthen threat detection capabilities and enhance defensive strategies against evolving cyber threats.

The MITRE ATT&CK Framework categorizes adversary behaviors into tactics and techniques. Tactics outline the high-level objectives of attackers, such as gaining initial access, executing malicious code, maintaining persistence, escalating privileges, evading defenses, and exfiltrating data. Techniques specify the specific methods and procedures adversaries use to achieve these objectives, providing detailed insights into how attacks are conducted.

Implementing the MITRE ATT&CK Framework significantly improves the cybersecurity resilience of embedded and IoT systems by offering a standardized approach to threat detection and response. By mapping adversary tactics and techniques, organizations can proactively identify potential threats and mitigate risks before they escalate. The framework's continuous updates and community contributions ensure that it remains relevant and effective in addressing emerging cyber threats, empowering defenders to stay ahead of adversaries.

2.1.7 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) provides a standardized framework for assessing the severity of security vulnerabilities. It assigns a numerical score to vulnerabilities based on their characteristics, aiding in prioritizing remediation efforts. The CVSS score comprises three main metric groups: Base, Temporal, and Environmental.

To calculate a CVSS score, researchers evaluate the following metrics:

1. **Base Metrics:** These include factors like the attack vector, attack complexity, privileges required, user interaction, scope of impact, and the potential impact on confidentiality, integrity, and availability.
2. **Temporal Metrics (optional):** These reflect the current state of a vulnerability, including exploit code maturity, remediation level, and report confidence.
3. **Environmental Metrics (optional):** These account for specific environmental factors that may modify the base score, such as security requirements and operational conditions.

The final CVSS score, ranging from 0.0 to 10.0, indicates the severity of the vulnerability, with higher scores indicating more severe risks. Engineers use this score to prioritize mitigation efforts, ensuring that the most critical vulnerabilities are addressed promptly to enhance the security of embedded and IoT systems.

Table 2.2: CVSS v3 rating

Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10

2.1.8 Microservice Architecture

Microservice architecture is a modern software design approach that structures applications as a collection of small, loosely coupled services, each responsible for specific business capabilities. Unlike monolithic architectures, where applications are developed as a single, interconnected unit, microservices promote modularization and independence. This architectural style is particularly favored for its agility, scalability, and resilience, making it well-suited for complex and dynamic environments like IoT systems.

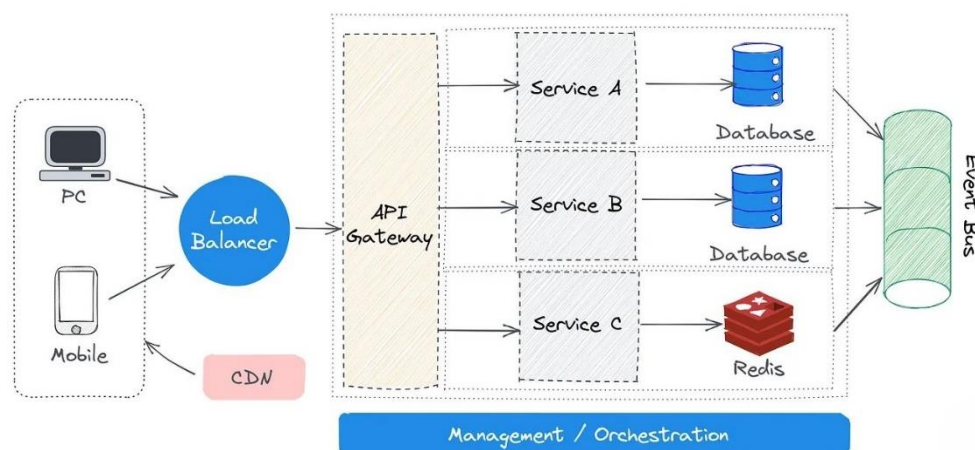


Figure 2.5: Microservice architecture

Microservices are designed to be independent and decoupled, allowing teams to develop, deploy, and scale services independently. Each service operates as a separate entity with its own bounded context, communicating with other services through well-defined APIs. This modular approach facilitates faster development

cycles and easier maintenance, enabling organizations to respond swiftly to changing business needs.

Scalability and elasticity are inherent benefits of microservice architecture. Services can be scaled horizontally based on demand, ensuring efficient resource utilization and performance optimization. This scalability is essential in IoT environments where workload fluctuations are common, allowing systems to handle varying levels of data processing and user interactions effectively.

Fault isolation and resilience are key features of microservice architecture. Failure in one service does not propagate to others, as services are isolated from each other. This isolation minimizes the impact of failures and enhances system reliability. By compartmentalizing functionalities, microservices enable applications to maintain operation even during component failures or updates. Implementing microservice architecture in embedded and IoT systems transforms how applications are developed and managed. It fosters agility, scalability, and fault tolerance, supporting rapid innovation and adaptation to evolving technological landscapes. This architectural approach empowers organizations to build robust and resilient systems capable of meeting the dynamic challenges of modern IoT environments.

2.1.9 Online and Offline Keys Management

Online and offline key management is crucial for securing embedded and IoT systems. This process involves the use of cryptographic keys to safeguard data, authenticate devices, and ensure secure communications. Effective key management is vital for maintaining system integrity, particularly in environments where devices are deployed in diverse and potentially untrusted locations.

The process begins with key generation, where cryptographic keys are created using secure algorithms to ensure their strength and randomness. These keys can be generated directly on the device (offline) or received from a central authority (online). Secure generation practices are essential to prevent unauthorized access and ensure the robustness of the keys.

Key distribution follows, where the generated keys must be securely transmitted to devices. This can be done through secure online communication channels or via physical methods in an offline setup, such as pre-loading keys during the

manufacturing process. Secure distribution ensures that only authorized devices receive the keys, maintaining the confidentiality and integrity of the communication.

Once distributed, keys need to be securely stored on the device using hardware security modules (HSMs) or trusted platform modules (TPMs) to prevent extraction or tampering. Additionally, key rotation and revocation are critical practices. Regularly rotating keys minimizes the risk of long-term exposure, and revoking compromised keys prevents their misuse, thereby protecting the system from potential security breaches.

Implementing effective online and offline key management enhances the security of embedded and IoT systems. By ensuring that keys are generated, distributed, stored, and managed securely, organizations can protect sensitive data and maintain system integrity. This comprehensive approach is essential for building trust and reliability in IoT deployments, enabling devices to operate securely in various and often challenging environments.

2.1.10 Intrusion Prevention System

An Intrusion Prevention System (IPS) plays a crucial role in securing FOTA (Flash Over-The-Air) clients that maintain both incoming and outgoing connections to servers. Key capabilities include:

1. **Real-time Threat Detection:** IPS continuously monitors incoming and outgoing traffic from FOTA clients. It employs signature-based detection to identify known attack patterns and anomaly-based detection to detect unusual behavior, thereby ensuring comprehensive threat coverage.
2. **Defense Against Unauthorized Access:** IPS actively prevents malicious actors from exploiting vulnerabilities in publicly exposed FOTA clients. It blocks unauthorized access attempts, mitigates intrusion attempts, and safeguards client integrity and confidentiality.
3. **Response and Mitigation:** Upon detecting suspicious activities, IPS assesses the severity of threats and takes immediate action. This includes blocking malicious packets, restricting unauthorized access, or alerting administrators to potential security breaches.

Implementing an Intrusion Prevention System enhances the security posture of FOTA clients by providing proactive threat detection and rapid response capabilities. By safeguarding against unauthorized access and potential breaches, IPS ensures the reliability and trustworthiness of FOTA operations in environments exposed to the internet.

2.1.11 Process Manipulation

Process manipulation is critical for securing embedded and IoT systems by preventing malicious processes from compromising the integrity of the FOTA (Flash Over-The-Air) client and its update process. It involves monitoring, controlling, and protecting the execution of processes to prevent an attacker from using a malicious process to gain remote control and install maliciously crafted firmware.

To detect unauthorized or abnormal behaviors, continuous process monitoring is essential. This involves utilizing system APIs, kernel-level monitoring, and behavioral analysis to identify deviations from normal process behavior. Early detection of potentially malicious processes is crucial to protect the FOTA client from being compromised.

Implementing stringent control mechanisms is another key aspect. Access control policies, privilege separation, and sandboxing limit the capabilities of processes, ensuring that unauthorized processes cannot gain control over the FOTA client and the update process. By enforcing these controls, the system adheres to security policies and operational guidelines, preventing unauthorized modifications.

Protection measures further enhance the security of critical processes involved in the FOTA client. Runtime integrity checks, code signing, and secure boot mechanisms prevent the execution of maliciously crafted firmware. These measures ensure that only verified and trusted processes can influence the FOTA client, maintaining system integrity and security.

Effective process manipulation techniques mitigate the risk of a malicious process gaining control over the FOTA client and its update process. Robust monitoring, control, and protection mechanisms prevent unauthorized access, maintain the integrity of the update process, and protect against remote code

execution (RCE) attacks. This ensures the secure operation of the FOTA client, even in the presence of potential threats.

2.1.12 Secure Booting

Secure booting is a foundational security feature for embedded and IoT systems, ensuring that devices boot using only trusted software. This process prevents unauthorized or malicious code from executing during the system startup, thereby protecting the integrity and security of the device.

The secure boot process begins with a hardware-based root of trust. This root of trust is typically embedded within the device's hardware, such as a Trusted Platform Module (TPM) or a similar secure element. During the boot sequence, the hardware verifies the digital signatures of the firmware and bootloader, ensuring they have not been tampered with and are from a trusted source.

Next, the firmware and bootloader continue the chain of trust by verifying the integrity and authenticity of the operating system kernel and other critical components. Each component is checked before it is executed, creating a secure boot chain that prevents the system from running any unauthorized code. This step-by-step verification process is crucial for maintaining the security of the entire boot process.

The impact of secure booting on the system is significant. By ensuring that only trusted software is executed, secure booting protects against a range of attacks, including rootkits and bootkits, which can compromise the system at its most fundamental level. This not only enhances the overall security of the device but also builds a foundation of trust for subsequent security measures, such as secure firmware updates and process manipulation protections.

In conclusion, secure booting is an essential security measure for embedded and IoT systems. By establishing a hardware-based root of trust and verifying each component in the boot sequence, it ensures that only trusted software is executed. This protects the system from unauthorized modifications and enhances its overall security, providing a robust foundation for additional security mechanisms.

2.2 Previous Studies and Works

This section critically examines previous studies and works relevant to secure FOTA frameworks, emphasizing their impact, methodologies, and contributions to the field. It aims to define and narrow the scope of the problem, avoid redundancy, select effective research methodologies, relate findings to existing knowledge, and identify areas requiring further exploration.

2.2.1 The Update Framework (TUF)

The Update Framework (TUF) has emerged as a cornerstone in the development of secure software update mechanisms. TUF addresses vulnerabilities associated with traditional update systems by introducing robust cryptographic techniques and metadata management. Metadata in TUF plays a pivotal role in verifying the authenticity and integrity of software updates, detailing crucial information such as version numbers, cryptographic hashes, and cryptographic signatures. This approach ensures that updates are tamper-resistant and verifiable, even in the presence of compromised components or malicious actors. Moreover, TUF's hierarchical signing model, encompassing **root**, **targets**, **snapshot**, and **timestamp** levels, establishes a flexible trust architecture that allows multiple entities to sign updates at different stages of distribution, enhancing security and reliability across diverse deployment environments.

2.2.2 Uptane

Uptane builds upon the principles of The Update Framework (TUF) to enhance the security of OTA updates, specifically tailored for automotive environments. While Uptane establishes robust standards and a framework for secure OTA updates, it focuses on providing guidelines and best practices rather than a specific implementation.

Uptane's architecture introduces a layered approach to update verification, featuring roles such as the **Director**, **Targets**, and **Image Repositories**. The Director manages the update process, ensuring authenticity and authorization before deployment. Targets verify update integrity against predefined

expectations, while Image Repositories store validated updates and facilitate their secure distribution to vehicles.

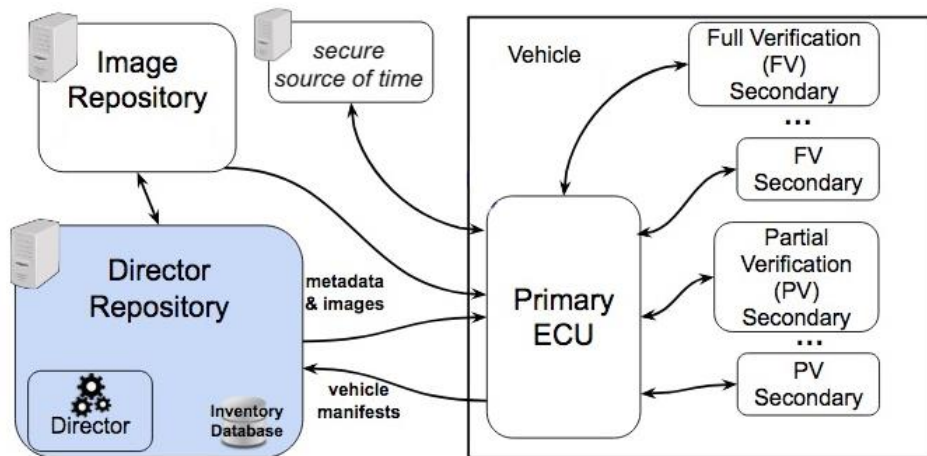


Figure 2.6: Uptane architecture

However, Uptane's threat modeling primarily addresses server-side threats, emphasizing network security and authentication mechanisms. It does not comprehensively account for physical attacks on target devices, such as unauthorized access to (ECUs). This limitation highlights the importance of implementing additional security measures to mitigate physical threats effectively and ensure comprehensive protection against unauthorized updates and tampering in automotive systems.

2.2.3 Comparative Analysis and Critique

A comparative analysis of Uptane highlights its strengths in establishing secure OTA update standards tailored for automotive environments. However, the framework's reliance on server-side threat modeling and its omission of physical security considerations pose challenges in addressing holistic security threats in real-world deployments. Future research should explore integrating additional security layers, such as physical security measures and advanced threat detection technologies, to enhance Uptane's effectiveness in safeguarding vehicle ECUs against evolving cyber threats and vulnerabilities.

Chapter 3:

System Architecture

3.1 System Characteristics

Before designing the system, it was imperative to identify the characteristics that constitute a successful solution addressing the previously discussed issues. As illustrated in the accompanying figure, the system must embody the following attributes:

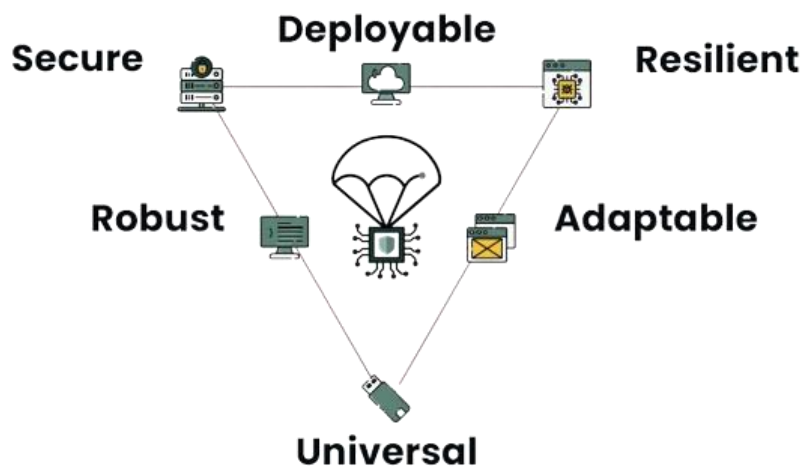


Figure 3.1: Framework characteristics

1. Deployable

The system should be easy to deploy across various environments with minimal configuration, ensuring swift and efficient implementation.

2. Resilient

The system should maintain operational integrity and functionality even when under attack, ensuring resilience against compromises and minimizing downtime.

3. Secure

Robust security measures must be integrated into the system to protect against unauthorized access, data breaches, and other malicious activities, ensuring the integrity and confidentiality of data.

4. Adaptable

The system should be capable of evolving in response to changing requirements, allowing for modifications and upgrades without significant overhauls.

5. Robust

The system must be able to perform reliably under a variety of conditions, demonstrating high levels of fault tolerance and error handling.

6. Universal

The system should be versatile enough to be applicable across different platforms, boards, technologies, and use cases, ensuring broad usability and interoperability.

These characteristics collectively ensure that the system is not only effective in solving the identified problems but also sustainable and scalable in the long term.

3.2 System Architecture

The proposed system architecture, illustrated in the diagram, consists of four main components designed to ensure the system's deployability, resilience, security, adaptability, robustness, and universality. This architecture includes two primary servers: the Director server and the Image server, and an optional Time server. Each server has distinct roles and responsibilities, working together to provide comprehensive functionality. The fourth element is the target, which can be a single primary ECU or a primary ECU with multiple secondary ECUs.

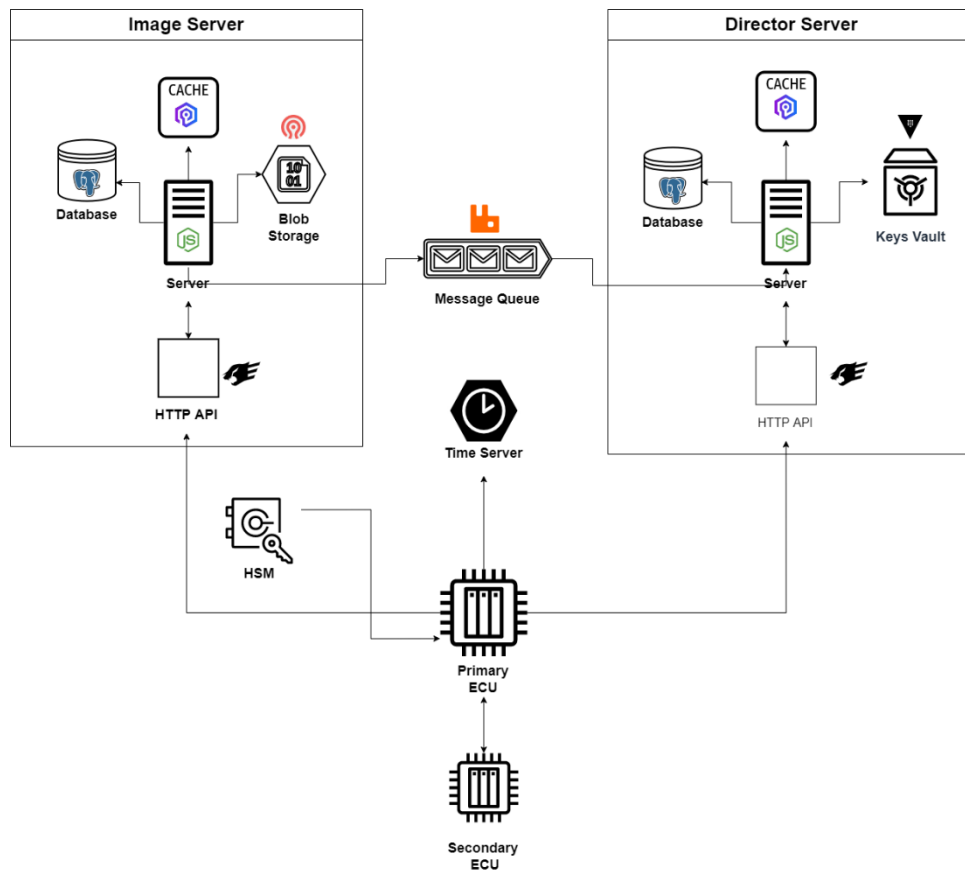


Figure 3.2: System architecture

3.2.1 Director Server

The Director server is responsible for creating and managing devices and ECUs, and cryptographically provisioning keys for secure updates. It supports functionalities such as single ECU updates, bulk updates via campaigns, and monitoring the status and reports periodically generated by each ECU and sent to the Director by the primaries. These reports include version status, intrusion

detection logs, and other useful diagnostics about each ECU. The Director server comprises several key components:

Node.js Server

This server handles the core Director service logic and coordinates the various components of the Director server.

Fastify HTTP API

The HTTP API serves as the interface through which external entities interact with the Director server, facilitating tasks such as device management, ECU provisioning, and update campaign management.

PostgreSQL Database

This database stores structured data required for registered devices, ECUs, status updates, and update campaigns.

Keys Vault

Essential for security, the keys vault manages cryptographic keys and secrets required for secure communications and data protection.

Valkey Cache

Utilizing Valkey, a Redis BSD alternative, this caching mechanism provides fast access to frequently requested data, improving system performance and reducing database load.

3.2.2 Image Server

The Image server is responsible for uploading, signing, and managing image files, which serve as versions to be selected by the Director server for specific ECUs or groups of ECUs, and later uploaded to the target. It primarily handles binary files, ensuring their efficient storage, retrieval, signing, and management. The Image server includes several critical elements:

Node.js Server

This server handles the core image service logic and coordinates the various components of the Image server.

Fastify HTTP API

The HTTP API acts as the interface through which external applications and users interact with the Image server, allowing for the submission, retrieval, and management of images.

PostgreSQL Database

This database stores structured data required for image management, ensuring reliable and scalable data storage.

Blob Storage

Blob storage is used for storing large binary objects, such as image files, ensuring the system can handle large volumes of image data efficiently.

Valkey Cache

Utilizing Valkey, a Redis BSD alternative, this caching mechanism provides fast access to frequently requested data, improving system performance and reducing database load.

3.2.3 Time Server

The Time server uses the NTS standard to provide a secure source of time for the primary ECUs. This ensures that the ECUs can retrieve the time securely without relying on unmanaged external servers, or a physical crystal, which can be tampered with.

3.2.4 Targets

The targets are the ECUs connected to the backend servers. They periodically package a version manifest, download any new updates specified by the Director, and then upgrade the current firmware. They are categorized into two types based on public internet connectivity, computing power, and storage capabilities:

Primary ECU

A Unix-based ECU with public internet connectivity, capable of sending the version manifest to the Director, retrieving metadata and images, and performing full verification on all downloaded images. It has substantial computing power and large storage to handle all required files for itself and the connected secondary ECUs. Ideally, a primary ECU is equipped with a hardware security module (HSM) to store communication keys and perform cryptographic operations securely. It also features a UFW firewall to monitor incoming and outgoing connections.

Secondary ECU

A bare metal or Unix-based ECU that lacks internet connectivity but is connected to a primary ECU via wired protocols. It has lower computing power and relies on the primary ECU for full verification, performing only minimal verification to ensure data integrity during transfer. Its storage capacity is limited to its metadata files and images.

3.2.5 Inter-server Communication

The Image Server and Director Server communicate through a Message Queue, which facilitates asynchronous messaging and decouples the servers, thereby enhancing system resilience and scalability. The message queue ensures reliable message delivery even in the face of intermittent network issues or server downtimes.

Chapter 4:

System

Implementation and

Deployment

4.1 Software Tools Used

During the project implementation, a variety of software tools were utilized to ensure the system's functionality, efficiency, and security. These tools were carefully selected to address specific needs within the project, ranging from core service logic to data management and secure communications.

Yocto Project (v4.0.17 “kirkstone”, Linux Foundation, 2022)

Integrated to manage the build environment and create customized Linux distributions tailored for embedded systems, ensuring seamless deployment and configuration of SecOTA on various hardware platforms.

BuildRoot (v2024.02.2, Buildroot Association, 2024)

Integrated to streamline the build process and create customized Linux-based firmware images tailored for embedded systems, facilitating seamless deployment and configuration of SecOTA across diverse hardware platforms.

C++ (v14.1.1, GNU Compiler Collection, 2024)

Employed to develop client software installed on target devices, enabling installation, verification, and update operations within the SecOTA framework.

Uncomplicated Firewall (UFW) (v0.36, Linux Foundation, 2020)

Implemented to protect and monitor incoming and outgoing network communications on primary ECUs, ensuring secure data transmission and system integrity.

OpenSSL (v3.2.2, OpenSSL Software Foundation, 2024)

Utilized for cryptographic operations, including secure communication and data integrity verification within the SecOTA framework.

Node.js (v18.20.3, Node.js Foundation, 2024)

Essential for handling core service logic in both the Director and Image servers, known for its robust asynchronous operations management.

Fastify (v4.28.0, NearForm, 2024)

Chosen for its high-performance HTTP API framework, optimizing communication between components with minimal overhead.

PostgreSQL (v16.1, PostgreSQL Global Development Group, 2023)

Selected for its reliability in storing structured data and supporting complex queries essential for system management.

Valkey Cache (v7.2.5, Linux Foundation, 2024)

Utilized for fast data access through caching mechanisms, reducing database load and enhancing overall system performance.

Ceph Blob Storage (v18.2.2, Ceph Foundation, 2023)

Employed for efficient handling and storage of large binary objects, ensuring scalability and effective management of image files.

Vault Secure Storage (v1.17.1, HashiCorp, 2024)

Integrated for secure storage solutions, ensuring encrypted data storage and access control mechanisms to safeguard sensitive information within the SecOTA ecosystem.

Docker (v23.0.3, Docker Inc., 2023)

Utilized for containerization of applications and services, facilitating easier deployment, scaling, and management of SecOTA components across different environments.

Kubernetes (v1.30.2, Cloud Native Computation Foundation CNCF, 2024)

Employed for orchestration and management of containerized applications, ensuring high availability, scalability, and resilience of SecOTA services in production environments.

Network Time Security NTS (RFC8915, 2020)

Implemented as a secure source of time synchronization, ensuring primary ECUs retrieve accurate and tamper-resistant time information, critical for secure OTA update operations.

4.2 Setup Configuration

The minimum hardware requirements for the main elements of the SecOTA framework are as follows:

4.2.1 Primary ECU

For the Primary ECU running unix-based OS and supporting public internet access, the minimal hardware requirements are:

Processor: Suitable for running unix-base OS

RAM: 2 GB

Storage: Multiple of two times the primary image size, with additional capacity for connected secondary ECUs

Networking: Ethernet, Wi-Fi, or LTE connectivity enabling public internet access

Additional: Hardware Security Module (HSM) recommended for key management

These specifications ensure that the Primary ECU can effectively run embedded Linux OS, accommodate storage requirements based on multiples of the primary image size for A/B partitioning, and support various network connectivity options necessary for secure, public-facing operations within the SecOTA framework.

4.2.2 Secondary ECU

For Secondary ECUs within the SecOTA framework, which may be bare-metal or Unix-based, the minimal hardware requirements are:

Processor: Suitable for running bare-metal or Unix-based applications

Storage: Adequate for storing metadata files and images

Networking: Wired connection to Primary ECU

Additional: Minimal computing power sufficient for basic verification of data integrity during transfers

These specifications ensure that Secondary ECUs can effectively support their role within the SecOTA framework, contributing to the secure and efficient deployment of OTA updates in embedded and IoT systems.

4.2.3 Servers Infrastructure

The server infrastructure for the SecOTA framework can be deployed according to various scenarios, accommodating different requirements for scalability, separation methods, and cost considerations:

Scenario 1: Single Machine

This scenario involves deploying all SecOTA services (Director, Image, and Time) on a single powerful server. It offers a cost-effective solution with simplified management but may have limited scalability and could pose a single point of failure risk.

Resources

CPU: At least 16 vCPUs

Memory: At least 64 GB RAM

Storage: At least 2 TB SSD

Network: 1 Gbps network interface

Breakdown

PostgreSQL DBs (2 instances)

CPU: 4 vCPUs

Memory: 16 GB RAM

Storage: 500 GB

Redis Instances (2 instances)**CPU:** 2 vCPUs**Memory:** 4 GB RAM**Storage:** 50 GB**RabbitMQ****CPU:** 2 vCPUs**Memory:** 4 GB RAM**Storage:** 50 GB**NodeJs Applications (2 instances)****CPU:** 4 vCPUs**Memory:** 8 GB RAM**React Applications (2 instances)****CPU:** 2 vCPUs**Memory:** 4 GB RAM**CEPH Instance****CPU:** 4 vCPUs**Memory:** 16 GB RAM**Storage:** 1 TB**HashiCorp Vault****CPU:** 2 vCPUs**Memory:** 4 GB RAM**Storage:** 50 GB**Nginx****CPU:** 2 vCPUs**Memory:** 2 GB RAM**Scenario 2: Logical Separated**

Logical Separated, the SecOTA framework is distributed across six distinct servers, each fulfilling specific roles essential for secure OTA updates. The Director Server manages device registrations, orchestrates update campaigns, and ensures overall update management. The Image Server handles storage, signing, and distribution of firmware images. RabbitMQ facilitates message

queuing and communication between components, while Nginx API Proxy secures API requests and manages communications. Ceph Storage provides scalable and reliable distributed storage for images and metadata. The NTS Time Server ensures secure time synchronization across the infrastructure, critical for maintaining update integrity. Finally, the Master Node in Kubernetes orchestrates containerized services, offering scalability, fault tolerance, and efficient resource utilization. This configuration supports medium to large-scale deployments, optimizing infrastructure for robust and secure OTA updates in embedded and IoT systems.

Director Machine (PostgreSQL DB, Redis, Hashicorp Vault, NodeJS)

CPU: 16 cores
RAM: 64 GB
Storage: 1 TB SSD

Image Machine (PostgreSQL DB, Redis, NodeJS)

CPU: 12 cores
RAM: 48 GB
Storage: 1 TB SSD

CEPH Machine

CPU: 8 cores
RAM: 32 GB
Storage: 1 TB SSD

Nginx Machine

CPU: 2 cores
RAM: 8 GB
Storage: 50 GB SSD

NTS Server Machine

CPU: 2 cores
RAM: 8 GB
Storage: 50 GB SSD

RabbitMQ Machine

CPU: 4 cores
RAM: 8 GB
Storage: 50 GB SSD

Master Machine (for managing Kubernetes)

CPU: 4 cores

RAM: 16 GB

Storage: 100 GB SSD

Additional Considerations

Networking: Ensure high-speed, low-latency networking between machines.

Backup: Implement regular backups for critical data, especially databases.

Security: Secure the network, implement firewalls, and ensure data encryption, especially for sensitive components like Hashicorp Vault.

Monitoring: Use tools like Prometheus, Grafana, or ELK Stack for monitoring and logging.

Scalability: Plan for future scalability by considering cloud-based or hybrid setups.

4.3 Experimental and Results

4.3.1 User Experience (UX) Research

The development of the SecOTA framework began with comprehensive User Experience (UX) research, including a competitive analysis comparing key features and usability aspects of existing FOTA solutions such as Aiken, Mender, and Otahere. Screenshots from the UX research phase illustrated key insights and user interface considerations, informing subsequent design decisions for the SecOTA framework.

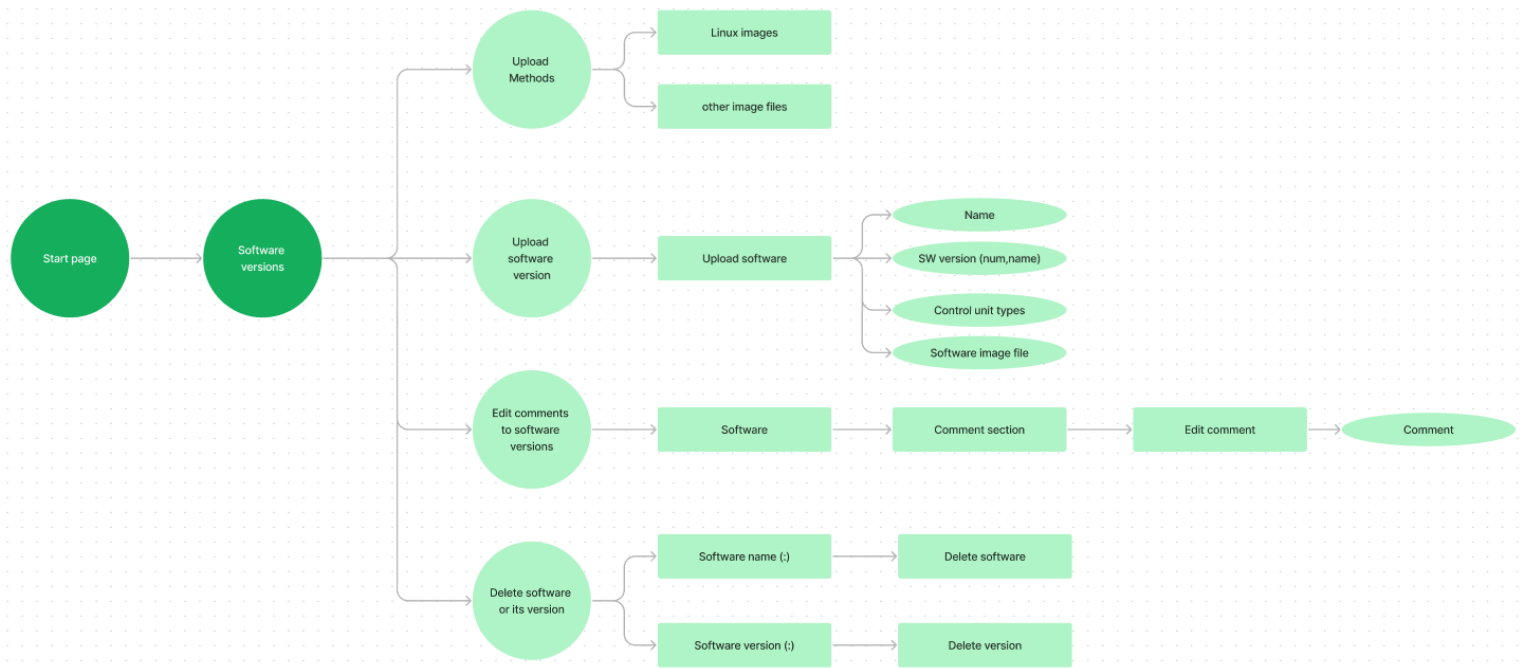


Figure 4.1: UX research for image server features

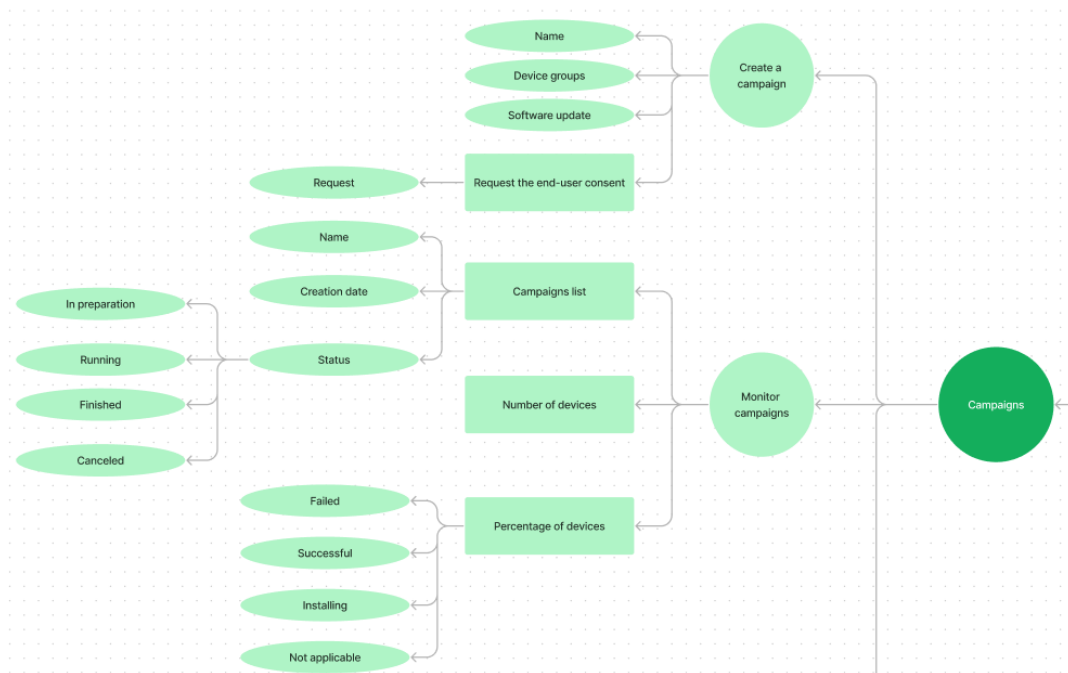


Figure 4.2: UX research snippet of the campaign features

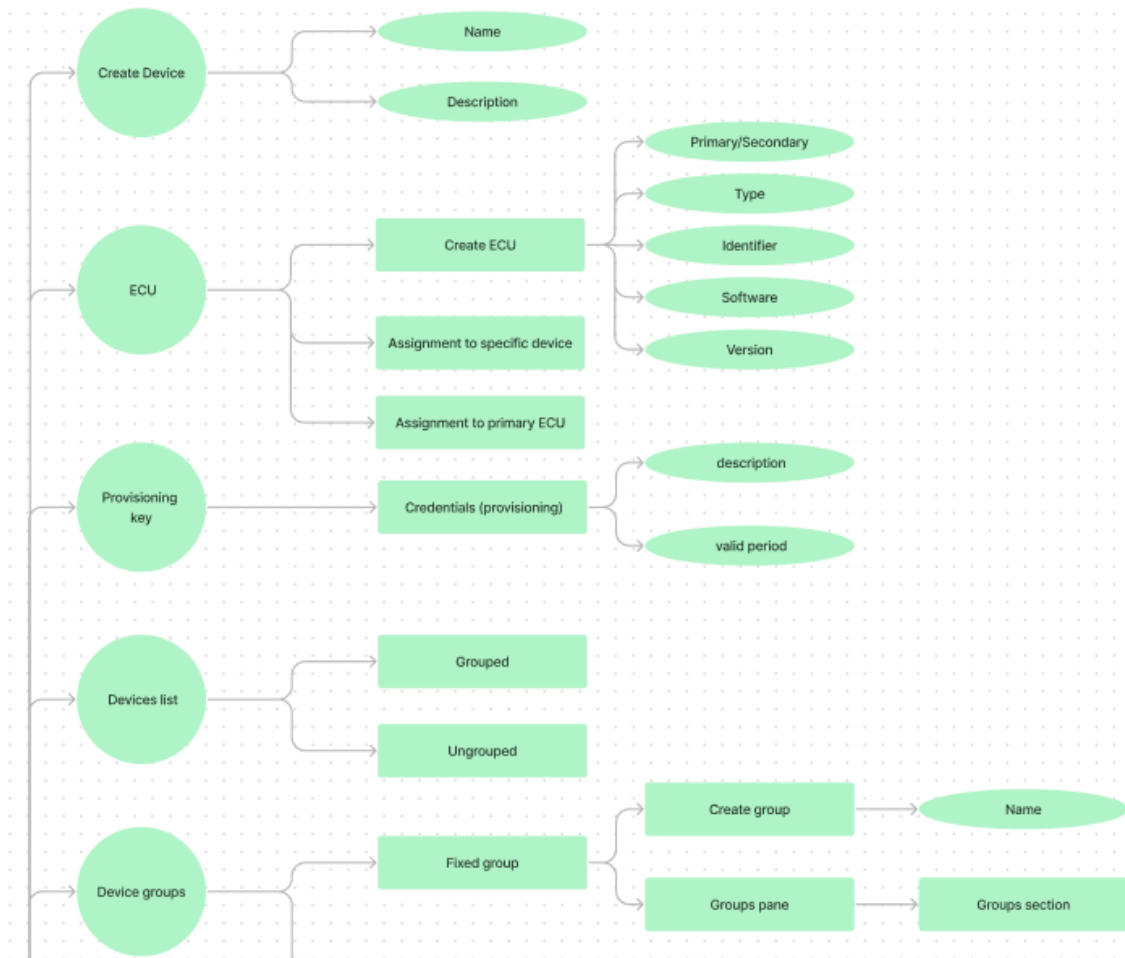


Figure 4.3: UX research snippet for the director server features

4.3.2 User Interface (UI) Design

Building upon UX research findings, the UI design of the SecOTA framework focused on usability and visual clarity. Snippets of the UI showcase the intuitive layout and functionality designed to streamline user interactions with backend functionalities.

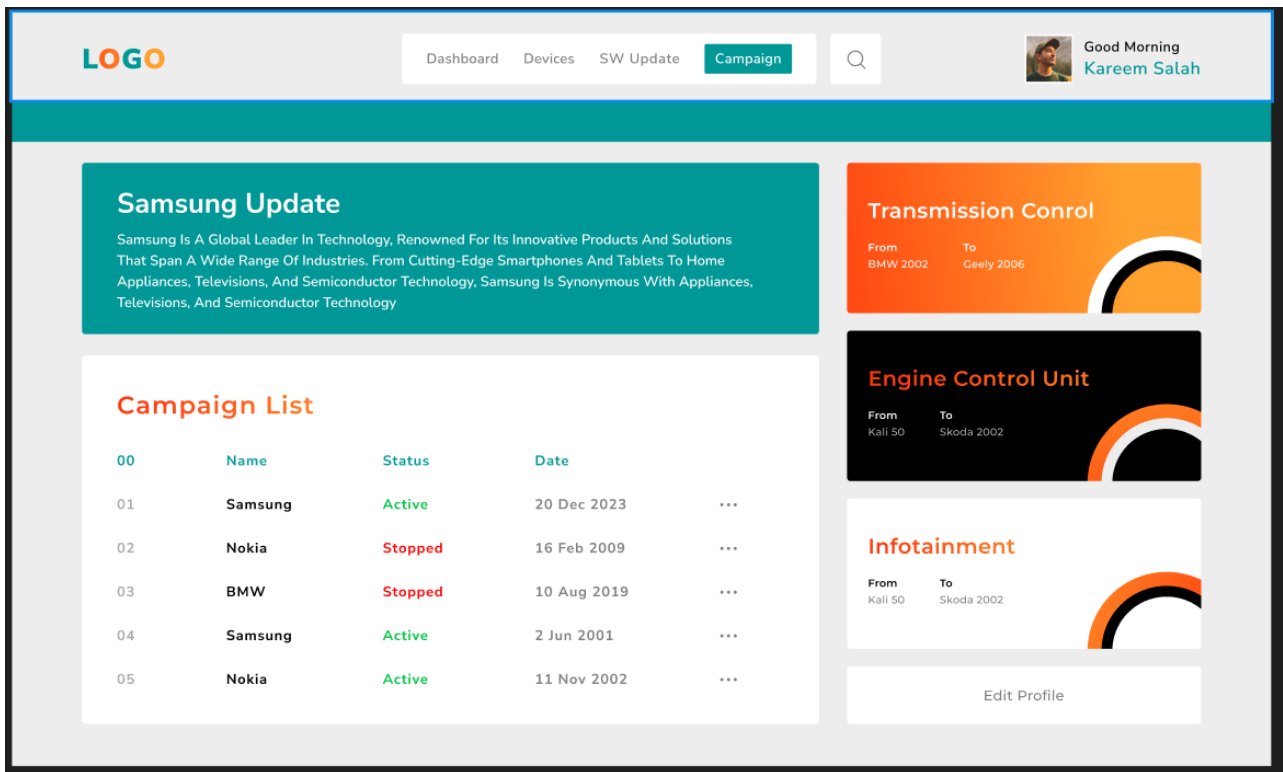


Figure 4.4: Campaign list UI screen

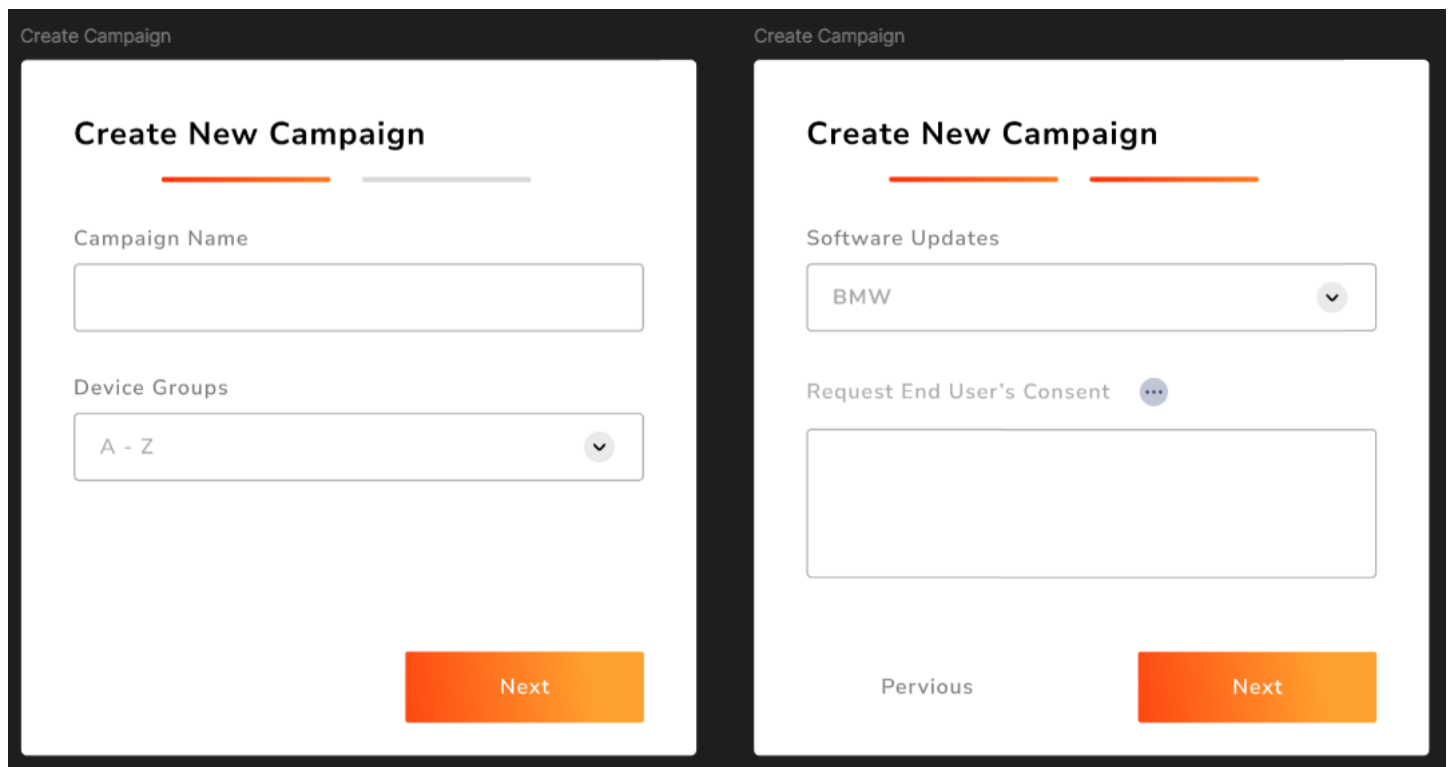


Figure 4.5: Campaign creation UI screens

4.3.3 Threat Analysis and Risk Assessment (TARA) Results

A rigorous Threat Analysis and Risk Assessment (TARA), conducted according to ISO 21434 standards, identified potential security threats and vulnerabilities. The results highlighted critical attack vectors and informed the implementation of robust security measures to mitigate risks.

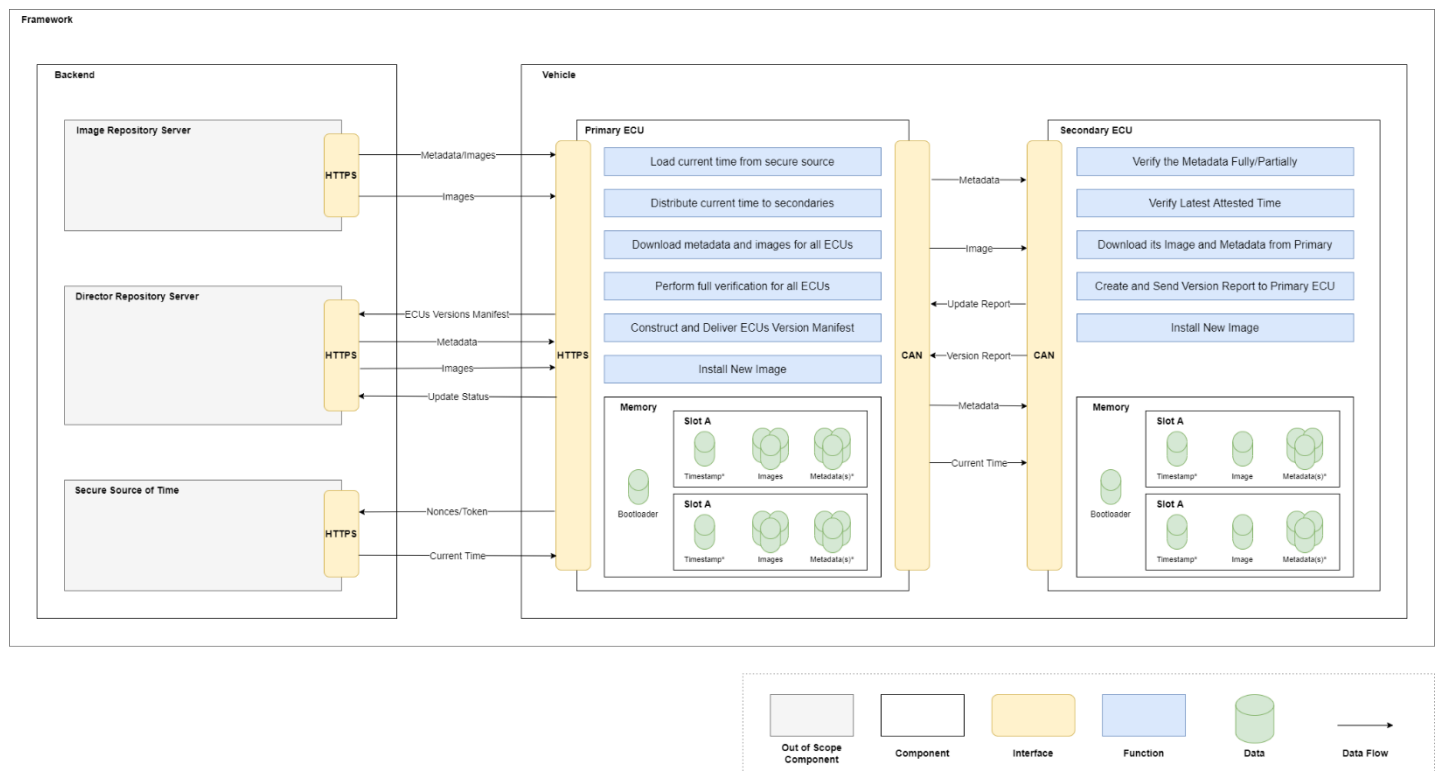


Figure 4.4: TARA Item definition

Throughout the document, each vulnerability or risk identified has been labeled as a Finding and categorized as a **Critical-Risk**, **High-Risk**, **Moderate-Risk**, or **Low-Risk**. Besides, each supplemental testing note is labeled as an Issue. These terms are defined below:

Table 4.1: Vulnerabilities severity definition

Critical Risk: These findings identify conditions that directly result in the complete compromise or unauthorized access of a network, system, application, or information. Also, it's easy to be exploited. Examples of Critical Risks include known command injection, and SQL injection, which could result in owning critical systems or services; unauthorized access; and disclosure of information.

High Risk: These findings identify conditions that could directly result in the compromise or unauthorized access of a network, system, application, or information. Examples of High Risks include known buffer overflows, weak or no passwords, and no encryption, which could result in denial of service on critical systems or services; unauthorized access; and disclosure of information.

Resolved Attacks

Table 4.2: Deprecated OpenSSL vulnerability card

#	Vulnerability Name	Severity
1	Deprecated OpenSSL	High
Description: Deprecated versions of OpenSSL expose systems to a variety of critical security vulnerabilities due to the absence of updates and reliance on outdated cryptographic algorithms.		
Impact:		
Lack of Security Patches: Deprecated versions do not receive updates, leaving the system vulnerable to new exploits and security holes.		
Outdated Cryptographic Algorithms: Older versions support cryptographic algorithms that are no longer secure, such as MD5 and SHA-1, which are susceptible to collision attacks.		
Susceptibility to Known Attacks: Systems using deprecated OpenSSL versions are exposed to known vulnerabilities that have been patched in newer versions, making them easier targets for attackers.		
Remediation: Update and rewrite cryptographic operations to use the latest version of OpenSSL. This ensures the use of current, secure algorithms and the application of all recent security patches.		

Table 4.3: Process manipulation vulnerability card

#	Vulnerability Name	Severity
2	Process Manipulation with Firmware Update Privileges	Critical
<p>Description: A vulnerability exists when a process with privileged access to update the ECU firmware can be manipulated by a malicious process running on the same system. This manipulation can lead to unauthorized firmware modifications, compromising the integrity and security of the ECU.</p> <p>Impact:</p> <p>Unauthorized Firmware Updates: Malicious processes can inject or alter firmware updates, leading to unauthorized code execution on the ECU.</p> <p>Compromised System Integrity: Unauthorized modifications can introduce vulnerabilities or malicious code into the firmware, affecting the overall system security and functionality.</p> <p>Persistent Malware: Malicious firmware can embed persistent malware, which remains on the system even after rebooting or other recovery attempts.</p> <p>Denial of Service: Manipulating the firmware update process can cause system instability or render the ECU inoperative, leading to a denial of service.</p> <p>Remediation: Sandbox privileged processes to limit their interaction with other processes and reduce the risk of manipulation.</p>		

Table 4.4: open ports and insecure connections vulnerability card

#	Vulnerability Name	Severity
3	Open ports and insecure connections	Critical
<p>Description: A vulnerability exists when a system has open ports and unsecured network connections, making it susceptible to unauthorized access, data breaches, and potential denial of service attacks. Open ports provide entry points for attackers to exploit, and unsecured connections can be intercepted or manipulated, compromising the integrity and confidentiality of the data.</p> <p>Impact:</p> <p>Unauthorized Access: Attackers can exploit open ports to gain unauthorized access to the system, leading to data breaches and system compromise.</p> <p>Data Interception: Unsecured connections can be intercepted by malicious actors, allowing them to eavesdrop on sensitive communications or inject malicious data.</p> <p>Denial of Service: Attackers can exploit open ports to launch denial of service (DoS) attacks, disrupting the normal operation of the system and causing outages.</p> <p>System Compromise: Open ports can be used to exploit known vulnerabilities in services running on those ports, leading to a full system compromise.</p> <p>Remediation: Utilize Intrusion Prevention Systems to continuously monitor network traffic, detect potential threats, and block malicious activities targeting open ports and unsecured connections.</p>		

Table 4.5: drop-request attack card

#	Vulnerability Name	Severity
4	Drop-request Attack	High
Description: blocks network traffic outside or inside the vehicle to prevent an ECU from receiving any updates.		

Table 4.6: slow retrieval attack card

#	Vulnerability Name	Severity
5	Slow Retrieval Attack	High
Description: Slows the delivery time of updates to ECUs so a known security vulnerability can be exploited before a corrective patch is received.		

Table 4.7: mix-and-match attack card

#	Vulnerability Name	Severity
6	Mix-and-match attack	Critical
Description: If attackers have compromised repository keys, they can use these keys to release arbitrary combinations of new versions of images.		

Table 4.8: Freeze attack card

#	Vulnerability Name	Severity
7	Freeze Attack	Critical
Description: Continues to send the last known update to an ECU, even if a newer update exists.		

Table 4.9: partial bundle installation attack card

#	Vulnerability Name	Severity
8	Partial bundle installation attack	High
Description: Allows only part of an update to install by dropping traffic to selected ECUs.		

Table 4.10: rollback attack card

#	Vulnerability Name	Severity
9	Rollback attack	High
Description:	Tricks an ECU into installing outdated software with known vulnerabilities.	

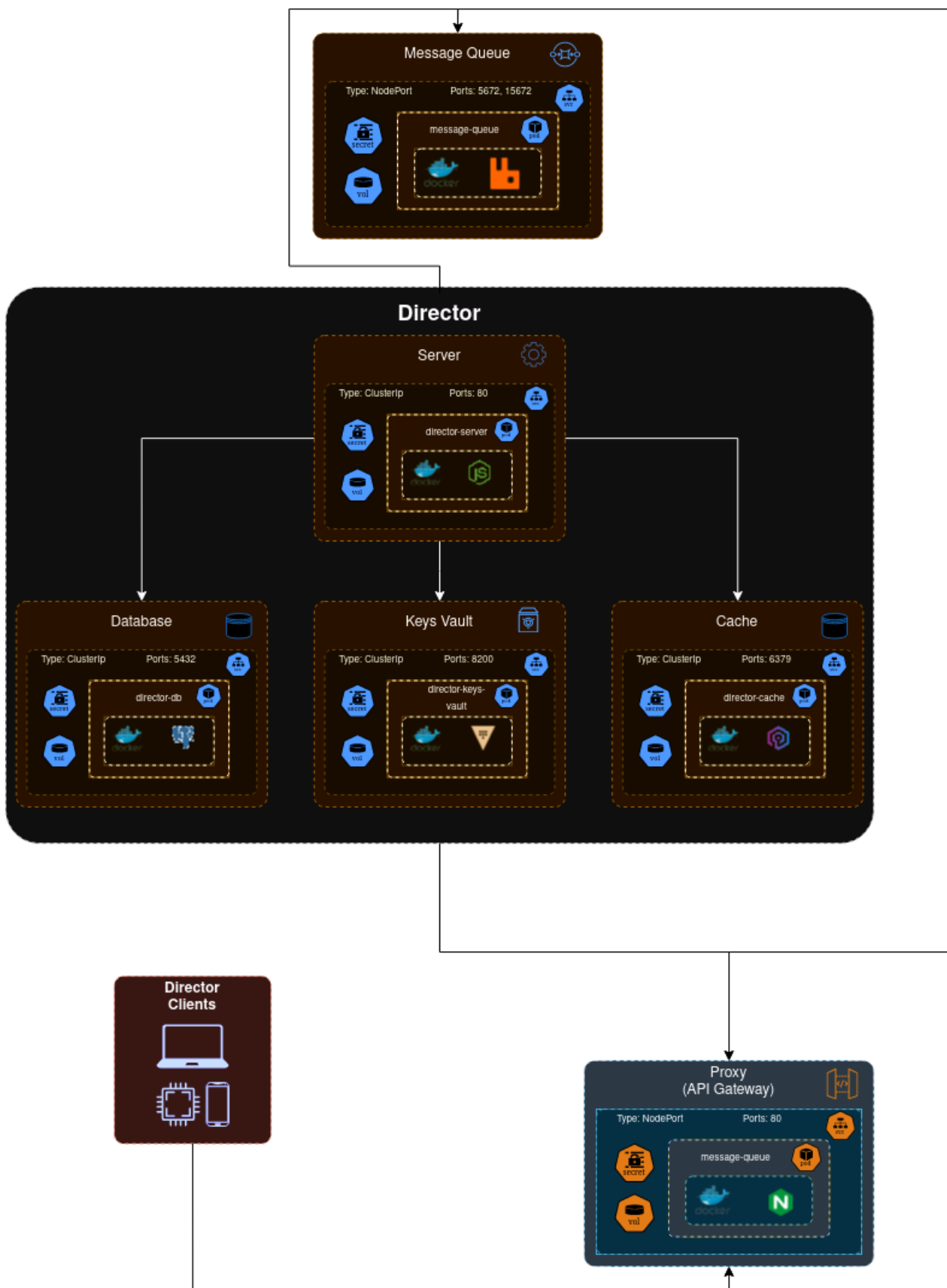
Table 4.11: Mixed-bundles attack card

#	Vulnerability Name	Severity
10	Mixed-bundles attack	High
Description:	Shuts down an ECU by causing it to install incompatible versions of software updates that must not be installed at the same time. Attackers can accomplish this by showing different bundles to different ECUs at the same time.	

Table 4.12: endless data attack card

#	Vulnerability Name	Severity
11	Endless Data Attack	High
Description:	Causes an ECU to crash by sending it an infinite amount of data until it runs out of storage.	

4.3.4 Deployment Diagram



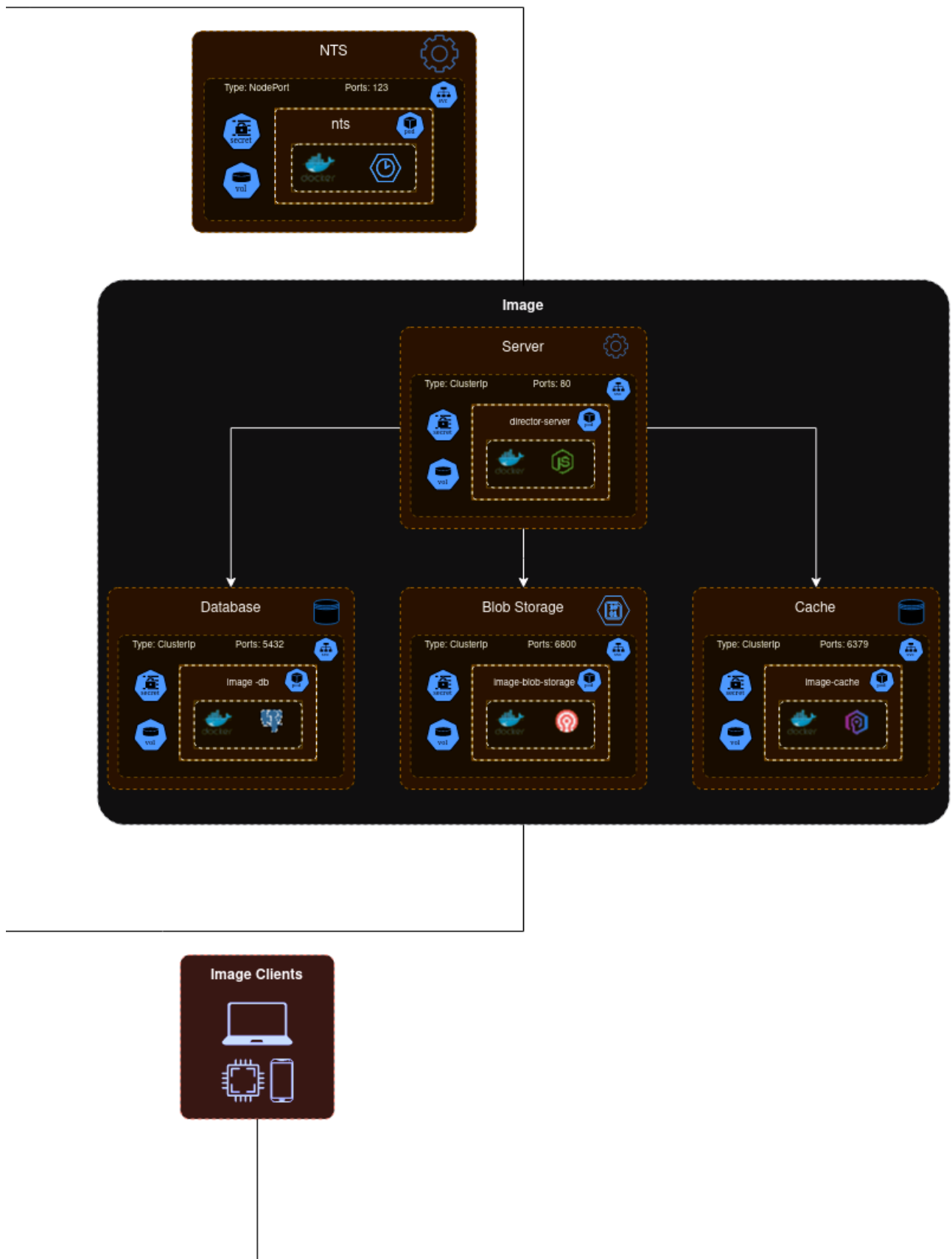
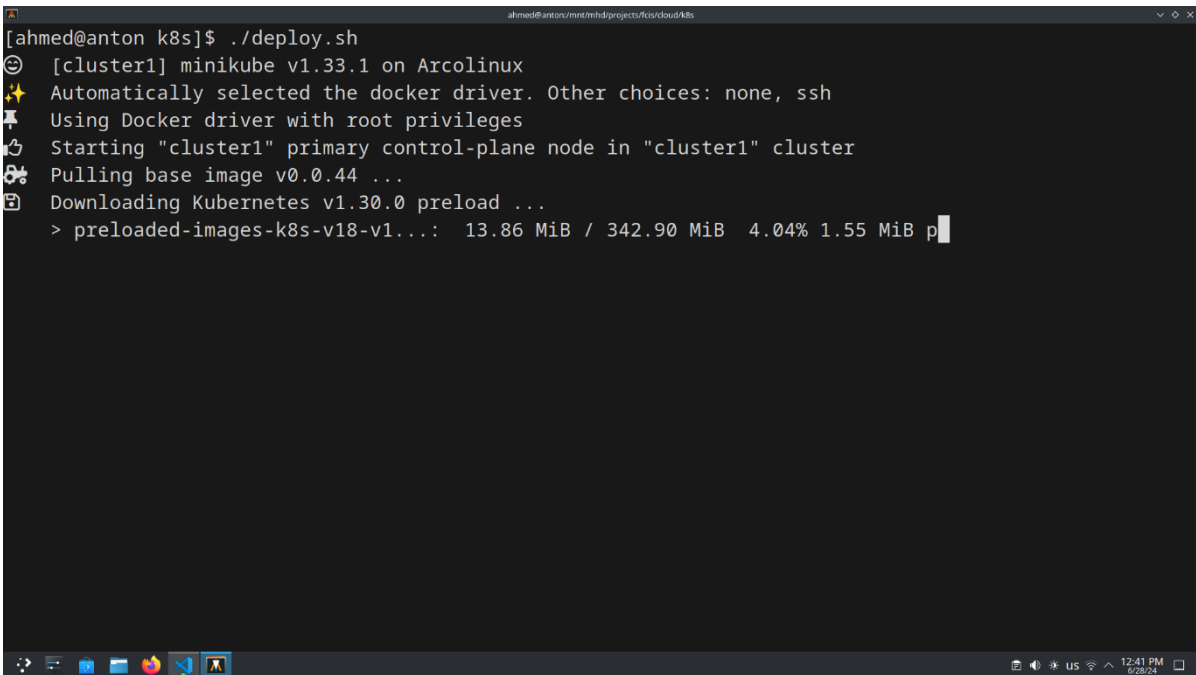


Figure 4.5: System deployment diagram

Chapter 5:

Application Run

5.1 Deploy Server Infrastructure

A terminal window titled 'ahmed@anton/minikube/projects/fco/cloud48s' showing the execution of a script to deploy a Kubernetes cluster. The script output includes: '[cluster1] minikube v1.33.1 on Arcolinux', 'Automatically selected the docker driver. Other choices: none, ssh', 'Using Docker driver with root privileges', 'Starting "cluster1" primary control-plane node in "cluster1" cluster', 'Pulling base image v0.0.44 ...', 'Downloading Kubernetes v1.30.0 preload ...', and a progress bar for 'preloaded-images-k8s-v18-v1...' showing 13.86 MiB / 342.90 MiB at 4.04% (1.55 MiB per second). The terminal has a dark background with light gray text and a standard Linux desktop taskbar at the bottom.

```
[ahmed@anton k8s]$ ./deploy.sh
[cluster1] minikube v1.33.1 on Arcolinux
Automatically selected the docker driver. Other choices: none, ssh
Using Docker driver with root privileges
Starting "cluster1" primary control-plane node in "cluster1" cluster
Pulling base image v0.0.44 ...
Downloading Kubernetes v1.30.0 preload ...
> preloaded-images-k8s-v18-v1...: 13.86 MiB / 342.90 MiB  4.04% 1.55 MiB p
```

Figure 5.1: Deploy Kubernetes cluster

```

service/image          ClusterIP 10.98.131.37 <none> 8002/TCP 80m
service/image-front    ClusterIP 10.103.135.165 <none> 8002/TCP 29s
service/kubernetes     ClusterIP 10.96.0.1 <none> 443/TCP 80m
service/nginx          NodePort 10.98.77.124 <none> 80:30000/TCP 80m
service/postgres-director ClusterIP 10.104.18.161 <none> 5432/TCP 80m
service/postgres-image  ClusterIP 10.98.27.163 <none> 27017/TCP 80m
service/rabbitmq       ClusterIP 10.104.87.91 <none> 5672/TCP,15672/TCP 80m

NAME                  READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/director  0/1    1            0          80m
deployment.apps/director-front  0/1    1            0          30s
deployment.apps/image     0/1    1            0          80m
deployment.apps/image-front  0/1    1            0          30s
deployment.apps/nginx     0/1    1            0          80m
deployment.apps/postgres-director  0/1    1            0          80m
deployment.apps/postgres-image  0/1    1            0          80m
deployment.apps/rabbitmq   0/1    1            0          80m

NAME                  DESIRED  CURRENT  READY  AGE
replicaset.apps/director-677d65bc9f 1 1 0 80m
replicaset.apps/director-front-d7bbdff6f 1 1 0 21s
replicaset.apps/image-86bb9cf4fd 1 1 0 80m
replicaset.apps/image-front-77fd68c56d 1 1 0 21s
replicaset.apps/nginx-5889559cc 1 1 0 80m
replicaset.apps/postgres-director-b6765cf94 1 1 0 80m
replicaset.apps/postgres-image-95c978cc8 1 1 0 80m
replicaset.apps/rabbitmq-66995df4f7 1 1 0 80m

```

Figure 5.2: Deployed pods

```

Resources in cluster1:

NAME                  READY  STATUS             RESTARTS  AGE
pod/director-677d65bc9f-lcbg6 0/1    ContainerCreating  0          80m
pod/director-front-d7bbdff6f-fmqsm 0/1    ContainerCreating  0          21s
pod/image-86bb9cf4fd-cnd4k 0/1    ContainerCreating  0          80m
pod/image-front-77fd68c56d-b6qpb 0/1    ContainerCreating  0          21s
pod/nginx-5889559cc-zvdp5 0/1    ContainerCreating  0          80m
pod/postgres-director-b6765cf94-vrhdn 0/1    ContainerCreating  0          80m
pod/postgres-image-95c978cc8-dtbq4 0/1    ContainerCreating  0          80m
pod/rabbitmq-66995df4f7-ljrz 0/1    ContainerCreating  0          80m

NAME                  TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/director      ClusterIP     10.111.19.225   <none>            8001/TCP          80m
service/director-front ClusterIP     10.109.94.213   <none>            8001/TCP          29s
service/image          ClusterIP     10.98.131.37    <none>            8002/TCP          80m
service/image-front    ClusterIP     10.103.135.165  <none>            8002/TCP          29s
service/kubernetes     ClusterIP     10.96.0.1       <none>            443/TCP          80m
service/nginx          NodePort      10.98.77.124    <none>            80:30000/TCP      80m
service/postgres-director ClusterIP     10.104.18.161   <none>            5432/TCP          80m
service/postgres-image  ClusterIP     10.98.27.163    <none>            27017/TCP         80m
service/rabbitmq       ClusterIP     10.104.87.91    <none>            5672/TCP,15672/TCP 80m

NAME                  READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/director  0/1    1            0          80m
deployment.apps/director-front  0/1    1            0          30s
deployment.apps/image     0/1    1            0          80m
deployment.apps/image-front  0/1    1            0          30s

```

Figure 5.3: Deployed pods 2

5.2 Integrate target with application

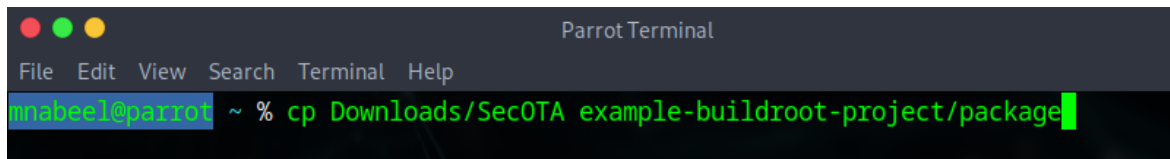


Figure 5.4: Move SecOTA package to the buildroot project



Figure 5.5: Open buildroot config file with nano

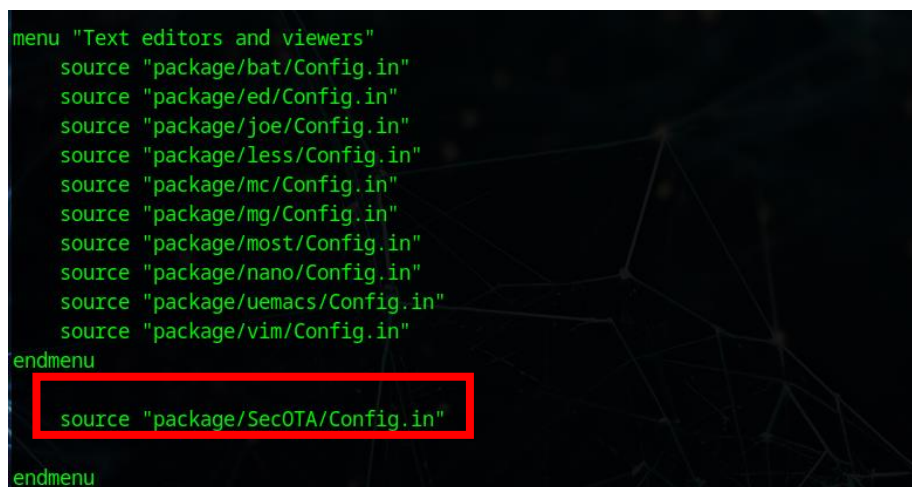


Figure 5.6: Add SecOTA package to the menu

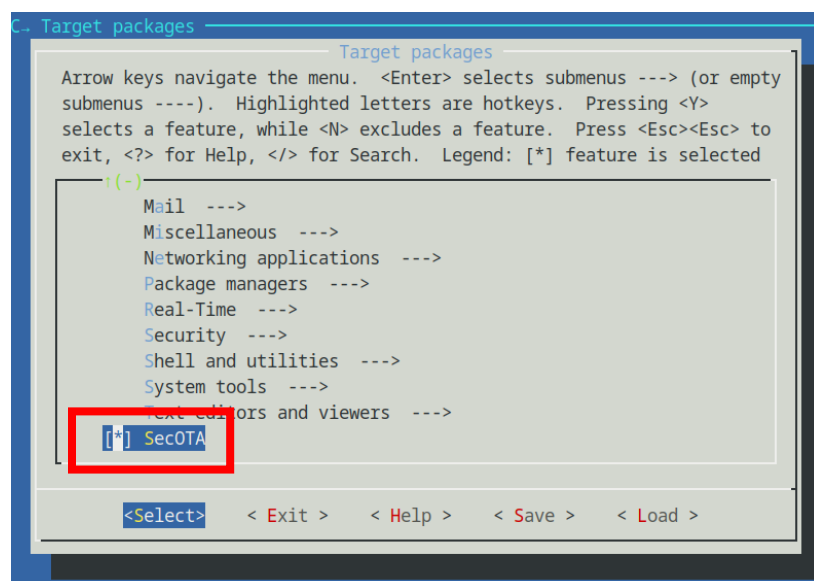


Figure 5.7: Choose SecOTA package in the menu

5.3 Devices Management

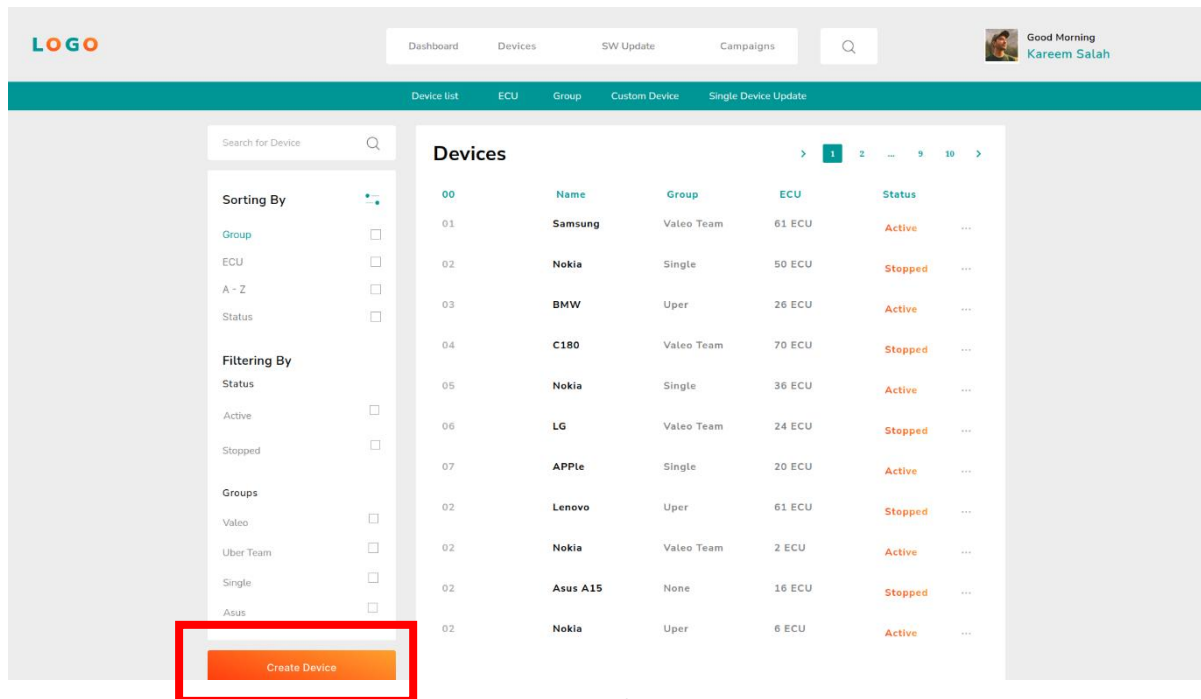


Figure 5.2: List device screen

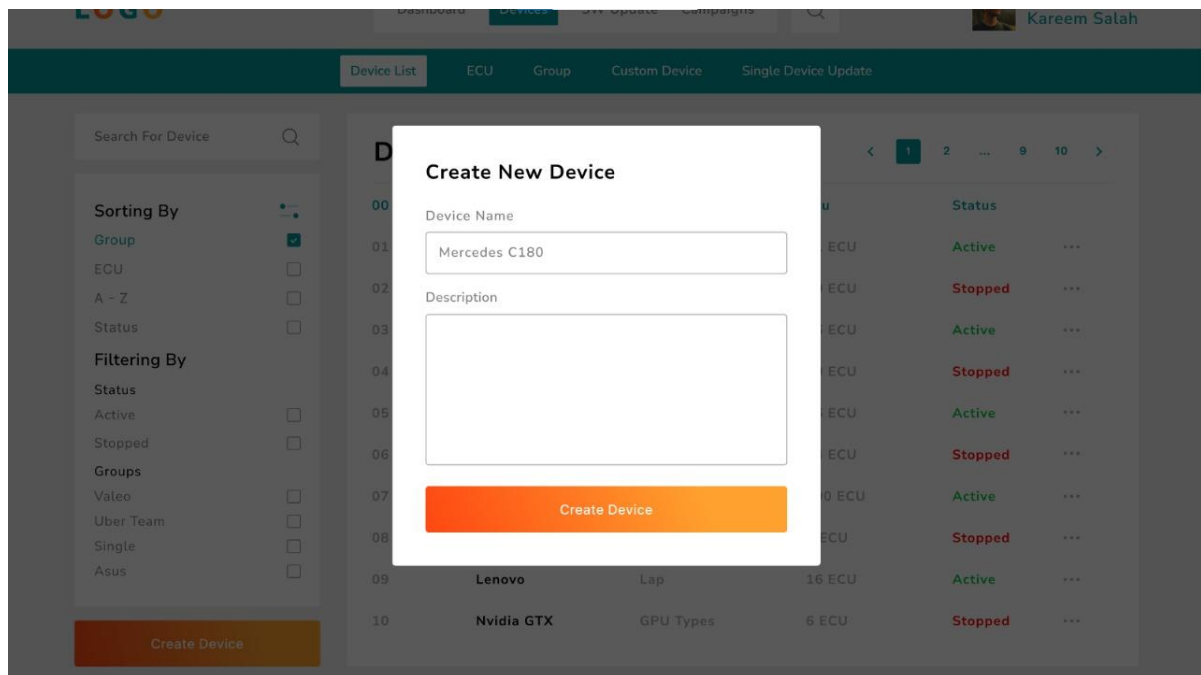


Figure 5.3: Create a new device

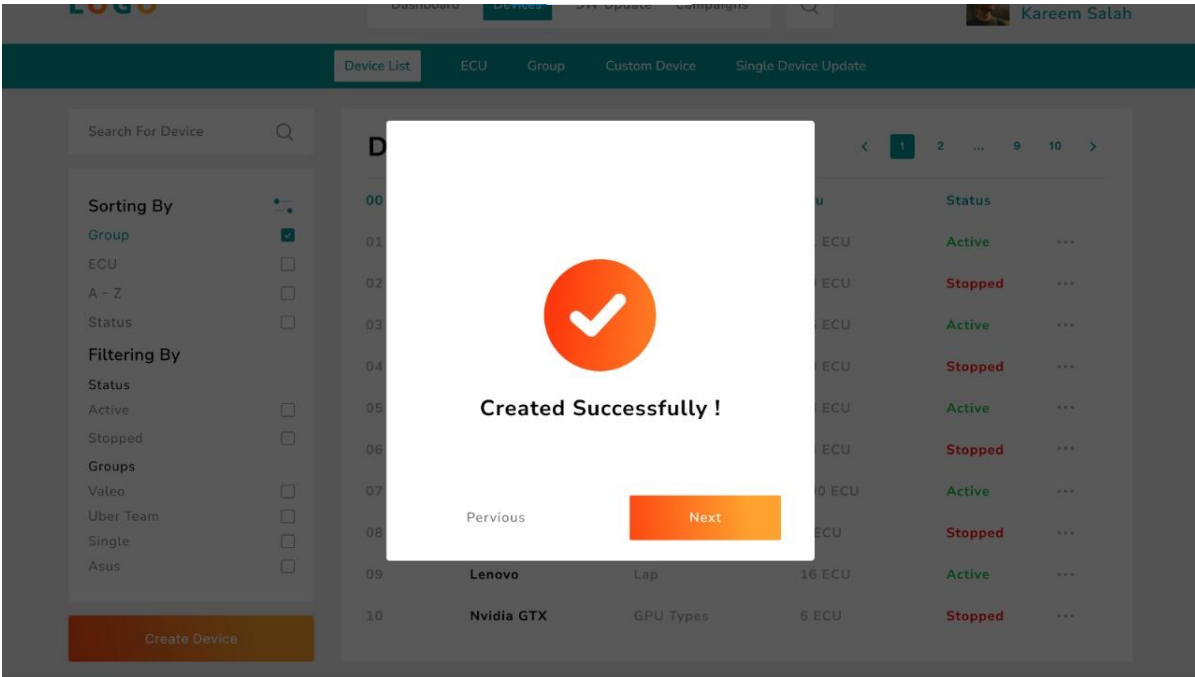


Figure 5.4: Successfully created a device

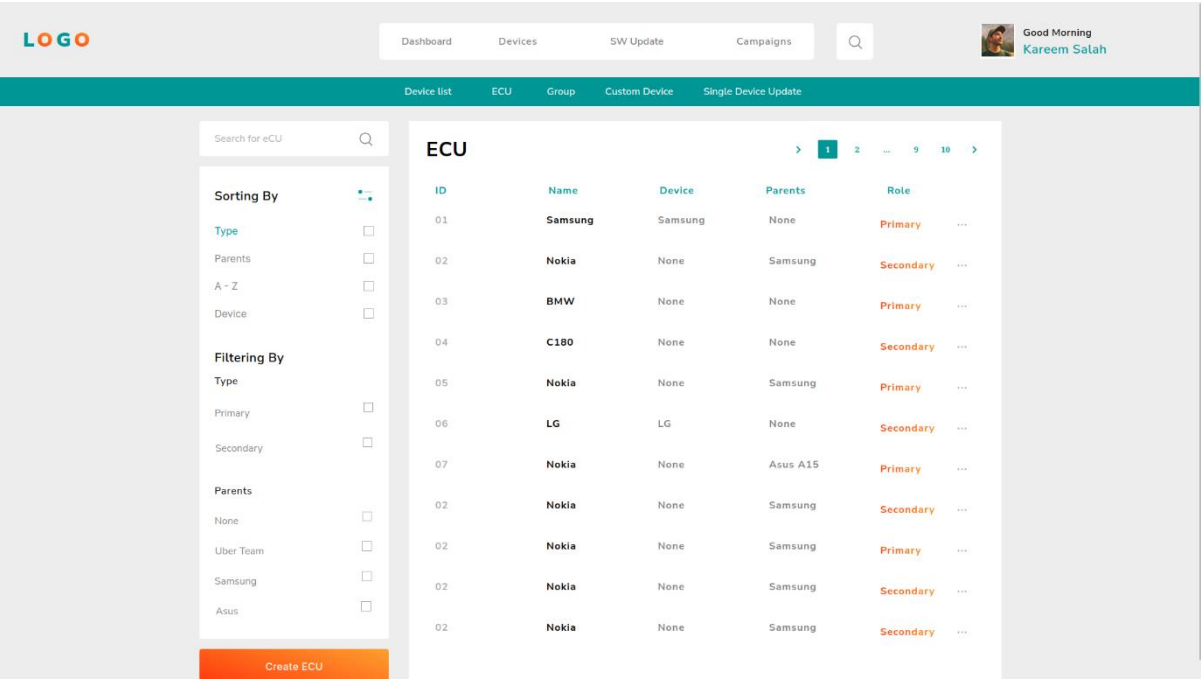


Figure 5.5: List ECUs screen

Create New ECU

#Identifier
ECU-6d3dcffb05b54fd1901ae15b89a12f82

ECU Name
Transmission Unit

Next

ID	Name	Role
01		Primary
02		Secondary
03		Primary
04		Primary
05		Secondary
06		Primary
07		Secondary
08		Primary
09		Secondary
10	Nvidia GTX	Primary

Figure 5.6: Step 1 of creating a new ECU

Create New ECU

ECU Role
Primary

ECU Type
Raspberry Pi 4

Previous Next

ID	Name	Role
01		Primary
02		Secondary
03		Primary
04		Primary
05		Secondary
06		Primary
07		Secondary
08		Primary
09	Lenovo	Secondary
10	Nvidia GTX	Primary

Figure 5.7: Step 2 of creating a new ECU

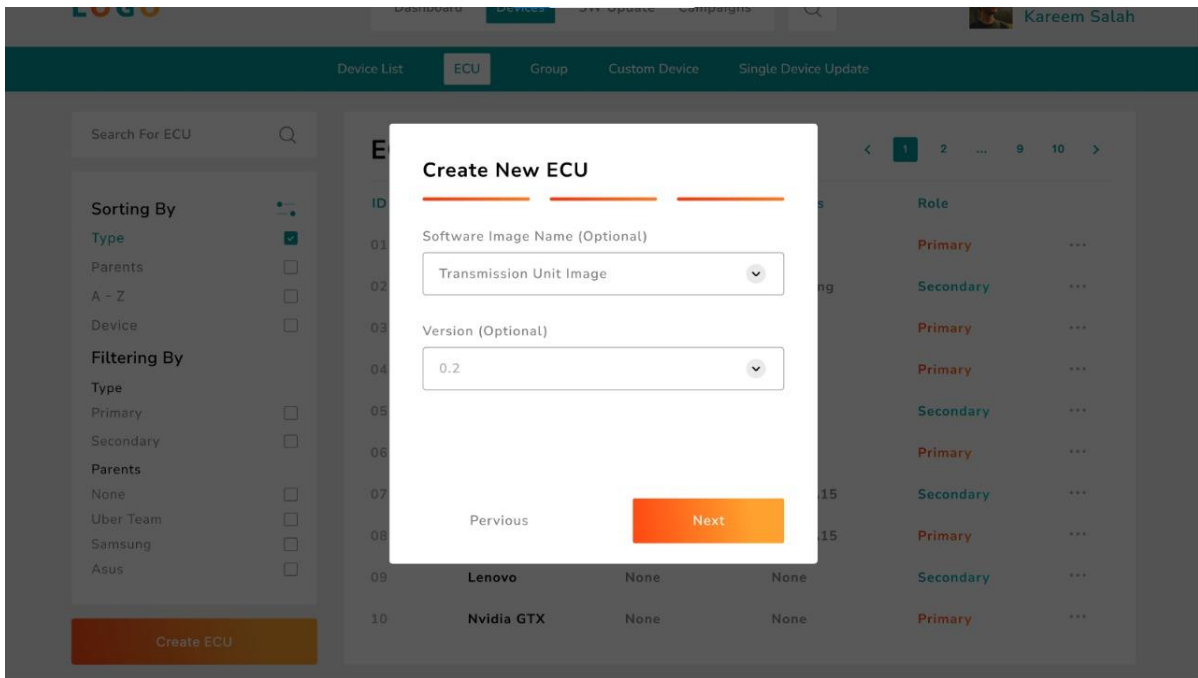


Figure 5.14: Step 3 of creating a new ECU

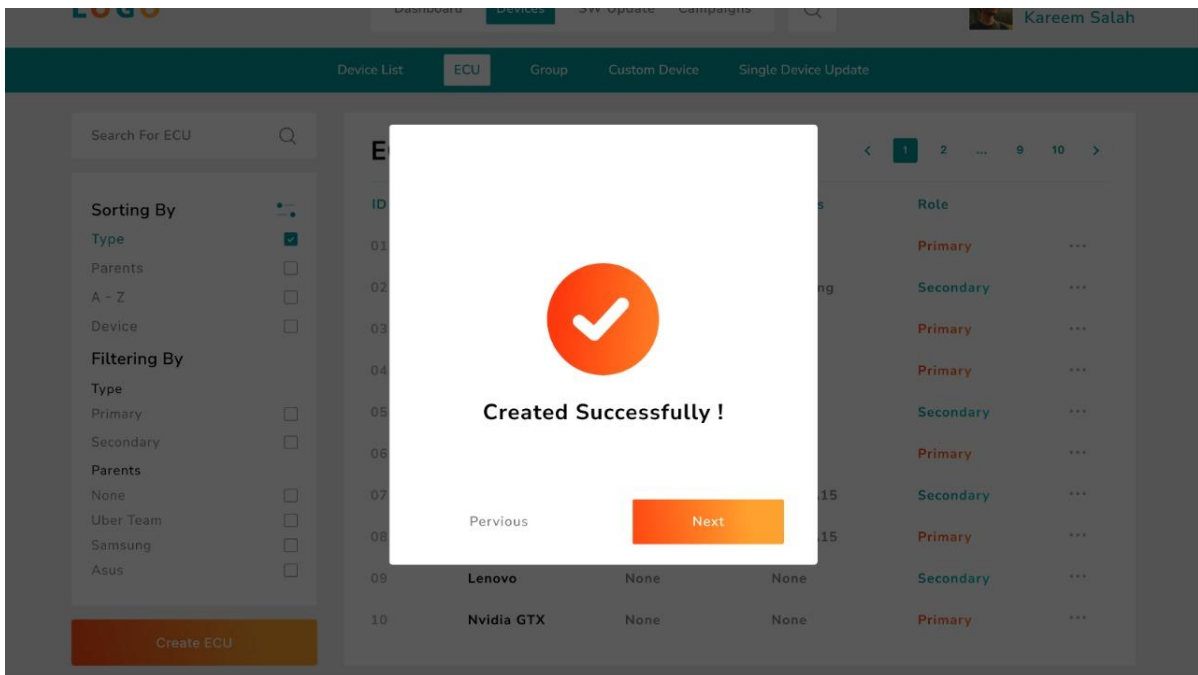


Figure 5.15: Successfully created an ECU

LOGO

Dashboard
Devices
SW Update
Campaigns

Q

Good Morning Kareem Salah

Search for Device

Sorting By

Campaign

A - Z

Status

Filtering By

Status

Active

Stopped

Campaign

Valeo

Uber Team

Single

Asus

Create New Sw

Software Updates

1

2

9

10

00	Name	Campaigns	Status	
01	Samsung	Valeo Team	Active	...
02	Nokia	Single	Stopped	...
03	BMW	Uper	Active	...
04	C180	Valeo Team	Stopped	...
05	Nokia	Single	Active	...
06	LG	Valeo Team	Stopped	...
07	APPIe	Single	Active	...
02	Lenovo	Uper	Stopped	...
02	Nokia	Valeo Team	Active	...
02	Asus A15	None	Stopped	...
02	Nokia	Uper	Active	...

Figure 5.16: Software updates list

LOGO

Dashboard
Devices
SW Update
Campaigns

Q

Good Morning Kareem Salah

Samsung Update

Samsung Is A Global Leader In Technology, Renowned For Its Innovative Products And Solutions That Span A Wide Range Of Industries, From Cutting-Edge Smartphones And Tablets To Home Appliances, Televisions, And Semiconductor Technology. Samsung Is Synonymous With Appliances, Televisions, And Semiconductor Technology

Campaign List

00	Name	Status	Date	
01	Samsung	Active	20 Dec 2023	...
02	Nokia	Stopped	16 Feb 2009	...
03	BMW	Stopped	10 Aug 2019	...
04	Samsung	Active	2 Jun 2001	...
05	Nokia	Active	11 Nov 2002	...

Transmission Control

From BMW 2002 To Geely 2006

Engine Control Unit

From Kali 50 To Skoda 2002

Infotainment

From Kali 50 To Skoda 2002

Edit Profile

Figure 5.17: Campaigns List

56

Chapter 6:

Conclusion and Future Work

6.1 Conclusion

Throughout this thesis, the SecOTA framework has been developed and implemented to address crucial challenges in secure Flash Over-The-Air (FOTA) updates for embedded and IoT systems. By integrating The Update Framework (TUF) adapted through Uptane and adhering to ISO 21434 automotive cybersecurity standards, SecOTA offers a robust solution for enhancing software update security and mitigating emerging cyber threats. This framework not only improves operational reliability but also reduces the risk of cybersecurity incidents related to compromised software.

SecOTA's significance lies in its comprehensive approach to securing FOTA mechanisms, which have historically lacked sufficient security measures. By deploying SecOTA, organizations can mitigate financial losses associated with software recalls, thereby ensuring safer and more efficient operations in automotive, industrial IoT, and consumer electronics sectors. Future enhancements could focus on integrating advanced anomaly detection and expanding compatibility with diverse hardware platforms to adapt to evolving cybersecurity landscapes.

Looking forward, continuous refinement and optimization of SecOTA based on ongoing threat assessments and industry feedback will be essential. Improving scalability and interoperability will further broaden its applicability across various embedded and IoT ecosystems. Collaboration with industry stakeholders

and regulatory bodies can establish standardized protocols for secure firmware updates, fostering trust and interoperability in global markets.

In conclusion, while this thesis has made significant strides in advancing secure FOTA solutions, ongoing research and development efforts are vital to further refine SecOTA and address emerging challenges. By embracing proactive cybersecurity measures and innovation, we can ensure the resilience and trustworthiness of connected devices in today's interconnected world.

6.2 Future Work

6.2.1 Enhanced Anomaly Detection and Threat Intelligence Integration

Future research could focus on enhancing SecOTA's anomaly detection capabilities by integrating advanced machine learning algorithms. These algorithms could improve the framework's ability to identify and respond to anomalous behaviors that may indicate sophisticated cyber threats. Additionally, integrating real-time threat intelligence feeds could further enhance SecOTA's proactive defense mechanisms, ensuring timely response to emerging cyber threats.

6.2.2 Expansion of Hardware Platform Compatibility

To broaden SecOTA's applicability, future efforts could focus on expanding compatibility with a wider range of embedded and IoT hardware platforms. This includes optimizing the framework to seamlessly integrate with diverse device architectures and ensuring interoperability across different manufacturers and models. Enhancing hardware platform compatibility will facilitate the broader adoption of SecOTA across various industry sectors.

6.2.3 Standardization and Certification

Collaboration with industry standards organizations and regulatory bodies will be crucial to establishing standardized protocols and certification processes for secure FOTA frameworks. Future work could involve working closely with these entities to define best practices, guidelines, and certification criteria for secure firmware updates in embedded and IoT systems. This initiative aims to promote trust, interoperability, and compliance with global cybersecurity standards.

6.2.4 Continuous Improvement and User Feedback Integration

Continuous refinement of SecOTA based on user feedback and ongoing threat assessments is essential. Future research should prioritize gathering feedback from industry stakeholders, end-users, and cybersecurity experts to identify areas for improvement and usability enhancements. Implementing an iterative development cycle will ensure that SecOTA evolves to meet the evolving cybersecurity landscape and user requirements effectively.

6.2.5 Exploration of Blockchain Technology Integration

Exploring the integration of blockchain technology could offer additional layers of security and transparency to SecOTA. Blockchain's decentralized and immutable ledger capabilities could potentially enhance the integrity and auditability of firmware updates, providing a secure and verifiable record of update transactions. Future research could investigate how blockchain can complement existing security measures in SecOTA.

6.2.6 Ethical Considerations and Privacy Protection

As SecOTA evolves, addressing ethical considerations and privacy concerns surrounding firmware updates will be paramount. Future work should include robust privacy-by-design principles and mechanisms to protect sensitive data during the update process. Ensuring compliance with data protection regulations and ethical guidelines will enhance user trust and mitigate potential risks associated with firmware update practices.

References

- [1] Uptane Framework, "Secure Software Update Framework for Automobiles."
- [2] E. Cebel, N. Donum, and H. Karacali, "Platform Independent Embedded Linux OTA Method," *The European Journal of Research and Development*, vol. 2, no. 4, pp. 243–252, 2022.
- [3] S. E. Jaouhari and E. Bouvet, "Toward a generic and secure bootloader for IoT device firmware OTA update," *International Conference on Information Networking (ICOIN)*, Jeju-si, Korea, 2022, pp. 90-95, doi: 10.1109/ICOIN53446.2022.9687242.
- [4] L.-C. Duca, A. Duca, and C. Popescu, "OTA Secure Update System for IoT Fleets," *International Journal of Advanced Networking and Applications*, vol. 13, Dec. 2021.
- [5] M. Opdenacker, "Implementing A/B System Updates with U-boot," *Embedded Linux Conference Europe*, 2022.
- [6] M. Nottingham, Ed., "The Network Time Security (NTS) Protocol," RFC 8915, Internet Engineering Task Force, Sep. 2020.