Politecnico di Torino

Politecnico di Torino

Master's Degree course in Computer Engineering

A.a. 2024/2025

Graduation Session October 2025

From Attacks to Defense: Security Strategies for Custom Embedded Devices

Supervisor:

Alessandro Savino

Stefano Di Carlo

Candidates:

Emanuela Bagnato

Internship tutor:

Carmelo Migliore

Santer Reply S.r.l

In partnership with:

Concept Reply, Santer Reply S.r.l.

Abstract

Cybersecurity has become increasingly important and critical as the world grows more interconnected and the Internet of Things (IoT) establishes itself as a global reality.
Today, most embedded devices operate as IoT components, making their protection a priority. Over time, several international standards have been developed to define best practices for cybersecurity in this field, and organizations are mostly required to comply with them.
After an in-depth study of the principal standards and best practices, this thesis focus shifted on the analysis of vulnerabilities in a custom embedded device. Starting from a static vulnerability assessment, targeted tests were performed, and based on the results improvements were subsequently introduced to enhance the device's security posture.
Particular attention was dedicated to the EN 18031-1 standard, which became a mandatory certification on August 1st of the current year, ensuring that the device was brought into full compliance with this standard.
Finally, the work highlights future directions, especially the challenges introduced by artificial intelligence, which will play an increasingly important role in securing next-generation embedded and IoT systems.

Summary

This thesis explores cybersecurity in embedded systems, focusing on best practices, industry standards, and specific vulnerabilities in connected devices. The primary goal is to analyze effective protection measures, identify risks, and propose mitigation strategies for IoT and critical systems.
The introduction presents the research context, objectives, methodology, and the overall structure of the thesis, highlighting the importance of embedded system security, particularly in industrial and IoT environments.
The Best Practices in Embedded Cybersecurity chapter details security guidelines and standards, including the OWASP Top 10 for IoT, ISA/IEC 62443, ETSI EN 303 645, and NIST IoT Security Standards. Key security domains are covered: secure access (strong authentication, RBAC, secure remote access), protection of debug interfaces, key and certificate management, Secure Boot mechanisms, secure OTA updates, data protection (in transit and at rest), and physical device security. Embedded operating system security, encryption, network security, and application resilience are also discussed.
Chapter 3, Threats and Vulnerabilities in Embedded Devices, analyzes major threats and vulnerabilities, such as firmware modification, default or hardcoded credentials, unencrypted communications, memory attacks, and network protocol attacks. Real-world case studies include Stuxnet, Amnesia:33, and Mirai, with corresponding mitigation strategies.
Chapter 4 introduces the Yocto Project, its architecture, core components (Poky, Bitbake, meta-layers), and advantages for developing secure embedded systems. It covers CVE management, package updates, security tools, and limitations.
After reviewing the relevant literature and analyzing the technologies employed, a practical assessment of the device was conducted, chapter 5 resents a practical case study assessing a specific device, including initial configuration, implemented security technologies (Secure Boot, dm-verity, dm-crypt, Mender OTA, firewall, VPN, SELinux), and mitigation strategies.
Some Penetration Testing Activities such as network reconnaissance, DoS simulation, encrypted partition testing, firmware fuzzing, and side-channel analysis are described in chapter 6 and the result of this testing is presented.

A final chapter, Test Results and Validation, presents the results of the tests where vulnerabilities were discovered, and recommendations for risk mitigation.
A particular focus was put into analyzing existing SELinux policies and developing custom policies for services such as Modbus and EdgeX, with validation and evaluation of security impact.
Considered the recent forced application of the Directive EN 18031 an evaluation of the device studied was done to evaluate compliance with the standard by analyzing relevant sections and improvements implemented in the studied device.
Given the widespread adoption of AI, a chapter is dedicated to exploring AI applications in embedded cybersecurity, including threat detection, vulnerability management, and potential AI-powered attacks. Ethical concerns, bias, transparency, human oversight, and best practices are discussed.
The conclusion summarizes the main findings, contributions, limitations, and future directions for improving security in embedded and IoT systems.

Acknowledgements

I would like to thank Professor Alessandro Savino for giving me the opportunity to carry out this thesis, and for the attention and availability shown throughout the process. Through his teaching, he sparked my interest in this subject, ultimately guiding me toward this thesis.

A special appreciation goes to my company tutor, Carmelo Migliore and to Riccardo Midena, for giving me the opportunity to learn so many new things, both in the field and about the professional world. I sincerely thank them for their almost daily support during the entire project, for providing continuous feedback, and for their trust in my abilities, and for all the jokes and fun moments that we shared. I would also like to thank everyone in the team for welcoming me, for always being willing to help whenever I needed it, and for making the entire experience more enjoyable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and Motivation

Lately, digital transformation has accelerated the integration of computing capabilities and introduced internet connection into everyday objects, transforming them into smart and interconnected systems. The Internet of Things (IoT), has brought significant improvements in every day life such as healthcare, transportation, industrial automation, and electronics. At the foundation of this new ecosystem are embedded devices: small, resource-constrained systems designed for specific functionalities that are now expected to operate in increasingly complex and critical digital environments.
IoT technologies are becoming increasingly important considering how much business benefit from them and the wide range of opportunities they create. Their quick expansion has not left time to adapt security measure so a new attack surface has been generated for ill-intentioned users to exploit. Unlike traditional IT infrastructures, embedded systems often lack computational capacity, memory, and energy capability required to implement strong security measures. As a result, IoT devices remain exposed to vulnerabilities ranging from minor misconfigurations to sophisticated exploits involving hardware tampering and side-channel analysis.
Cybersecurity for embedded systems is now a serious concern. Weaknesses and vulnerabilities in IoT embedded devices can compromise infrastructures, threaten producer and consumer privacy, and put physical safety at risk. High-profile incidents such as the Mirai botnet, the Stuxnet worm, and the Amnesia:33 vulnerabilities illustrate that embedded security breaches can cause worldwide disruptions. From these attacks, the urge to rethink security strategies for devices that were not originally designed to withstand persistent and well-funded adversaries arises.

1.2 Industrial Relevance

The importance of cybersecurity in the embedded world also has a deep impact in the economic and industrial sectors. Modern supply chains rely heavily on connected devices for logistics, monitoring, and automation. Vulnerabilities in a single component can propagate through entire ecosystems and apart from the disrupt of the service they provide they can now cause financial losses and reputational damage to the industry. For example, a compromised IoT sensor in industrial environments may lead to data leakage and also to production downtime and physical hazards.
Moreover, businesses rely on consumer trust to succeed in the market. Devices where robust protections is absent and are more easily broken risk being abandoned by users or banned by regulators. Security, therefore, is not only a technical requirement but also a competitive differentiator. Companies need to impose themselves on the market so it is important for them to demonstrate compliance with standards and implement transparent security practices while also contributing to safer digital infrastructures.

1.3 Regulatory Landscape

The growing dependence on internet and connected technologies has led governments and standardization organization to enforce stronger and stricter requirements. In Europe, the *Radio Equipment Directive (RED)* introduced delegated acts mandating cybersecurity and privacy protections for wireless devices while also considering the financial aspect of it. Most recently, the *EN 18031-1* standard has become mandatory, setting a baseline for cybersecurity certification of IoT devices. Similar efforts exist worldwide, such as the *NIST IoT Cybersecurity Standards* in the United States and the *ETSI EN 303 645* standard for consumer devices.
These frameworks share a common principle: security must be embedded in the design process, not added as an afterthought. This is a pillar in all security systems. Manufacturers are now forced to comply with harmonized standards, implement secure development lifecycles, and prove the resilience of their devices against defined threat models just to have the ability to put their product on the market. While regulatory compliance is essential, it does not guarantee immunity to newer or unforeseen attack vectors. This gap is the reason research is important not only to interprets and applies standards but also to anticipates future threats and adapts defenses accordingly.

1.4 State of the Art in Embedded Security

The literature on embedded cybersecurity reveals recurring weaknesses. Authentication mechanisms are often simplistic. Passwords does not usually follow best practice standards and sometimes are hardcoded. In addition, consumer usually leave default passwords facilitating attackers. Firmware integrity is frequently neglected, making devices susceptible to modification and injection of malware that consequently disrupt the service. Communication protocols, particularly lightweight ones designed for IoT, are commonly deployed without encryption or integrity, enabling eavesdropping and man-in-the-middle attacks. Hardware-level threats, including side-channel leakage and fault injection, present even more difficult challenges since are not easily discovered.
To counter these risks, researchers and practitioners have developed a wide range of defensive mechanisms. Secure boot processes establish a chain of trust from hardware to software, preventing unauthorized firmware and code execution. Encrypted secure storage and safe transport protocols protect sensitive data in transit and at rest. Role-based, attribute-based and the new context-aware access control models limit the impact of compromised credentials and help define access controls. Moreover, secure over-the-air (OTA) update infrastructures ensure that patches can be deployed at scale without exposing devices to downgrade or injection attacks while keeping the device functioning.
Despite this improvements and new standards, several open issues still exist. Embedded platforms are subject to the tight constraints derived from the hardware itself. It is difficult but critical to achieve a balance between strong protection and efficient operation. Security solutions must also account for lifecycle management, from manufacturing to end-of-life. Finally, new challenges are emerging thanks to artificial intelligence, that created new opportunities for enhanced defense but can also serve as a new attack vector.

1.4.1 Relevant Case Studies

Well-documented incidents, occurred during the last couple of years, demonstrate the fragility of embedded systems:

- **Stuxnet** revealed how malicious firmware manipulation can sabotage industrial systems, proving that cyberattacks can have tangible physical consequences. Moreover, it showed the important of training personnel.

- **Mirai** exploited weak authentication in IoT devices, creating one of the largest botnets ever witnessed. This DDoS ended up disrupting internet services in a vast part of the world.

- **Amnesia:33** showed how vulnerabilities in TCP/IP stacks affect a vast array of embedded platforms, showing that even common libraries can be entry points for attacks.

These cases exemplify the persistent tension between functionality, cost, and security.

1.5 Research Gap

Although numerous frameworks and best practices exist, there remains a gap between theoretical recommendations and their practical application in custom embedded platforms. Companies and industry usually have to make decision on how much security they want to implement based on the functionality of each product. Since adding security can be a costly operation, performance and efficiency often take priority.
Much of the existing research focuses either on high-level standards or on specific technologies in isolation. Less attention is given to holistic integration, where multiple defensive layers are combined, validated, and measured against real threats. Finally, the emergence of AI-driven attacks poses new challenges that are not yet addressed in current industrial practices since the use of it is still related to social attacks rather than physical. This thesis seeks to bridge that gap by providing a systematic, experimental, and standard-aligned approach.

1.6 Objectives of the Thesis

This thesis is driven by the need to strengthen the resilience of custom embedded devices against evolving threats while ensuring compliance with modern regulatory standards. Its specific objectives are:

- Evaluate existing defensive mechanisms, including secure boot, authentication, encryption, and update protocols, highlighting their strengths and limitations, and group as many best practices as possible to provide a comprehensive overview of the best practices derived from existing standards.

- Analyze the threat landscape relevant to embedded and IoT devices, identifying both traditional and possibly new attack scenarios.

- Conduct a vulnerability assessment and a penetration testing activity on a custom embedded device with a security solution already implemented. The goal was to expose real weaknesses and quantify the risk they pose.

- Design and implement security features that address the identified gaps and eventual vulnerabilities discovered in the test stage, and ensure the device's compliance with the EN 18031-1 directive.
- Explore the role of artificial intelligence as both a catalyst for next generation defenses and as a potential tool for adversaries.

1.7 Methodology

The research methodology adopted combines theory and practice:

1. **Theoretical analysis** A review of existing standards, best practices, and scientific literature on embedded cybersecurity, with a focus on the Yocto Project, as it is the framework used in the target embedded device and the most common in the embedded world.
2. **Experimental evaluation** Deployment of a custom embedded device as a test platform, with initial configuration, security feature integration, and systematic vulnerability assessment.
3. **Penetration testing** Execution of targeted attack scenarios in a controlled environment, focusing on network services, authentication, boot processes, and cryptographic modules.
4. **Validation and compliance checks** Mapping the implemented security measures against the requirements of EN 18031-1, identifying areas of non-compliance and providing solutions if any is find.
5. **Forward-looking exploration** Investigating the impact of artificial intelligence in both defending and attacking embedded platforms.

1.8 Structure of the Thesis

The thesis is structured into eleven chapters. Chapter 2 presents industry standards and best practices, while Chapter 3 outlines the main threats and vulnerabilities. Chapter 4 introduces the Yocto Project, its relevance to embedded security, and its selection for this work, as it is the framework used in the target embedded device. Chapters 5 and 6 detail the vulnerability assessment and penetration testing activities, respectively. Chapter 7 presents the test results and mitigation strategies. Chapter 8 discusses the customization of SELinux policies, and Chapter 9 addresses compliance with the EN 18031 directive. Chapter 10 explores the opportunities and risks introduced by artificial intelligence. Finally, Chapter 11 concludes the work by summarizing findings, contributions, and future directions.

Chapter 2

Best Practices in Embedded Cybersecurity

2.1 Security Guidelines and Industry Standards

As the number of connected devices continues to grow, the cybersecurity of embedded systems becomes increasingly critical. Systems and appliances that were once isolated are now online, and therefore must be designed from the very beginning with protection against cyber threats in mind.
With threats becoming increasingly sophisticated and widespread, it is critical for developers and organizations to rely on established security practices and recognized industry standards to build robust and secure systems.
These frameworks provide structured recommendations, regulatory compliance checkpoints, and technical best practices that help mitigate vulnerabilities and strengthen system resilience. They also serve as a common reference for evaluating the security status of embedded platforms across different industries.

2.1.1 OWASP Top 10 for IoT

«The OWASP Internet of Things Project is designed to help manufacturers, developers, and consumers better understand the security issues associated with the Internet of Things, and to enable users in any context to make better security decisions when building, deploying, or assessing IoT technologies.» [1]
The manifest was created back in 2018 but it is valid to this days.

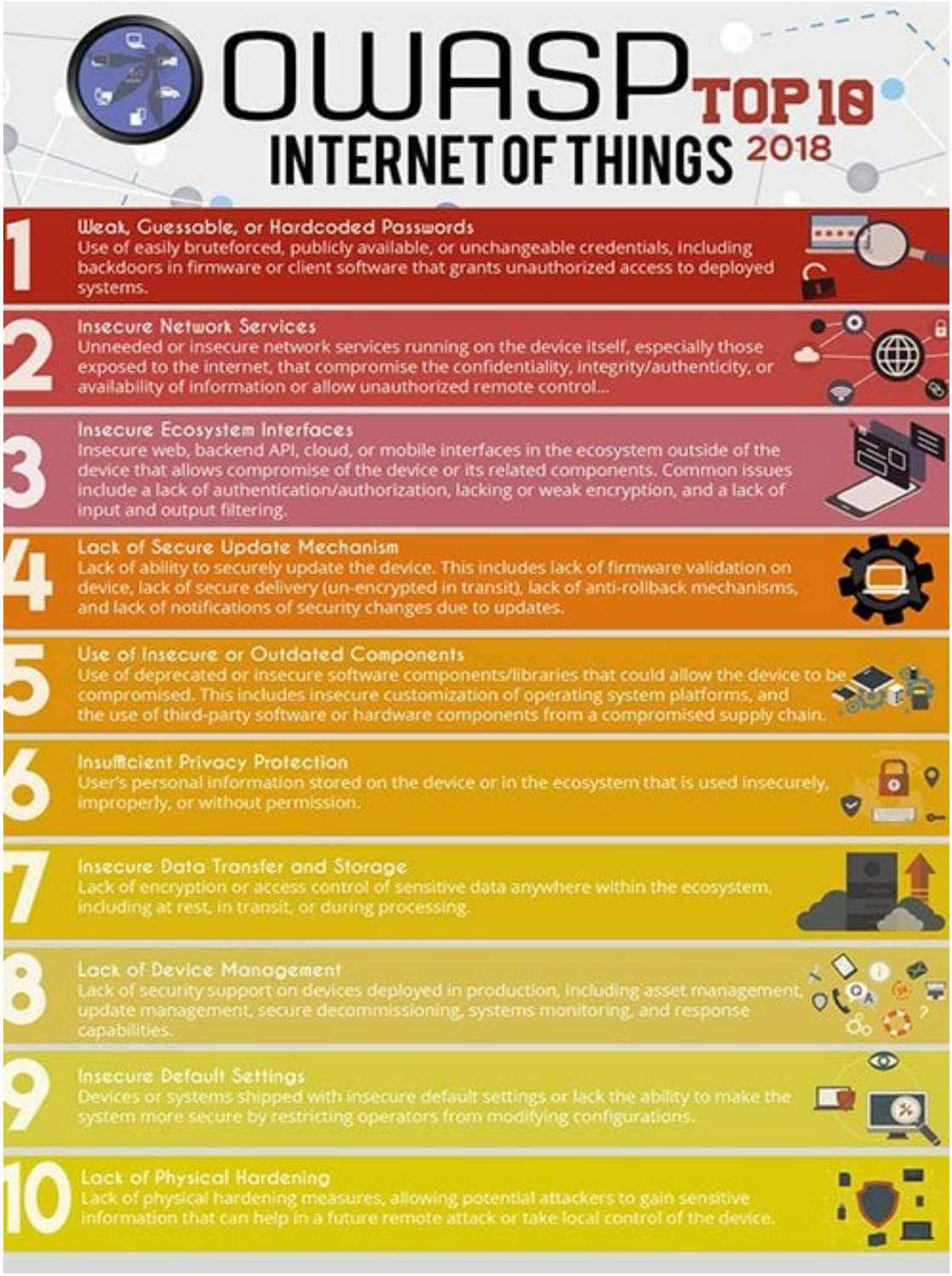

Figure 2.1: OWASP IoT Top 10

2.1.2 ISA/IEC 62443

The International Society of Automation (ISA) and the International Electrotechnical Commission (IEC) have joined forces to improve the cybersecurity of Industrial Automation and Control Systems (IACS).
An IACS is defined as a collection of personnel, hardware, software, and policies involved in the operation of the industrial process. These systems directly affect the safety, security, and reliability of the operation. Since IACS are physical-cyber systems, cyberattacks can have serious consequences that include, but are not limited to:

- Endangerment of public or employee safety or health.
- Damage to the environment.
- Loss of product integrity.
- Loss of proprietary or confidential information.
- Financial loss

They developed and published the ISA/IEC 62443 series. These documents describe a systematic engineering-based approach to addressing the cybersecurity of IACS. They are arranged in four groups, each targeting a specific focus and audience. [2]

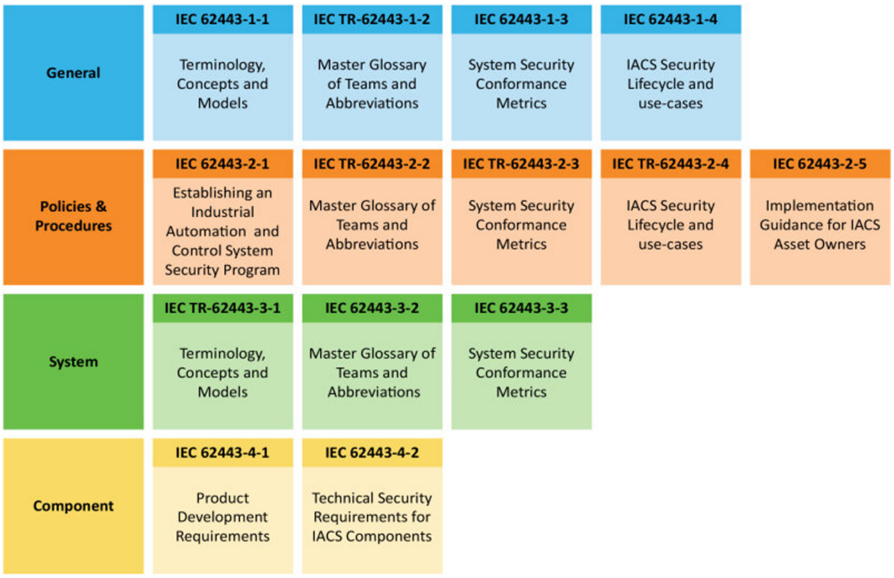

Figure 2.2: ISA/IEC 62443 four groups

2.1.3 ETSI EN 303 645

The document brings together recognized good practices in security for Internet-connected consumer devices, expressed as high-level outcome-focused provisions. Its objective is to support parties involved in the development and manufacturing of consumer IoT products by providing guidance on securing their products. The provisions are primarily outcome-focused, rather than prescriptive, giving organizations and companies the flexibility to innovate and implement security solutions appropriate for their products. The focus is on the technical controls and organizational policies most relevant to addressing common and significant security shortcomings. Overall, the document defines a baseline level of security intended to protect against basic attacks on fundamental design weaknesses. [3]

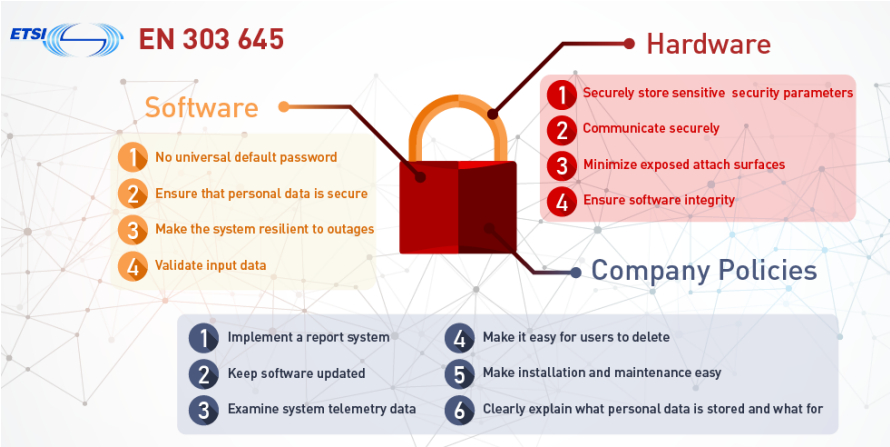

Figure 2.3: ETSI EN 303 645

2.1.4 IoTSF

The IoT Security Foundation's Best Practice Guides provide concise essential advice on measures to help secure IoT products and systems. They emphasize basic principles of good practice, while acknowledging that many devices cannot meet every requirement due to real-world constraints.In such cases, designers and developers must consider the trade-off between constraints and risks, and document the implications for where and how the device may be used if a downgrade in security happens. While primarily intended for IoT designers and developers, the Guidelines also offer useful information for consumers and aim to empower other parties in the supply chain to ask the right questions. [4]

2.1.5 NIST IoT Security Standards

NIST has developed a suite of documents and programs tailored to enhance IoT security. [5, 6]

- **NIST SP 800-213** helps to identify and define cybersecurity requirements for IoT devices as integral components within larger systems.
- **NIST SP 800-213A** offers a comprehensive list of both technical and non-technical security capabilities, referring to federal controls and the Cybersecurity Framework.
- The **Cybersecurity for IoT Program** promotes the development of standards and tools through multi-stakeholder collaboration.
- **NIST IR 8425 and 8425A**) provides a baseline for security outcomes specific to consumer IoT devices and routers, with a specialization in consumer focused profiles.

2.1.6 Integrating Standards in Embedded Development

Integrating established security standards into the device development lifecycle is essential to translate theoretical guidelines into effective protections. This involves embedding security checkpoints at each phase: initial design and architecture, coding, testing, deployment, continuous maintenance and updates. By doing so, developers ensure that security considerations are not an afterthought but a priority. Standards such as those described previously provide detailed requirements and best practices that can be mapped onto development activities, such as threat modeling, secure coding, and vulnerability assessments.
Furthermore, integrating standards promotes better collaboration among multi-disciplinary teams, aligning hardware, software, and security experts toward a common goal. This approach results in more resilient embedded systems capable of coping with evolving cyber threats and meeting regulatory demands.

2.2 Secure Access

A secure embedded system must ensure that only authorized users and processes can access the device's resources and functionalities. This requires a multi-layered approach that combines physical safeguards, logical controls, and cryptographic protections to prevent unauthorized access attempts and, if they occur, to respond effectively to incidents.
In the context of custom embedded devices, the following measures must be implemented while considering hardware constraints such as limited memory, processing power, and battery capacity. However, security should never be sacrificed for performance, especially for functions that protect critical system operations or sensitive data.

2.2.1 Strong Authentication

Ensuring that only legitimate users can access the device starts with robust authentication mechanisms.

- **No weak or default passwords:** Many consumer IoT devices are sold with universal default usernames and passwords, which are often weak (e.g. admin). There are several ways to avoid this issue. One option is to generate unique, pre-installed passwords for each device using a secure random generator, ensuring that the mechanism used reduces the risk of automated attacks. Another approach is to require users, on first login, to choose a password that follows cryptographic best practices, such as a minimum length, a mix of character types, and resistance to dictionary and brute-force attacks. Strong authentication can also be achieved without using passwords by relying on alternative authentication mechanisms such as digital certificates or cryptographic tokens, which are particularly effective for automated and machine-to-machine authentication.

- **Multi-factor authentication (MFA):** Users, usually, are authenticated based on one of three factors: something you know (knowledge), something you have (possession), and something you are (inherence). MFA strengthens security by requiring users to present at least two distinct types of evidence to prove their identity.

- **Rate limiting and account lockouts:** To guarantee strong authentication, it is important to be able to mitigate brute-force attacks by enforcing account lockouts after repeated failed login attempts and by limiting the number of attempts within a specific time window.

- **Salted and hashed passwords:** Passwords should never be stored in plain text, as this would give attackers direct access in the event of a breach. Instead, systems should store the salted hash of the password. Both the hash and the salt must be stored in secure memory, preferably leveraging hardware-based secure storage.

[3, 4]

2.2.2 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a model that authorizes user access to systems, applications, and data based on a user's predefined role. It is widely regarded as a best practice for enterprise systems because it enables organizations to adopt a granular approach to Identity and Access Management (IAM) while streamlining authorization processes and access control policies.
RBAC policies address key cybersecurity vulnerabilities by enforcing the principle of least privilege (PoLP). Through PoLP, user roles grant access to the minimum level of permissions required to complete a task or fulfill a job.
By limiting access to sensitive data, RBAC reduces the risk of accidental data loss and intentional data breaches. In particular, it helps mitigate lateral movement, a common cyberattack technique in which adversaries exploit initial access to expand their reach across a system. Because RBAC restricts what each account can access, it limits the potential damage caused by a compromised user credential.
The model also offers operational benefits, such as the ability to quickly add access permissions for contractors, vendors and other third-party users. Moreover, it strengthens organizational resilience by separating administrative functions from operational ones, thus reducing the impact of compromised accounts.
To remain effective, RBAC must be accompanied by strong governance practices. These include maintaining a clear mapping between roles and responsibilities, including specialized roles for maintenance and debugging. In addition, organizations should conduct regular reviews of user accounts to ensure that unused or obsolete credentials are promptly revoked. [7]

Attribute-Based Access Control with context-aware

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. Its attribute-driven approach provides both rich user context and fine-grained access control. Since ABAC is attribute-based, it supports dynamic authorization, granting or revoking access to data or applications in real time. ABAC is useful in environments that are scaling and in situations where identity or resource policy management has become complex.
Building on this foundation, context-aware ABAC extends the model by explicitly

integrating situational and dynamic parameters into authorization decisions. For instance, access to a system may be granted only if certain contextual conditions are met, such as the request originating from a trusted device, occurring during defined working hours, coming from an approved location, aligning with expected user behavioral patterns, or taking place within a secure network segment. By continuously evaluating contextual information ABAC enables organizations to implement adaptive and risk-informed access control policies, which are particularly well aligned with modern Zero Trust architectures. [8, 9]

Aspect	**Advantage of ABAC over RBAC**
Scalability	Permissions scale with innovation; administrators do not need to manually update existing policies for new resources.
Policy management	Requires fewer policies, since access can be defined by attributes rather than creating different policies for each job function. Those policies are easier to manage.
Adaptability	Teams can dynamically respond to change and growth; permissions for new resources are automatically granted.
Granularity	Enables fine-grained policies that enforce least privilege, e.g., actions are allowed only if resource tags match user roles.
Integration	Leverages attributes from corporate directories (SAML/OIDC session tags) to allow or deny permissions dynamically.

Table 2.1: Comparison of ABAC advantages over RBAC

While no document states verbatim that “context-aware ABAC is the most modern and recommended model" these advantages are recognized in several authoritative sources, including NIST SP 800-162, NCCoE Practice Guide 1800-3, and the FICAM Roadmap (2011), which highlight ABAC as a flexible and scalable model well suited to modern enterprise environments.

2.2.3 Secure remote access

With the increasing proliferation of internet-connected devices, employees and other verified users frequently access organizational resources from diverse locations and devices, making traditional access controls insufficient. Secure remote access encompasses a set of security strategies designed to prevent unauthorized access to networks, systems, and sensitive data when connecting from a different network or

location at any time. Key best practices include:

- **Access Controls:** Apply IP whitelisting for critical administrative access, and implement network segmentation to isolate sensitive data and limit potential breaches.

- **Mutual Authentication:** Ensure both endpoints verify each other before exchanging data, especially when connecting devices to cloud services or control systems.

- **Single Sign-On (SSO):** Allow authenticated users to access selected applications with a single set of login credentials, reducing password overload while maintaining security.

- **Encryption:** Use SSH for shell access, TLS for web interfaces, APIs for data exchange, and VPNs for remote access. Disable insecure services such as Telnet, unencrypted FTP, HTTP, and outdated SSL/TLS versions. Strong encryption ensures that data transmitted during remote sessions remain confidential and protected from interception attempt.

- **Endpoint Security:** Verify antivirus and firewall software are in place, systems are patched, keyloggers or other malicious processes are not running, and sensitive data is not left behind in caches.

- **Monitoring and Auditing:** Continuously monitor remote access logs for anomalies. Conducting periodic security audits and keeping software and systems up-to-date helps identify vulnerabilities and address them quickly, minimizing the risk of exploitation.

- **Employee Training:** Educate users about security best practices, including recognizing phishing attempts, using strong passwords, and following secure access procedures.

Collectively, these measures form a layered defense strategy that mitigates the risks of remote access while maintaining operational flexibility in modern, distributed environments. [10, 11]

2.2.4 Protection of debug interfaces

Debug ports such as JTAG or UART expose low-level system functions and memory, making them attractive targets for attackers. The following best practices are recommended:

- **Disable Port in Production:** Debug interfaces are convenient during device development and debugging, but it is good practice to disable them in the release version of the device/firmware. If not required after development and testing or for maintenance and serving, debug ports should be permanently disabled in production devices.
- **Password Protection and Locking:** When the debug interfaces, like JTAG, must remain available, access should be protected with strong authentication. Many modern micro-controllers allow JTAG to be locked down and only be accessible with proper credentials.
- **Fuse Settings:** Many embedded processors include fuse bits that can be "blown" to permanently disable low-level debug access. This is a common practice in production environments to prevent misuse and exploitation once devices are deployed. Additionally, there are configurable bits in the micro-controller's non-volatile memory to disable certain debug interface functions.
- **Tamper Detection:** Employ mechanisms that disable the debug interface or erase sensitive data when unauthorized access attempts are detected. In high-security environments, tamper events can be configured to trigger defensive actions such as wiping cryptographic keys or erasing sensitive data.
- **Secure Debugging Interfaces:** Use secure debug protocols that encrypt communications and authenticate users, ensuring that even physical access does not turn into exploitation.
- **Physical Security:** Restrict physical access to debug pins. In many cases, the simplest form of security is to ensure that the interface is not exposed or is located in a secure environment. It is also possible to use epoxy or conformal coatings to cover the JTAG pins, making physical access more difficult, remove entirely the connectors from the board or scattering debug pins as hidden test points.
- **Code Obfuscation and Encryption:** If an attacker gains access to the debug interfaces it is useful to increase the difficulty of reverse engineering and firmware tampering by encrypting sensitive code and data and applying obfuscation techniques.
- **Regular Security Audits:** Continuously audit systems to ensure that the interfaces remain secure by identifying new vulnerabilities and maintaining strong defenses over time.
- **Secure boot and code signing:** Implementing secure boot and code signing ensures that only trusted firmware can run, avoiding malicious code injection through these interfaces.

Securing the debug interfaces consequently requires a multi-layered strategy that combines physical safeguards, software controls, and secure design practices. [12, 13]

2.2.5 Key and certificate management

Proper management of cryptographic keys and digital certificates is essential to ensure that only authorized entities can authenticate and exchange data. Keys should always be generated in a secure environment, relying on hardware-backed random number generators to prevent predictable outputs. Once created, they must never appear in plain-text within software, as this would expose them to extraction through reverse engineering or memory analysis.
For long-term protection, sensitive material should be stored in tamper-resistant components such as Hardware Security Modules (HSMs) or Secure Elements, which provide safeguards against both physical and logical attacks. Secure access policies must also account for the full life-cycle of cryptographic material. Regular key rotation and certificate renewal reduce the risk associated with long-term use. Moreover, any compromised or expired credentials need to be promptly revoked through mechanisms such as Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) validation.
Although this topic naturally arises in the context of secure remote access, its relevance extends across multiple areas of embedded cybersecurity, including secure boot, firmware updates, and device identity. A detailed analysis of keys and certificates management best practices will be presented in a dedicated chapter.

2.3 Secure Boot

Secure Boot is a security technology that verifies the legitimacy of software before it executes during startup. This helps prevent malware, any software designed to disrupt, damage, or gain unauthorized access to a computer, from infecting the system at the critical stage prior to the operating system loading.
Security is based upon the concept of trust: everything in the system ultimately agrees that some root point in the system can be trusted, providing a secure foundation upon which the rest of the system can be securely built. The higher the risks and the potential losses, the stronger the assurance required that the root of trust is truly reliable.
The same approach need to be followed when securing IoT devices. To have a device function securely the boot process must be validated in the first place, and without that assurance up front anything else that operates on the device cannot be entirely trusted.
For instance, an attacker may replace an existing executable file with one containing malware. On a device without Secure Boot, the system could restart and execute the malicious code, resulting in a full compromise. From that point on, no operation performed by the device could be considered trustworthy. In contrast, with Secure Boot enabled, integrity checks during startup would detect the altered file and trigger protective countermeasures. [14, 4]

2.3.1 Chain of Trust

A Secure Boot process relies on a chain of trust, where each stage of the boot sequence is verified before execution. The process begins from a hardware-based root of trust and extends to the operating system or application firmware. The main stages are:

1. **Root of Trust (ROM-based secure boot)**: Implemented in immutable storage (ROM, OTP, or eFuse), it holds reference keys or hashes and verifies the authenticity of the first-stage bootloader.

2. **First-stage Bootloader (FSBL)**: Verified by the Root of Trust, it performs minimal initialization and validates the second-stage bootloader.

3. **Second-stage Bootloader (SSBL)**: Handles early device initialization, basic logging, and firmware update checks. It verifies the operating system or the application firmware before execution.

4. **Operating System / Application Firmware**: Runs only after the chain has been successfully validated, ensuring the device operates with trusted code.

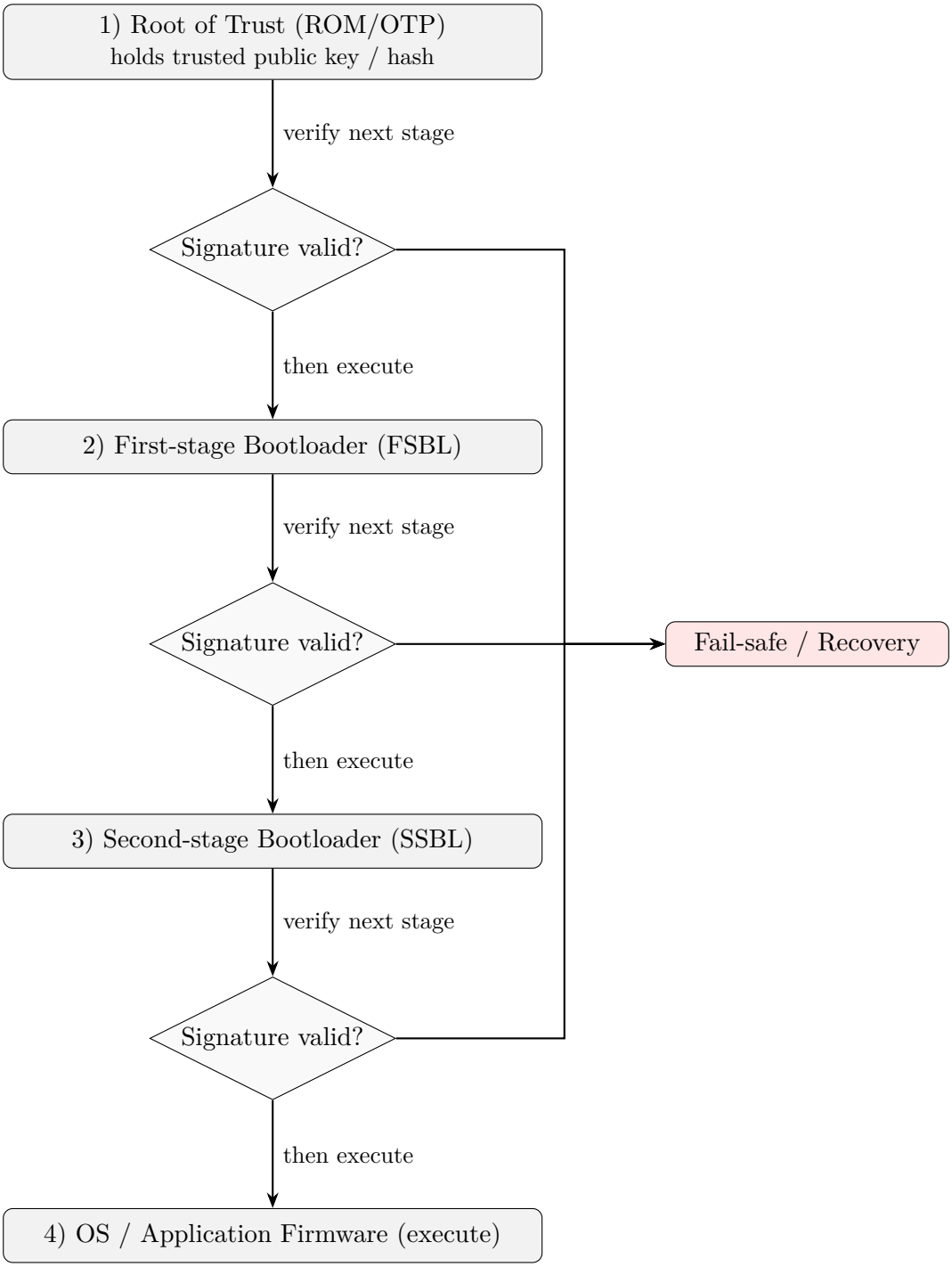

Figure 2.4: Secure Boot chain of trust: ROM-based Root of Trust → FSBL → SSBL → OS/Application, with per-stage signature verification and fail-safe on error.

At each verification stage, if a check fails, the device must transition into a secure failure state rather than executing unverified code. This prevents unauthorized access and may involve recovery procedures, such as reverting to a known good firmware image or disabling functionality until a trusted update is available.
In addition to this sequential flow, several best practices strengthen the overall security of the boot process:

- **Enable basic logging:** Record startup parameters and potential anomalies to support diagnostics and forensic analysis.
- **Firmware update:** If a firmware update is present, confirm its authenticity, apply it securely, and restart the system to ensure the device operates only with trusted code.
- **Verify external services:** Ensure that supporting services such as power, DNS, -or NTP are available and legitimate, preventing potential misconfigurations or attacks.
- **Hardware-backed protection:** Use secure hardware such as Secure Access Modules (SAM) or Trusted Platform Modules (TPM) to store critical data and perform cryptographic operations required during boot. Secure hardware has limited secure storage so it must only hold the read-only first stage of the bootloader and all data required to verify the authenticity of firmware.
- **Stage-by-stage validation:** Each piece of boot code must be validated immediately before execution to mitigate Time-of-Check to Time-of-Use (TOCTOU) attacks. Digital signatures are used to confirm that the code is genuine and has not been tampered with.
- **Hardware integrity assurance:** At every stage, verify that only expected hardware components are present and match the expected configuration.
- **Sequential enforcement:** Do not allow later boot stages to execute until the previous ones have been successfully verified.
- **Graceful failure:** If any stage fails, the system must fall back to a secure state to prevent unauthorized access to underlying systems, code or data. Any code run must have been previously authenticated.

2.3.2 How it works

Verifying a bootloader executable file is usually done using public/private keys. The manufacturer securely stores their private key and never reveals it. When a device is manufactured, the public key associated with that private key is placed

inside a secure storage on the device.
The bootloader code is developed using a secure development process, and then a cryptographic hash of it is digitally signed with the manufacturer's private key. When a bootloader firmware is needed to be installed on the device, the signature is checked against the public key on the device to confirm that it is a genuine hash value from that manufacturer. The firmware code is then hashed again and compared with the signed hash. If the signature matches, the firmware code has not been altered and the new firmware can be installed. Whenever the device boots, the installed bootloader is again verified before being allowed to run.
The bootloader code knows which sub-systems is expected on the device so it can checks that they exist and are functional. Another check is to make sure that there are no unexpected subsystems present. If all is correctly implemented basic logging can start, recording various initial parameters and any issues found.
At some point an update should be installed so the boot process will verify the new software as described before.
If everything is working correctly up to this point, functional applications can be verified in the same way and then run on the device.
One way to secure a boot process is to burn the boot code into a write-once chipset on the manufacturing line. This will ensure the code cannot be tampered with after the device is built. It also means any future changes cannot be made. Aside from software bugs, if new security vulnerabilities are discovered they cannot be patched on hardware and the device start to become an increasing risk in the network.
Alternatively, keys (or other data) for verifying each boot stage can be locked into the chipset. Even in this case, if the key data is ever compromised then all devices with that key suddenly become 'at risk', unless there is also a way to securely update the key.
One other option could be to use external services to provide the verification, but this offers limited possibilities as it requires access to trusted external services at an early boot stage. [4]

2.3.3 Secure Boot in edge devices

Secure Boot is a vital security feature that protects edge devices from malware and unauthorized software during the boot process. Secure Boot is important for edge devices because they're frequently deployed in unsecured locations. Edge devices are particularly vulnerable to physical tampering and unauthorized access. Additionally, the sensitive data they handle, such as customer information and financial data, makes them prime targets for cyberattacks.
The process needed to enable Secure Boot varies depending on the device manufacturer and operating system. It typically involves accessing the UEFI (BIOS) settings during startup. However, customizing Secure Boot requires generating

custom certificates and enrolling them in the UEFI firmware. This can be a complex process, but can be simplified by creating a custom UEFI with your certificates already enrolled. Edge devices can further benefit from a customized Secure Boot deployment, which can provide:

- **Tailored security:** Custom Secure Boot policies allow to control exactly which software is permitted to run the your device, ensuring only trusted applications have access.
  By limiting the execution of unauthorized software, the risk of malware infections and other security breaches is reduced, especially when devices are on the move or installed in publicly accessible locations.
- **Improved control over device behavior:** Customization can be used to restrict, deny or allow certain features, such as booting from specific storage devices, like USB, or running specific operating systems.
- **Compliance with industry standards:** For certain industries, such as healthcare or finance, customizing Secure Boot may be necessary to comply with specific regulations and security standards.

[14]

2.4 Secure Update Mechanisms

Keeping software updated is one of the most critical security measures in IoT, since unpatched vulnerabilities can quickly become entry points for attackers. For this reason, secure update mechanisms must be designed to ensure robustness, reliability, and user trust. Managing software updates successfully usually relies on communication of version information for software components between the device and the manufacturer.
All software components in consumer IoT devices should be securely updateable, with exceptions justified such as first-stage bootloaders that needs to be immutable as previously stated.

Robust Update Devices must implement mechanisms that allow updates to be installed securely and reliably. This means that there are adequate measures to prevent an attacker misusing the update mechanism. Measures can include the use of authentic software update servers, integrity protected communications channels, verifying the authenticity and integrity of software packages.
To prevent downgrade attacks, an anti-rollback policy based on version checks should be enforced. Recovery techniques such as watchdog timers, dual-bank flash memory, or recovery partitions ensure that the device can revert to a known good version or factory state if an update fails.
Update mechanisms can range from the device downloading the update directly from a remote server, transmitted from mobile application or transferred over a USB or other physical interface.

User-friendly Update mechanisms should be simple and practical for end users. Users should be able to enable, disable, or postpone installation of security updates, depending on the device's intended use. If an update is difficult to apply, the chance that a user will repeatedly defer updating the device, leaving it in a vulnerable state increase.

Timeliness Security updates must be delivered promptly, with urgency aligned to the severity of the addressed vulnerability. Manufacturers should establish a clear management and deployment plan, while also maintaining transparency toward consumers regarding the level of update support.
Update deployment often depends on multiple dependencies on other organizations so it can be useful for the manufacturer to consider the entire software supply chain in the development and deployment of security updates. It is generally advisable to separate security updates from larger feature updates, to avoid unnecessary delays in the delivery of critical fixes.

For some devices, particularly those that rely on associated services, randomized update checks performed by the service may be more appropriate than device-initiated checks.

Cryptography best practice The authenticity and integrity of each software update must be verified before installation. Constrained devices, like IoT devices, can have power limitations that make performing cryptographic operations costly. In such cases, verification can be delegated to a trusted peer device, which then transmits the verified update over a secure channel.

Fail-safe mechanism A device should be able to detect invalid or malicious updates and respond appropriately. Beyond rejecting the update, it may also notify the user, report the incident to a monitoring service, or log the event for later review. In addition, mitigating controls can be placed to prevent an attacker from bypassing or misusing an update mechanism. Restricting the amount of data exposed to an attacker during the update process reduces the potential for exploitation.
The manufacturer should clearly inform users when a security update is required, providing information on the risks it mitigates. Furthermore, devices should notify users if applying an update will temporarily disrupt functionality. Particularly, devices that fulfill a safety-relevant function are expected not to turn completely off in the case of an update as this could result in hazardous situations. [3]

Lifecycle management Manufacturers and system operators should manage devices across their full lifecycle, including:

- Maintaining an inventory of deployed devices.
- Tracking software versions and compatibility.
- Planning for routine and emergency updates.
- Handling ownership changes securely.
- Securely managing devices at end of life.

2.4.1 OTA updates

Automatic mechanisms are essential for delivering software updates at scale. An Over-the-Air (OTA) update refers to the secure deployment of software or firmware to a connected device over a network. Depending on the update target, the term may appear as SOTA (Software Over-The-Air), FOTA (Firmware Over-The-Air),

or simply software update.
Once the need for an OTA update process for a new connected product is established, the update infrastructure must be designed to meet four key criteria: robustness, cybersecurity, operational, and ecosystem integration [3, 4, 5]. The OTA update process relies on secure communication. Ensuring end-to-end security throughout the infrastructure, the update procedure, and its management is critical for maintaining cybersecurity, safety, and regulatory compliance.
The OTA update infrastructure must adhere to *secure-by-design* principles [5]. Encryption and strong cryptography are baseline requirements across all stages of the process. To mitigate "man in the middle" cyberattacks, the device running the OTA client should authenticate and secure the connection using industry-standard protocols such as secure TLS to get the update, modify accordingly the inventory data, and deliver status information. The client must verify the server's identity by using a Certificate Authority from the device OS or by using pre-distributed keys, like self-signed certificates.Moreover, devices should expose no unnecessary open ports, as these significantly increase the attack surface [4].
The device must further ensure that the update package (artifact) originates from a trusted source, typically verified using the public part of a key pair employed to sign the update.
An OTA update infrastructure plays a pivotal role in managing and mitigating software vulnerabilities by enabling remote patching across thousands or even millions of devices. Leveraging a secure end-to-end infrastructure with robustness built in, manufacturers can deploy security patches and bug fixes across device fleets, with control and granularity [3]. This approach eliminates the logistical and financial burden of manual or physical updates, which are neither practical nor scalable in IoT ecosystems. By automating patch deployment, manufacturers can ensure that their entire fleet remains secure without the costly burden of on-site interventions [6] [15].

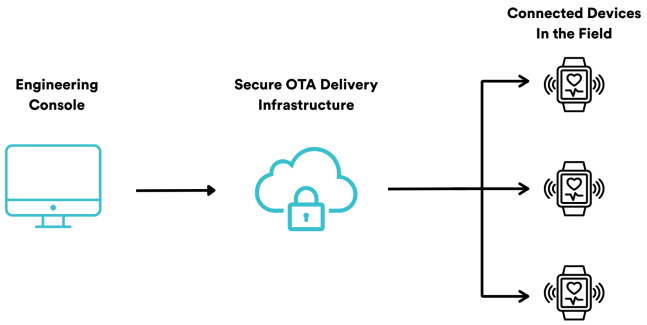

Figure 2.5: OTA updates

Operational complexity

OTA infrastructures also provide a foundation for advanced operational practices, such as predictive maintenance and real-time monitoring. By integrating OTA updates into established CI/CD pipelines, manufacturers can accelerate feature delivery, address vulnerabilities more rapidly, and continuously improve deployed products [4].
Effective OTA solutions must also include fleet management capabilities, allowing operators to control and deploy updates across heterogeneous environments and geographical regions. Automation reduces administrative overhead and minimizes deployment errors, while features such as dynamic grouping ensure that devices sharing specific attributes automatically receive the appropriate updates. A mTLS proxy container is also a very important capability for manufacturers who, during mass production, want to sign client certificates during production, so that devices automatically get accepted into the OTA update infrastructure server when activated by the end customer, sometimes several months after manufacturing. [15]

Connectivity Constraints and Update Optimization

An OTA update mechanism must be aware connectivity can be not sufficient or failing entirely, and with it the quality of service for devices. For example, devices should only attempt to contact update servers once a stable connection is detected and should be capable of resuming interrupted transfers.
The OTA update infrastructure should also support incrementally updates of the software image by transfering only the differences between the previous image and the new one. By doing so, the amount of data required is minimal and the devices remain up to date [15].

2.5 Data protection

In the era of Industry 4.0, the integration of IoT devices in production and business contexts represents a strategic evolution to increase efficiency, control, and competitiveness.
IoT devices continuously collect a multitude of data points in real or near real time. The data collected can be transmitted and analyzed and include fundamental information about the functioning of machines, logistics and production cycles. This information is also used to initiate different operations, autonomously or manually, and to analyze both the surrounding environment and the usage of the devices themselves. IoT sensors can collect various types of data as shown in the table below. All these data can be migrated into a data warehouse where they are transformed and used to advanced business intelligence and provide companies

Category	Data Points	Generic Description
Position Data	Position, navigation, and speed; proximity, presence; position, orientation, angles	Collect information regarding the location, movement, or spatial context of people, objects, or devices.
Status Data	Temperature; lights, images, optical data (when used for monitoring)	Collect basic information regarding the current state of the device and its condition, such as activation/deactivation status, availability/non-availability, or environmental values.
Automation Data	Usage, time, and duration; sounds and vibrations; lights, images, optical data (when used for automation)	Provide input to trigger or optimize automated operations, supporting decision-making processes and system control.

Table 2.2: Classification of IoT Data Types

with a wealth of information which, adequately collected, stored, and processed, give an unprecedented in-depth view of user behavior.
Consumer IoT devices must ensure the secure handling of personal data. The confidentiality of data transmitted between the device and associated services, often cloud-based, need to be protected using cryptographic methods suitable for the sensitivity of the data, with additional measures for sensitive personal data whose disclosure could result in significant harm. Protection typically includes mechanisms for verifying data integrity.
All external sensing capabilities of the device should be documented and made accessible to users in a clear and transparent manner. Data input via user interfaces, APIs, or network communications has to be validated to prevent exploitation through malformed or malicious data.
Manufacturers are expected to provide users with clear information regarding the personal data collected, its use, and the parties involved, including third-party services. When processing relies on user consent, the consent must be obtained through an explicit opt-in mechanism, and users must have the ability to withdraw consent at any time. Telemetry data collected should be limited to what is strictly necessary for the intended functionality, and users must be informed about its

collection and use [3, 16, 17, 18].

2.5.1 Data in Transit

IoT devices frequently communicate through lightweight protocols like MQTT, CoAP or HTTP. Usually, these protocols are employed in public network without any little to none security level making them particularly vulnerable to threats and attacks such as sensitive data theft and traffic manipulation along the communication channel.
It is important for manufacturers to adopt strategies to ensure that their IoT devices follow best practices for data security. The following measures are recommended:

- **Secure lightweight protocols:** The protocols mentioned before now offer "secure by design" versions, such ah MQTT-over-TLS, HTTPS and CoAPS (CoAP-over-DTLS). These protocol ensure confidentiality and integrity of data in transit without adding computational and network overhead. This is particularly important for embedded devices, which often operate under strict resource constraints yet require efficient and secure communication.
- **End-to-end encryption:** Data transmitted by IoT devices should be protected throughout the entire communication path, from the source device to the destination. The use of TLS and DTLS with efficient cryptographic algorithms, such as Elliptic Curve Cryptography (ECC), can provide high security while reducing processing time and power consumption.
- **Strong Authentication:** Encryption alone is not sufficient to guarantee secure communication. Devices must be able to verify the identity of the endpoints they communicate with. Some type of strong authentication is needed, such as X.509 certificates or pre-shared keys (PSK) in resource-constrained environments. This can help mitigate spoofing or impersonation.
- **Integrity:** It is essential that transmitted data remain unaltered. Mechanism such as Message Authentication Code or Lightweight digital signature (e.g., Ed25519) can be employed to verify that the received data matches the original, thus protecting against man-in-the-middle attacks and any type of packet modification and injection.

2.5.2 Data at Rest

Many IoT devices store data locally, including system logs, configuration files, credentials, telemetry data, and other sensitive information. This introduces significant risks: if the device is physically compromised or its internal logic is altered, data can be stolen, manipulated, or used for further attacks. Therefore,

protecting data at rest is as crucial as protecting data in transit, and best practices must be followed.

- **Local data encryption:** The protection of memory (flash, EEPROM, or embedded databases) should be ensured using strong symmetric algorithms such as ChaCha20 or AES-256. The choice depends on the need to balance hardware resource availability with the required level of security. This ensures that even if an attacker gains physical access to the memory, the data remains unreadable without the correct decryption key.
- **Secure storage using dedicated hardware:** When possible, employing dedicated modules such as a TPM (Trusted Platform Module) or embedded HSM provides enhanced security. These components safeguard cryptographic keys in an isolated environment, preventing extraction even if the main device is compromised.
- **Credential protection:** The use of static or default credentials remains one of the most exploited vulnerabilities in IoT attacks. It is therefore essential to implement secure provisioning processes, including protected initial key distribution and periodic key rotation, to reduce the risk of long-term compromise.
- **Anti-tampering mechanisms:** To prevent physical attacks, many devices employ logical key destruction techniques.
- **Secure cloud backups:** Since IoT devices have limited storage, it is common to synchronize data with cloud infrastructure. However, backups must be transferred over encrypted connections and stored in secure environments, protected by encryption and fine-grained access policies, to prevent them from becoming an additional vulnerability.

[18, 17]

As a general rule, all sensitive data and cryptographic keys should be stored ideally in hardware-backed secure elements but if a device does not support such hardware, using a separate, protected partition with strict access controls is an acceptable alternative, ensuring that even in case of compromise, secrets remain isolated from application-level code.

2.5.3 Data Storage and Integration

The options of data storage today are more numerous than ever: companies can keep data on internal servers (on-premise), choose a cloud storage solution, or adopt a hybrid approach that combines the best characteristics of both. IoT requires

storage solutions with high capacity and adaptability, providing to companies the scalability necessary to manage budgets. Cloud integration has become essential as it provides access to multiple platforms (e.g., Google Cloud, Amazon Web Services, Microsoft Azure) and facilitates the extraction, transformation, and loading of data, as well as the migration processes necessary to transfer them across systems. In addition, cloud integration offers versatility, allowing data to be moved in a very short time between applications, accelerating the generation of business intelligence. A key advantage of this approach is its ability to combine and consolidate multiple data streams, which is particularly relevant given the continuous proliferation of IoT devices and the diversity of formats in which they collect and store information. The protection of this information cannot be underestimated, since an attack can compromise the entire operational continuity and irreparably damage the company's reputation. The degree of protection required against unauthorized viewing, changing or deletion of data depends on the sensitivity of that data. A "data classification scheme" defines a number of classes or levels of sensitivity for data and is key to its protection. Classifying data according to the scheme means that the right level of security can be identified and applied, based on the nature of the data being processed. Data classification will also help ensure compliance with legal regulations [4].

2.6 Physical device security

Physical security is often neglected when discussing security of embedded systems. Physical attacks such as tampering, theft, or hardware manipulation can compromise the security of the device and enable further cyberattacks. For this reason, it is recommended to implement tamper-evident mechanisms, restricted access policies, secure boot mechanisms, hardware-base encryption and environmental monitoring and logging. These measures are particularly important in public or unsecured environments where direct access to the device may be possible.

2.6.1 Restricting Physycal Access

Restricting unauthorized access is the first and most fundamental defense layer. In public or semi-public environments where device treating sensitive data are deployed, access control systems such as electronic badges, PIN pads, or biometric authentication ensure that only authorized personnel can reach the devices. Physical access authorizations apply to employees and visitors. Organizations determine the strength of authorization credentials necessary to be consistent with applicable laws, executive orders, directives, regulations, policies, standards, and guidelines. If an area is design to be public, physical access authorizations may not be necessary.

Business need to periodically review the access list detailing authorized facility access and eventually remove individuals from the list when access is no longer required.
Security controls used to perform physical access to system distribution and transmission lines include disconnected or locked spare jacks, locked wiring closets, protection of cabling by conduit or cable trays, and wiretapping sensors. Such controls may also be necessary to prevent eavesdropping, physical tampering or modification of unencrypted transmissions.
Detection and prevention activities can employ many types of anti-tamper technologies. Anti-tamper programs help to detect hardware alterations through counterfeiting and other supply chain-related risks. Tamper-evident seals and secure enclosures help prevent unauthorized opening of devices and manipulation of internal components. Organizations can implement tamper detection and prevention at selected hardware components deciding where to implement tamper detection and where tamper prevention. Moreover, secure boot and firmware signing are security measures that provide a degree of protection against tampering.
Intrusion alarms and motion sensors provide an additional layer of defense by detecting attempts of unauthorized physical access. [19] [20] They can be used to alert security personnel when unauthorized access to the facility is attempted. Alarm systems work in conjunction with physical barriers,and security guards by triggering a response when these other forms of security have been compromised or breached. Physical intrusion alarms can include different types of sensor devices, such as motion sensors, contact sensors, broken glass sensors and surveillance video cameras installed at strategic locations throughout the facility.

2.6.2 Hardening Against Physical Attacks

For many systems and components, particularly hardware, there are technical methods available to verify authenticity and detect alterations. These methods include techniques such as optical or nanotechnology tagging, physically unclonable functions, side-channel analysis, cryptographic hash checks, digital signatures, and visible anti-tamper seals or labels. Additional controls may involve monitoring for abnormal performance, which can signal tampering or counterfeit parts. If a system or component is suspected to be counterfeit or modified, the supplier, contractor, or original equipment manufacturer can either replace the item or carry out forensic analysis to trace its origin. Organizations may also train staff to recognize signs of suspicious hardware or component deliveries. [21]

Side-Channel

A side-channel attack, also known as a sidebar attack or an implementation attack, is a security breach that gathers information either from the system's execution process or by process manipulation. These attacks are based on observing and analyzing the physical behavior of encryption systems during cryptographic operations [22]. They aim is to obtain sensitive information, including cryptographic keys, by measuring simultaneous hardware emissions. This kind of attack can happen in electromagnetic, acoustic, power, optical, timing, memory cache, hardware-related, and other ways.
Simple Power Analysis (SPA) is a technique that involves direct interpretation of the power consumption measurements collected during cryptographic operations. Because SPA can reveal the sequence of instructions executed, it can be used to break cryptographic implementations where the execution path depends on the data being processed. Techniques for preventing simple power analysis are generally easy to implement. Avoiding procedures that use secret intermediates or keys for conditional branching operations will mask many SPA characteristics. In cases such as algorithms that cannot avoid branching, this can require creative coding and result in a serious performance penalty. Additionally, the micro code in some microprocessors cause large operand dependent power consumption features. For these systems, even constant execution path code can have serious SPA vulnerabilities. [23]
Implementing countermeasures to protect against side-channel attacks, such as noise injection, power balancing, and electromagnetic shielding, can help prevent adversaries from extracting sensitive data through unintentional information leakage. At hardware level, design logic gates, circuits, and silicon layouts is helpful to reduce physical leakage. This is done by utilizing tamper-resistant chips or shielding sensitive signals through the careful routing of metal layers.
Dynamic voltage variation is introduced randomly, displays with special shielding effects are utilized to reduce EMF emissions, and power analysis countermeasures, such as power gating and balancing of power consumption, are employed, making it challenging to directly correlate current consumption with logical operations.
Some other techniques that can be implemented into the hardware are physically unclonable functions (PUFs), physically random functions (PRFs), or performance counters. [22]

Fault injection

Fault injection countermeasures aim to protect embedded systems and hardware devices from glitching errors and alteration of program behavior. These countermeasures can be implemented at both the software and hardware levels.
At the software level, redundancy techniques can be implemented by replicating

critical operations and comparing the two outputs. A new scheme that not only detects faults injected by a malicious adversary but also automatically corrects single nibble/byte errors introduced by low-multiplicity faults was presented by Gay, Mael, et al. Moreover, when checking condition it is important to be sure that all possibilies are considered, even default ones to avoid potential data manipulation [REF-1141].
Introducing random delays before critical operations, making timing unpredictable is another software protection, together with program flow integrity protection. One such example of the latter is tagging the source code to indicate the points not to be bypassed [REF-1147].
At the hardware level, sensors can detect variations in voltage and current and shields provide physical barriers to protect the chips.
When a glitch is detected, it is best for security to trigger a CPU reset rather than forcing a repeat of the faulted instruction, as resetting the CPU would greatly slow down an attacker's repeated attempts to glitch the same instruction.
Examples of hardware mechanisms for glitch detection include fast-reacting voltage monitors and Brown-out Detection (BOD) circuits for voltage anomalies, tunable replica circuits (TRCs) for voltage and clock glitches, and phase-locked loops (PLLs) to detect electromagnetic fault injections (EMFI).
Comparison of external clocks to a high frequency ring oscillator based on internal reference clock can detect clock glitches while optical sensors and wire meshes can sense tamper events such as decapsulation, which may mitigate optical fault injection. Finally, shadow registers can enhance fault resiliency by introducing redundancy in stored data. [24]

Shielding e ruggedization

Components need to be physically protected from impacts, dust, humidity, or tampering. Shields are applied on the walls and cathodes to protect parts from being covered with undesired coating. They can also include creating an electrical shielding function with a defined or a floating potential.
Another type of shield is the thermal shield, applied for protection of radiation heat to keep the heat losses to the wall low. This is the main function in tool machines. In component applications, water-cooled shields can be applied to cool the substrate and enable an increased deposition rate.
Finally, dust shields are sometimes applied for protection against dust falling directly on the substrates. All shields must be easily replaceable and have to be maintained so they must be clearly identifiable. Rapid replacing shields increases the availability of the system. It is very important that shields are installed only on the locations they are intended for, since mixing of shield positions may create gaps deny the possibility for expansion, leading to bending and even short circuits.

Additionally, shields must be designed in a way that they can be easily cleaned by bead blasting, without influencing their designed shape. [25]
Ruggedized electronics are designed to prevail long-term, even when exposed to extreme conditions such as: severe winds, dust, dirt, rain, snow, ice, extreme temperatures changes and vibrations. For outdoor use, ruggedized electronics represent the most dependable choice. These devices are designed, built, and tested to provide enhanced durability, resilience, and robustness, as they must withstand exposure to environmental conditions typical in the manufacturing, transportation, military, and construction industries.[26]

Hardware Obfuscation and Camouflaging

Obfuscation aim to modify the design to hide the true functionality of the hardware, making it harder for adversaries to understand and replicate the design. Camouflaging instead involve the design of hardware components to make them look alike or include dummy components in the system, making it difficult to distinguish between critical and non-critical parts.
Partition the hardware design into different parts, each manufactured by separate entities, so no single manufacturer has access to the complete design is also a way to further enhances security. Finally, verification mechanisms, such as logic locking, can be employed to ensure that specific portions of the design can only be unlocked with the correct key, guaranteeing that the final product matches the intended specification. [27]

2.6.3 Physical Monitoring and Environmental Logging

Monitoring and logging are essential for detecting and responding to physical compromise attempts. Maintaining access logs of individuals who enter secured areas or interact with devices creates accountability and allows forensic analysis after a security incident.
As discussed, embedded systems can integrate environmental sensors to detect anomalies such as unexpected temperature changes, humidity spikes, or vibration patterns that may indicate tampering. Logging this information and linking it to alerting systems provides real-time visibility into physical threats and ensures that any suspicious activity can be promptly investigated so continuous monitoring is fundamental to enhance security. It is also important to track and log every individual who physically accesses the device or its enclosure.
Maintaining secure audit logs for forensics, enable organizations to reconstruct how and when an attempted compromise occurred. For this reason it is of high priority to securely store these logs.

2.6.4 Operational Safeguards

Physical device security efficiency depends on robust operational practices. Personnel must be trained to handle sensitive devices correctly, ensuring they are not inadvertently exposed to risks. Training means avoiding leaving equipment unattended in public areas but also teaching staff to recognize and report suspicious signs like broken seals or unusual packaging.
Secure disposal procedures represent the last line of defense. Before decommissioning or recycling devices, all sensitive data, cryptographic material, and critical components must be securely wiped and destroyed. This prevents attackers from retrieving valuable information from discarded hardware. Moreover, devices in transit should be shipped with proper chain-of-custody documentation and with tamper-evident packaging and encryption of onboard data.
Operational safeguards ensure that even when technical protections are in place, human and organizational factors reinforce the security posture, reducing the likelihood that oversight or negligence undermines overall defense.

2.7 Encryption and key management

Proper key management is essential to guarantee the security of every IT systems. Keys should be generated securely, stored in hardware-backed elements or protected partitions, and accessed only by authorized users and processes. Keys life cycle has to be managed too, especially considering that they can be compromised. Regular key rotation and revocation procedures need to be in place. Minimizing the exposure of keys to the application layer and following a secure provisioning process are essential best practices to maintain confidentiality and integrity over time.

2.7.1 Secure key generation

Keys must be produced using a cryptographically secure random number generator (CSPRNG) or, ideally, a true random number generator (TRNG). Unfortunately in cryptography a true random number generator is difficult to obtain so of a deterministic random bit generator (DRBG) is considered compliant with best practice, provided it is seeded with sufficient entropy [28]. Devices with constrained entropy sources should consolidate multiple entropy sources or fall back to secure provisioning. By ensuring the randomness of key material and combining it with hardware isolation, systems significantly lowering the risk of key compromise.

Hardware Security Modules

For long-term protection, sensitive material should be stored in tamper-resistant components such as Hardware Security Modules (HSMs) or Secure Elements, which provide safeguards against both physical and logical attacks. In embedded contexts, lightweight Secure Elements or Trusted Platform Modules (TPMs) often serve a similar function, acting as a hardware root of trust. By offloading critical cryptographic operations to such components, organizations significantly reduce the attack surface while ensuring compliance with standards such as FIPS 140-3 or Common Criteria (EAL).
HSM is equipped with one or more cryptographic processors, designed to protect the entire life cycle of a cryptographic key, used to allow or deny access to digital information. Generally these modules can be internal cards or external devices that connect directly to a computer or to a network server. In the PKI field, HSMs can be used by Certification Authorities (CA) and Registration Authorities (RA) to generate, manage and store cryptographic key pairs for strong authentication. Some characteristics that make HSMs important are:

- **Completeness:** HSMs are complete solutions for cryptographic processing, generation and storage of keys. They include an integrated package of hardware and firmware.

- **Performance:** HSMs have the advantage of being optimized to reduce latency and increase the efficiency of the cryptographic process.

- **Regulatory compliance:** Since an HSM must preserve the integrity of private keys in a PKI, it must adhere to the main operational and security standards defined by third-parties.

- **Centralized management:** Keys are stored in a single unit. This allows an adequate level of key protection.

- **Layered Protection:** HSM protects cryptographic keys to guarantee confidentiality of information.

- **Backup and Restore:** HSMs must ensure the backup of the managed keys. Since they are part of critical infrastructures, it is preferable that they are mounted in clusters, in order to obtain a high availability system.

- **Control mechanisms:** HSMs possess control mechanisms to monitor the system, such as logs and alerts. They also have anti-tampering features.

[29]

2.7.2 Cryptographic Algorithms

Choosing the appropriate cryptographic algorithms is essential for securing every device but in embedded devices in can be a challenge since resource and physical constraints need to be balanced. Symmetric algorithms such as AES-256 or ChaCha20 are recommended for encrypting data at rest or in transit, offering strong security efficiency and robustness with minimal overhead. For asymmetric operations, RSA with a minimum of 2048 bit is still acceptable although Elliptic Curve Cryptography (ECC) such as Ed22519 nowadays provides equivalent security with significantly smaller key sizes, making it suitable for resource-constrained devices. Regarding cryptographic hash functions, SHA-256 and SHA-512 are still accepted but since a collision was found in the SHA-2 family a transition to SHA-3 should be employed to verify data integrity and support digital signatures.

2.7.3 Certificates and PKI

Digital certificates enable authentication, integrity verification, and non-repudiation. Trusted certification authority (CA) provide the binding between private and public keys in order to verify identities across heterogeneous IoT ecosystems. Deploying a Public Key Infrastructure (PKI) enables organizations to automate certificate issuance, validation, and revocation, which is essential when managing potentially millions of devices.
Best practices for certificate management include:

- **Validating full certificate chains:** Root CA trust is ensured.
- **Revocation mechanism:** Certificate Revocation Lists (CRLs) or the Online Certificate Status Protocol (OCSP) are useful to make sure a certificate is valid.
- **Short-lived certificates:** Window of exposure is minimal in event of compromise.
- **Certificate renewal:** Automating certificate renewal processes is important, as manual renewal is error-prone and impractical in large-scale deployments. Protocols such as the Automated Certificate Management Environment (ACME, RFC 8555) are increasingly adopted as best practice to enable secure, standardized, and scalable certificate issuance.

Additionally, the manufacturer should provision devices with unique certificates, rather than sharing common credentials, to prevent systemic compromise.

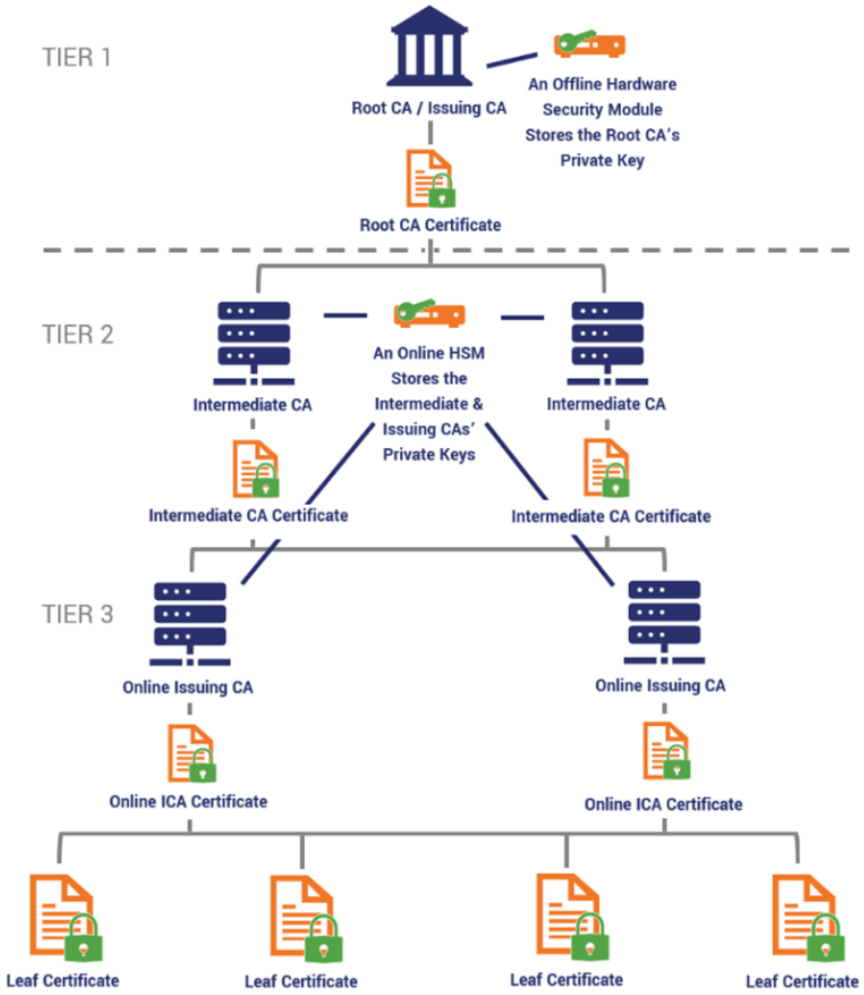

Figure 2.6: Three tier architecture PKI

2.8 Network security and logging

2.8.1 Network Architecture

Best practices for securing IoT networks includes segmentation, isolation, and proactive monitoring. IoT devices should be logically separated from critical networks using VLANs, dedicated firewalls, and network zoning, in order to reduce the impact of potential attacks. A **VLAN (Virtual Local Area Network)** is a logical division of a physical network. It is an isolated broadcast domain used to contain broadcast traffic within public and private networks by grouping devices based on function, location, or security requirements, allowing traffic control and

reducing the potential for lateral movement in case of compromise. [30]
A **dedicated firewall** is a device or software that monitors and controls incoming and outgoing network traffic based on predefined security rules. By deploying firewalls organizations can enforce stricter access controls and prevent unauthorized connections between devices and other parts of the network [31].
Network zoning involves creating separate network areas or "zones" with different security policies depending on the sensitivity or criticality of the devices they contain. This ensures that a compromised device in one zone cannot easily affect devices in more critical zones. Network zoning is the foundation of a defence-in-depth [32].

2.8.2 Network Protocol

Secure communication should rely on lightweight but robust protocols appropriate for constrained devices, such as MQTT-SN over TLS or CoAP over DTLS, ensuring confidentiality and integrity without overloading device resources.
TLS (Transport Layer Security) is a cryptographic protocol that establishes an encrypted channel between devices and servers, ensuring that data cannot be eavesdropped or modified and that endpoints can authenticate each other. Best practices recommend using TLS version 1.2 or higher, with strong key lengths (e.g., 2048-bit for RSA or 256-bit for AES) and modern cipher suites to protect against known cryptographic attacks. Applying these protocols consistently across the IoT network is a key best practice.

2.8.3 Logging and Monitoring

Effective logging and monitoring are essential to detect anomalies, support incident response, and maintain operational resilience in IoT environments. Logs should capture critical events such as device startup and shutdown. Event logging must be reliable, accessible with the right authorization, and in most cases, confidential so they need to be stored securely. Logs should be protected against tampering by guaranteeing integrity. Moreover, they should make it impossible for attackers to cover they work. [33].
Simple battery-powered or resource-constrained Considering the constraints of embedded devices logs are usually forwarded to a local hub or a central log management system to reduce overhead while maintaining visibility. Devices must be synchronized to an accurate time source so that timestamps can be correlated across multiple systems [21].
Additional best practices include defining log levels to balance resource use and

detail, rotating log files to manage storage, and making sure that sensitive information, such as passwords or cryptographic keys, are never logged.
Regular monitoring and analysis are critical to extract useful insights, detect potential and real faults, identify security breaches, and support forensic incident investigations. By following these recommendations, IoT systems can achieve a robust, reliable, and secure logging infrastructure that reinforces operational resilience. [4]

2.9 Application security and resilience

Application security and resilience in IoT ecosystems require both robust software protection and operational continuity. This subsection presents the key aspects to consider, with a focus on secure design, resilience, and development practices.

2.9.1 Input Validation

Input validation ensure only correctly formed data is entering the workflow in an information system, preventing malformed data from getting stored persistently in the database and causing malfunction. It also can contributes significantly to reducing the impact of injection attacks, buffer overflows and XSS.
Input validation should implement programming techniques that allows check at both syntactic and semantic levels. Some of these techniques are:

- Use native data type validators available in web frameworks.
- Using JSON Schema or XML Schema to validate input.
- Apply strict type conversion with exception handling.
- Establish minimum and maximum ranges for numbers and dates, and length checks for strings.
- Restrict input to predefined sets of allowed values for small parameters.
- Use regular expressions for structured data that covers the entire input string.
- Apply denylisting of known dangerous patterns as an additional defense, but always prefer allowlisting as the primary method.

[34]

2.9.2 Minimization of Exposed Interfaces and Secure Defaults

Reducing the attack surface is critical for securing IoT applications. At the software level, this means exposing only the APIs and services that are strictly necessary for device functionality, while disabling or removing unused interfaces and endpoints. Devices should also be deployed with secure default configurations: unique application level credentials must be enforced during setup, secure communication protocols (e.g., HTTPS, MQTT over TLS) should be enabled by default, and insecure options, such as plaintext authentication, deprecated cipher suites or obsolete version of the security communication protocols, must be avoided. [3]

2.9.3 Graceful Recovery and Safe-State Mechanisms

IoT applications must be resilient to crashes, unexpected failures, and outages of data networks or power. To address these challenges, several mechanisms are commonly adopted.

- **Watchdog timers** automatically reset the system if the application becomes unresponsive, ensuring that devices do not remain inoperable after a software crash. [21]
- **Safe-state defaults** provide fallback operating modes in which the device continues functioning in a limited but predictable and secure way. [35]
- **Automatic recovery routines** enable services to restart after unexpected failures, restoring connectivity and resynchronizing critical data where possible. [21]

Moreover, is important to implement graceful degradation and recovery. The first one ensure that an acceptable level of security is sustained during unexpected conditions while the latter facilitate the recommencement of normal operation. It is possible to make sure a system implement correctly graceful degradation and recovery through automated triggering and coordination. [36]

2.9.4 Application-Layer Denial-of-Service Protection

Application DDoS attacks are distributed denial-of-service (DDoS) attacks targeting application services, aiming to make them unavailable by overwhelming them with excessive traffic. The sudden surge in requests can saturate servers, networks, and IoT devices, causing service disruptions. Often, compromised computers, servers, and IoT devices (forming botnets) are used to amplify the attack. [37]
To mitigate application-layer DDoS attacks, several best practices are recommended:

- **Rate Limiting:** Enforce thresholds for requests per user or per IP address to prevent overwhelming the application [38, 37].
- **Traffic Filtering and Throttling:** Use web application firewalls (WAFs) or API gateways to detect and block malicious traffic patterns [38, 37].
- **Caching and Load Balancing:** Offload repeated requests to caches and distribute traffic across multiple servers to maintain service availability [35].
- **CAPTCHA and Challenge-Response Mechanisms:** Challenge suspicious clients to verify they are human, reducing automated bot traffic [39, 40].
- **Anomaly Detection and Monitoring:** Continuously monitor traffic patterns and system metrics to detect abnormal spikes early [21, 33, 3].
- **Segmentation and Isolation:** Isolate critical services from less critical ones to prevent a localized DDoS from cascading through the system [21, 41, 35].
- **Auto-Scaling and Redundancy:** Use elastic resources or redundant infrastructure to absorb traffic surges [21].

Implementing a combination of these strategies can significantly reduce the impact of application-layer DDoS attacks, ensuring the availability of IoT services even under difficult conditions.

2.9.5 Secure Development Practices

Some of the best practice to ensure a secure development process are:

- **Threat Modeling:** Identifying possible threats and attack vectors from the early stage of the design phase, considering all constraints given by the device and the organization.
- **Code Reviews and Peer Assessment:** Systematic review of source code to detect wrong or incorrect coding patterns, improper input handling and validation, hardcoded credentials, or other vulnerabilities. This can be done manually or automatically with tools.
- **Fuzz Testing:** Automated injection of malformed or unexpected inputs to discover buffer overflows, crashes, and logic flaws in the application code.
- **Static and Dynamic Analysis:** Use of static analysis tools to detect vulnerabilities in source code or binaries, and dynamic analysis to identify weaknesses during execution.

- **Dependency and Library Management:** Keeping third-party libraries and dependencies up to date, tracking known vulnerabilities (CVEs), and applying patches promptly to minimize exposure.

- **Continuous Security Testing:** Integration of automated security tests into the CI/CD pipeline, ensuring that vulnerabilities are detected early and fixes are verified before deployment.

- **Documentation and Traceability:** Maintaining detailed records of security requirements, design decisions, tests results, and mitigation measures.

2.10 Handling known vulnerabilities (CVEs)

An important aspect to consider in securing a device is the management of known vulnerabilities. The term CVE stands for Common Vulnerabilities and Exposures and refers to a globally recognized catalog of unique identifiers for publicly known vulnerabilities. It is maintained by the MITRE Corporation and financed by the Department of Security USA Cybersecurity section.
The CVE catalog is more like a dictionary than a database. Each entry has a name, a description and a number serving as the identifier (ID CVE). Moreover, a score is assigned to each vulnerability. A CVE does not mean immediately that there is an exploit for it but that one can be found by an attacker. [42]
In the embedded world, this issue is more relevant because devices often have long life cycles, limited computational and storage resources, infrequent or even absent update mechanisms and are deployed in uncontrolled environments, expanding the attack surface and emphasizing the necessity for systematic vulnerability management.
For a vulnerability to become a CVE some criteria need to be met:

- The vulnerability needs to be resolved independently from other bugs or defects.

- The supplier has to recognize the possible impact on security. Alternatively, a report demonstrating such negative impact need to exists.

- The vulnerability interest just one code base or product. Defects that interest more than one product have a different ID CVE for each product.

[43]

2.10.1 The Vulnerability Management Process

Each day a number of vulnerabilities are discovered. In order to protect a device a cyclical process that span the whole product life cycle need to be put in place. There are four stages: [33, 44]

1. **Identification:** It can be done by continuous monitoring official sources such as the National Vulnerability Database (NVD) or by integrating into DevSecOps pipelinse an automated control trough API. The latter also helps early detection and reduce exposure time.

2. **Assessment:** After identification, vulnerabilities must be evaluated in terms of the impact they have on security. For this purpose the CVSS (Common Vulnerability Scoring System) is used. It assing a numeric value from 0 to 10 based on the exploitability and impact metrics such as base, temporal, and environmental. [43, 44, 33]

3. **Mitigation:** Not all vulnerabilities can be addressed simultaneously. Based on the CVSS scores some need to treated as high priority and a patch need to be applied almost immediately while others can be managed in time. Mitigation strategies may include secure patch deployment, over-the-air (OTA) updates, or temporary measures such as network segmentation, disabling vulnerable features, or virtual patching at gateway or middleware levels. [33, 45]

4. **Disclosure:** To reduce exposure time and improve mitigation vulnerabilities have to be disclosed. Coordinated Vulnerability Disclosure (CVD) ensures a structured and transparent process for reporting vulnerabilities to the vendor. A proper CVD program includes a dedicated reporting channel, defined response timelines (SLA), and coordination with CERT/CSIRT teams when needed. [3, 33]

As every security element, vulnerability management should not be considered as an isolated post-deployment activity but as an integral component of the design and development. This means that after deployment some operations need to be performed, such as: continuous monitoring, maintaining a detailed documentation of the state of each vulnerability found, and defining the responsibility of a vendor in terms of time. [19, 35] [4]

2.10.2 Summary and Best Practice

In conclusion, effectively managing known vulnerabilities (CVEs) in embedded devices requires a structured approach. This process begins with the continuous monitoring of vulnerability databases and advisories to ensure detection of relevant

threats happens in time. Identified Assessment of the vulnerabilities found using standardized frameworks such as CVSS, while carefully considering the specific constraints and operational context of IoT devices must be done. Patching and mitigation must be prioritized according to score and exploitability and patching should follow secure and reliable management mechanisms. OTA updates are essential to maintain device security throughout its operational life cycle. When immediate patching is not feasible, temporary mitigation can help reduce exposure. Moreover, establishing a disclosure policy, following industry standards, helps with timely resolution of security issues. By embedding vulnerability management into every stage of the product life cycle, organizations can enhance long-term resilience, safeguard user trust, and ensure alignment with regulatory and best practice frameworks.

Chapter 3

Threats and Vulnerabilities in Embedded Devices

3.1 Firmware modification attacks

Firmware modification is a critical concept in cybersecurity. Firmware modification attacks represent a general and effective strategy well-suited for exploiting embedded devices. The goal is to introduce arbitrary and persistent changes to the victim's firmware by leveraging common design flaws found within embedded software. These kind of attacks can be categorized into several types, including hardware and firmware trojans, ransomware, wiper, and various forms of malware targeting Industrial Control Systems (ICS). Firmware modification attacks can compromise entire families of devices sharing the same systemic design flaws, regardless of operating system versions and hardware architecture.
Firmware modification attacks often aim to inject malware into the target embedded device. They can be carried out as standalone attacks, by manipulating firmware update mechanisms, or as secondary payloads following initial exploitation via traditional attack vectors (e.g., software vulnerabilities or memory modification attacks). [46, 47]

3.1.1 Enabling Factors

Authentication and firmware signature verification cannot fully prevent firmware modification attacks on embedded systems with vulnerable attack surfaces.

Vector / Variant	Description
Firmware Update Manipulation	Many devices have update mechanisms that are not sufficiently protected by proper user authentication. Devices that require authentication before updates can often be compromised via trivial administrative interface attacks.
Network Boot Vulnerabilities	Devices that boot their firmware or operating system over the network often use insecure protocols, like TFTP, and because of this, are vulnerable to standard OSI Layer 2 attacks.
Platform Independence	Attacks that manipulate firmware update features generally work across multiple models and architectures, without relying on specific software vulnerabilities.
OS Limitations	Embedded operating systems typically lack fine-grain privilege separation and execution isolation. Sometimes they are present in the device but vendors decide not to utilize them. Even if present and used, kernel or privileged process vulnerabilities can still allow persistent firmware modifications.
Supply Chain Exploitation	Weak access controls, insecure channel communication, or missing on-device verification may result in compromised firmware installation. Common issues include incomplete inspection procedures in bootloaders and weak verification mechanisms in update agents.
Targeting Update Processes	Attackers can intercept or tamper with updates delivery, injecting malicious firmware that gets installed without verification.
Exploiting In-Device Vulnerabilities	Vulnerabilities in existing firmware or privileged processes can be exploited to inject malicious code that remains inactive until triggered.
Physical Manipulation	Gaining physical access allows attackers to directly reprogram nonvolatile memory (e.g., flash) and install malicious firmware.

Table 3.1: Vectors and Variants of Firmware Modification Attacks

3.1.2 Real-World Examples

Autonomous Firmware Zombie Attack (AFZA)

Traditionally, "zombie" attacks involve botnets and direct network control. In such attacks, an attacker infects multiple devices, gaining full or partial control. Each infected device, known as a zombie, becomes part of the botnet, which the attacker uses to launch coordinated attacks. These attacks often aggregate all zombies' processing power and network connections to execute large-scale offensives [48].
The "Autonomous Firmware Zombie Attack" (AFZA) represents a paradigm shift in the threat landscape of ICS. Unlike conventional cyberattacks that rely on continuous remote control and command via compromised networks, the AFZA attack leverages the inherent vulnerabilities associated with the trusted yet insecure communication protocols and the segregated network environment typical of ICS. This novel attack vector involves the injection of maliciously crafted firmware into industrial devices, possibly during manufacturing, maintenance, or through compromised supply chain processes and insecure connections. Once infected, these devices operate autonomously, executing predefined malicious activities when specific conditions are met without further external command.

PANdora's Box

Vulnerabilities in Palo Alto Firewalls allowed attackers to bypass Secure Boot, modify UEFI/system configurations, and install persistent and stealthy malware.[49]

BootKitty and the Rise of Bootkits (2024)

Recently, security researchers have been analyzing and publishing details about "Iranukit" and "Bootkitty," malware targeting Linux systems. Bootkitty was covered by the media and was demeed the first UEFI bootkit for Linux, created as a proof-of-concept, demonstrating the feasibility of Linux-targeted UEFI bootkits, even if it is not malicius itself. [49]

AMD Sinkhole/SinkClose (2024)

In mid-2024, researchers identified a significant vulnerability, "Sinkhole," in AMD processor chips dating back to 2006. This flaw allows attackers to install persistent malware after having infiltrated the system through the System Management Mode. These malicious entities can remain undetected even after the reboot or reinstall of the operating system. This pose a severe threat to system integrity. AMD has acknowledged the issue and released mitigation options for several product lines. [49]

LogoFAIL (2023)

Some vulnerabilities are called LogoFAIL because they exist in UEFI image parsers that display the manufacturer logo when the system boots up. They affect both Windows and Linux systems.[49]

PixieFail (2023)

The "PixieFail" vulnerabilities are a set of nine critical security flaws in the IPv6 stack of in Tianocore's EDK II. These vulnerabilities are mainly found in the Preboot Execution Environment (PXE) of the UEFI specification. This type of exploit is particularly dangerous as it occurs before the operating system loads, bypassing many traditional measures such as antivirus software or operating system-level security features. [49]

3.1.3 Impact and Consequences

- **Persistent, Long-Term Access**: modified firmware provides attackers with a foothold that survives operating system reinstallation.
- **Hardware Control**: compromised firmware allows attackers to manipulate hardware components and disrupt operations.
- **Malware Injection**: attackers can introduce backdoors, ransomware, or other malware hidden within the device firmware.
- **Denial of Service**: malicious modifications can permanently disable devices, stop services, or cause important data loss.
- **Critical Infrastructure Disruption**: in ICS and smart grid environments, firmware attacks can manipulate industrial processes such as controlling circuit breakers, resulting in potentially catastrophic outcomes.

3.1.4 Countermeasures (Overview)

To defend against firmware modification attacks it is important to implement a layered strategy including secure boot to load only trusted firmware, cryptographically signed updates to verify authenticity, buffer and stack overflow protection to prevent code injection, and robust firmware analysis to detect malicious code or vulnerabilities in the verification process. Regularly patching devices with the latest firmware is also crucial to address known weaknesses.

3.2 Use of default or hardcoded credentials

A recurring vulnerability in embedded and IIoT devices is the use of default or weak passwords. Default credentials are often hardcoded by vendors to facilitate initial setup and are rarely changed by end users. Weak passwords, on the other hand, are chosen by users themselves but lack sufficient length, complexity, or unpredictability. [50]

3.2.1 Consequences

- **Unauthorized Access**: Weak or default passwords allow attackers to enter systems, accounts, or networks they should not have access to, ranging from personal devices to critical business infrastructures. This can compromise network security and privacy. Once inside, the attacker can extract sensitive information, impersonate the real user, or disrupt operations.

- **Account Takeover**: Attackers can perform fraudulent activities by obtaining access to one account and leveraging the information found there to access other accounts, especially if the same password is reused. This domino effect can compromise personal and professional digital data.

- **Data Breaches**: When attackers infiltrate one account, they can often navigate through an entire network and open the door to large-scale leaks of sensitive data, including personal information, intellectual property, trade secrets, and customer records.

- **Identity Theft**: Stolen credentials are frequently used to impersonate individuals, enabling fraudulent activities such as opening credit lines or scams.

- **System Compromise and Ransomware**: Attackers may escalate privileges, gain persistent access, or deploy ransomware, locking users out and demanding payment to restore data.

- **Financial Losses**: Individuals and organizations can suffer monetary damage. In a business context, a breached account can cost millions to the company because it can lead to stolen funds or intellectual property. For individuals, the theft of banking or credit card information can have immediate financial implications.

- **Website Takeover**: For website managers and owners, weak credentials can result in attackers seizing control of websites, compromise of content, stealing customer data, or redirecting traffic to malicious platforms.

- **Reputation Damage**: For businesses, a security breach can damage their reputation, irreparably harm a business's brand and customer relationships.
- **Legal Consequences**: Organizations may face lawsuits, fines, and regulatory penalties for failing to implement sufficient password security measures.

[50]

3.2.2 Common Attacks

Passwords are the gatekeepers to everyone personal and professional lives. From social media accounts to online banking, these strings of characters protect the most sensitive information. However, password attacks are a constant threat, as adversaries continuously develop techniques to bypass weak or poorly managed credentials. Understanding the most common attack methods is crucial for mitigating risk and strengthening authentication security.

Brute-force Attack

It is the most straightforward technique, in which an attacker systematically tries every possible password combination until the correct one is found.
In the manual cases, attackers enhance brute force with social engineering, leveraging personal information such as activities, hobbies or public data from social network and the web to guide their guesses. The strings are then entered manually by the attacker.
In the automated case, a software tool generates random strings and pairs them with an identifier (e.g., a corporate email). Modern tools can automate this process at high speed, making short or simple passwords especially vulnerable. This highlights the importance of using sufficiently long and complex passwords to withstand exhaustive guessing attempts. [51, 52]

Dictionary Attack

Attackers rely on lists of likely passwords, often compiled from previous data breaches and sold on underground markets. Unfortunately, analysis of such leaks reveals that many passwords are reused globally with alarming frequency. In this context, the attacker repeatedly attempts access until the security barrier is breached. Short passwords, as well as predictable strings, make the attacker's job easier.
A dictionary is a collection of strings that are not limited to actual words in the dictionary. They could also include popular names of pets, movie characters, and people. Attackers developed software to automatically test these strings. These

dictionaries include also predictable variations, such as substituting letters with numbers. This behavior is common in workplace environments where frequent mandatory password changes are enforced and users add minimal change to the previous password to make it more "secure".
The widespread reuse of leaked passwords and the presence of tools to automate the process significantly increase the success rate of this technique. A successful dictionary attack can lead to the theft of sensitive data, account compromise, and even the spread of stolen credentials into future attack campaigns.
The stolen credentials themselves may be added to new dictionaries for future credential stuffing attempts. Furthermore, unauthorized system access may enable privilege escalation or even the deployment of ransomware. [51, 52]

Reverse Brute-force Attack

In this case, the attack approach is the opposite of the traditional brute force attack, instead of attempting many candidate passwords against one account, the attacker tests a single weak password against a large set of usernames.
Typically, these are weak passwords that appear frequently in dictionaries available to attackers. To exploit this, the attacker needs a large collection of usernames or email addresses, relying on the statistical likelihood that at least one user will have chosen the weak password in question. The premise of this attack is that, within a large pool of identifiers, there will be at least one individual who does not pay adequate attention to security. Given the popularity of predictable strings like “123456” or “password”, this strategy often succeeds within large user populations, underscoring how even one careless choice can expose entire systems to compromise. The fact that reverse brute force attacks remain widely used today demonstrates both their effectiveness and the persistent lack of security awareness among many users. [51, 52]

Credential Stuffing

Credential stuffing password attacks are similar to brute-force attacks where the attackers use trial-and-error to gain access and exploits the tendency of users to reuse the same password across multiple platforms. Attackers take username–password pairs stolen from one breach and test them against other services, hoping that at least some will still be valid. Large-scale automation makes credential stuffing highly efficient, and its success rate grows proportionally with the number of credentials exposed in previous incidents.
Over the years, numerous breaches of websites and cloud-based services have resulted in a massive number of compromised credentials sold by cybercriminals or gave away on the dark web. As with brute-force attacks, automated tools make these password attacks incredibly successful. [51, 52]

Password Spraying

Instead of trying numerous guesses on a single account, which could trigger lockout mechanisms, attackers test a small set of common passwords across many accounts. Usually, they'll conduct reconnaissance first to limit the number of login attempts and prevent account lockup. By rotating attempts carefully, they reduce the risk of detection while still achieving a high probability of success. [51, 52]

Keylogger Attack

Beyond guessing strategies, attackers may also deploy malware. A keylogger is a type of spyware that records a user's activity by logging keyboard strokes. Keyloggers may be delivered as malicious attachments, hidden in software bundles, or installed directly on compromised systems. Since planting hardware on a device takes a lot of extra work, the threat actors are more likely to install malware on a computer or device by luring a user to click on a malicious link or attachment. Unlike guessing attacks, this method circumvents password complexity entirely, directly harvesting sensitive data such as login details or credit card numbers. [51, 52]

Phishing

Instead of breaking the password through computation, attackers trick victims into revealing their credentials on fraudulent websites or through deceptive emails. Although not a technical attack on the password itself, phishing continues to be highly effective, exploiting human trust rather than computational weaknesses. [51, 52]

3.2.3 Countermeasures (Overview)

These attacks can have everyone as a target, from individuals to large companies. Knowing what a password attack is helps you understand why strong security measures are important. Effective defenses require both technical safeguards, such as multi-factor authentication and account lockout policies, and user awareness to reduce susceptibility to social engineering and poor password hygiene. [52]

3.3 Unencrypted Communications

Unencrypted communications are a vulnerability in the network communication protocols of web, API and infrastructure. This vulnerability occurs when packets are sent over the network without cryptographic protection, making the data readable by an attacker inside the channel. The information sent include sensitive

data such as usernames, passwords, financial information, personal messages, and system commands. Unencrypted communications are usually used in outdated protocols like HTTP, Telnet, FTP or inside systems that fail to adhere to encryption standards.
The consequences can range from credential theft and identity fraud to unauthorized system access and disruption of critical infrastructure.[53]
In modern IIoT and embedded systems, unencrypted communication represents a significant attack surface that can be used to compromise confidentiality, integrity, and availability of data and services.

3.3.1 Common Attacks

Sniffing Attack

Sniffing refers to the process of intercepting and inspecting data packets as they travel across a network. It is like eavesdropping on a conversation between devices. This technique is commonly used for troubleshooting network issues, but it can also be exploited for malicious purposes. Sniffing makes it possible to analyze the data, revealing details like source and destination addresses, content, and even sensitive information such as login credentials and credit card numbers.
An attacker can place a network interface card in promiscuous mode and capture surrounding traffic using tools like Wireshark or tcpdump [54, 45]. Using unsecured public networks, such as those in cafes or airports, increase the risk of being exposed to a sniffing attack. Sniffing is often the first step toward more complex attacks [55].

Man-in-the-Middle (MITM) Attack

A Man-in-the-Middle (MITM) attack is a cyberattack in which an attacker intercepts communications between two online targets, in order to steal sensitive information. By secretly positioning themselves between the two parties, the attacker can capture sensitive data and carry out malicious activities, including unauthorized purchases, financial account theft, and identity fraud. [56]
Beyond interactions between a user and an application, a MITM attacker may also intercept private communications between two individuals. In such cases, the attacker can redirect and forward messages between the parties, sometimes altering or replacing messages to manipulate the conversation [55, 57]. This is common in unprotected Wi-Fi environments or when insecure protocols are used. The danger lies in the fact that both victims continue to see the connection as legitimate, unaware of the manipulation.

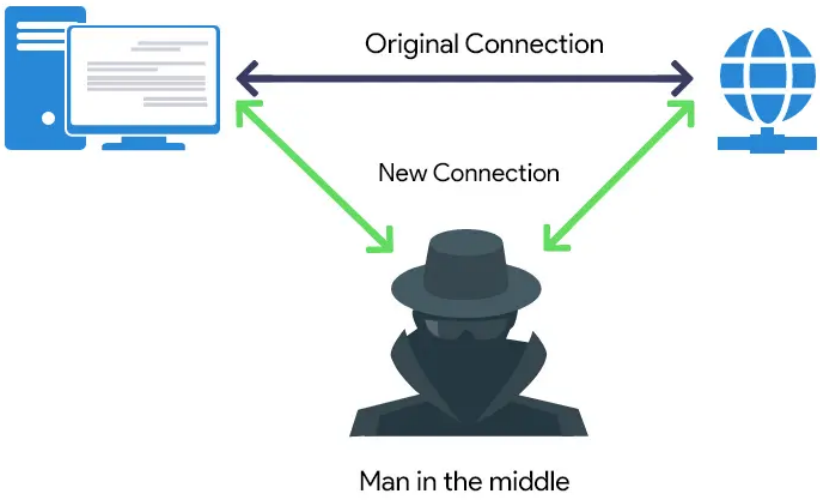

Figure 3.1: Man-in-the-middle attack

Session Hijacking

Session hijacking occurs when an attacker steals a session identifier (such as a cookie or authentication token) to impersonate a legitimate user [58, 1]. Once the attacker takes over the session, they can act with the victim's privileges, for example, making transactions, sending emails, or accessing restricted data. Hijacking is often made possible by sniffing unencrypted traffic or exploiting weak session management.

Replay Attack

In a replay attack, the attacker captures legitimate communication (such as authentication requests or transactions) and retransmits it later to trick the system into executing the action again [58, 45]. Since the data is not encrypted or bound to a unique session, the attacker can, for example, resend a payment authorization or log in without knowing the actual credentials.

Data Tampering / Injection Attack

Unencrypted traffic allows adversaries not only to read but also to modify communications [59, 1]. Attackers may alter commands, inject malicious payloads, or manipulate transmitted parameters (e.g., prices in an e-commerce request). This compromises both the integrity and reliability of the communication.

ARP Poisoning Attack

Address Resolution Protocol (ARP) poisoning is a local network attack in which an adversary sends forged ARP messages to associate their MAC address with the IP address of another device (often the gateway) [54, 45]. This enables the attacker to intercept, modify, or block traffic between devices. When communications are unencrypted, ARP poisoning can easily escalate to credential theft, MITM, or data manipulation.

Credential Theft

Since unencrypted communication often includes credentials (usernames, passwords, tokens) sent in plain text, attackers can directly capture them during transmission [60, 1]. This is particularly dangerous in protocols such as Telnet, FTP, or HTTP, where credential and other login information are not protected by encryption.

3.3.2 Countermeasures (Overview)

Unencrypted communications expose devices and networks to multiple threats. Effective countermeasures include:

- **End-to-End Encryption:** Protects data in transit using protocols such as TLS/SSL [57, 58].
- **Secure Authentication:** Employ strong password policies, multi-factor authentication, and token-based sessions [1].
- **Network Security:** Use VPNs, firewalls, and secure Wi-Fi configurations (WPA) to prevent unauthorized access [54, 45].
- **Session Management:** Implement short lived session, secure cookies, and re-authentication for sensitive operations [59].
- **Integrity Checks:** Apply cryptographic integrity validation, like MACs and digital signatures to prevent tampering [57].
- **Monitoring and Logging:** Detect anomalies and potential attacks such as MITM, ARP poisoning, and replay attacks through continuous monitoring [55].

3.4 Memory attacks

Memory-based attacks, known as fileless or non-malware attacks, are swiftly becoming a significant threat in the field of cybersecurity. They describe a category of

cyber threats in which an attacker exploits vulnerabilities in a computer system's memory to execute malicious activities, and avoid being detected by traditional file-based antivirus solutions.
Memory-based attacks differ in terms of mode of operation and stealth characteristics. These intrusions omit the installation of persistent copies of malware within a system's hard drive bypassing signature-based detection and leaving minimal traces for forensic investigation.They leverage existing system tools and processes to exploit vulnerabilities of an in-memory space, so the malicious code can be directly executed on the RAM. This makes memory-based attacks covert, short lived, and incredibly challenging to detect and remediate using usual security measures. [61]

3.4.1 Common Attacks

Buffer Overflow Attacks

Buffers are memory storage regions that temporarily hold data. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer and push past the boundaries of the buffer. As a result, the program overwrites adjacent memory locations causing some of that data to leak into other buffers, corrupting the data they were holding.
The attack typically involves violating programming languages, manipulating the coding error, and overwriting the bounds of the buffers they exist on to carry out malicious activities and compromise the system. Most buffer overflows are caused by the combination of manipulating memory and mistaken assumptions about the composition or size of data.
There are different kinds of buffer overflows. Stack-based buffer overflows are the most common and leverage the stack, the memory space used to store user input, local variable and return pointers, allocated during the execution time of a function. When an attacker sends data containing malicious code to an application to overwriting the return pointer, control is transferred to the attacker.
Heap-based buffer overflows are more difficult to carry out than the stack-based approach. The attacker needs to flood the dynamically reserved memory space for a program beyond the memory it uses for current runtime operations. An attacker can corrupt function pointers or allocator metadata, gaining code execution with the privileges of the vulnerable process.
Flaws in buffer overflows exist in application servers, web servers and also in custom web application codes. The latest are given less scrutiny by security teams but are less likely to be discovered by hackers so they are more difficult to exploit.
Common consequences of this kind of attack include system crashes, loss of access control, and arbitrary code execution that an attacker can also use to exploit other vulnerabilities and subvert other security services. [62, 63, 64]

Format String Attack

Most programming languages have a mechanism to convert arbitrary data into a string. Usually there are some formatting mechanism to specify how the developer wants the output. Some of these mechanisms are quite powerful and privileged, especially in memory-unsafe languages.
The Format String attack happens when the submitted data of an input string are evaluated as a command by an application. By doing so, the attacker can execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system. The attack is possible when the application does not provide proper validatation of the submitted input. It is possible for an attacked to define a well-crafted input that could change the behavior of the format function (e.g. printf, sscanf), allowing the possibility of denial of service or the execution of arbitrary commands.[65]

AMD BadRAM (2024)

BadRAM is a new attack that creates aliases in the physical address space of the Dynamic Ram (DRAM) modules. By manipulating the Serial Presence Detect (SPD) chip on a memory module, an adversaries can trick the system into believing the DRAM is larger than it actually is. [49]

Cold Boot Attack

A Cold Boot Attack is a hardware-based cyber attack that targets the contents of a computer's random access memory (RAM) after a system reboot, exploiting the fact that the contents of the RAM do not vanish immediately after a the power off especially in systems that prioritize fast boot-up and shutdown processes over memory clearing.
RAM is a volatile memory and its data disappear gradually. This process can be slowed significantly if the memory modules are kept at a low temperature. Cooling techniques, such as the use of compressed air sprays or liquid nitrogen, can extend the retention time of data in memory. The residual data can persist for a brief period, potentially allowing retrieval of the information previously stored. The data stolen can range from encryption keys and passwords to other critical and sensitive data.
Researchers at Princeton University took advantage of this concept to publish a new attack marketed toward machines that have full disk encryption (FDE), where the encryption key is stored unencrypted in RAM while the computer is fully booted. [66] During a cold boot attack, an attacker gains access to a computer's RAM by physically removing it or by rebooting the computer with a specially

crafted bootable device. The cold boot attack, also known as the RAM dump attack (Anderson and Anderson, 2010), relies on the fact that most PCs can boot from an external USB device. [67]
Cold boot attacks present several security concerns even if they are considered rare compared to other attacks because they require physical access to the targeted system or component. However, they can still occur in certain situations: when the attacker has specific knowledge of the targeted system and its vulnerabilities, when the device is left unattended in public, when there are no physical security measure or when there is no access control to critical equipment. Cold boot attacks can bypass software-based security measures and can be executed quickly, circumventing encryption and other protective technologies, underlining the importance of combining cryptographic solutions with strong physical security measures and robust system configurations, in a multi layer security approach.[68]

Rowhammer Attack

Rowhammer is a security vulnerability in DRAM, enabling an attacker to change the data stored within memory cells. While DRAM offers many desirable advantages, rowhammer vulnerability, poses a significant threat to computing systems. Rowhammer attacks take advantage of the physical proximity of these cells within the memory array.
A typical rowhammer attack involves the repetitive access of a specific row within a DRAM chip which can cause bit flips in neighboring rows, causing unexpected changes in their values. The attack is assumed to be possible thanks to three potential causes: bridging between neighboring rows thanks to conductive channels between separate wires and capacitors in DRAM, electromagnetic coupling when a voltage alteration in a word-line introduce noise into a neighboring word-line, hot carrier injection that can escalate charge leakage from victim cells leading to the alteration of the stored data.
Since, typically, software and processors employ virtual addresses and physical addresses, rather than DRAM addresses, an an understanding of DRAM address mapping is required to execute this kind of attack. Rowhammer attacks targeting an operating system have the potential to facilitate privilege escalation and can target cryptographic keys stored in memory. If an attacker successfully flips bits in the memory where encryption keys are stored, they can decrypt sensitive data, compromising the confidentiality of encrypted communications or stored data. [69]

3.4.2 Countermeasures (Overview)

General countermeasures involve following secure coding practices, validating all inputs, employing memory-safe programming techniques and use built-int

protections such as memory isolation. Additional protections such as memory encryption, Secure Boot, and access control, help further reduce exposure. Runtime defenses—including stack canaries and ASLR, reduce the potential of buffer and heap overflows. Layering these measures with monitoring and anomaly detection further strengthens overall system security.

3.5 Network protocol attacks

A network protocol is a set rules and regulations for communication among devices over a network. The protocols specify the message format, transmission methods, how devices respond to communication and how they authenticate. Some of the most used protocols are Transmission Control Protocol (TCP), Internet Protocol (IP), Hypertext Transfer Protocol (HTTP), Domain Name System (DNS) and Transport Layer Security (TLS).
These protocols, and the many others, have a number of vulnerabilities that are prone to be used by hackers leading to their active exploitation and posing serious challenges to network security. There are attacks that target different layers of the network stack, from the application layer down to the data link layer, and may result in unauthorized access, data leakage, service disruption, or the propagation of malware. [70]

3.5.1 Common Attacks

DNS Cache Poisoning

DNS cache poisoning, also known as DNS spoofing, is a type of cyber attack posing a significant and persistent threat, especially over the past decade. It targets the Domain Name System (DNS), which maps human-readable domain names into machine-readable IP addresses, facilitating seamless communication between users and online resources and enabling computers to communicate efficiently over the Internet.
DNS cache poisoning aims to redirect traffic from legitimate websites to malicious ones by manipulating DNS cache records. This manipulation exploits the fact that DNS operates over UDP rather than TCP, which lacks a handshake mechanism and therefore allows attackers to forge packets more easily. With UDP, there is no guarantee that a connection is open or that the recipient is ready to receive. UDP is vulnerable to forging because an attacker can send a message via UDP and pretend it is a response from a legitimate server by forging the header data.
When a user attempts to access a website, their operating system sends a request to a DNS resolver. [71] DNS resolvers handle these queries and temporarily store IP addresses in their caches to answer future queries much more quickly, without

the need to communicate with the multiple servers involved in the typical DNS resolution process, guided by the time-to-live (TTL) associated with each record. Since resolvers cannot independently verify the accuracy of the received data, forged responses are often accepted uncritically, causing the resolver to store incorrect IP addresses. This vulnerability exists because DNS was originally designed for a trusted environment of universities and research centers, where the threat of malicious actors was negligible.

Despite these major vulnerabilities in the DNS caching process, DNS poisoning

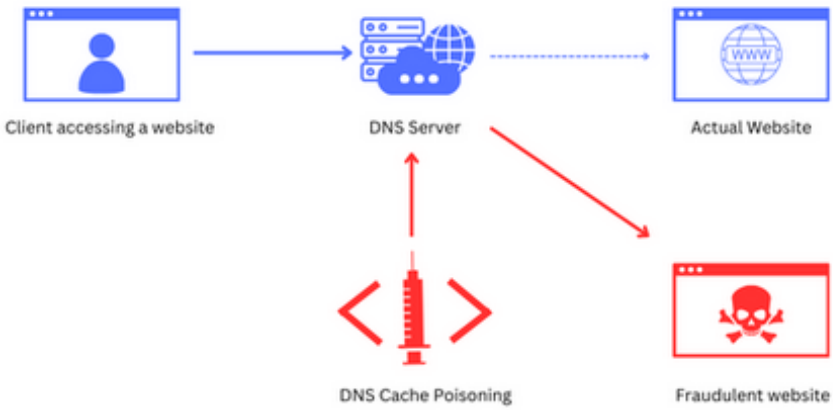

Figure 3.2: DNS Cache Poisoning

attacks are not simple to execute. Since the DNS resolver actually queries the authoritative nameserver, attackers only have a few milliseconds to send the forged response before the legitimate response from the authoritative nameserver arrives. Successful attacks require knowledge or guesses about several factors, including which queries are not cached, the port number used by the resolver, the request's identification number, and the authoritative server that will respond. In addition to remote packet spoofing, attackers can also gain access to the DNS resolver through other means. If a malicious actor operates, compromises, or physically gains access to a DNS resolver, they can easily tamper with or alter the cached data. When DNS cache poisoning succeeds, users are unknowingly exposed to phishing, malware, or other malicious activities. The consequences extend beyond individual users, undermining trust in domain resolution and the reliability of online services. [72, 73]

HTTP Response Splitting

HTTP Response Splitting is a vulnerability that occurs when data enters a web application through an untrusted input included in an HTTP response header without being validated for malicious characters. Malicious input is injected into various standard and user defined HTTP headers through use of Carriage Return (CR), Line Feed (LF), Horizontal Tab (HT), Space (SP) characters as well as other compliant special characters, and unique character encoding.
The underlying platform must be vulnerable to the injection of such characters that not only give attackers control of the remaining headers and body of the response the application intends to send, but also allow them to create additional responses entirely under their control bypassing security controls and enabling unauthorized access to sensitive data or the execution of further attacks. The vulnerability is often exploited through discrepancies in how different HTTP agents, such as web servers, proxies, load balancers, caching servers, application firewalls, or client browsers, parse and interpret HTTP messages. HTTP Response Splitting is usually the result of the usage of outdated or incompatible HTTP protocol versions as well as lack of syntax checking and filtering of user input in the HTTP agents receiving HTTP messages in the path.
Key prerequisites include a vulnerable or compromised server or domain, controlled data, differences in HTTP parsing between intermediary and client agents, and headers that can be manipulated by users. Target systems often operate on HTTP/1.0 or HTTP/1.1, supporting features such as Keep-Alive connections, pipelined requests, and chunked responses.
The attackers performs network reconnaissance by monitoring important traffic to identify the network path and parsing of the HTTP messages with the goal of identifying potential targets. They then experiment with a variety of ambiguous HTTP requests to observe responses from HTTP infrastructure in order to identify discrepancies in the interpretation and parsing of HTTP requests. Once the behavior is understood, the attacker sends crafted HTTP requests to inject malicious headers, scripts, or objects into the responses seen by client or intermediary agents.
The ability of the attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including: Cross-User Defacement, Cache Poisoning, Cross-site Scripting (XSS) and Page Hijacking.[74, 75]

TLS Downgrade

A TLS downgrade attack is a technique that exploits backward compatibility in the security protocol to force clients and servers to use outdated and vulnerable cryptographic standards. The strength of TLS protection lies in the encryption algorithms and security parameters that it works on. These algorithms and parameters differ from one SSL/TLS version to another. When a security element

of a TLS version is found to be seriously vulnerable, that version of SSL/TLS is deprecated and is replaced by a newer version. TLS is designed to ensure the confidentiality and integrity of online communications, but its earlier versions suffer from well-known cryptographic weaknesses and the fallback mechanism, introduced to maintain compatibility with legacy systems, creates an opportunity for attackers to force parties into insecure configurations.

The attack typically occurs in a man-in-the-middle scenario. The adversary intercepts the handshake and alters the messages exchanged, tricking both parties into using weaker algorithms or versions of legacy protocols. Once the downgrade is successful, the attacker can exploit existing vulnerabilities to decrypt data in transit, manipulate communications, or impersonate one of the endpoints.

Downgrade attacks have existed since the early versions of TLS. SSL 2.0 suffers from the "ciphersuite rollback" attack, where the attacker limits SSL 2.0 strength to the weakest ciphersuite. SSL 3.0 is vulnerable to the "version rollback" attack that works by modifying the client's proposed version from SSL 3.0 to SSL 2.0, so that all SSL 2.0 weaknesses are inherited. Another design flaw in SSL 3.0 allows a theoretical attack named the "key-exchange rollback" where the attacker modifies the client's proposed key-exchange algorithm from RSA to DHE, which makes the communicating parties have different views about the key-exchange algorithm, leading to the generation of breakable keys, impersonation of each party to the other, and decryption of the application data.

Several notable attacks illustrate the severity of downgrade vulnerabilities. The POODLE attack, by exploiting the "downgrade dance", forces the client into falling back to SSL 3.0 and then exploits a flaw in block cipher padding to progressively recover sensitive data needing only 256 requests to crack an encrypted message of one byte. The FREAK attack leverages exploited "export-grade" RSA keys, left over from historical export restrictions, which remain trivially breakable with modern computing power. Logjam targets the Diffie-Hellman key exchange by downgrading it to parameters with insufficient cryptographic strength. Similarly, DROWN combined weaknesses in SSLv2 with modern TLS deployments, allowing adversaries to decrypt sessions by exploiting servers that reused the same RSA keys across protocols. Another case of downgrade attack is the "Forward Secrecy rollback" attack, in which the attacker exploits an implementation vulnerability to make the client fall back from Forward Secrecy mode to non Forward Secrecy. A downgrade attack in TLS 1.0 and TLS 1.1 falls under a family of attacks named Security Losses from Obsolete and Truncated Transcript Hashes (SLOTH). This attack is possible due to the use of non collision resistant hash functions like MD5 and SHA-1.

Defending against downgrade attacks primarily requires deprecating insecure protocols and weak cryptographic suites. Today, only TLS 1.2 and TLS 1.3 provide adequate guarantees, while all earlier versions are considered insecure and have

been deprecated by major browsers and platforms. Regularly updating systems, disabling legacy cipher suites, and preventing protocol fallback remain the most effective strategies to significantly reduce the attack surface and preserve the security of encrypted communications on the Internet. [76]

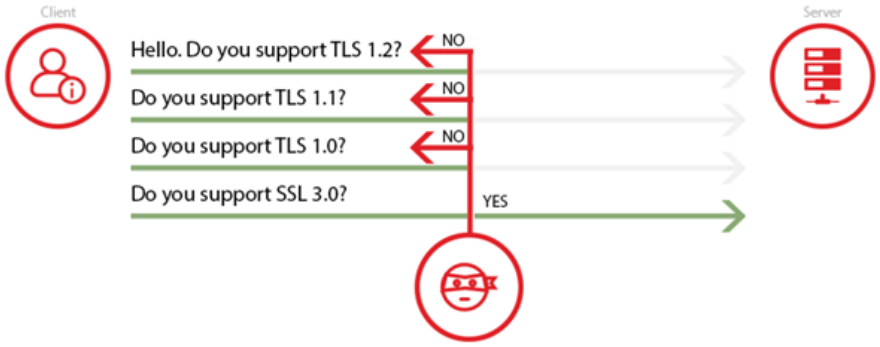

Figure 3.3: TLS Downgrade attack

TCP/IP Spoofing and SYN Flood Attacks, RST injection

The TCP/IP protocol is the most used protocol suite for data communication and it was originally design under the assumption that all the hosts participating in the communication have no malicious intent. This lack of built-in authentication has made it vulnerable to various forms of abuse, among which SYN flooding and RST injection are significant examples.
The TCP SYN flood works by exhausting the TCP connection queue of the host, denying legitimate connection requests. The SYN flood attack exploits a vulnerability of the TCP three-way handshake: a server needs to allocate a large data structure for any incoming SYN packet regardless of its authenticity. Whenever a server receives a SYN packet, it allocates resources and replies with a SYN/ACK, keeping a slot open while waiting for the client's final ACK. During SYN flood attacks, the attacker sends SYN packets with source IP addresses that do not exist or are not in use. Since the source IP addresses used in SYN flood attacks can be nonexistent, the server will not receive confirmation packets (ACKs) for the requests. More and more requests will accumulate and fill up the memory stack. Therefore, no new request, including legitimate requests, can be processed and the services of the system are disabled. Generally, the space for the memory stack allocated by the operating system is small so even a small scale SYN flood attack can be disruptive. The half-open connection remains in the backlog queue, consuming memory and processing capacity.[77]
A common technique used to strengthen SYN flood attacks is IP spoofing. Attackers can forge the source address of IP packets to hide their identity and also to redirect

the blame for attacks. Since the spoofed addresses often do not correspond to real machines, the server never receives the final ACK packets. At the same time, spoofing complicates defensive measure. When attackers inject packets with spoofed source addresses into the Internet, routers forward those packets to their destination just like any other packet without checking the validity of the packets source addresses. This combination of SYN flooding with spoofed IP packets forms one of the most widespread denial-of-service strategies on the Internet.[78]
Another related attack is TCP reset (RST) injection. In this scenario, an adversary

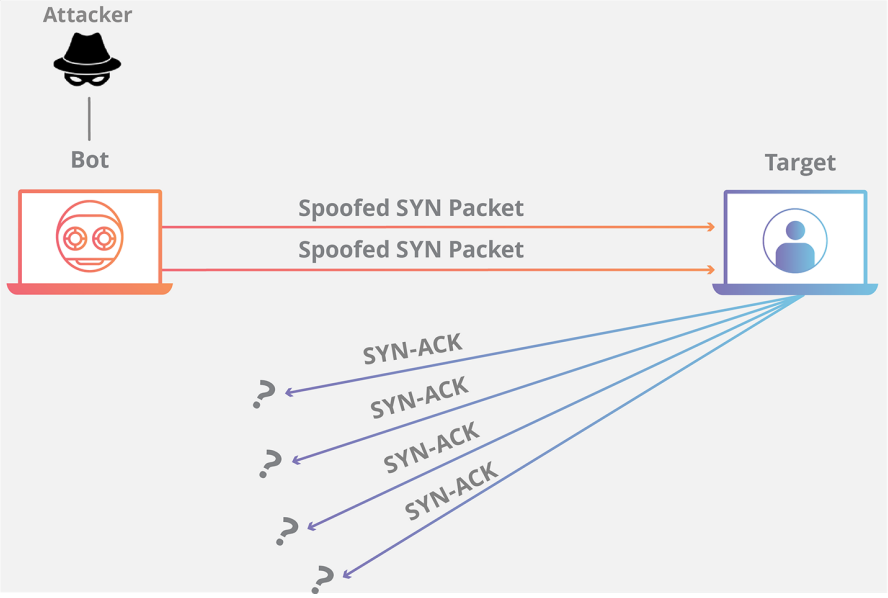

Figure 3.4: SYN flooding with Spoofed IP

injects one or more TCP packets with the RST flag set and sends them to one of the communicating parties that has made a HTTP GET request. An attacker may generate a RST segment that would be acceptable to a TCP receiver by guessing in-window sequence numbers. The intention is to force the premature termination of an active connection by making the recipient believe that the other side has requested a reset. [79]
As specified in [RFC 793], the reset will be considered valid and the connection closed. This so-called blind reset attack highlights how the absence of authentication in TCP allows attackers to manipulate or disrupt ongoing communications.

IP Fragmentation Attack

IP fragmentation was originally introduced to solve compatibility issues between networks with different Maximum Transmission Units (MTU) playing an important

role in ensuring the successful transmission of data. When a packet is larger than the MTU of the transmission medium, it must be split into smaller fragments that are later reassembled by the destination host.
In normal network operations, IP fragments typically arrive in the order in which they are sent but not always. IP supports out-of-order transmission and overlapping fragment reassembly behaviors which are difficult to analyze. Assembling IP fragments can also become complicated since the fragments should be kept in memory until the final fragment is received, in order to complete the assembly of the entire packet. While this mechanism ensures interoperability between heterogeneous networks, it also introduces significant security risks. One of the main concerns lies in the fact that reassembly occurs before the authentication of upper-layer protocols, leaving systems vulnerable to manipulation.
Many routers, firewalls, and IDS systems do not adequately address the security concerns of fragmentation. NIDS sensor does not always reassemble IP fragments in a similar way of the target host, so there will bi a mismatch of the signature of the packet to its database value. Since the payload of an exploit may be spread across multiple fragments, a poorly configured IDS that fail to reassemble them correctly, will allow malicious content to slip through undetected. This means that fragmentation can be used as a method of obfuscating an attack or bypassing detection. One method is the "insertion attack", where an attacker sends packets to an IDS and to a target device that will be accepted by the IDS but rejected by the target. An "evasion attack" is the opposite: packets will be rejected by the IDS but accepted by the target. The idea behind either attack is to send different data streams to each device.[80]
Another possible attack is the IP fragment flood attack. As said before, each fragment must be stored in memory until the entire packet is received; an adversary can send a very large number of incomplete or overlapping fragments, preventing proper reassembly. This forces the target system or the IDS to allocate excessive buffer resources leading to dropped packets, severe performance degradation, or even complete crashes. In this sense, IP fragmentation floods represent a particularly effective variant of Denial-of-Service attacks.[81, 82]

ICMP Attacks

The Internet Control Message Protocol (ICMP) is used to check the health and connectivity of a device, typically through echo requests and echo replies, also known as pings. Although essential for network diagnostics, attackers can exploit ICMP to disrupt services or establish covert communication channels.
ICMP flood is a type of denial-of-service attack, or DoS attack, that was highly popular among attackers in earlier years. In an ICMP flood attack, sometimes referred to as a “ping flood attack,” attackers overload the bandwidth of a targeted

network router or IP address, or overload a device's ability to forward traffic to the next hop downstream by overwhelming it with crafted ICMP packets. Threat actors can configure ICMP packets from real source IPs or using spoofing techniques. As the device tries to respond, all of its resources, memory, processing power, interface rate, bandwidth are consumed and it can no longer serve legitimate requests or users, making it inaccessible to any network devices or next-hop upstream/downstream devices connected to that targeted endpoint.
While it has fallen out of favor as a primary attack vector, it is often used alongside other methods to create highly complex attacks that are more difficult to mitigate. These multi-vector attacks create enormously large and complex DDoS attacks.
ICMP amplification, also known as a Smurf attack, follows the same techniques as traditional ICMP but does not require the attacker to leverage their own hacked resources but exploit open resolvers or other vulnerable infrastructure.
Furthermore, most types of ICMP packets have the capability of carrying data in its payload which makes them suitable for establishing covert channels. The use of ICMP for covert communication presents one big advantage over the classical use of TCP or UDP. ICMP packets use fewer parameters than TCP or UDP. Unlike TCP or UDP, ICMP does not use port numbers, removing an additional filtering parameter that firewalls can use to block suspicious traffic. This simplicity makes ICMP tunneling a widely used technique for covert communication.[83, 84]

DHCP Spoofing

The Dynamic Host Configuration Protocol (DHCP) is a client-server protocol that provides network configuration settings and allocates IP addresses to clients. While essential for simplifying network management, DHCP lacks built-in authentication, which makes it a target for adversarial exploitation.
In a DHCP spoofing attack, the adversaries may redirect network traffic to their own systems by spoofing DHCP traffic and acting as a malicious DHCP server on the victim network. This malicious server responds to client DHCP requests by providing falsified network configurations. Attacker may collect network communications, including passed credentials, especially those sent over insecure, unencrypted channels. For example, the rogue server may list itself as the default gateway or assign adversary-controlled DNS servers to the victim devices. Malware can act as a DHCP server and provide adversary-owned DNS servers to the victimized computers allowing the adversary to achieve the AiTM position.
The attacker can also perform a Denial of Service (DoS) attack by flooding the legitimate DHCP server with requests, exhausting its IP address pool and preventing it from serving legitimate clients causing what is called a DHCP starvation. This manipulation disrupts the normal communication flow, making it easier for the attacker to carry out further malicious activities.[85]

Understanding the potential risks of DHCP Spoofing is crucial for maintaining network security. Key risks associated with DHCP spoofing include data interception, unauthorized network access, network disruption, credential theft, and an increased likelihood of further attacks, such as man-in-the-middle or denial-of-service scenarios. Because of its ability to undermine fundamental trust in network communication, DHCP spoofing represents a significant threat to modern infrastructures.

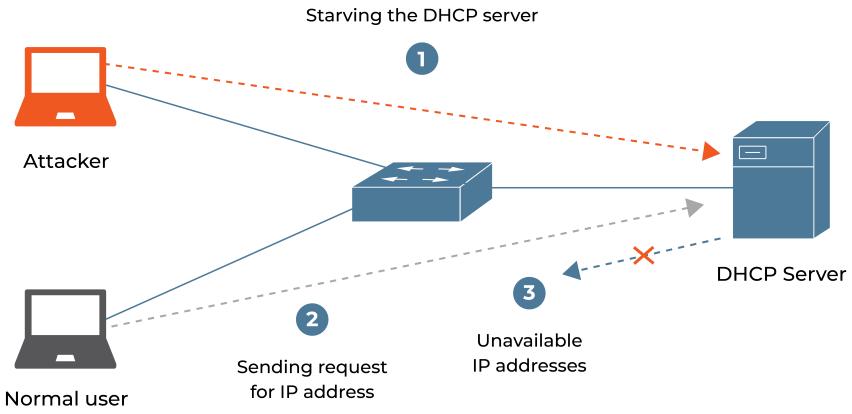

Figure 3.5: DHCP Spoofing and Starvation

ARP Cache Poisoning

The ARP protocol is used to resolve IPv4 addresses to link layer addresses (MAC) within a local network segment where devices communicate with each other by using MACs. If a networked device does not have the link layer address of a particular networked device, it send out a broadcast ARP request to the local network to translate the IP address to a MAC address. The device with the associated IP address replies with its MAC address and the device that made the ARP request will then use as well as store that information in its ARP cache. This cache is sometimes referred to as an ARP table, as the addresses are stored in the form of a table.
ARP, however, is a stateless protocol and lacks any form of authentication. Therefore, devices may wrongly add or update the MAC address of the IP address in their ARP cache. In a local area network (LAN) using ARP, this weakness allows adversaries to exploit the protocol through ARP cache poisoning (or ARP spoofing) positioning themselves between the communication of two or more networked devices. In this attack, adversary may reply with their MAC address, deceiving the victim by making them believe that they are communicating with the intended networked device. This can be done by replying faster than the legitimate device,

or by sending a gratuitous ARP reply that maliciously announce false IP–MAC bindings across the network segment.
Once successful, the adversary positions themselves as a man-in-the-middle (MitM) between communicating devices and can intercept, relay, or modify network traffic. Sensitive information such as credentials, personal data, or financial details can be harvested, particularly when transmitted over insecure or unencrypted protocols.
ARP spoofing has similar effects to other forms of spoofing. Noticeable effects of ARP spoofing can range from none to a full loss of connection to the network if the attacker implements a denial-of-service attack. Whether the effects of the attack are seen depends on the goals of the hacker. If they are only looking to spy on or modify information between hosts, they go unnoticed. If they are looking to instigate a further attack at some point they will become obvious when their end goal is reached.
ARP spoofing can lead to significant disruption, such as total loss of network connectivity or large-scale compromise through ransomware. Regardless of its form, ARP spoofing is considered highly dangerous because it grants attackers unauthorized access to private communications and undermines the trust and reliability of local network infrastructures.[86, 87]

NTP Attacks

The NTP (Network Time Protocol) protocol is designed to allow Internet-connected devices to synchronize their internal clocks and plays an important role in Internet architecture. An NTP amplification attack is a volumetric, reflection-based DDoS attack in which an attacker exploits a functionality of the NTP server to overload a target network or server with an amplified amount of UDP traffic. The target and its surrounding infrastructure become inaccessible to legitimate traffic.
All amplification attacks leverage a disparity in bandwidth cost between an attacker and the targeted web resource. When this disparity is amplified across various requests, the resulting traffic volume can disrupt network infrastructure operations. By sending small queries that generate large responses, the attacker is able to obtain maximum results with little effort. Multiplying this amplification so that every bot in a botnet makes similar requests allows an attacker to remain undetected and reap all the benefits of increased attack traffic.
NTP amplification is very similar to DNS amplification. In this kind of attacks, an adversary reflects and amplify traffic coming from unprotected servers in order to conceal the real origin of the attack and increase its effectiveness. Amplification attacks use devices with low-bandwidth connections that make many small requests for large records. During these requests, the attacker forges the sender's address so that it is that of the designated victim.
NTP amplification is made possible by exploiting the **monlist** command enabled

on some NTP servers. This command is enabled by default on older devices and responds with the last 600 source IP addresses of the requests that have been made to the NTP server. The **monlist** response will be 206 times larger than the initial request. This means that an attacker with 1 GB of Internet traffic can deliver an attack of over 200 gigabytes, a massive increase in the resulting attack traffic. The attacker usually uses a botnet to send UDP packets with spoofed IP addresses to an NTP server that has the monlist command enabled. The spoofed IP address on each packet refers to the real IP address of the victim. Because the size of the response is typically considerably larger than the request, the attacker is able to amplify the volume of traffic directed at the victim. Eventually the surrounding network infrastructure is overloaded with a flood of traffic that causes a denial of service (DoS).
The recommended mitigation is to disable the monlist command on NTP servers or upgrade to the latest version (NTP 4.2.7 or later), which permanently removes this functionality.[88, 89]

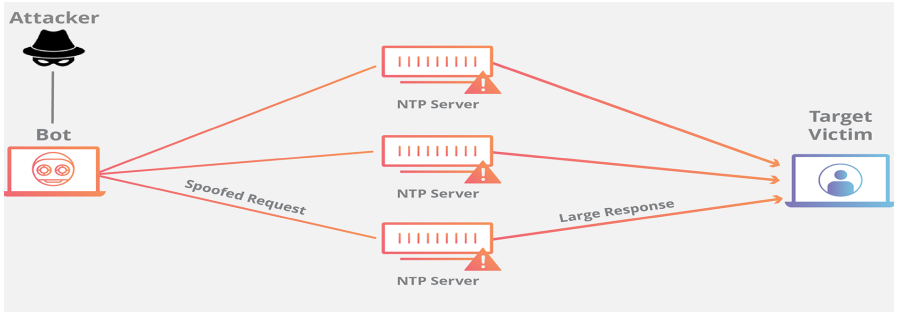

Figure 3.6: NTP Amplification Attack

3.5.2 Countermeasures (Overview)

The network protocol attacks described above exploit weaknesses in fundamental Internet protocols, which were not originally designed with strong security mechanisms. Consequently, defending against these threats requires a layered approach combining technical, organizational, and procedural measures.
On the technical side, cryptographic protocols and secure configuration can mitigate spoofing, cache poisoning, and downgrade attacks. Network defenses such as IDS/IPS, firewalls, and deep packet inspection help detect abnormal traffic and block volumetric threats like SYN floods. Segmentation and access control reduce the impact of ARP or DHCP spoofing, while authentication and secure time synchronization mitigate risks in critical infrastructures. Organizational measures, including timely system updates, patch management, and regular vulnerability assessments, further reduce the attack surface.

Overall, countermeasures should follow a defense-in-depth approach, integrating protocol hardening, cryptography, monitoring, and incident response to provide robust protection against diverse network protocol attacks.

Attack	**Main Countermeasures**
DNS Cache Poisoning	Use DNSSEC, restrict recursive queries, monitor DNS traffic
HTTP Response Splitting	Input validation, sanitization of headers, use modern web frameworks
TLS Downgrade	Enforce TLS 1.3, disable legacy protocols and weak ciphers, certificate pinning
TCP/IP Attacks	SYN cookies, rate limiting, intrusion detection/prevention systems (IDS/IPS)
IP Fragmentation Attack	Packet reassembly checks, anomaly detection, firewalls with fragment inspection
ICMP Attacks	Filter unnecessary ICMP types, rate limiting, monitor ICMP traffic
DHCP Spoofing	DHCP snooping, static IP assignments in sensitive networks, authentication
ARP Cache Poisoning	Dynamic ARP Inspection (DAI), static ARP entries for critical devices, VLAN segmentation
NTP Attacks	Use authenticated NTP, restrict NTP sources, monitor time synchronization anomalies

Table 3.2: Overview of countermeasures against common network protocol attacks

3.6 Real-world case studies

3.6.1 Stuxnet

The Stuxnet worm was discovered in 2010 on Iranian systems and quickly attracted attention due to its advancement and the use of four zero-day vulnerabilities. Unlike typical malware, Stuxnet was not designed to spy and steal information but to sabotage centrifuges in the Natanz power facility, with the goal of disrupting the Iranian nuclear program. It is believed that the attack was carried out through a compromised USB drive, which allowed the worm to penetrate an air-gapped network, meaning that the creators of the worm required a person to deliver the worm and infect the network.
Stuxnet specifically targeted Supervisory Control and Data Acquisition (SCADA) systems and Programmable Logic Controllers (PLCs). It exploited an automatic process from connected USB drivers, a connection to shared printers, and two other vulnerabilities related to privilege escalation. The latter is a computer process that allowed the worm to execute software in computers even when they were on lock-down.
Stuxnet is larger than comparable worms and it was written in several different programming languages with some encrypted components. Stuxnet goal was to infect computers running the Microsoft Windows operating system since it provided a valid certificate for it. Once inside the network, it was able to alter the speed of centrifuge rotors, forcing them to alternate between high and low speeds and causing irreparable damage while simultaneously masking these changes through a rootkit, so that operators believed everything was functioning normally.
Furthermore, the worm was also able to communicate with other infected machines and C&C servers in Denmark and Malaysia in order to update itself and transmit information about what it had found.
It is difficult to know exactly how the malware was created, but there can be no doubt that its development required considerable resources in manpower, time and money. The creators of the worm had extensive knowledge about the Iranian facilities and they must have needed a team of five to ten programmers working full-time for at least six months. Several antivirus experts have asserted that only a state could have done it.
At the international level, Stuxnet had the effect of being a wake up call for states; they realized that they needed to enhance their cybersecurity policies and/or strategies and required a comprehensive cybersecurity strategy that extends to critical infrastructures and the private actors that manage them.
The Stuxnet attack also directly affected the technology sector. Companies that had developed software with vulnerabilities that were exploited to infect and control the computers in Iran were forced to react in order to contain the malware. Microsoft

issued patches to solve the relevant zero-day exploits.
The case of the Stuxnet worm showed that infecting computers by means of a USB drive is effective as a means of targeting air-gapped networks. Therefore, states should particularly focus on computer users and the issue of using unknown USB drives. [90]

3.6.2 Amnesia:33

AMNESIA:33 is the name given by Forescout Research Labs to a set of 33 zero-day vulnerabilities affecting four widely used open-source TCP/IP stacks: uIP, FNET, picoTCP, and Nut/Net, which serve as foundational connectivity components for millions of IoT, OT, networking and IT devices. The TCP/IP stacks affected by AMNESIA:33 can be found in operating systems for embedded devices, systems-on-a-chip, networking equipment, OT devices and a myriad of enterprise and consumer IoT devices. Four of these vulnerabilities are critical and allow for remote code execution. Exploiting them allows attackers to take control of a device, using it either as an entry point into a network, a pivot for lateral movement, a persistence point, or the final target of an attack.
For enterprise organizations, this means they are at increased risk of having their network compromised or having malicious actors undermine their business continuity. For consumers, this means that their IoT devices may be used as part of large attack campaigns, such as botnets, without them being aware.
A key challenge in patching AMNESIA:33 lies in the fact that the affected TCP/IP stacks are open-source and not controlled by a single vendor. As a result, a single vulnerability can propagate silently across multiple code bases, products, companies and development teams. It is estimated that more than 150 vendors and millions of devices are exposed, though the real number may be far higher.
It is difficult to assess the full impact of AMNESIA:33 because the vulnerable stacks are widely spread, highly modular with components, features and settings being present in various combinations and code bases often being forked, and incorporated in undocumented embedded subsystems. For the same reasons, these vulnerabilities tend to be very hard to mitigate.
Many of the vulnerabilities reported within AMNESIA:33 arise from bad software development practices, such as an absence of basic input validation. They relate mostly to memory corruption and can cause denial of service, information leaks, DNS cache poisoning or remote code execution.
A further complicating factor is the absence of a complete Software Bill of Materials (SBOM) for most IoT and OT devices. Because components are often sourced across complex supply chains, neither end users nor vendors can always determine which embedded software libraries are present.
Consequently, there is no way to know precisely how many devices are affected

by AMNESIA:33, and actual device numbers may far exceed current estimates of millions of vulnerable devices in enterprise networks.[91, 92]

3.6.3 Mirai

The Mirai botnet, composed primarily of embedded and IoT devices, stormed the internet in late 2016 when it overwhelmed several high-profile targets with massive distributed denial-of-service (DDoS) attacks. Mirai targeted a variety of IoT and embedded devices including DVRs, IP cameras, routers, and printers. However, the final composition of the botnet was strongly influenced by the market share and design choices of a few consumer electronics manufacturers. According to the source code of Mirai, the foundation of the botnet consists of a Command & Control (CNC) server, a MySQL database server, a Scan Receiver, a Loading server (or Loader), and a DNS server.
An attacker can launch a DDoS attack by sending a dedicated command from a Remote Terminal to the CNC server through Telnet. Simultaneously, the command is recorded historically on the MySQL database server and then the target of the attack is forwarded to the infected IoT devices (or bots). In response, the bots that are currently alive would follow the CNC's order by sending a flood of network packets to the victim server that is targeted.
Additionally, an infected IoT device is capable of exploring the network for other vulnerable IoT devices from a wide range of IP addresses. When a vulnerable device is found the bot report this finding (including the IP address, user credential, type of service, etc.) to the Scan Receiver. The Loader, often collocated with the Scan Receiver, continuously monitored the system output stream (stdout), where the Scan Receiver logged new discoveries.
As soon as new information was captured, the Loader attempted to log in to the vulnerable device and upload the Mirai malware. After the new IoT device is infected, it will then be set up as a new bot and the new bot must register itself with the CNC server. Each infected device retrieved the CNC server's IP address from a hard-coded DNS server, a design decision that also applied to communication with the Scan Receiver. Because of this design, as long as the DNS server is alive the attacker can move all other servers to a different IP address.
Because the Telnet protocol does not use encryption, a forensic investigator can capture the network traffic or recover the previous network packets from the memory dumps of the attacker's terminal. These packets often contained usernames, passwords, and issued commands. Similarly, the Loader stored bot executables for different CPU architectures on disk, enabling forensic analysts to retrieve them from the server's image. Each Mirai executable hard-coded the IP address of the DNS server in order to resolve the CNC and Scan Receiver addresses. This design choice also enabled investigators to repurpose the malware as a probe. By executing

Mirai binaries in a sandbox, researchers could monitor their behavior and collect intelligence about future attacks carried out by the botnet. [93, 94]

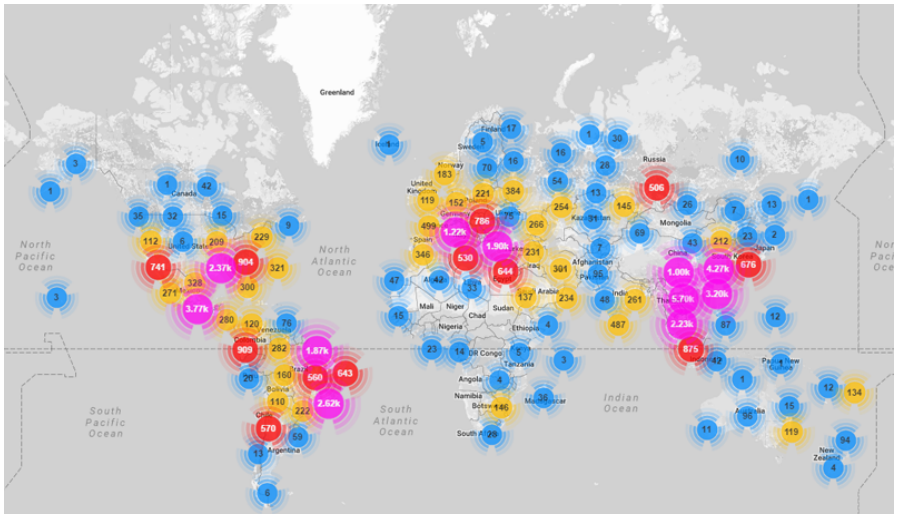

Figure 3.7: Mirai diffusion worldwide

Chapter 4

The Yocto Project in Embedded Security

4.1 Introduction to Yocto: architecture and philosophy

The Yocto Project is an open source collaboration that enable developers to create custom Linux-based systems for embedded products and other targeted environments, regardless of the underlying hardware architecture. The project delivers a flexible collection of tools and a community where developers worldwide can share technologies, software stacks, configurations, and best practices that can be used to generate tailored Linux images for embedded and IOT devices, or any scenario where a specialized Linux distribution is needed.
Over the years, Yocto has become the industry-standard toolkit for building custom embedded Linux distributions and the number one platform to validate new SoC designs, build Board Support Packages (BSPs), generate Software Bill Of Materials (SBOMs) and support Reproducible Builds. The tools allow users to build and support customizations for multiple hardware platforms and software stacks in a maintainable and scalable way. The project provides a standard to deliver hardware and software support promoting collaboration between suppliers and consumers in each sector.
Historically the project grew from, and works with the OpenEmbedded Project, which provides the build system that Yocto uses.
In this work, the Yocto Project is analyzed in depth because it is the framework used in the target embedded device. All subsequent security assessments, customization experiments, and practical evaluations are based on this specific environment. [95]

4.1.1 General Workflow

The Yocto build process follows a series of structured steps:

1. **Configuration:** Developers specify architecture, build policies, patches and system configuration details.
2. **Source Retrieval:** The build system fetches and downloads the source code from where is specified in the code. Standard methods such as tarballs or source code repositories systems such as git can be used.
3. **Preparation:** The sources are extracted into a local workspace, patches are applied and common steps for configuring and compiling the software are executed.
4. **Packaging:** The software is installed into a staging area and packaged using the selected format (deb, rpm, or ipk).
5. **Quality Checks:** Sanity checks and QA verifications run throughout the build process.
6. **Package Feed:** After the binary files are created, a binary package feed is generated.
7. **Image Creation:** The root file system image is assembled for deployment.

This workflow can change and extended depending on what components and tools are actually being used. [95]

4.2 Key Features and Advantages

Some of the main strengths of the Yocto Project include:

- **Wide Industry Adoption:** Supported by semiconductor, operating system, software vendors and by service providers that adopt Linux.
- **Architecture Independence:** Supports Intel, ARM, MIPS, AMD, PPC and other architectures. Most vendors create and supply BSPs that support their hardware. Moreover, the Yocto Project fully supports device emulation using QEMU.
- **Portability of Images:** Images can be moved between architectures without changing development environments. Commercial Linux vendors such as Wind River and Mentor Graphics can provide ongoing support for Yocto-built systems.

- **Flexibility:** Yocto allows the customization of internal Linux distributions across product groups to suit the needs of the products.
- **Ideal for Embedded/IoT:** The Yocto Project allow to add the feature support or packages that are absolutely needed for the device.
- **Comprehensive Toolchain Capabilities:** Standard toolchains are provided, but customization for hardware-specific features or third-party toolchains is supported.
- **Mechanism Rules Over Policy:** Developers are free to set policies based on the design needs because the focus is on the mechanism.
- **Layer Model:** Functionality is grouped into layers, reducing redundancy, simplifying extensions, and isolating changes. Further information is provided below.
- **Partial Builds:** It is possible to build and rebuild individual packages as needed since there is a Shared State Cache (sstate) schema in place.
- **Predictable Release:** Major releases occur on a six-month cycle usually in October and April. The most recent releases support point releases to address common vulnerabilities and exposures.
- **Rich Ecosystem:** Since The Yocto Project is an open source project, a rich community and active contributors are important to support development.
- **Binary Reproducibility:** The Yocto Project allows precise specification of dependencies, achieving a very high level of binary reproducibility.
- **License Manifest:** Yocto generates license manifests for compliance and legal review.

[95]

4.3 Layer Model

The Layer Model is a key Yocto feature for embedded and IoT Linux creation that differentiates the Yocto Project from simpler build systems. Layers are repositories containing related sets of instructions that tell the OpenEmbedded Build System what to do. They support collaboration, sharing, reuse and customization.
Layers in the Yocto Project are hierarchical and can override previous instructions or settings at any time, enabling customization of collaborative layers to fulfill specific product requirements. A modular layer structure allows logical separation of

build information. Consolidating all metadata into a single layer complicates future modifications and reduces reusability. By isolating hardware-specific configurations in dedicated layers, metadata that is common across multiple hardware platforms can be shared through separate layers.
The Yocto Project typically employs multiple layers, each conforming to a conventional structure. This standardization enables BitBake to reliably locate different types of metadata during the build process, ensuring consistency and maintainability. There are many layers working in the Yocto Project development environment. [95]

4.4 Core components

The Yocto Project includes a variety of open-source components and tools, some of which are central to the build process and others that support additional functionality.

4.4.1 Metadata

Metadata is a fundamental element of the Yocto Project since it is used to construct a Linux distribution. It is contained in the files that are used when building an image. In general, Metadata includes recipes, configuration files and other information referring to the build instructions, as well as the data used to control what gets built and how they are built.
Configuration Files contains global and user defined variables as well as hardware configuration information. They are used by the build system to determine what to build and put into the final image to support a particular platform.
Recipes are the most common form of metadata. Recipes contain a list of settings and tasks for building packages which are then used to build the binary image. Recipes specify the source code location, required patches, dependencies on libraries or other recipes, and configuration and compilation options. They are organized within layers.
OpenEmbedded-Core (oe-core) is a collection of foundational recipes, classes and associated files that provide a validated, quality-assured base among many different OpenEmbedded-derived systems, like the Yocto Project.It represents a carefully reduced subset of the original OpenEmbedded repository, ensuring stability and maintainability.
Metadata also encompasses commands and data indicating software versions, sources, and any modifications or patches applied to address bugs or customize functionality for specific use cases.

4.4.2 Poky

Poky serves as a reference embedded distribution and a reference test configuration created to provide a base level functional distribution which can then be customize. Poky is used to validate Yocto Project components, and as a starting point for users to download Yocto. Poky is an integration layer that stands on top of oe-core.

4.4.3 Build System – "Bitbake"

Bitbake is the core build engine of Yocto which parses recipes and configuration data. It creates a dependency tree to order the compilation, schedules the processing of the included code, and finally, executes the building of the specified, custom Linux image.
BitBake recipes include all the package dependencies, source code locations, configuration, compilation, build, install and remove instructions. Recipes also use standard variables to store the metadata for the package. Related recipes are consolidated into a layer. During the build, process dependencies are tracked and native or cross-compilation of the package is performed.
For cross-builds, it can automatically generate an Extensible SDK (eSDK), which is a custom development toolkit allowing application developers to integrate their code and libraries into the build environment, making them accessible to other developers working on the same image.

4.5 Security in the Yocto Project

Security is a process, not a product, and thus at any time, a number of security issues may be impacting Poky and OE-Core. It is a responsibility of the maintainers, users, contributors and anyone interested in the issues to investigate and possibly fix them by updating software components to newer versions or by applying patches to address them. [96]

4.5.1 CVE management and package updates

The Yocto Project provides an infrastructure for monitoring and addressing security vulnerabilities by integration with the public Common Vulnerabilities and Exposures (CVE) database.
The project maintains an updated list of known issues for the software packages included in Poky and OE-Core. It also tracks the number of unpatched vulnerabilities as well as the progress of applied fixes. This monitoring is performed for both the development branch and all actively supported releases.
Not all entries in the CVE database are relevant: some are too old or there are

some incomplete records so they do not explicitly affect Yocto Project components. When CVE checks are enabled, the build system attempts to map each compiled component against the CVE database. At the end of the process, it produces a detailed report at both the recipe and image level.
This report generally contain:

- Basic information on the software component like names and versions.
- Details about the CVE such as description and its reference to the National Vulnerability Database (NVD).
- A list about all the CVEs potentially affecting a specific software component.
- The status of each CVE which can be:
  - **Patched:** means there is a patch file that address the vulnerability and it has been integrated in the newest update.
  - **Unpatched:** means there are no patches to solve the vulnerability and further investigation is required.
  - **Ignored:** means that, after careful analysis, it has been decided that the vulnerability is irrelevant, for example it may affect the other operating systems.

By default, the connection to the CVE database is performed without using an API key, which limits the request rate and can result in slower retrieval of vulnerability data.
Once the CVE check is enabled, the build process produces detailed reports for each compiled component, highlighting potential vulnerabilities and their status.
A key challenge in this mapping process lies in the correct identification of software components within the CVE database. When the correspondence between a component and the recorded vulnerabilities is inaccurate, two main issues may arise. On one hand, false positives can occur, where vulnerabilities are incorrectly associated with a component that is not actually affected. On the other hand, false negatives may emerge, where relevant vulnerabilities are deemed not important due to mismatches in component naming or versioning.
To mitigate these issues, it is necessary to refine the mapping by ensuring a more precise alignment between the component identifiers used in the build environment and those recognized by the database, possibly including information such as vendor or release version when relevant.
Considering that not all the CVEs in the NVD are complete or free of errors, it is preferable to provide corrections directly to the database maintainers when this problems are found, rather than attempting local workarounds that may create inconsistencies across projects. Feedback to NVD about CVE entries can be

provided through the NVD contact form.
When evaluating vulnerabilities, it is recommended to follow a structured process, which should include:

1. Consulting the latest data from the CVE database.
2. Reviewing how upstream developers of the affected software have addressed the issue (e.g., applied patches or released updated versions).
3. Comparing the adopted solutions with those of other Linux distributions.
4. Following security notices released by other Linux Distribution.
5. Following specialized open-source security mailing lists to obtain early warnings and insights into vulnerabilities and their fixes.

[96]

4.5.2 Security tools and configurations

Securing a customized image requires a combination of best practices, tools, and design choices. Since every device has its own risk profile and operational environment, there is no universal recipe for security. Not all situations are identical when it comes to making an image secure. Within the Yocto Project ecosystem, several mechanisms are available to strengthen the security of the generated images in ways that are specific to this build system. Security needs inevitably vary depending on the device and its intended use, but Yocto provides configurable options and tools that can be adapted to different contexts.
One of the most relevant practice is to maintain Poky and OE-Core in line with their upstream stable or long-term support (LTS) branches. Regular synchronization ensures that the system benefits from the continuous application of bug fixes and security patches delivered by the community.
Yocto also allows the control of debugging features and services enabled in the final image. During development, debugging functionality may be necessary, but before deployment it should be disabled or removed to prevent the introduction of vulnerabilities. The same applies to unnecessary services or software packages, which can be excluded at the image generation stage, thereby reducing the attack surface.
Finally, when supported by the target hardware, Yocto provides configurations to enable secure boot mechanisms, ensuring that only software images verified through cryptographic signatures can be executed. This represents an additional layer of protection against unauthorized modifications of the system software. It is also important to remove any software from the image if needed.

Yocto does not guarantee security by default, but it gives developers the means to build it into every layer of their system. [95]

4.5.3 Security-related meta-layers

The Yocto Project offers specific tools to enhance the security of generated images through the meta-security layer available in its official source repositories. This layer provides a comprehensive set of utilities for kernel hardening, runtime protection, integrity verification, intrusion detection, and vulnerability scanning. It also includes support for sandboxing and other mechanisms to mitigate potential security risks.
The meta-security layer must be explicitly added to the build environment configuration. Meta-security depends on several other layers and core packages, including OE-Core. It effectively acts as a toolbox for security, providing pre-configured tools and utilities designed to support secure embedded and IoT systems. [95]

4.6 Yocto's limitations for secure development

While the Yocto Project provides a powerful framework for building custom embedded Linux systems, it also presents several challenges that can affect secure development.
One major limitation is the steep learning curve. The framework's complexity requires significant time and experience to master, particularly when managing multiple layers, custom recipes, and cross-compilation toolchains.
Another important aspect is the need for continuous maintenance. Security relies on keeping Poky, OE-Core, and any security-related layers up to date with the latest stable or LTS releases. Failing to do so can leave the system exposed to known vulnerabilities.
Additionally, the effectiveness of security measures can be compromised by misconfigurations. Errors in enabling necessary features or layers, or improper configuration of distribution settings, may prevent security tools from functioning correctly or leave unnecessary services active in the image.
Finally, integrating security measures into the build process requires careful planning. Even with tools like CVE-check or meta-security, developers must understand how to interpret reports and apply fixes appropriately to ensure a genuinely secure system.

Chapter 5

Vulnerability Assessment: Case Study

5.1 Purpose and scope of the assessment

The purpose of this assessment is to evaluate the security implementation of the IoT gateway device, which is deployed in an industrial environment for environmental monitoring, such as temperature checks, and data logging.
Each feature has been analyzed to determine how it is configured with respect to the specific technologies chosen for this device. Security controls are then reviewed against industry best practices to verify that they provide the intended protection. Both static and dynamic analyses, including practical tests, were conducted to identify potential vulnerabilities.
The results of these assessments are summarized in this report, while any vulnerabilities identified are detailed in Chapter 7 along with recommended mitigations. When applicable, penetration testing activities are presented in Chapter 6.

5.2 Device overview and initial configuration

The device on which the vulnerability assessments has been done is the IOT-GATE-iMX8, also known as **compulab**. It is an industrial IoT Edge Gateway, ideal for industrial control and monitoring, that provides extensive wired and wireless connectivity ensuring reliability, flexibility, and long-term availability.
The CPU is the NXP i.MX8M Mini CPU, a quad-core Cortex-A53 processor with a Cortex-M4 co-processor, and supports from 1 GB to 4GB LPDDR4 RAM. Primary and secondary storage up to 128GB eMMC flash is available. Primary storage is soldered on-board while the secondary storage is an optional module. Connection

can be established with numerous option such as LTE modem (Simcom SIM7600G), WiFi 802.11ax, Bluetooth 5.1, dual Ethernet ports (1 Gbps and 100 Mbps), as well as multiple serial interfaces, RS485/RS232, CAN-FD, and 3 USB-2 ports.
The device supports industrial protocols such as Modbus RTU, Modbus TCP, and MQTT. Regarding security, secure boot is implemented with i.MX8M Mini HAB module and features like TPM 2.0 is supported.
Expansion capabilities are provided through custom I/O add-on boards, enabling CAN, RS485, RS232, digital I/O, and 4–20mA inputs. The device is optimized for industrial use, featuring a fanless design in aluminum, rugged housing for harsh environments, an operating temperature range from -40°C to 80°C and a wide input voltage support: from 8V to 36V. Moreover, it supports DIN-rail and wall mounting and operates reliably in 24/7 deployments, offering a 5-year warranty and 15-year availability.
From a software perspective, the gateway is fully compatible with Debian Linux and the Yocto Project BSP, and supports U-Boot, Docker, and cloud platforms like Microsoft Azure IoT. It also enables OTA updates via Mender, ensuring lifecycle management. The combination of all of these features makes the IOT-GATE-iMX8 is a versatile solution for demanding industrial IoT applications. [97]

5.3 Security technologies implemented

5.3.1 Secure Boot

To enable Secure Boot on i.MX8 platforms, the firmware images must be signed using NXP's Code Signing Tool (CST). The process begins with obtaining the CST package and generating a PKI tree through the use of a script. By executing this script, four Super Root Keys (SRKs) are generated and the CA flag is enabled.
Once the PKI is established, the SRK table and fuse table are created and the four SRK certificates are aggregated into binary files that will later be programmed into the device's fuses. The IoT-gate-imx8 fuses are one-time programmable so no

mistakes can be made when executing the u-boot fuse commands.
To fully enable Secure Boot, the generated SRK hash must be programmed into the OTP fuses of the device, and the Secure Configuration fuse bit need to be set. This bit instructs the ROM code to enforce signature verification at every boot so that it will only load and execute images that include a valid Command Sequence File (CSF) signed with one of the provisioned SRKs.
This procedure establishes a chain of trust, ensuring that only authenticated software images can be executed by the device. Secure Boot is then activated in U-Boot. The bootloader can start in High Assurance Boot (HAB) mode by validating the digital signatures before uploading the firmware or the kernel. The **blob** command is enabled in U-boot to manage the signed Command Sequence File needed by the HAB process.
This fuse configuration, together with the HAB logic integrated in U-Boot, effectively activates Secure Boot and establishes the immutable hardware root of trust.
The assessment was conducted through three complementary steps:

- **Configuration review:** The secure-boot configurations were examined to verify the presence of the right configuration parameter, ensuring that the bootloader was compiled with HAB support and the appropriate cryptographic algorithms.
- **Process validation:** The procedures for key generation and image signing were reviewed to confirm that NXP's recommended methodologies were properly implemented.
- **Functional Testing:** The device was tested by trying to boot unsigned images or improperly signed binaries. The test succeeded, demonstrating that it was impossible to boot the device from the corrupted images.

Through this combined approach, the Secure Boot implementation was found to be correct, and no vulnerabilities were identified in the chain of trust up to this point.

5.3.2 dm-verity

After the Secure Boot checks succeed, the kernel is loaded and executed, and dm-verity makes sure that the kernel can only access an authentic and immutable file system by checking the integrity at run-time of the file system root or other protected partition.
One way to verify a block device, which is the storage level underlying the file system, s to directly hash its contents and compare them with a stored value. However, attempting to verify an entire block device like this can take a considerable amount of time and consume significant device power, making the device unusable.
The dm-verity feature allows to examine a single block device only when accessed,

and determine whether it matches the expected configuration. To achieve this, it uses a cryptographic hash tree. When the hash is verified latency is introduced but it is considered minimal considering the already costly operation of reading a block.
This functionality ensures that the device at the time of check is in the same state as the one it was in the last active session and helps detect persistent rootkit that can escalate privileges and compromise the device.
To verify the signature of the hash and confirm that the device's system partition is protected and unmodified a key included in the boot partition is used. Since the hash values are stored in a page tree, only the first-level root hash needs to be trusted to verify the rest of the tree. The ability to modify any block would be equivalent to breaking the cryptographic hash. If the verification fails, the device generates an I/O error indicating that the block cannot be read [98].
The general algorithm for constructing the hash tree is as follows:

1. Choose a random salt value in hex encoding.
2. Decompress the system image into blocks of 4K.
3. For each block, compute the salted SHA-256 hash of the block.
4. Concatenate these hashes to form a level.
5. If the blocks are not a multiple of 4K, pad the level with zeros until it reaches the block boundary.
6. Link the level into the hash tree.
7. Repeat steps 2–6 using the previous level as the input for the next one, until a single hash remains.

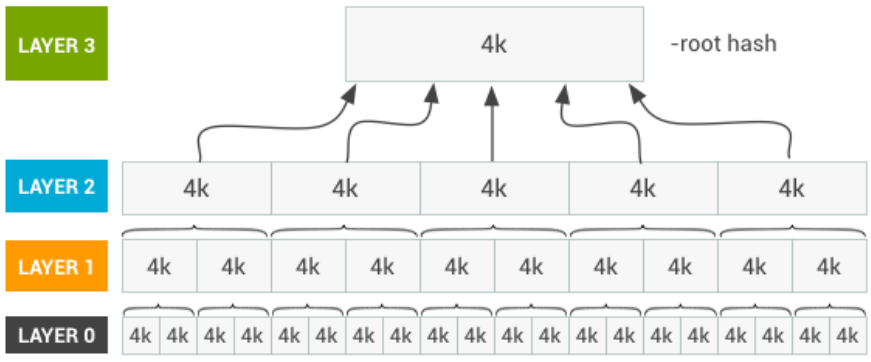

Figure 5.1: dm-verity hash table

The root hash resulting, together with the used salt, is then used to create the dm-verity mapping table identifying the block device for the kernel and the location of the hash in the tree structure.
The dm-verity table is then signed and this signature is the first one validated. The dm-verity table and its signature are grouped together into the Verity metadata. The entire metadata block is version-controlled, so it can be extended if necessary, for example to add a second type of signature or change the ordering [98].
To perform practical tests and verify that no vulnerabilities are present in the implementation of dm-verity the first step is to ensure that the kernel is actually using dm-verity during the boot phase. A first approach is to check with the

```
dmesg | grep verity
```

that will show the log of the boot phase and if dm-verity is enabled specific messages will appear indicating that the kernel has initialized dm-verity and the protected partition are verified. After this, the

```
dmsetup ls --target verity
```

command shows which partition, there can be more than one in a device, is protected with dm-verity, and the

```
dmsetup table <partition_name>
```

will then shows the verity table for that partition. Thanks to the result of this command it is possible to know all the configuration parameters and make sure that they are following the best practice, such as the 4K block size mention beforehand. Moreover this information can be used to perform the next step of the test done with the following command

```
veritysetup verify \
  --hash-offset=$((HASH_BLOCKS * BLOCK_SIZE)) \  # Offset where hash blocks start
  --data-blocks=${DATA_BLOCKS} \                 # Total blocks to verify
  --data-block-size=${BLOCK_SIZE} \              # Size of each data block
  --hash-block-size=${BLOCK_SIZE} \              # Size of each hash block
  --hash-alg=sha256 \                            # Hash algorithm
  --fec-device=${ROOTFS_DEV} \                   # Device storing FEC data
  --fec-offset=$((FEC_START * BLOCK_SIZE)) \     # Offset of first FEC block
  --fec-roots=${FEC_ROOTS} \                     # Number of FEC root blocks
  --root-hash=${ROOT_HASH} \                     # Expected root hash
  ${ROOTFS_DEV} ${ROOTFS_DEV}                    # Data device, Hash device
```

Although usually the root-hash is just another parameter discovered thanks to the **dmsetup table** command, there are cases, such as this, where the root hash is signed and the table reports the descriptor of the key used to verify its signature,

not the root hash itself. This design choice reflects the security model in which the root hash must be validated against a trusted key stored in the kernel keyring. However, both the original root file system image and the build tools used to generate it are accessible in the development environment allowing to perform the **verify** command nonetheless. The data device and the hash device are the same since they reside on the same partition, as it is common for embedded images.
This tool recomputes the entire hash tree of the root file system and compares it against the trusted root hash stored in the boot partition. Unlike the runtime integrity checks that dm-verity carries out block by block during normal operation, this command forces a full traversal of the hash tree. Each data block of the root file system is read, its hash is recomputed, and then compared against the corresponding value in the stored hash tree up to the root hash. If FEC (Forward correction errors) is present, the procedure uses the related data to correct some compromised blocks before deeming the verification a failure. The result of this offline check was successful meaning that the file system in use is both authentic and immutable.
Later, a final controlled tampering test was performed. One important feature of using dm-verity is that the image created with it is always read-only. This is an inherent consequence of the function: if a block is modified, the check will fail and the file system will not start. If a root file system is mounted, it is impossible to modify it directly.
To perform the following test, a modifiable copy of the device created to verify that the verity implementation works correctly. After the copied imaged was connected to a loop device, a block was modified and dm-verity was then activated. In this way, it was possible to rerun the verify command and see if it fails. The command indeed failed, generating I/O errors in the logs.
The performed assessment confirmed that dm-verity is correctly implemented on the device and no vulnerabilities were discovered. The mechanism provides strong runtime integrity verification, rollback protection and secure update capabilities.

5.3.3 dm-crypt, dm-integrity and secure partitioning

While dm-verity provides runtime integrity verification for the read-only root file system, other parts of the system, such as secure storage, need protection for both confidentiality and integrity. In our setup, this is achieved using dm-crypt for encryption and dm-integrity for block-level integrity and confidentiality protection.
dm-crypt is a Linux kernel device-mapper crypto target that provides block-level encryption. It can encrypt whole disks, partitions, software RAID volumes, logical volumes, as well as files. It appears as a block device, and can be used for file systems, swap, or as an LVM physical volume.
It ensures only the confidentiality of data stored on a device by transparently

encrypting and decrypting blocks as they are written and read. It makes it impossible to read data block without the correct key, which means that even if physical access is obtained the data stay unreadable.
However if a block is modified or corrupted, it will still be decrypted if the correct key is available, potentially resulting in corrupted data. To solve this problem dm-crypt is usually combined with dm-integrity [99].
dm-integrity is a Linux kernel device-mapper target that provides block-level integrity protection. It computes and stores a checksum or cryptographic tag for each data block, allowing detection of accidental corruption or intentional tampering [100].
Inside the compulab, a data partition, separated from the root file system, is initialized with dm-integrity using the following command:

```
integritysetup format ${DATA_DEVICE} \     # Device to protect with dm-integrity
    --tag-size ${TAG_SIZE} \               # Integrity tag size per block
    --integrity ${INTEGRITY_ALGO} \        # Algorithm used for integrity
    --integrity-key-file ${INTEGRITY_KEY_FILE} \  # Key file for integrity checks
    --sector-size ${SECTOR_SIZE}           # Size of each block/sector in bytes
```

By analyzing the code, it is possible to confirm that the parameters have been selected according to cybersecurity best practices while also considering the constraints of the embedded device. For example, the integrity algorithm is HMAC-SHA256, a keyed-hash algorithm widely recommended, and the tag size is chosen large enough to provide sufficient security against tampering but not too large to produce overhead in memory and storage.
Some tests have been performed to ensure the correct implementation of dm-integrity. One of these consisted in writing some data into the volume and then reading them again to make sure no errors appeared. Later, a block was modified manually on the partition, and the system correctly detected the corruption and returned an error.
Inside the data partition protected with integrity a secure storage is created thanks to dm-crypt. This secure vault is encrypted using LUKS (Linux Unified Key Setup) and is intended to store sensitive information, such as keys, SSH credentials and everything that is deemed to kept secret.
A file acting as a block device is created and formatted as a LUKS volume. LUKS is a standard provides block-level encryption and centralized key management. Using it in collaboration with dm-crypt allow to inherit the characteristic of LUKS, such as secure multiple keyslots to open the vault with up to 8 different keys, strong encryption algorithms, and protection against brute-force attacks thanks to the PBKDF2 applied to the passphrase.
In the system just one keyslot is used to reduce the attack surface since there is no need to have multiple users and it is easier for an embedded device to have just one random key. This key itself is securely stored, after being generated randomly

by the CAAM module, inside the CAAM itself, which will be discussed later.
After creating the volume, the following command opens it.

```
cryptsetup luksOpen ${DATA_DIR}/.cryptvault secure --key-file /tmp/key
```

The key is provided as a file, previously generated, instead of using it in plain-text or via a passphrase. In this way the key never actually leave the CAAM and never gets written persistently on disk.
Doing so allows a file system to be created on the volume and permissions to be set to restrict read/write access only to authorized users. This approach ensures that even if the device is physically accessed by an attacker, the sensitive data inside the vault remains confidential.
To safely test the implementation of dm-crypt in the secure vault, a temporary copy of the encrypted volume has been created. This allows experimenting with incorrect keys or attempting to open the vault without risking the production data. When using a wrong key, **cryptsetup** failed to open the volume as expected, as it did attempting to open it without any key. Additionally, trying to read the volume while it was not open failed, as expected.
Moreover, trying to mount the partition as a non-root user was tested. It was impossible to mount the secure vault or even access it because the partition is not visible to unauthorized users.
The implementation of dm-crypt and dm-integrity in the system aligns with several key cybersecurity best practices. dm-crypt ensures the confidentiality of sensitive information stored, while dm-integrity guarantees that any modifications to the data partition are detected, providing tamper-evident protection. Together with dm-verity this creates a multiple layers defense-in-depth approach. Access controls enforce the principle of least privilege, as only authorized users can access the encrypted vault. Hardware-assisted key management via CAAM protects encryption keys against exposure. Overall, these measures collectively safeguard both system and user data at rest.

5.3.4 Access control and authentication

Static code analysis and practical tests were performed to verify that access control and authentication mechanisms are correctly enforced. This section summarizes the observations.
The IoT gateway enforces a robust access control model regarding both system and user accounts. Standard system accounts such as root, daemon, bin, and sys are present, along with service-specific users with no duplicate UID 0 accounts or accounts with empty passwords.
Privilege management relies on standard groups. No user belongs to the root or wheel groups, and no group has sudo privileges by default. Authorized users

have sudo access restricted to specific commands necessary for their role. Service accounts (edgex, redis, sshd, etc.) are configured with non-interactive shells (/bin/-false or /bin/nologin) to prevent interactive logins.
Remote administrative access is provided exclusively via SSH with public key authentication; password-based authentication is disabled for administrative accounts. Public keys are validated against entries in the authorized_keys file and verified via the SSH handshake. Session controls are enforced through ClientAliveInterval and ClientAliveCountMax, terminating inactive sessions to reduce exposure. Both Ed25519 and RSA-4096 keys are accepted according to current standards.
Mandatory Access Control (MAC) is implemented via SELinux in enforcing mode. Dashboard access uses username/password authentication over TLS. Password hashes are securely stored using the Argon2 algorithm with unique salts. Password change is enforced on first login for all users except the highest level one, and programmatic password reset is available through the API.
Practical tests were conducted to verify the robustness of authentication and access control. This test included:

- Attempting SSH logins using invalid public keys and verifying that access was denied.

```
ssh -i invalid_key user@iot-gateway-ip
```

- Attempting password-based SSH logins to ensure they are correctly disabled for administrative accounts.

- Verifying session timeout enforcement by leaving SSH sessions idle beyond ClientAliveInterval limits.

- Checking file permissions and access rights for `.ssh` directories and `authorized_keys` files to confirm only root access.

- Testing dashboard login with valid and invalid credentials to verify proper authentication handling.

```
curl -k -X POST https://iot-gateway.example.com/api/login \
    -H "Content-Type: application/json" \
    -d '{"username":"valid_user","password":"valid_password"}'
curl -k -X POST https://iot-gateway.example.com/api/login \
    -H "Content-Type: application/json" \
    -d '{"username":"valid_user","password":"wrong_password"}'
```

- Attempting password change on first login and API-based password reset to ensure enforcement and cryptographic verification with Argon2 hashes.

These tests confirmed that the access control mechanisms work as intended for regular operation. However, some problems and vulnerabilities regarding the management of the dashboard password have been found.

5.3.5 Overlay filesystem and read-only root

In order to maintain a read-only root filesystem while still allowing runtime modifications, the system employs an overlay filesystem. Writable layers are created on top of specific directories, such as **/etc** and **/var**, and are stored in the integrity-protected data partition managed by dm-integrity. This approach allows the system to handle temporary or user-generated data without compromising the security and integrity of the core root filesystem.
Of course, it is important that access control to these layers is correctly enforced. In this system, only root has complete access to the files inside the overlay layers. To mitigate potential risks in case an authorized user escalates privileges, it is also recommended to avoid placing critical configuration files inside these layers so that, even if root access is obtained, the potential damage is limited. This means leaving out or restricting permission to those configuration files that can disrupt the functionality of the compulab if changed or that can be used to perform some attacks. For example, if an attacker becomes root and accesses the configuration files of Redis, they could disable authentication, change passwords or read sensitive information stored in Redis. Moreover, directories such as shadow could be used to create hidden accounts allowing the attacker to later obtain complete access to the system without needing to perform a privilege escalation each time.
The overlay file system was analyzed to ensure that permissions are set according to the principle of least privilege. Some directories in the overlay file system remain writable to allow services to create configuration files or runtime data at startup. However, once created, critical files are immediately set to read-only, even if they reside in an overlay directory. By inspecting the overlay layers and the requirements of each service, it was observed that several directories could safely remain read-only, as no service or user needs to modify their contents during normal operation. This approach reduces the attack surface and ensures that critical configuration files cannot be tampered with, even in the unlikely event of a privilege escalation.

5.3.6 CAAM: integrated components in the project

The acronym CAAM stands for Cryptographic Accelerator and Assurance Module and it is hardware that can be found on many i.MX NXP devices such as the one under test. CAAM is an multi-function accelerator that supports the most common cryptographic functions in many security protocols. This includes AES128, AES256, DES, 3DES, SHA1, SHA224, SHA256, RSA-4096 and a random number generator

with a true entropic seed. CAAM includes a DMA engine that is descriptor based to reduce processor-accelerator interaction [101, 102].
In addition to the performance gained with using the CAAM for cryptographic operations, the application can also improve security by using encrypted keys and secure memory partitions with the CAAM. When describing the keys and blobs used with the CAAM, the term black keys and blobs are used to describe when the key has been encrypted by the hardware. The key generation is automated using *ssh-keygen* and *openssl*, and all entropy is sourced directly from *dev/random* and the hardware random number generator provided by the CAAM ensuring secure key material.
To sum up, CAAM isolates cryptographic operations from the main CPU, protecting keys and sensitive operations from software-based attacks and also manages hardware-backed key storage, secure key derivation, and provides a high-quality hardware random number generator. This hardware layer isolation and security strengthen the system's resilience against a broad spectrum of attacks, including those targeting cryptographic operations.
Testing has been performed trough code review and some on device test. The code review confirmed several best practice regarding the use of the module. The script initializes the system by generating a local key and IV, encrypting the integrity key using

```
openssl enc -aes-256-cbc
```

with the generated key/IV, and creating a CAAM black key blob via

```
caam-keygen create blackkey ecb -t $sslKey
```

The final artifacts are stored as base64-encoded blobs in the boot environment which is furthermore protected from unwanted access. Temporary files, including `/tmp/key`, `/tmp/iv`, and `/tmp/keyRandom`, intermediete plaintext files and hex dumps are removed after use. They are created and used entirely within the `initramfs`, residing in volatile memory. They never reach persistent storage, which effectively eliminates the risk of forensic recovery from disk.
On-device testing validated that the CAAM-produced blob can be unwrapped by the CAAM and used for subsequent cryptographic operations. The encrypted integrity key is handled entirely within the initramfs during setup by integritysetup and cryptsetup, remaining in volatile memory and never written to persistent storage. Therefore, there is no practical exposure window on disk.
The CAAM-related directory is located on an encrypted partition protected with dm-crypt and is accessible only to the root account with elevated privileges, providing an additional layer of storage protection.
These outcomes indicate correct use of CAAM for hardware-backed key protection and secure storage in this device context, assuming the underlying dm-crypt and

boot-environment protections are enforced. The script's behavior and the observed runtime results align with best practices for hardware-rooted key management and entropy sourcing.
Some not critical vulnerabilities were found and are documented in chapter 7.

5.3.7 firewalld

firewalld provides a dynamically managed firewall with support for network and firewall zones that define the trust level of network connections or interfaces. It supports IPv4, IPv6, Ethernet bridges, IP sets and support IPv4 and IPv6 NAT. Changes can be done immediately in the runtime environment without restarting the firewall daemon.
The firewall exposes a D-Bus interface, which is used to add, modify, or remove rules, and thanks to offline tools like `firewall-offline-cmd` alter configuration files directly. Runtime configuration is temporary and lost upon restart or reload, whereas permanent configuration is applied consistently. With the runtime environment it is possible to use this phase for settings that should only be active for a limited amount of time. If the runtime configuration has been used for evaluation, and it is complete and working, then it is possible to save this configuration to the permanent environment.

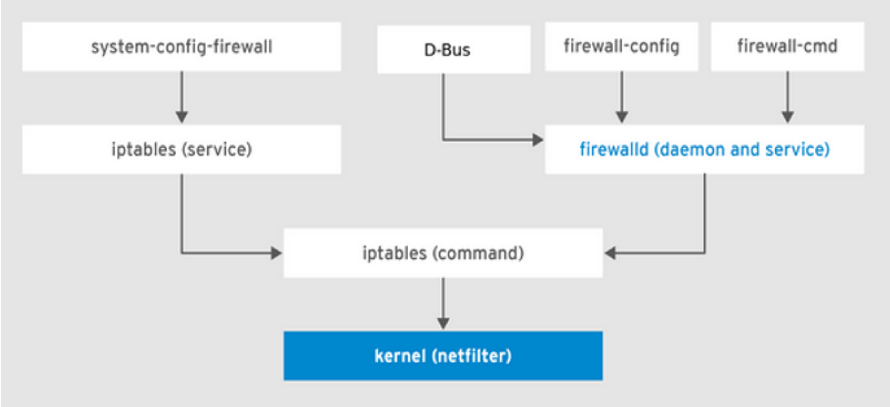

Figure 5.2: firewalld configuration

Firewalld simplifies traffic management thanks to the concept of zones and services. Zones are predefined sets of rules that assign trust levels to interfaces or sources. The allowed traffic depends on the network to which the computer is connected and the security level assigned to that network. Firewall services are predefined rules that cover all necessary settings to allow incoming traffic for a specific application and are applied within a zone. A connection can belong to only

one zone, but a zone can be used for multiple network connections [103, 104].
By default, firewalld blocks all incoming connections except those explicitly allowed, ensuring a secure baseline. The default zone is **public**, which permits only selected connections such as SSH or HTTPS. The default zone can be changed. Other zones provide different levels of trust and access:

- `trusted`: All incoming connections are accepted. Suitable for VPN interfaces or fully trusted networks, but overly permissive if assigned to public interfaces.
- `internal`, `home`, `work`: Intended for LAN environments, allow selected incoming traffic depending on service definitions.
- `external`: Used for gateways or NAT-enabled interfaces, with masquerading enabled.
- `dmz`: (Demilitarized Zone) Isolated network zone for publicly accessible services with limited internal access.
- `block`: Rejects all incoming connections with ICMP host-prohibited messages; only outbound connections initiated by the host are allowed.
- `drop`: Drops all incoming packets silently; only outbound connections initiated by the host are allowed.

The firewalld architecture consists of a core layer, which handles configuration backend like `iptables`, `ip6tables`, `ebtables`, and `ipset`, and the D-Bus layer that provides the main interface for online management of the firewall. This design allows dynamic adjustments, integration with services, and flexible rule definitions, while also supporting automatic kernel module loading, logging, whitelisting, and timed rules. Overall, firewalld enables dynamic, host-based network security that follows best practices for segmentation and controlled access in complex network environments.
By checking the status of the service on the device it is confirmed that firewalld is active and running as a background daemon and the configuration follows the zone and service models explained before. There are rules in place that define the level of trust making sure that only allowed traffic is permitted. With SELinux in enforcing mode, firewalld cannot send certain D-Bus messages. The SELinux policy deliberately restricts the ability of the firewalld process domain to send certain D-Bus messages to processes running under the unconfined domain. This behavior is designed to enforce strict process isolation and minimize the risk of privilege escalation through D-Bus communication. As a result, firewall-cmd operations may time out in enforcing mode, while in permissive mode they succeed, allowing to test the correct configuration of the firewall on the device. Naturally, these operations require root privileges. The runtime configuration was verified using

```
firewall-cmd --get-active-zones
firewall-cmd --zone=trusted --list-all
```

The primary active zone is `trusted`, assigned to the VPN interface (`tun0`). This zone accepts all incoming and outgoing traffic by default, without restrictions, masquerading is enabled, and a port forwarding rule is configured. The public zone is configured and only allows specific service (ssh) and port to be open for TCP and UDP connections but no interface is assigned to these rules are currently not enforced since the only active zone is the trusted one and only that connection can be established. Other network interfaces such as `eth0` were unassigned, so the interface is handled by the default zone, `public`.
The system has no global IPv6 addresses, only '::1' for loopback and 'fe80::/64' for link-local communication and *ip6tables* modules are missing, meaning the IPv6 firewall is disabled.
`ipset` is not available, which means advanced list-based filtering (e.g., blacklists/whitelists) cannot be used.
`ebtables` is also not available so bridge traffic filtering is disabled but there are no active bridges present.
Attempts to load the `nf_conntrack` module initially failed due to BusyBox `sysctl` limitations, but further verification showed that stateful connection tracking for IPv4 was functional.
The evaluation demonstrates that firewalld provides flexible, host-based network security and correctly enforces the configured rules on the embedded device, provided that all network interfaces are assigned to appropriate zones, necessary kernel modules are available, and SELinux policies permit required operations.
Overall, the dynamic firewall framework works as intended but some feature can become problems so some improvements can be done and are documented in chapter 7.

5.3.8 DNSMasq

The DNSMasq module is a lightweight DNS and DHCP server designed to provide DHCP or DNS services inside a private network. It should not be used as a public DNS server. The only prerequisite to use DNSMasq is a configured network interface, meaning that the LAN should have a valid IPv4 address and a valid subnet mask. After installing the module, both the DNS and DHCP servers can be configured via the web interface [105].
On this device, the DNS function has been disabled, `port=0`, so the service is only acting as a DHCP server. Any DNS resolution through this service would require custom configuration, which is not present in this deployment.
The DHCP server can be configured with the following options:

- **IP range start:** The first IP address that will be assigned to clients.
- **IP range end:** The last IP address that will be assigned to clients.
- **Lease time:** The duration for which the IP address will be assigned to the client, expressed in hours.
- **Gateway:** The gateway IP address for client configuration. If left empty, the gateway address configured for the local node will be assigned to clients.

[105]
The DHCP server in the device is bound to the `wlan0` interface, an IP range is assigned explicitly and there is a lease time for allocated addressed, 12 hours. No gateway is explicitly defined, so clients receive the default gateway of the node.
The module provides additional configuration options that can be accessed manually through configuration files. The directories that accept custom files are located in the module's root directory, under the state directory. Custom configuration files can be placed in dnsmasq.d (.conf) and custom host entries in dnsmasqhosts.d. These options enhance flexibility, but in the current deployment, no custom files are present.
The DNSMasq module installed on the IoT gateway was subjected to a series of functional and security tests to validate its configuration and behavior.
DHCP allocation tests were performed using multiple client connections on the wlan0 interface to verify that IP addresses were correctly assigned within the defined range and that the lease time was enforced.
DHCP conflict handling was tested by manually assigning IPs within the DHCP pool and gateway assignment tests confirmed that clients received the correct default gateway when none was explicitly defined.
DNS functionality was verified by checking that the server does not provide DNS resolution in this deployment, consistent with the intended design.
All tests produced the expected results, and no vulnerabilities were identified, the module is correctly configured for its intended use in a private, controlled network, following the cybersecurity best practice of minimizing the attack surface by disabling unnecessary services.

5.3.9 OpenVPN

OpenVPN is an open-source flexible VPN protocol designed to secure internet traffic by creating an encrypted connection between devices, including site-to-site connections and remote client access. In addition to strong encryption it provides authentication, and support for multiple network configurations guaranteeing a high level of security. OpenVPN offers features like network bridging, full tunneling, and split tunneling, which are crucial for complex enterprise network configurations.

The protocol operates in two modes: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP mode provides reliability guaranteeing that data packets are delivered in order and retransmitted if lost, though at the expense of speed. UDP mode, which is commonly the default, is faster but does not guarantee the order of packets, making it less reliable but more efficient for streaming and real-time communications. The UDP protocol also provide better protection against DoS attacks and port scanning than TCP. Support for IPv4 and IPv6 is present and they can be used simultaneously.
OpenVPN uses the OpenSSL library to handle encryption and decryption, providing a secure channel for data traversal. It uses the SSL/TLS protocol for key exchange, allowing for strong encryption mechanisms that include up to 256-bit encryption with advanced cipher suites. This ensures that even if data packets are intercepted, they can not be deciphered without the corresponding keys. Keys are usually RSA keys and are set by default to 1024 but should be increased to at least 2048. This can slow a little the SSL/TLS renegotiation handshake and the one-time Diffie Hellman parameters generation process but it offer higher security. By default, OpenVPN uses Blowfish, a 128-bit symmetrical cipher but is supports any cipher which is supported by the OpenSSL library, and as such can support ciphers which use large key sizes. One of the security benefits of using OpenVPN is the use of an X509 PKI where the root CA is not needed on the server machine.
OpenVPN provides several mechanisms to add security layers in order to avoid relying on just one component. The tls-auth directive adds an HMAC signature to all SSL/TLS handshake packets allowing to perform integrity verification. Any packet with the incorrect HMAC signature can be dropped without further processing. The tls-auth HMAC signature can protect against: DoS attacks or port flooding on the OpenVPN UDP port, port scanning to determine which server UDP ports are in a listening state, buffer overflow vulnerabilities in the SSL/TLS implementation, and unauthorized SSL/TLS handshake initiations by cutting them of earlier. OpenVPN has been very carefully designed to allow root privileges to be dropped after initialization becoming

```
user nobody
group nobody
```

On Linux, OpenVPN can be run entirely unprivileged. This configuration is the best in terms of security but is a little more complex to implement. To work with this configuration, OpenVPN must be configured to use `iproute` interface and `sudo` package so that interface properties and routing table can be modified. This configuration uses the Linux ability to change the permission of a `tun` device, so that unprivileged user may access it. [106, 107]
The OpenVPN instance on the device is configured to operate as a client, connecting to a remote VPN server. The client configuration includes directives such as

`client`, `remote`, and `nobind`, and it uses UDP transport `proto udp` for the VPN tunnel. Authentication is performed with certificates. The CA certificate, the client certificate and the private key are all stored inside the crypt secure partition. Server verification is enabled via `remote-cert-tls server`.
Data encryption is provided by the AES-256-CBC cipher, and privileges are dropped after initialization by running the process under the unprivileged nobody user and group.
The `update-systemd-resolved` helper, ensures that DHCP/DNS options received via the VPN are pushed to `systemd-resolved` over D-Bus, preventing DNS leaks. Additional security mechanisms are used to control execution of scripts in a secure manner.
A set of functional and local security checks were conducted to validate the client configuration and assess potential vulnerabilities.

```
sudo openvpn --config /etc/openvpn/server.conf
ps -eo user,group,cmd | grep openvpn
openvpn --show-tls
openssl x509 -in <secure/partition/path/*crt> -text -noout
ls -l <secure/partition/path/*.key>
sudo -u nobody test -r <secure/partition/path/*.key>
```

VPN connection only happens when valid certificates are present, while connections with missing or corrupted certificates are rejected, demonstrating correct enforcement of certificate-based authentication and server verification.
Process inspection verified that privileges are properly dropped, with the OpenVPN process running under the nobody user after initialization. Cipher negotiation and certificate parameters were checked, confirming that AES-256-CBC is in use and that certificate key sizes meet recommended security standard. Finally, key files were verified to be protected and inaccessible to unprivileged users, preventing accidental exposure of sensitive material.
Overall, these tests indicate that the OpenVPN client is securely configured for its intended role, with no client-side vulnerabilities identified. While the configuration follows core best practices some hardening options can be added.

5.3.10 SELinux (overview)

The device enforces SELinux in targeted mode, providing mandatory access control across all processes and services. Custom policies ensure that each component, including system daemons and user applications, operates within its assigned security domain. File and process labels are correctly applied, and autorelabeling is supported, ensuring that label integrity is maintained even after updates or file system changes. Functional verification confirmed that the device runs as expected without policy violations, demonstrating that SELinux is effectively enforcing the

intended access control. A detailed discussion of SELinux configuration, policies, and enforcement is provided in Chapter 8.

5.3.11 CVE-check with Yocto

The CVE (Common Vulnerabilities and Exposures) system, created in 1999 by MITRE Corporation, provides a uniform standard for identifying and tracking publicly disclosed security flaws and bugs. Each CVE entry in the list is assigned a unique ID, allowing researchers and organizations to refer to the same vulnerability consistently across different systems and coordinate the development of security tools and solutions. The MITRE Corporation maintains the CVE list, but a security flaw that becomes a CVE entry is often submitted by organizations and members of the open source community. There are thousands of CVE IDs issued every year.
Complex products, like an operating system (OS), can accumulate hundreds of CVEs. This means that their security posture decline over time, particularly when the product enters its end-of-maintenance phase, so bug fixes and security patches stop being issued, and end-of-life phase when all first-party support ends. This is particularly relevant for IoT devices, where studies estimate that over 90% of compromises stem from exploits of known, unpatched vulnerabilities.
To mitigate this risk, vulnerability scanning must be a continuous process, not a one-time activity before product release. Preferably, scans should happen directly on the device but, they can be performed against an inventory database if the vendor maintains full control of all the software running on the device [108].
In the analyzed system, the Yocto Project provides the `cve-check` class, which can be enabled to perform scans on packages for public CVEs against the National Vulnerability Database (NVD). This feature can be used to scan individual packages but also images by scanning all packages included by that specific image. It utilizes the NATIONAL VULNERABILITY DATABASE, from which it performs the lookup. Enabling the feature only requires adding the following directive to `local.conf`:

```
INHERIT += "cve-check"
```

When building the image, a report is generated with all the CVEs affecting the software components in the system. Each reported CVE must then be reviewed to assess its actual impact on the device. In many cases, vulnerabilities affect services that are not enabled or relevant to the use case so they can be ignored or whitelisted. CVEs with a severity score below 5 may be de-prioritized, while high or critical issues must be addressed either by upgrading to a patched release, backporting a security patch, or marking the CVE as patched using the `CVE_STATUS` variable. For Poky and OE-Core master branches, updating to a more recent software component

release with fixes is the best option, but patches can be applied if releases are not yet available [108].
CVE checks must be performed regularly, as new vulnerabilities are disclosed on a daily basis. Whenever a high-severity issue is identified, a patch release is issued to address it promptly. In most cases, however, it is more efficient to group multiple CVE fixes together and schedule their application at defined intervals.
For the device under test, at the time of evaluation all CVEs were correctly identified, and critical ones were patched. A whitelist was maintained for non-relevant vulnerabilities, and it is periodically updated. Moreover, the `cve-check` process is integrated into the CI/CD pipeline, ensuring that vulnerability scans are automatically performed at every build. This guarantees early detection of new issues, consistent monitoring of the software bill of materials, and faster remediation when high-severity CVEs are disclosed. The process ensures that the system remains aligned with security best practices as new CVEs are disclosed.

5.3.12 Redis, EdgeX

EdgeX Foundry is an open source, vendor neutral, flexible, interoperable, software platform for IoT edge computing, hosted by the Linux Foundation. It can be considered an edge middleware, operating between physical sensing and the technology systems. It provides a modular, microservice-based architecture that enables devices, sensors, and actuators to interoperate with higher-level IT and cloud applications.
Version 2, the one implemented in the device, introduced a refined API set, improved performance, and a more consistent security model, including a simplified reverse proxy and enhanced secret management.
Edgex micro services are organized into 4 service layers, and 2 underlying augmenting system services.

- **Device Services:** connectors that interact with physical devices and translate native protocols (e.g., Modbus, BACnet, MQTT) into EdgeX's common data models.
- **Core Services:** intermediary between the north and south sides of EdgeX, provides persistence, metadata, and command/control functions. This layer is divided into:
  - **Core Data:** persistence repository for sensor data collected from south side objects.
  - **Command:** service that facilitates and actuate requests from the north side to the south side.

  - **Metadata:** repository and management service of metadata about the things that are connected to EdgeX Foundry. It also provides the capability to provision new devices and pair them with their own device services.
  - **Registry and Configuration:** provides other EdgeX Foundry micro services with information about associated services.

- **Application Services:** forwarding processed data to external IT or cloud systems.

- **Supporting Services:** features such as scheduling and notifications.

The 2 underlying System Services of EdgeX Foundry are Security, secret storage and an API gateway for access control, and System Management, which provides service orchestration and monitoring.
EdgeX Foundry's reference implementation database (for sensor data, metadata and all things that need to be persisted in a database) is Redis. [109]

Redis Redis is an open source, in-memory database, used as a message broker in EdgeX. Redis is durable and uses persistence only for recovering state and is well suited to edge environments thanks to its low latency and high throughput. Its function is to store sensor readings, metadata and service states as well as provide a fast buffer for real-time operations rather than long-term data storage.
Redis Enterprise offers several database security controls to help protect data against unauthorized access and to improve the operational security of the database. Best practices include:

- Enforcing strong passwords and disabling default users that are present for backwards compatibility.

- Configuring TLS for data in transit.

- Enabling client certificate authentication to prevent unauthorized access to sensitive data.

- Using trusted certificates instead of self-signed ones.

- Verifying and testing database backups as well as implementing a disaster recovery strategy.

[110]
In the tested device, EdgeX Foundry v2 is deployed with Redis configured strictly as a local service, meaning it is only accessible to EdgeX microservices and has no external network exposure. This design minimizes the attack surface, as Redis cannot be reached from outside the device. Redis remains primarily an ephemeral

store used for runtime operations. Data durability is supported through snapshot and append-only persistence.
Credentials and sensitive configurations are handled by EdgeX's dedicated secret store (Vault) rather than Redis, ensuring proper isolation and protection of authentication material.
The Yocto `cve-check` integration confirmed that no critical unpatched Redis vulnerabilities were present in the tested build.
Overall, the adoption of EdgeX Foundry v2 with Redis provides a modern, scalable, and secure middleware layer for industrial IoT. The local deployment of Redis, combined with improved security components introduced in EdgeX v2 and continuous vulnerability management, ensures compliance with best practices and robustness against common exploitation vectors.
To confirm the correct deployment and isolation of EdgeX Foundry v2 and its Redis database, a series of practical tests were performed. These tests focused on verifying that Redis is accessible only to local microservices, that secrets are isolated from the database, and that the API and dashboard enforce secure access.

- Redis local-only binding was verified to ensure Redis listens only on 127.0.0.1 using:

  ```
  netstat -tuln | grep 6379
  ```

- EdgeX API functionality was tested to ensure API responds correctly and is reachable only from authorized sources with the following commands:

  ```
  curl -v http://localhost:port/api/v2/ping
  curl -v http://<device-ip>:port/api/v2/ping
  ```

- TLS validation for the dashboard and API was checked to ensure certificate enforcement:

  ```
  curl -vk https://<device-ip>:<dashboard-port>
  ```

All tests passed successfully, confirming that the security configuration is effective and that the platform operates according to best practices for secret management, access control, and data isolation.

5.3.13 Mender: secure OTA update management

Mender is a secure and robust over-the-air software update framework designed to handle a large number of devices where the build system is a simple workstation or a standard component such as Yocto, which is the one used in the device under test. A **Mender Artifact** is created in the format required by the target device and then uploaded to the **Mender Server**, which is responsible for deploying updates to the device. It also monitors the current software version present on each device and schedules the roll out of new releases.
The device runs the **Mender Client** in managed mode, which means it acts as a daemon and regularly poll the server, automatically applies updates and commits them. If there is, the update client downloads the artifact and later performs the installation.
Mender implements a client/server model and supports a dual redundant scheme also known as A/B scheme, which ensure that the device always returns to a working state on failure. During an update of the OS, the client writes the new version to the inactive partition. After, the client verifies the checksum and if everything is correct it sets a flag in the bootloader that will cause it to flip the active and inactive partitions around on the next reboot. On the first boot following an update, the Mender Client has the job to commit the update and a flag in the bootloader indicates that the update was applied successfully. If something goes wrong before committing the update, the bootloader will roll back to the previous version by flipping the active and inactive partitions back again.
Mender security model relies on TLS for server/client channel confidentiality and server authentication, per-device RSA or Elliptic Curve key pair for device identity and optional artifact signature verification to ensure updates come from a trusted source. This ensures that the Mender Client will only connect to a verified server, and no man-in-the-middle attack is possible. The Client initiates all communication by connecting to the server, so no open ports are required on the device in order to use Mender. As an additional layer of security, Mender Enterprise supports Role Based Access Control to limit authorization of users and configurable API rate limits. [111] To validate the implementation on the target device a comprehensive test suite covering artifact verification, A/B update flow, rollback and resilience, protection of Mender/U-Boot environment variables and secure storage of keys was executed.
By analyzing the configuration file it is possible to see that the optional artifact signature is enabled and that the `ArtifactVerifyKey` is stored in the secure storage partition and no access to it is granted to unauthorized users. A first test was to check that the implementation worked correctly so a new artifact was created, deployed to the device, and written to the inactive partition. The checksum verification completed without errors, after which the bootloader flag

was updated to instruct the system to boot from the new partition. The device successfully rebooted into the updated image, and once stability was confirmed, the Mender client issued the commit. During laboratory validation, where access to the bootloader environment was available, additional checks were performed on the U-Boot variables using

```
fw_printenv
fw_setenv
```

The tests confirmed that variables such as

```
mender_boot_part, mender_boot_part_hex, mender_saveenv_canary
```

changed consistently during the update cycle and returned to a stable state after commit. These checks also demonstrated that in case of a failed update the system correctly rolled back to the previous partition, ensuring robustness against incomplete or corrupted deployments. In a production environment, however, direct access to the bootloader environment is neither possible nor desirable for security reasons so verification relies on client logs and on the observable system behavior. During laboratory test, an additional robustness check was performed by manually altering critical U-Boot variables. After reboot, as expected, the device did not remain in a corrupted state but the boot process automatically restored the correct values and completed successfully. This confirmed that the update mechanism is resilient against unintended or malicious modifications of boot parameters, which in any case cannot be changed without access to the U-Boot environment and the corresponding unlock codes.
In addition, rollback functionality was checked by interrupting the update process before the commit stage. The system detected the incomplete deployment, automatically reverted to the previous working partition, and restored normal operation after reboot.
To further validate the security mechanisms, an additional test attempted to upload an unsigned, or incorrectly signed, artifact: as expected, the device detected the signature problem and rejected the installation, preventing untrusted code execution.
Overall, these tests confirm that the secure OTA update mechanism is robust, reliable, and fully aligned with industry best practices, providing strong protection against corrupted updates, unauthorized modifications, and other potential threats.

5.3.14 Grafana: monitoring, logging and exposure risks

Grafana Agent is a distribution of the OpenTelemetry Collector designed to be flexible, performant, and compatible with multiple ecosystems. Grafana Agent is based around components wired together to to send metrics, logs, and traces to

the Grafana observability after the collection of telemetry data.
The agent is fully vendor-neutral and integrates seamlessly with the Prometheus and OpenTelemetry ecosystems as well as the Grafana open-source stack, including Loki, Tempo, Mimir, and Pyroscope. Grafana can be deployed on any number of machines to collect millions of active series and terabytes of logs, while maintaining efficiency.
Moreover, it provides robustness and reliability, combined with powerful programmability and debugging capabilities and comes with ready-to-use integrations for systems such as MySQL, Kubernetes, and Apache. [112, 113]
In the analyzed system, it is deployed via the standard Yocto recipe, with additional CA certificates for secure backend communication. The certificates and configuration files are installed with restrictive permissions (0644 for files, 0755 for the binary), preventing unauthorized modification.
The agent uses configuration parameters such as LOKI_URL for the Loki logging endpoint, MIMIR_URL for metrics ingestion, and INSTANCE_NAME to identify the device in dashboards. In the current deployment, these endpoints are configured to communicate over HTTPS, protected with TLS, using the installed CA certificates, providing authentication and integrity of telemetry data.
Functional validation included starting the agent under systemd, verifying logs, and confirming network behavior. Commands used include:

```
systemctl status grafana-agent
journalctl -u grafana-agent
netstat -tulpen | grep grafana-agent
```

Results confirmed that the service started correctly, telemetry data was sent without errors, and no unintended listening ports were exposed. Certificates were properly recognized, and TLS connections were successfully established to the backend endpoints.
Overall, the Grafana Agent deployment follows security best practices: files and certificates are protected with correct permissions, the agent communicates securely with verified backend, and no excessive network exposure is present. This ensures a robust observability setup with minimal risk of telemetry compromise, while maintaining flexibility for future enhancements such as integrating additional secure components.

Chapter 6

Penetration Testing Activities

6.1 Scope and rules of engagement

All penetration tests were carried out in a controlled laboratory environment with explicit authorization. Tests were designed to be non-destructive where possible; destructive or availability-impacting tests (for example, high-rate DoS floods or on-device firmware replacement) were executed only when explicitly permitted and clearly labeled.

6.2 Reconnaissance and port scanning

Reconnaissance activities represent the first phase of a penetration test, where the attacker attempts to gather information about exposed services and interfaces. Port scanning allows identification of reachable services and verification of firewall rules. This commands were used to check service fingerprinting and do a general initial scan:

```
nmap -O <DEVICE_IP>
nmap -sU <DEVICE_IP>
```

6.2.1 firewalld Assessment

To complement the inspection of the internal `firewalld` configuration, external validation was performed using the `nmap` network scanner. This ensured that the effective packet filtering behavior observed on the network interfaces was consistent with the configured zones and services. The following tests were executed:

- **Port verification in the `public` zone.** Targeted scan of TCP ports commonly associated with allowed services (e.g., SSH on 22, HTTPS on 443) confirmed that only explicitly permitted ports were reachable:

  ```
  nmap -p 22,443 <DEVICE_IP>
  ```

- **Full port sweep.** A comprehensive scan across all TCP and UDP ports verified that no additional services were exposed:

  ```
  nmap -sS -sU -T4 -p- <DEVICE_IP>
  ```

- **Zone separation check.** Scans through the VPN interface (`tun0`, trusted zone) confirmed unrestricted connectivity.

- **Masquerading and port forwarding validation.** Where port forwarding rules were defined, `nmap` confirmed correct behavior.

Results confirmed that firewall enforcement aligned with both the intended security posture and the configured policies. This validated that no unintended exposure existed through `firewalld`.

6.2.2 DNSMasq / DHCP tests

A dedicated network penetration test was performed to assess the attack surface of DNSMasq and the DHCP capability it provides.
The scans confirmed that DNSMasq was listening only on the intended interface (`wlan0`) and that no additional TCP or UDP ports were exposed. Commands executed:

```
nmap -sS -p- <gateway_ip>    # full TCP port scan
nmap -sU -p- <gateway_ip>    # full UDP port scan
nmap -O <gateway_ip>         # OS detection and service fingerprinting
```

Results confirmed that the system exposes only the expected DHCP service, with no unexpected open ports or services, minimizing the attack surface.

6.3 Brute-force attacks (SSH / Dashboard API)

A brute-force attack systematically attempts many passwords against a login endpoint until the correct one is found. If no throttling or lockout mechanisms are enforced, an attacker can guess weak credentials in a reasonable time.
All brute-force checks were performed in a controlled laboratory environment on the target device with explicit authorization. Tests were limited and rate-controlled to avoid service disruption.

SSH SSH was found to be resilient to brute-force in this deployment. Password authentication is disabled for administrative accounts; only public-key logins are allowed. Session controls (`ClientAliveInterval / ClientAliveCountMax`) and acceptable `MaxAuthTries` are in place. Attempts to authenticate using invalid keys were denied as expected. Brute-force tools (e.g. hydra) were not used against SSH because key-only auth is enforced.

Dashboard (HTTP API) The dashboard password was found to be vulnerable to brute-force attacks. The Admin password was found to be weak/hardcoded and there is no effective rate limiting or lockout for repeated login attempts. As a consequence the dashboard authentication endpoint (HTTPS) can be brute-forced until the correct credentials are supplied.

```
for pw in weakpass1 weakpass2 weakpass3; do
  http_code=$(curl -s -o /dev/null -w "%{http_code}" -k \
    -X POST "https://iot-gateway.example.local/api/login" \
    -H "Content-Type: application/json" \
    -d "{\"username\":\"admin\",\"password\":\"$pw\"}")
  echo "$pw -> $http_code"
  sleep 2
done
```

The server responded with 200/401 codes but did not trigger any lockout or throttling during these controlled attempts. For mitigation applied see chapter 7.

6.3.1 Future Work

Full brute-force campaigns with tools like `hydra` or `wfuzz` were not performed, but could simulate large-scale automated attacks in lab conditions.

6.4 Privilege escalation attempts

Privilege escalation refers to exploiting misconfigurations or vulnerabilities to gain higher privileges than those originally assigned, moving from a restricted user to root. Attackers typically attempt this by exploiting SUID binaries, weak file permissions, or misconfigured sudo rules.
In the tested device, practical checks included:

- **SUID/SGID binaries inspection.** Search the filesystem for programs with the SUID bit set and inspect results for unexpected or risky entries:

  ```
  find / -perm -4000 -type f -exec ls -ld {} \; 2>/dev/null
  find / -perm -2000 -type f -exec ls -ld {} \; 2>/dev/null  # SGID
  ```

- **Group membership and permission checks.** Enumerate groups and user memberships, then attempt non-destructive group-based checks to see if any misconfigured group grants unintended access:

```
getent group | grep -E 'sudo|wheel|root'
groups <username>
id <username>
```

- **sudo policy review and controlled execution trials.** Enumerate allowed sudo commands for users and, when safe and explicitly allowed by policy, run minimal permitted commands to confirm behavior:

```
sudo -l    # list sudo rights for current user
```

All non-destructive tests indicated the system is resistant to simple privilege escalation attempts. No unexpected SUID/SGID binaries or group misconfigurations that would allow privilege escalation were found, and controlled sudo checks did not permit unauthorized privilege gain. Attempts to escalate from a non-privileged user to root in the lab environment were unsuccessful.

6.4.1 Future Work

Kernel-level privilege escalation (via known CVEs) was not attempted, but could be simulated by introducing vulnerable kernels in a lab environment.

6.5 U-Boot / Secure Boot penetration attempts

Secure Boot prevents unauthorized firmware from loading at startup. U-Boot, the bootloader, enforces integrity and rollback checks.
From a penetration-testing perspective the objective is to verify whether an attacker with local access can tamper the boot chain by changing U-Boot environment variables or cause the device to boot unsigned or replayed firmware. A **Replay attack** in this context means attempting to reuse a previously valid artifact (image or CSF) to force booting into an older or unauthorized state.
Only actions that simulate an attacker modifying boot environment variables and observing resulting boot behavior were performed. All steps were executed in a controlled lab with an unlocked bootloader. It is not possible to do these operations with a locked bootloader. After reading the full U-boot environment, these test were performed:

- **Modify a critical boot variable (lab only):** attempt to change the active boot partition to an alternate value and record the command exit status.

```
fw_setenv mender_boot_part 2
echo "fw_setenv exit: $?"
```

- **Replay test (lab):** attempt to point the boot flags to a previously valid artifact (or manually try to re-present an older signed artifact) and observe acceptance or rejection.

  ```
  fw_setenv mender_boot_part <older_slot>
  reboot
  journalctl -k -b -1 | grep -i -E 'verity|signature|hab|csf|rollback'
  ```

  If attempting to upload a previously valid artifact via the update server, monitor the client logs to confirm signature verification rejection:

  ```
  journalctl -u mender-client -f
  ```

Changing critical variables in the lab caused the boot flow to detect an invalid/unauthorized boot path; either the boot failed verification or the bootloader refused to hand off to an unsigned/invalid image.
The device did not remain permanently broken, the system returned to a validated state and the `fw_printenv` variables returned to a consistent configuration after recovery.
Replay attempts were rejected by the verification layer, dm-verity, preventing rollback to older images in this testbed.

6.6 Encrypted partitions and integrity (dm-crypt, dm-verity)

Penetration tasting can be used to verify that LUKS/dm-crypt prevents unauthorized access to encrypted volumes and that dm-verity detects filesystem image tampering at boot. Since it was not possible to perform potentially disruptive tests, just one alongside the the validation tests was done.
Attempt LUKS open with incorrect key to confirm that an incorrect key is rejected by the LUKS keyslots and that the encrypted container cannot be opened without valid credentials.

```
cryptsetup luksOpen /dev/mmcblk2pX test --key-file wrong.key
```

The command returned a non zero exit status and the container was not opened. No plaintext block devices were exposed and system logs indicated an invalid key error. This demonstrates correct rejection behavior for a straightforward wrong key attempt.

6.6.1 Future work (lab only — potentially disruptive)

The following penetration tests are valid follow-ups to increase assurance but were intentionally avoided on production hardware because they can be destructive or require full lab recovery procedures:

- **Rate-controlled brute-force / key guessing:** validate system response to repeated wrong key attempts and check for throttling or lockout behavior.
- **LUKS header tampering / header backup tests:** corrupt or replace LUKS headers to verify recovery processes and observe failure modes.
- **Race / rollback injection during update:** try to flip boot variables or swap images during an update window to detect race conditions that might allow rollback or bypass.
- **Cold-boot / memory dump attempts:** attempt ephemeral key recovery from RAM during initramfs operations (requires specialized hardware and lab controls).

6.7 Mender OTA penetration checks

OTA systems are a high-value target because a successful bypass allows persistent and remote code execution. Penetration tests here attempt to install unsigned or tampered artifacts, replay previously valid artifacts to induce rollback, and manipulate the boot selection process to force execution of non-committed partitions. The aim is to confirm the client and bootloader verification layers cannot be bypassed by an attacker with network and/or local access in a lab scenario.

- **Replay previously valid artifact to force rollback:** attempt to point U-Boot/Mender state to an older slot/artifact or re-present an older signed artifact, by manipulating local boot env, to see if rollback protection or anti-replay prevents boot into the older image.

  ```
  fw_setenv mender_boot_part <older_slot>
  reboot
  ```

- **Attempt replay via update server (artifact replay attack):** re-send a previously captured/archived artifact with original metadata to the client to test replay prevention in the update protocol (server-side and client-side).
  In the lab, a previously archived artifact was re-uploaded to the Mender

server and delivered to the client. The client's verification process and logs were monitored during the update to check whether freshness and rollback protections could be bypassed.

In the lab all invalidly signed artifacts were rejected by the client verification steps. Attempts to force rollback by switching boot partitions or re-presenting older artifacts were detected and blocked by the verification layers (dm-verity/metadata checks). No path was found to execute unsigned code without compromising the chain of trust or possessing the unlock key.

6.7.1 Future penetration tests (lab-only)

The following test were not executed but represent meaningful penetration test activities that could uncover vulnerabilities if performed in a controlled environment.

- **Attempt delivery of invalid or unsigned OTA artifacts:** An attacker with access to the update delivery infrastructure might attempt to introduce unsigned or tampered artifacts. Penetration testing in the lab would involve crafting artifacts with invalid signatures or corrupted metadata and observing whether the client properly rejects them, verifying that no inactive partition is activated and that the boot chain integrity remains intact.
- **Network-level MITM attempts during OTA delivery:** A penetration test in the lab would simulate MITM conditions (e.g., TLS interception or packet alteration) to verify that the Mender client enforces end-to-end artifact integrity and rejects any tampered or replayed updates.

6.8 CAAM (Cryptographic Accelerator) checks

Hardware crypto engines (CAAM) protect keys and perform sensitive operations inside a controlled execution environment. Penetration testing focuses on attempts to extract key material, to trick the CAAM into wrapping/unwrapping keys to an attacker-controlled output, or to misuse CAAM APIs. Tests emphasize active misuse attempts rather than passive inspection.

- **Attempt to extract CAAM blob and unwrap outside CAAM context:** capture the CAAM blob and try to run unwrap/decrypt operations using non-CAAM tools or by feeding the blob into userland crypto utilities to determine if keys can be exposed in plaintext.

```
fw_printenv -n caam_black_blob > /tmp/caam_black_blob.base64
```

```
base64 -d /tmp/caam_black_blob.base64 > /tmp/caam_black_blob.bb
caam-decrypt /tmp/caam_black_blob.bb AES-256-CBC \
/tmp/dataintegritykey /tmp/key
```

The blob could only be unwrapped successfully by CAAM hardware. No exposure of plaintext keys to disk or memory was observed. This confirmed that blob confidentiality is enforced and external tools cannot bypass CAAM protections.

6.8.1 Future penetration tests (lab-only)

The following tests were not executed but represent meaningful penetration test activities that could uncover vulnerabilities if performed in a controlled environment. **Attempt to coerce CAAM API into returning key material via malformed requests:** exercise on-device CAAM utilities with malformed parameters to detect improper error handling or accidental leakage to logs or files.

```
/usr/bin/caam-tool --unwrap --blob /tmp/caam_black_blob.bb --nonce malformed
```

6.9 Denial-of-Service (DoS) penetration tests

Active attackers frequently use resource exhaustion and protocol floods to disrupt device availability. Penetration tests simulate realistic attacker behaviors (spoofed sources, varying packet sizes, protocol variants) to evaluate whether the device or its networking stack fails open, crashes, or otherwise exposes service degradation beyond normal tolerance. Tests are strictly rate-controlled and executed only in lab.

- **Controlled spoofed UDP flood targeting service port:** simulate an attacker using randomized sources and high packet rates to saturate the service or trigger stack bugs in the network or application layer.

  ```
  hping3 --udp -p 1195 --flood --rand-source <vpn-server-ip>
  tcpdump -i <iface> -w dos_flood.pcap
  # monitor resource metrics on device
  sar -n DEV 1 10
  top -b -n 1
  ```

- **Burst-limited stress test to detect transient failure modes:** send a finite, large number of packets to observe transient crashes, memory leaks, or resource exhaustion that only manifest under burst conditions.

```
hping3 --udp -p 1195 -c 10000 --rand-source <vpn-server-ip>
tcpdump -i <iface> -w dos_burst.pcap
# collect dmesg/logs after test for faults
dmesg | tail -n 200
journalctl -b --priority=err
```

- **ICMP flood (network stack resilience check):** generate a high-rate ICMP echo request stream to test resilience of the network stack under common flooding conditions.

```
hping3 --icmp --flood <vpn-server-ip>
tcpdump -i <iface> -w icmp_flood.pcap
sar -n DEV 1 10
```

In all scenarios, flooding traffic led to degraded network availability as expected (increased latency, dropped packets, elevated CPU utilization), but the device remained stable. No kernel crashes, memory leaks, or persistent service failures were observed.

6.10 OpenVPN penetration tests (future work)

In the current campaign, no OpenVPN penetration tests were executed. This limitation was due to the OpenVPN server not being accessible from the lab client environment, following constraints from the customer. Given the critical role of VPN tunnels in ensuring confidentiality and integrity of communications, future work should include the following activities:

- **Replay of captured handshakes:** An attacker might attempt to reuse previously valid TLS handshake packets to bypass freshness checks or desynchronize the session state. This can be done by capturing OpenVPN handshakes with `tcpdump` and replay them using `tcpreplay`, monitoring both client and server logs for acceptance or rejection.
- **Malformed handshake injection:** Use mutation tools (e.g., `boofuzz`, `TLS-Attacker`) to generate abnormal handshakes and observe stability.
- **Cryptographic downgrade scenarios:** If cipher suite negotiation is not properly enforced, an attacker might attempt to force weaker algorithms or parameters. To do this it should be possible to configure a lab OpenVPN endpoint with downgraded ciphers and attempt forced negotiation via crafted packets.

- **State desynchronization attempts:** Dropping, duplicating, or replaying data packets may reveal weaknesses in session management.

6.11 Other recommended penetration tests (general)

In addition to OTA, CAAM, and VPN-focused activities, the following penetration test directions are suggested for a more complete assessment:

- **U-Boot deeper fuzzing / exploit simulation:** fuzz command parsing and environment handling (requires unlocked bootloader). Tools: AFL, QEMU.
- **Kernel / local privilege escalation checks:** simulate known CVEs in isolated lab to assess exploitability. Tools: Metasploit, PoC code.
- **Side-channel analysis on CAAM:** timing/power-based leakage assessment (requires specialized hardware).
- **Application-level DoS / API flood:** flood web/API endpoints with `wrk` or `ab` to test throttling and rate-limiting.
- **MITM scenarios application level:** evaluates whether the dashboard and its transport layer correctly enforce strict cipher policies, reject renegotiation/downgrade attempts, and detect interception. Simulate MITM conditions using a controlled proxy or custom TLS interception tooling to attempt downgrade and renegotiation attacks.
- **Credential replay:** If an attacker obtains client credentials (private keys, tokens) they may try to re-use them to authenticate to the dashboard or OTA service. Penetration testing should assess whether the system detects anomalous client usage (geolocation endpoint mismatch, concurrent sessions) and whether key/credential reuse can be used to perform unauthorized actions.

6.12 Summary of penetration testing outcomes

Most of the on-device and network-level penetration checks performed in the lab produced no critical exploitable vulnerabilities. The main actionable findings were related to dashboard password policy and the absence of throttling/lockout, these were fixed during validation.

Test Area	**Performed / Observed**	**Future Work**
Reconnaissance / Port Scanning	Nmap scans, firewall zones, DHCP/DNS verified. No unexpected open ports.	None required; repeat if network changes.
Brute-force (SSH / Dashboard)	SSH key-only auth effective; Dashboard vulnerable to weak/hardcoded passwords.	Large-scale automated brute-force in lab, test rate-limiting / lockout.
Privilege Escalation	Checked SUID/SGID binaries, group membership, sudo rights; no unauthorized elevation possible.	Kernel CVE simulation in lab.
U-Boot / Secure Boot	Modified boot variables, replayed older artifacts. Boot verification and dm-verity blocked unauthorized boots.	Deeper U-Boot fuzzing, exploit simulation.
Encrypted Partitions / dm-verity	Attempted wrong-key LUKS open rejected. No data exposed.	Brute-force / key guessing, header corruption, rollback injection, cold-boot attacks.
Mender OTA	Replayed artifacts locally and via server; invalid signatures rejected.	Deliver unsigned/tampered artifacts, network-level MITM (lab).
CAAM	Blob unwrapped only by hardware; no key leakage.	Malformed API requests, check for accidental leakage.
Denial-of-Service	Controlled UDP / ICMP floods; device remained stable, minor CPU increase and packet loss.	Application-level API floods.
OpenVPN	Not performed; server inaccessible.	Replay handshakes, malformed handshake injection, cipher downgrade.
Other / Misc.	N/A	Side-channel CAAM analysis, MITM attacks on dashboard, credential replay, kernel/local CVE checks.

Table 6.1: Summary of Penetration Testing Activities and Future Work

Chapter 7

Test results and validation

7.1 Introduction

The following section presents the results of the security assessment conducted on the IoT gateway device. It is important to note that the issues identified are not critical vulnerabilities that could be exploited in the current configuration. Rather, they are minor implementation aspects, configuration choices, or residual risks that could theoretically lead to problems under specific circumstances or in future deployments.
These findings were identified through a combination of static code analysis, configuration review, and practical testing. Each issue is described along with the mitigation strategy or improvement that has been implemented or recommended.

7.2 Vulnerabilities discovered and Mitigation Strategies

7.2.1 Authentication

Initial analysis revealed weaknesses in the management of the dashboard administrative password and the absence of brute-force protections for such passwords. These issues have since been addressed.
Mitigation: All user types (including Admin) are now required to change their default password the first time they log in, so there are no more hardcoded passwords that stay the same. Best practice regarding the strength of the password has been enforced: minimum length, inclusion of upper/lowercase letters, numbers, and special characters, and prevention of password reuse.
Regarding the brute-force protection some security measure are now in place. An IP and username based lockout mechanism was introduced, blocking further

attempts after five consecutive failures for a defined time window. Moreover, a global rate-limiting mechanism was added to introduce incremental delays in case of excessive failed login attempts across multiple accounts, mitigating automated attacks.
Testing after these implementation has been performed and they confirm that the security is significantly strengthened.

7.2.2 Overlay System

Through static analysis, a number of potential risks were identified. Some directories have write permissions that could, in theory, be exploited. Additionally, some configuration files and other directories are unnecessarily readable or writable, even though they are never modified or accessed during normal operation.
Mitigation: At the overlay filesystem layer, these risks were addressed by properly setting file permissions and limiting write access to only the runtime files that actually require it.

7.2.3 CAAM module

Although the initialization script correctly leverages CAAM to protect keys and wraps sensitive material into hardware-bound black blobs, all operations are performed within the `initramfs`, so there is no persistent exposure. Nevertheless, a few improvements could still be considered.
Firstly, the current method of generating key material by truncating base64-encoded output `keyRandom::32` and `iv::16` does not provide full entropy for 32-byte or 16-byte keys.
Mitigation: Replacing base64 truncation with direct raw byte generation.
Another issue is the use of the `ecb` flag in the `caam-keygen` command. Even if this is only a tool option and not an actual use of ECB for data encryption, ECB mode is cryptographically weak and its presence requires careful consideration.
Consideration: In this context the ecb flag was found to serves only as a tool-specific parameter and does not result in the use of ECB mode for actual data encryption so it does not represent a security vulnerability.
Finally, the script relies on `/dev/random` for entropy, which on embedded devices may block or provide limited throughput unless properly seeded by CAAM.
Mitigation: Seed with CAAM RNG or use /dev/urandom if appropriate.
These risks do not compromise the overall design, as the U-Boot environment is protected by an unlock code and the final artifacts are hardware-encrypted blobs, but they highlight areas where improvements can be made.

7.2.4 firewalld

Although the evaluation confirmed that `firewalld` is active and enforces rules correctly, some residual risks and limitations were identified:

- **Trusted zone on VPN interface**: the active zone `trusted` applied to `tun0` accepts all incoming traffic without restrictions. While this is acceptable for authenticated VPN tunnels such as the one used in the case study, it would be overly permissive if the VPN endpoint were misconfigured or compromised. **Recommendation:** Assigning a more restrictive zone (e.g., `internal` will reduce exposure.
- **Unassigned Ethernet interface**: the physical interface `eth0` falls back to the default `public` zone, which is restrictive by design, but leaving the assignment implicit may cause ambiguity.
  Recommendation: Explicit assignment is recommended. However, as explained in chapter 5, this is not an immediate risk since the public zone is not active.
- **IPv6 firewall disabled**: the device currently has only loopback and link-local IPv6 addresses, so the absence of `ip6tables` does not pose an immediate risk. **Recommendation:** If global IPv6 addressing is enabled in the future, firewalling should be configured accordingly.
- **Advanced features unavailable**: the absence of `ipset` and `ebtables` disables list-based filtering and bridge traffic filtering, respectively.
  Recommendation: They are not required in the current deployment but may limit flexibility and can be needed for future settings.
- **Connection tracking behavior**: initial errors when loading the `nf conntrack` module were observed, caused by BusyBox `sysctl` limitations. Subsequent tests confirmed that IPv4 stateful inspection operates correctly but altogheter the error can cause inconsistencies.
  Recommendation: Monitor behavior in constrained environments to ensure consistency.

Overall, the identified points are not critical vulnerabilities but configuration and design aspects where explicit assignments or additional modules could further harden the deployment.

7.2.5 OpenVPN

Some recommended hardening options in the OpenVPN configuration are currently commented out, most notably tls-auth. Without tls-auth (or tls-crypt), the VPN

client is more exposed to unauthenticated UDP packets targeting the OpenVPN port. This can make it easier for attackers to perform DoS attacks, port scanning, or attempt to exploit SSL/TLS handshake vulnerabilities, because all handshake packets are processed by the server before authentication.
Mitigation: Enable tls-auth or tls-crypt adds an HMAC signature to all handshake packets, allowing the server to drop unauthenticated packets early, reducing the attack surface, preventing resource exhaustion attacks, reducing the effectiveness of unauthenticated UDP probing and mitigate coarse-grained scanning or flood attempts.
Additionally, the configuration uses AES-256-CBC, which is a secure symmetric cipher, but it is a non-AEAD (Authenticated Encryption with Associated Data) cipher. Non-AEAD ciphers provide confidentiality but do not protect the integrity of the encrypted data by themselves. This could allow certain attacks, CBC mode ciphers are susceptible to padding oracle attacks, on data integrity or allow subtle manipulation of encrypted packets if an attacker manage to intercept the traffic.
Mitigation: Update the cipher configuration to an AEAD cipher (e.g., AES-256-GCM) in both server and client configurations to provide both encryption and authentication in a single operation and prevent tampering inside the VPN.
Further hardening could also include enabling `persist-tun` and `persist-key` to minimize transient leaks of sensitive state information during re-connections.

7.3 Conclusion

The validation and mitigation phase confirmed that no critical vulnerabilities are present in the current configuration of the IoT gateway. The issues identified were limited to password management, brute-force resistance, file system permissions, cryptographic key generation practices, firewall zoning, and VPN configuration choices.
All of these aspects have been either remediated through configuration changes or addressed with clear recommendations for improvement. The applied mitigations, such as mandatory password changes, enforcement of strong password policies, lockout and rate-limiting mechanisms, stricter filesystem permissions, and enhanced VPN settings, significantly strengthen the security posture of the device.
At the time of assessment, residual risks are considered minor and do not affect the operational security of the system. The gateway can therefore be regarded as adequately secured for its intended deployment, provided that recommended best practices continue to be applied and the configuration is periodically reviewed as new threats and software updates emerge.

Table 7.1: Summary of identified risks and recommendations

Area	**Issue**	**Recommendation**
Authentication	Weak/hardcoded dashboard password; no brute force protection	Enforce password change on first login, strong password policy, and rate limiting/blocking after failed attempts
Overlay System	Unnecessary read/write permissions	Restrict to runtime-only files
CAAM module	Base64 truncation lowers entropy	Generate raw key bytes
	`ecb` flag in keygen is misleading	Clarify tool-only parameter
	`/dev/random` may block	Use CAAM RNG or `/dev/urandom`
firewalld	VPN zone too permissive	Use more restrictive zone
	Ethernet interface implicit	Assign explicitly
	IPv6 not filtered	Add IPv6 rules if needed
	Missing `ipset`/`ebtables`	Not critical, but limits flexibility
	Conntrack errors in BusyBox	Monitor in constrained setups
OpenVPN	`tls-auth`/`tls-crypt` disabled	Enable for DoS/scan protection
	AES-CBC cipher (non-AEAD)	Switch to AES-GCM
	State leaks on reconnect	Enable `persist-tun`/`persist-key`

Chapter 8

Custom SELinux Policies

8.1 Overview of SELinux

Security-Enhanced Linux (SELinux) is a Linux security architecture that allows administrators to perform fine-grained control over system access. Originally developed by the United States National Security Agency (NSA) as a set of kernel patches using the Linux Security Modules (LSM) framework, SELinux enforces mandatory access control (MAC) policies that define which subjects (processes) can access which objects (files, directories, sockets, etc.).
An SELinux security policy is a collection of SELinux rules that specify allowed interactions between processes and system resources. A policy is a key component of SELinux and is loaded into the kernel by SELinux user-space tools. By default, SELinux denies all access not explicitly permitted by the loaded policy.
The most commonly used policy types are *targeted* and *MLS* (Multi-Level Security): targeted is suitable for most systems and activities while MLS is typically reserved for highly sensitive government environments.

Policy Type	Use Case	Description
Targeted	Common systems	Only specific critical processes are confined; default on RHEL/CentOS
MLS	High-security environments	Multi-Level Security, used in government systems
MCS	Optional, container isolation	Multi-Category Security, isolates workloads in containers

Table 8.1: SELinux Policy Types

Usually, the `etc/sysconfig/selinux` file includes the indication of wether SELinux is disabled or set to enforcing or permissive mode, and which policy is loaded.
Labeling and type enforcement are the most important concepts in SELinux. Labels provide a logical way to put elements together, and the kernel manages these labels during system startup and enforces access control by checking these labels against the policy rules.
When an application or process, requests access to an object, SELinux checks the permissions for subjects and objects using the Access Vector Cache (AVC) where the access decision are previuosly cached.
If a decision cannot be made locally using the cache, requests are forwarded to the SELinux security server for evaluation, which checks the file and the security context of the application or process. This context is applied based on the SELinux policy database, and the access is then either granted or denied.
This mechanism allows the enforcement of a MAC policy independent of traditional discretionary access control (DAC). Even if DAC settings in a home directory are modified, the system can be protected by enabling a SELinux policy that prevents other users or processes from accessing the directory. [114] When writing custom SELinux policies, three main types of files are typically created:

- **.te (Type Enforcement) files:** These files define the core rules of the policy, specifying which domains (processes) can access which types (files, directories, ports, etc.) and under what permissions. *Example:* Allow a process labeled `modbus_t` to read files labeled `modbus_conf_t`:

  ```
  allow device_modbus_t device_modbus_etc_t:file { read };
  ```

- **.fc (File Context) files:** These files assign SELinux labels (types) to filesystem objects. Each file or directory in the system that needs to be managed by SELinux is mapped to a type. *Example:* Assign the `modbus_conf_t` type to a configuration directory:

  ```
  /etc/edgex/device-modbus(/.*)?
  gen_context(system_u:object_r:device_modbus_etc_t, s0)
  ```

- **.if (Interface) files:** Interface files provide reusable abstractions and definitions, allowing different policy modules to reference common rules or types. They simplify policy maintenance and help enforce consistent rules across multiple modules. *Example:* Define an interface to allow reading a configuration file:

```
interface(‘device_modbus_log_access’) {
    allow $1 device_modbus_log_t:file { append write open };
}
```

Together, these files define the behavior of the custom SELinux policy and are compiled into a policy module that are loaded into the kernel with the following commands:

```
checkmodule -M -m -o device_modbus.mod device_modbus.te
semodule_package -o device_modbus.pp -m device_modbus.mod
semodule -i device_modbus.pp
```

In a Yocto-based build, these steps are handled automatically: the `.te`, `.fc`, and `.if` files are compiled into policy modules during the image build process and installed on the target system.

8.2 Analysis of Existing Policies

Common SELinux commands used to understand the status of how SELinux is implemented in the device under test include:

```
getenforce        # shows current mode (Enforcing, Permissive, Disabled)
setenforce 1/0    # enable/disable enforcing
sestatus          # display current policy status
```

The results shows that SELinux is enable and in Enforcing mode in standard condition.
The device under study comes with a set of pre-defined SELinux policies. An initial analysis of these policies revealed that they are generally functional and allow standard operation of services.
To analyze the policies, the following workflow was used:

1. Explore the policy repository (where the `.te`, `.fc`, `.if` files are) to understand existing types and domains.
2. Check labels assigned to active processes using `ps -Z` and `ls -Z`.
3. Monitor AVC denied messages during service execution:

   ```
   ausearch -m avc -ts today
   ```

4. Identify excessive permissions and unnecessary access of some services.

Overall, the default policies provide a solid baseline, but opportunities exist to define more specific constraints for critical services. Some of them run under overly permissive labels (e.g., `initrc_t`), granting them access to resources beyond what is strictly necessary. While functional, this reduces the security benefits of SELinux.

8.3 Writing Custom Policies for Specific Services

To improve security, custom SELinux policies for selected services were developed, focusing on reducing over-permissive labels and enforcing least-privilege access. As an example, the **Device-modbus** service, which handles communication with industrial Modbus devices is analyzed. A custom SELinux policy was created to restrict this service to only the files, directories, and network ports it requires, reducing over-permissive access and enforcing least-privilege operation.

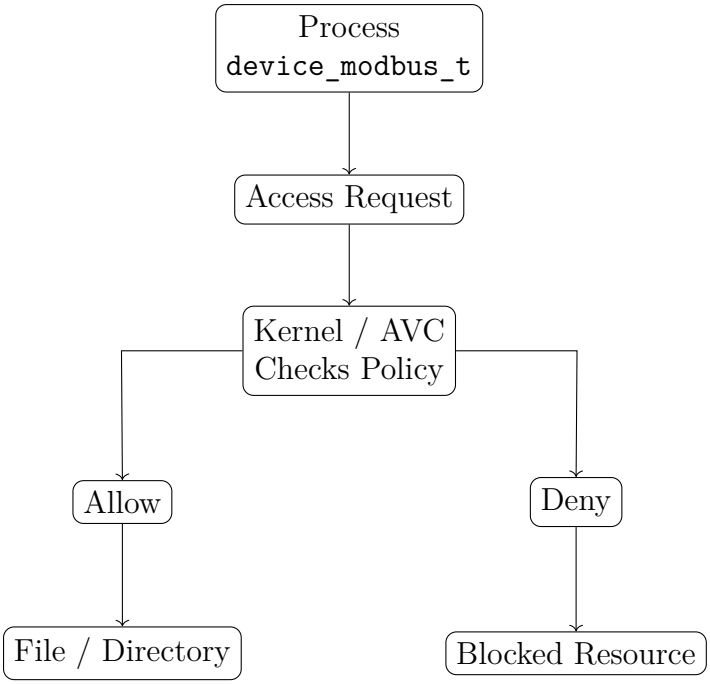

Figure 8.1: SELinux access decision flow for a process.

8.3.1 Defining Service Requirements

The first step is to understand what resources the service actually needs. This includes:

- Files and directories the service must read/write.

- Network ports it uses.
- Inter-process communication (IPC) requirements.

From this, the necessary SELinux types and domains are drafted to represent the service and its resources.

8.3.2 Iterative Policy Development

Policies are written incrementally. Initially, restrictive rules are defined, and the service is launched under this new policy in `Permissive` mode. Any access attempts denied by SELinux are captured in AVC logs. Each denied access is then evaluated, and an explicit `allow` rule is added if the access is required for normal operation. This process repeats until the service functions correctly without unnecessary permissions.

Example AVC Log

An example AVC denial for the Device-modbus service could look like this:

```
type=AVC msg=audit(1696000000.123:456):
avc:  denied  { read } for  pid=1234 comm="device_modbus"
name="other_config.conf" dev="sda1" ino=56789
scontext=system_u:system_r:device_modbus_t:s0
tcontext=system_u:object_r:other_service_etc_t:s0 tclass=file
```

This log indicates that the process `device_modbus`, domain `device_modbus_etc_t`, attempted to read a file labeled `other_service_etc_t`, which is denied by the current policy.

8.4 Testing and Validation of Policies

Policy validation relies heavily on monitoring AVC denials during normal service operation. Services are started and exercised with typical workflows, and any denials are reviewed. Policies are adjusted iteratively, and the process is repeated until:

- The service operates correctly under the custom policy in `Enforcing` mode.
- Only the minimal required accesses are allowed.
- No unnecessary privileges are granted.

Automated tools such as `audit2allow` were used cautiously to translate AVC denials into candidate rules, but manual review ensured that only truly required permissions were granted.

```
tail -f /var/log/audit/audit.log    # real-time AVC monitoring
audit2allow -w -a                   # suggestions for missing rules
```

8.5 Security Impact and System Behavior Changes

Implementing custom SELinux policies enhances system security by enforcing strict access controls tailored to each service. The main impacts observed are:

- Reduced risk of privilege escalation and lateral movement between services.
- Minimal disruption to service operation once policies are correctly tuned.
- Improved observability of unintended access attempts via AVC logs.

Previously, a Modbus service could read configuration files of other services, posing a risk of compromise. After confinement in `modbus_t`, access to unrelated directories such as `/etc/other_service` is blocked.
Overall, custom policies provide a practical way to tighten security without disrupting functionality, demonstrating the value of SELinux in modern embedded and IoT systems.

Chapter 9

Compliance with Directive EN 18031

9.1 Overview of EN 18031

EN 18031 specifies common security requirements and related assessment criteria for internet-connected radio equipment. EN 18031 is a document that specifies common security requirements and related assessment criteria for internet connected radio equipment. [115] The directive includes three key articles relevant to cybersecurity that CEN/CENELEC used to publish the three standards on January 30, 2025:

- **Article 3.3(d) – Network protection:** Devices must include safeguards to prevent damage to the network or its components and avoid interference with other devices or services (EN 18031-1). This ensures that devices cannot compromise networks, cause malfunctions, or be exploited in cyberattacks.
- **Article 3.3(e) – Strengthening personal data protection:** Radio devices must incorporate measures to protect user privacy and personal data (EN 18031-2). This safeguards confidentiality and integrity of personal information processed or transmitted by the devices.
- **Article 3.3(f) – Reducing the risk of fraud:** Devices must implement functionalities such as enhanced user authentication controls to minimize fraudulent transactions (EN 18031-3). This prevents devices from being used to steal sensitive data, credentials, or bank information.

These cybersecurity requirements aim to improve technological resilience, ensuring a high level of security for radio devices on the European market. They reduce the risk of cyberattacks on IoT devices, smartphones, and other radio equipment, enhance consumer trust in connected devices, and help prevent issues that could

compromise both network security and end-user privacy.
The **Directive 2014/53/EU (Radio Equipment Directive, RED)** provides the main European regulatory framework for placing radio equipment on the market, establishing essential requirements for safety, electromagnetic compatibility, and efficient use of the radio spectrum. To respond to the growing number of connected devices, the EU strengthened the directive through the adoption of **Delegated Regulation (EU) 2022/30**, introducing new obligations on cybersecurity, privacy, and fraud prevention, effective from August 1, 2025.
The RED Directive regulates the EU market entry of devices such as mobile phones, wireless routers, wearable devices, baby monitors, remote controls, and smart IoT devices. It defines three categories of essential requirements: safety of persons and property, electromagnetic compatibility, and efficient use of the radio spectrum.
Additionally, it allows for optional functional requirements (Art. 3.3) activated through delegated acts, including network protection, privacy, and fraud prevention. Following these standards provides a presumption of conformity with the RED Directive, while non-standard approaches require assessment with a notified body. Manufacturers and other operators in the supply chain must ensure that radio equipment:

- Is designed and manufactured in compliance with the essential requirements under Article 3.
- Can be used in at least one Member State without violating applicable radio spectrum regulations.
- Complies with EN 18031 standards or is prepared for assessment by a notified body.

Overall, EN 18031 establishes unified guidelines to design, implement, and maintain secure and reliable systems, reducing the risk of compromise and ensuring the protection of sensitive information.
Member States must establish effective sanctions for violations, including administrative and criminal penalties. Sanctions must be effective, proportionate, and dissuasive, applying to manufacturers, importers, and distributors who place non-compliant products on the market. In general, the updated European regulatory framework strengthens digital security and consumer trust by ensuring that radio equipment is designed safely, responsibly, and transparently[116, 115].

9.1.1 EN 18031-1

The EN 18031-1 standard provides a comprehensive framework for securing critical systems, defining requirements and best practices to ensure protection, integrity, and resilience. The document, after defining what network and security assets are, covers several key aspects, including:

- **Access control mechanisms:** Define where control is needed and ensure that the implementation of such is secure.
- **Authentication mechanisms:** In case access control is applied, ensure secure use of authenticators and its update, password strength, and protection against brute-force attacks.
- **Secure update mechanisms:** Ensure the update process is present and protected.
- **Secure storage mechanisms:** Protect data stored on the device from corruption and theft.
- **Secure communication mechanisms:** Guarantee confidentiality, integrity, and replay protection of transmitted data.
- **Resilience mechanisms:** Mitigate the effects of Denial-of-Service (DoS) attacks and restore systems to a defined state. Tests need to be provided.
- **Network monitoring mechanisms:** Detect to DoS attacks and other threats.
- **Traffic control mechanisms:** Ensure that data are not generated by a compromised equipment.
- **Confidential cryptographic keys:** Ensure generation follows best cryptographic practices.
- **General equipment capabilities:**
  - Up-to-date software and hardware with no publicly known exploitable vulnerabilities.
  - Limit exposure of services via network interfaces; configure and document needed services, and block unused interfaces.
  - Input validation for all input received via external interfaces.

9.2 Current Relevance and Impact

As of August 1, 2025, compliance with EN 18031 becomes mandatory for all internet-connected or functionally sensitive radio devices placed on the European market [115]. This regulatory enforcement ensures that devices incorporate protections against network compromise, unauthorized access, and fraud, while also safeguarding privacy and personal data. By adhering to these standards, manufacturers not only mitigate security risks and prevent potential cyberattacks but also enhance consumer trust in their products.
Furthermore, compliance facilitates market access within the EU by providing a presumption of conformity with the Radio Equipment Directive (RED), ensuring that devices meet the essential requirements for safety, electromagnetic compatibility, and efficient use of the radio spectrum.

9.3 Application of the directive to the case study

The device examined in this thesis falls exclusively under the scope of EN 18031-1. It is categorized as network equipment, which means that only requirements related to network and security assets are in scope of the assessment.
The device does not process personal data and is not used to perform any kind of monetary transaction, therefore EN 18031-2 and EN 18031-3 are not relevant.
Table 9.1 summarizes the compliance status of the ten requirements defined in EN 18031-1.

9.4 Gap analysis and implemented improvements

The analysis identified a few requirements that were initially only partially satisfied. The main gaps and the measures implemented to achieve compliance are summarized below:

- **Authentication:** Brute-force protection for certain passwords was missing. This has now been implemented, ensuring that authentication mechanisms are fully robust.
- **Resilience:** Improvements have been made to improve resilience. Further enhancements can be considered in future updates.
- **Network monitoring:** The system still relies on the SIM modem module for GTP monitoring since it is done following best practice. An additional firewall rule was implemented to filter ICMP packets. The solution provides a baseline level of protection.

- **CVE management:** CVEs in the Yocto environment are updated regularly, and automatic scanning can be added if required to further enhance security.

The device now demonstrates a substantial alignment with EN 18031-1 requirements and can be certificated as compliant.

Requirement	Short description of implementation	Compliant
Access control	Enforcement of both DAC and MAC policies	Yes
Authentication	Appropriate mechanisms with secure handling of authenticators and strong password	Yes
Secure updates	Mender OTA with A/B partitioning, dm-verity, and Secure Boot	Yes
Secure storage	Dedicated encrypted partition with dm-crypt, LUKS1, and CAAM	Yes
Secure communication	TLS-secured channels and WPA2 protection	Yes
Resilience	Limited testing; external penetration tests performed	Partial
Network monitoring	SIM modem supports GTP; third-party penetration tests covered ARP and ICMP scenarios	Partial
Traffic control	firewalld rules combined with OpenVPN tunneling	Yes
Cryptographic keys	Secure key generation following best practices (RSA 4096, ED25519)	Yes
General capabilities	Regular CVE monitoring with Yocto, unnecessary interfaces disabled, input validation enforced	Yes
Cryptography	Best cryptographic practices across all security-critical components	Yes

Table 9.1: Compliance of the case study device with EN 18031-1 requirements

Chapter 10

Artificial Intelligence in Embedded Cybersecurity

Artificial Intelligence (AI) is emerging as a powerful technology in cybersecurity, enabling the automation of repetitive tasks, the acceleration of threat detection and response, and the improvement of overall accuracy in defending against cyber-attacks. Advisory organizations such as the National Institute of Standards and Technology (NIST) advocate for more proactive and adaptive security approaches, emphasizing real-time assessments, continuous monitoring, and data-driven analysis to strengthen protection and resilience.

AI is an intriguing tool that can analyze millions of events in real time, track a wide range of threats, and anticipate potential attacks, supporting timely and effective countermeasures. For this reason, it is increasingly integrated into cybersecurity frameworks, either to automate security operations or to enhance the effectiveness of human analysts. The growing enthusiasm of researchers from both AI and cybersecurity fields have resulted in quite a number of studies to solve problems related to the identification, protection, detection, response and recovery from cyberattacks.

At the same time, the same features that make AI valuable for defenders can also be exploited by adversaries. Attackers may use AI to automate large-scale attacks, improve phishing campaigns, identify vulnerabilities more efficiently, or even develop adaptive malware capable of evading traditional defenses. This duality of AI is fueling what many researchers describe as an "AI arms race", where both attackers and defenders continuously adopt and adapt AI technologies to gain an advantage. [117]

10.1 Using AI to benefit cybersecurity

In the digital age, cybersecurity has become more important than ever, as increasingly sophisticated threats can no longer be effectively mitigated with traditional techniques alone. AI is uniquely suited to perform tasks such as detecting cyberattacks more accurately than human analysts, prioritizing responses based on real-world risk. It can identify and flag suspicious emails or messages commonly used in phishing campaigns, simulate social engineering attacks to help security teams uncover potential vulnerabilities before they are exploited, and rapidly analyze incident-related data to enable swift containment of threats.
Organizations are adopting Artificial Intelligence (AI) to enhance their security strategies. Stopping breaches before they occur would not only help protect the data of individuals and companies, but also lower IT costs for businesses.

10.1.1 Threat Detection and Response

One of the most important applications of AI in cybersecurity lies in threat detection and retaliation. Machine learning algorithms and natural language processing can process enormous datasets to identify patterns and anomalies that could point to the presence of a cyber threat. This can be achieved by deploying intrusion detection systems, which use AI-powered algorithms to monitor network traffic for suspicious behavior that may indicate a security compromise. Moreover, advanced persistent threats (APTs) can be hidden within networks. In addition, predictive analytics enables organizations to anticipate and neutralize threats before they materialize, allowing for a more proactive defense posture.
A particularly challenging domain is botnet detection, which often relies on pattern recognition and proxy synchronization analysis. Botnets are typically controlled by a master script and involve a large-scale of compromised devices and "users" performing the same query to a server during a single attack. Since it is very difficult to contain or even identify by a human, automated systems are used to identify these connection patterns almost instantly, enabling rapid detection of distributed denial-of-service (DDoS) attacks and timely mitigation. [117]

10.1.2 Vulnerability and Risk Management

Vulnerability management is an essential use of AI in cybersecurity. Automated vulnerability scanning solutions can identify and prioritize weaknesses that need to be fixed immediately. AI also supports automated penetration testing, policy enforcement, and patch management, reducing the time and resources required to maintain a secure infrastructure. By accelerating these processes, organizations can mitigate risks more effectively and minimize their exposure to known vulnerabilities.

At the same time, AI plays an increasingly important role in compliance and governance. It can monitor adherence to regulations and internal policies, assess hazards by analyzing massive amounts of data to detect potential dangers and weaknesses and suggest appropriate mitigation solutions. Moreover, automated policy enforcement ensures that violations are promptly identified and addressed, strengthening governance frameworks and improving overall resilience.

10.1.3 Speed, Accuracy and Scalability

One of AI's main advantages for cyber security is its ability to deliver real-time threat identification and response. By examining network traffic, AI algorithms can quickly identify any unusual behavior that might indicate a security flaw, allowing security teams to react promptly and mitigate damage. Compared to traditional methods, AI-powered solutions are also more accurate in detecting potential risks, reducing false positives and uncovering threats that human analysts might overlook. Since it is faster and more precise as compared to human analysts at processing and analyzing the workload for security experts is lessened, allowing them to concentrate on jobs that are more challenging and sophisticated. In addition, AI-powered security solutions they can analyze data from numerous sources and handle huge traffic volumes without noticeably degrading performance so they provide higher scalability.
Another significant advantage of AI in cybersecurity is the ability to improve and learn over time. By continuously monitoring security systems and reviewing security incidents, AI algorithms can discover patterns and gain insights that improve their efficacy and accuracy making it easier to stay one step ahead of hackers. [118]

10.2 AI-powered automated attacks

Even though AI has become a fundamental technology in every enterprise IT toolkit, it has simultaneously turned into a weapon for cybercriminals.
AI-driven cyberattacks make use of artificial intelligence and machine learning (ML) models to automate, accelerate, and optimize different stages of an attack. This can involve detecting weaknesses, launching campaigns along specific attack vectors, progressing through attack chains, planting backdoors in systems, stealing or manipulating information, and disrupting normal operations.
Malicious actors exploit AI-based tools to craft convincing phishing emails, social engineering attempts, and other AI-generated content capable of bypassing traditional security defenses. Such attacks are becoming more advanced, deceiving users into revealing sensitive data or carrying out harmful actions.
As with all machine learning systems, algorithms used in AI-enabled attacks can

improve and adapt over time. This allows AI-powered threats to evolve in ways that evade detection or establish attack patterns that security solutions fail to recognize. AI-powered cyberattacks share several key characteristics:

- **Attack automation:** AI tools allow attacker to automate attack research and execution.
- **Efficient data gathering:** AI can automate or accelerate reconnaissance, enabling the research phase to be shorter and potentially improve the accuracy and completeness of the adversaries analysis.
- **Customization:** AI Data Scraping capability can be used to create personalized, relevant, and timely messages that serve as the foundation for phishing attacks and other attacks that leverage social engineering techniques.
- **Reinforcement learning:** AI algorithms continuously change to provide more accurate insights for corporate users and help adversaries improve their techniques or avoid detection.
- **Employee targeting:** AI can be used to identify individuals inside an organization that are high value targets.

[119]

10.2.1 Type of AI-Powered Attacks

AI-driven social engineering attacks

AI-driven social engineering attacks exploit AI algorithms to support in the research, planning, and execution of these attacks.
Social engineering refers to cyberattacks that aim to manipulate human behavior to achieve malicious objectives, such as sharing sensitive data, transactions of money or high-value items or gaining access to systems, applications, databases, or devices. In this context, AI algorithms can assist attackers by identifying corporate victims and individuals within an organization who may serve as a gateway into the IT environment. Most of these algorithms work by developing a persona and corresponding online presence to engage the target, designing scenarios that appear plausible and producing tailored content such as personalized messages or synthetic multimedia to establish trust and manipulate the target. [119]

AI-driven phishing attacks

AI-driven phishing attacks exploit generative AI to create highly realistic and tailored emails, SMS messages, phone calls, or social media accounts with the aim

of deceiving victims. The goal is the same as traditional social engineering but AI can also automate real-time communication, like chatbots, making the interaction with victims nearly indistinguishable from human dialogue.
Attackers can use these tools to attempt to connect with countless individuals simultaneously. In many cases, chatbots pose as customer support or service agents in an attempt to gather personal information and account credentials, reset account passwords, or access a system or device.
Another type of AI-generated attack is performed by manipulating social media and online platform to amplify propaganda, spread misinformation, or manipulate user behavior. This presents a significant challenge for businesses and consumers as it can lead to reputational damage, loss of trust, and even negatively influence the public opinion.
The impact of a successful phishing attempt can be severe. Once the threat actor gains knowledge of sensitive information, they can use it for various malicious purposes, such as identity theft, financial fraud, or unauthorized access to protected systems or accounts. Protection requires a proactive and multilayered approach including user education on the risks of phishing, adoption of multi-factor authentication, timely updates of security software, and the deployment of spam filters capable of detecting and blocking suspicious messages.
In conclusion, AI-enhanced phishing represents an intense and evolving threat to both individuals and organizations. It is crucial to remain vigilant and exercise caution when dealing with digital communications, especially those requesting sensitive information. [120, 119]

Deepfakes

A deepfake is an AI-generated video, image, or audio file that is designed to deceive people. While deepfakes commonly appear on the internet for entertainment or to confuse, they can also be used maliciously. In the context of cyberattacks, deepfakes are frequently employed as part of social engineering attacks. For instance, an attacker may use existing footage of a corporate leader or client to produce a convincing video or voice recording that instructs a target to perform actions such as changing a password, transferring funds, or granting system access.
AI-generated deepfakes leverage the ability to manipulate audio, video or image content mimicking real people, making it appear legitimate. These falsified materials can be distributed widely and almost instantly, including on social media platforms to create stress, fear or confusion among those who consume it. The implications can be severe both personally and professionally, including identity theft and defamation.
To mitigate the risk, people should exercise caution with urgent voice or video messages asking for money or sensitive information. It is important to only respond

to known contacts and verify the identity of anyone asking for confidential details before giving them. Personal information should never be shared in response to unsolicited communications. [119, 120]

Adversarial AI/ML

Adversarial artificial intelligence (AI) or machine learning (ML) refers to attempts by malicious actors to disrupt the performance, deceive, or decrease the accuracy of AI/ML systems through manipulation or deliberate misinformation.
Such attacks exploit weaknesses in model design, training, or deployment. Common adversarial AI/ML techniques include:
Poisoning attacks. These target the the AI/ML model training data used to build the model itself. Attackers can compromise the model's accuracy or objectivity by injecting fake or misleading information into the dataset.
Evasion attacks. These focus on the input data provided to an AI/AM model. These attacks apply subtle changes to the input to cause the model to misclassify information and negatively impact the model's predictive capabilities.
Model tampering. These attacks manipulate the parameters or structure of a pre-trained model. An adversary makes unauthorized changes to the model to compromise its ability to create accurate outputs or embed malicious behaviors within the system. [119, 120]

Malicious GPTs

A generative pre-trained transformer (GPT) is a class of AI model designed to generate coherent and context-aware text in response to user prompts.
A *malicious GPT* refers to a modified or deliberately misused version of such a model that produces harmful, deceptive, or misleading outputs.
A malicious GPTs can be exploited to automate and scale malicious activities, generate attack vectors, such as malware code or scripts, or support attack materials. This capability lowers the barrier of entry for cybercriminals and increases the efficiency of coordinated attacks, amplifying risks for both individuals and organizations. [119, 120]

Ransomware attacks

AI-enabled ransomware is a type of ransomware that incorporates artificial intelligence to enhance its effectiveness and automate stages of the attack lifecycle. AI can be used to research potential targets, identify system vulnerabilities, or optimize the encryption of data once access has been achieved. AI can also be used to adapt dynamically the ransomware files over time, making them more difficult to detect with traditional cybersecurity tools. These capabilities significantly increase

the challenges faced by security teams and professionals.
The integration of AI with natural language processing (NLP) also enables attackers to craft realistic communications that support ransomware distribution, such as phishing emails or fraudulent messages with minimal linguistic errors. This lowers the likelihood of early detection and increases the probability of successful delivery.
Overall, AI-enhanced ransomware illustrates how artificial intelligence can amplify the scale, speed, and stealth of cyberattacks, requiring enhanced security measures to combat them. [119, 120]

Multi-Factor Authentication Bypasses

Multi-factor authentication (MFA) is a security measure that help protect against unauthorized access to account and system by requiring more than one type of verification.
However, new techniques in artificial intelligence have introduced new ways to bypass MFA. AI-based tools can simulate legitimate user behavior and generate convincing authentication requests that trick users into disclosing credentials. By analyzing previous authentication requests, these tools can create variations that closely mimic real requests, making it difficult for users to understand the difference between genuine and malicious.
Generative AI can also target messages to make them appear to originate from trusted sources. Such messages exploit psychological factors such as urgency or curiosity, prompting users to willingly provide their credentials bypassing MFA protections.
Mitigation strategies include using security tools that can spot unusual authentication behavior and keeping systems under continuous monitoring for signs of compromise. User awareness is just as important: regular training helps people recognize and resist phishing attempts aimed at bypassing MFA.
MFA is still a key defense, but new bypass methods show the need for flexible security measures that mix technical protections with informed user behavior. [120]

Password hacking and Investment Scams

Password hacking Cybercriminals are starting to rely on AI-enhanced algorithms to accelerate password deciphering. These techniques allow for faster and more accurate guessing of credentials, increasing the efficiency and profitability of brute-force and dictionary-based attacks. As a result, attackers are provided with access to multiple accounts, particularly when users reuse or recycle credentials across platforms.
Mitigation strategies include employing password managers which will create unique, long and complex passwords, using encrypted storage and enabling MFA to add an additional security layer.

AI-powered investment scams Cybercriminals also exploit generative AI to design fake websites and social media profiles that promote fraudulent investment opportunities. These schemes often promise low risk with high returns, luring victims into transferring funds before the attackers disappear, with no return of the money. The consequences extend beyond financial loss, eroding trust in online platforms and digital financial services.
Countermeasures involve verifying investment opportunities through trusted channels, avoiding unsolicited online promotions, and consulting professional advisors before transfering funds.

10.2.2 Challenges for Security Teams

Security teams face multiple challenges when defending against AI-generated cyberattacks:

- **Highly dynamic threats:** AI attacks continuously modify tactics, techniques, and procedures to avoid detection, making static defenses hard.
- **Detection evasion:** Attackers use AI to optimize malware, phishing, and network traffic, hiding from signatures or anomalies based detection systems.
- **Increased scale and speed:** AI enhances the automation and efficiency of attacks, making it difficult for human analysts to keep up.
- **Blindspots in modeling:** AI models may fail to detect new attacks that are not represented in their training data.
- **Difficulty attributing attacks:** Constantly changing tactics, techniques and procedures make it harder to trace attacks to specific groups.
- **Skill gaps:** Defending against AI attacks requires specialized data science and ML expertise.
- **Lack of labeled training data:** Effective defensive AI relies on large volumes of attack data, which is, at the moment, scarce.
- **Data isolation:** Insights learned from attacks in one organization are often not shared, reducing the effectiveness of others defenses.
- **Explainability issues:** Inability to explain the reasoning behind AI model detections interfering with investigation, tuning, and collaboration.

10.2.3 Techniques Used in AI-Generated Texts

AI-generated texts use various techniques to create realistic and convincing content for malicious purposes. Natural Language Processing (NLP) is often used since it allows AI models to analyze linguistic patterns and generate human-like text.
Generative AI tools, trained on large datasets, leverage ML to mimic human writing styles, producing content that appears authentic.
Attackers can also use prompt engineering to manipulate AI models by providing specific cues, increasing the likelihood of successful phishing attempts.
Finally, adversaries exploit the natural imperfections of language models, such as occasional grammatical errors or inconsistencies, to make the generated content seem more realistic and genuine. [120]

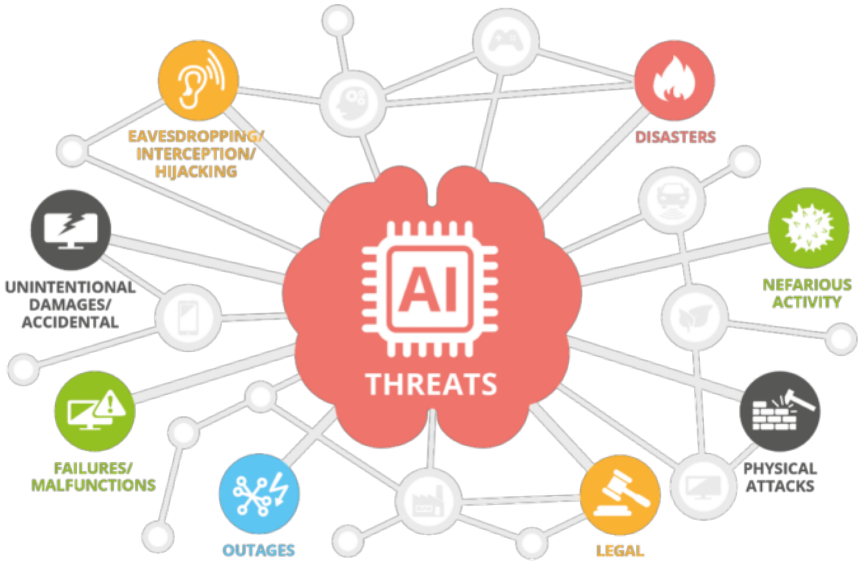

Figure 10.1: AI Threats

10.3 Mitigation

These attacks challenge traditional security measures and require security teams to adapt and strengthen their defenses continuously. It is crucial for cybersecurity professionals and businesses to understand the potential threats posed by AI-generated attacks and implement appropriate security measures to mitigate the risks. Policymakers and national security agencies must adapt their threat models and security standards to address the evolving landscape of AI-generated threats. [119]

Category	**Recommended Actions**
Continuous Security Assessments	Deploy monitoring, IDS and endpoint protection. Establish baselines for system and user activity, integrate UEBA, and monitor AI/ML inputs and outputs for anomalies in real-time.
Incident Response Planning (NIST)	Develop a document covering preparation, detection/-analysis to determine severity, containment/eradication to limit the spread and apply patches, and recovery to prevent future attacks.
Employee Awareness	Train employees on AI-powered attacks, social engineering, deepfakes, and adversarial AI techniques. Teach staff to recognize suspicious activities or outputs.
AI-Powered Defensive Solutions	Leverage AI for monitoring, analysis, patching, prevention, and remediation. Set alerts for high-risk activities and prioritize responses efficiently.
Ethical and Responsible AI Use	Ensure human supervision, technical robustness, data privacy and governance, transparency, diversity/non-discrimination, social and environmental well being, and accountability of AI systems.

Table 10.1: Key Recommendations for Defending Against AI-Generated Cyberattacks.

10.4 Risks, ethical concerns, and future directions

Cybercrime, as crime itself, is a human problem, and generative AI is simply another tool in an attacker's arsenal. While threat actors disregard ethical considerations, even legitimate organizations risk breaching ethical standards if AI models are developed or used carelessly. Laws like Europe's General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), even if developed before the rise of generative AI, serve as valuable guidelines for informing ethical AI strategies.

10.4.1 Privacy and Data Protection

Generative AI models are often trained on vast amounts of internet data, including copyrighted material or inadvertently scraped personal information. This raises concerns about:

- **Data leaks and misuse:** Sensitive data collected for training may be exposed to unauthorized parties.
- **Biometric and personal data:** Systems using AI for authentication like facial or fingerprint recognition, process highly sensitive data, which, if compromised, has long-term consequences.
- **Re-identification risks:** Cross-analysis makes it feasible for anonymous data to be linked back to individuals.

10.4.2 Bias, Fairness, and Accountability

AI systems inherit biases from their training data. In cybersecurity, this can result in unfair profiling, for instance if an AI-powered email filter disproportionately flags messages from a particular cultural or linguistic group. Furthermore, when AI autonomously blocks IPs or quarantines files, errors raise accountability questions that someone need to be held responsible for between developers, deploying organization, or system operators.

10.4.3 Transparency and Explainability

Many AI models function as "black boxes," making it difficult for professionals to understand or justify their decisions. This lack of interoperability creates the following:the following:

- **Low trust:** Stakeholders may be reluctant to rely on unexplained AI outcomes.
- **Hidden biases:** If the logic of the model is difficult to understand, discriminatory patterns are harder to identify and mitigate.
- **Security gaps:** Attackers may exploit vulnerabilities in opaque models through inputs.

Techniques, such as simplified models, visualization, and rule-based outputs, are easier to explain so they are critical to improve transparency.

10.4.4 Human Oversight and TEVV

AI should augment, not replace, human intelligence. Testing, Evaluation, Validation, and Verification (TEVV) are vital throughout the development cycle. Human supervision ensures that models remain accountable, biases are reduced through diverse auditing, and AI drift, which is self-reinforcing errors from training on its own outputs, is avoided.

10.4.5 Job Displacement and Societal Impacts

Automation of threat detection and routine tasks risks displacing cybersecurity professionals. Rather than reducing personnel, organizations should reinvest savings into retraining staff for AI-adjacent roles. This human-centered approach sustains both technical resilience and ethical integrity.

10.4.6 Best Practices for Ethical AI in Cybersecurity

Practice	**Description**
Transparent communication	Ensure stakeholders clearly understand AI's capabilities and limitations.
Bias mitigation	Regularly audit training data and models to reduce discrimination.
Accountability frameworks	Define roles and responsibilities for AI-driven actions.
Responsible data handling	Collect only necessary data, anonymize sensitive information, and secure it with encryption.
Continuous ethical training	Keep teams updated on evolving AI ethics standards.
Regular audits and monitoring	Review AI systems periodically to identify new risks.
Collaboration with the AI community	Share knowledge and best practices to address ethical issues collectively.

Table 10.2: Best practices for ensuring ethical AI in cybersecurity

[121, 122, 123]

10.5 Future Directions

The future development of AI in cybersecurity will mostly depend on the ability to reconcile technical robustness with ethical responsibility.

On the technical side, one of the main priorities is the improvement of AI models that are capable of resisting manipulation attempts to deceive detection systems. The adoption of real-time and multi-modal detection capabilities, allows AI systems to combine heterogeneous data sources, from network traffic to IoT telemetry, so that it AI can provide faster and more accurate responses to evolving threats. Moreover, combining AI-driven methods with traditional rule and signature based techniques represents a path toward more reliability and resilience.
Future AI systems must be designed to operate in large-scale, distributed environments such as cloud infrastructures, edge networks, and critical IoT ecosystems. Additionally, privacy preserving techniques need to be improved to ensure that the growing reliance on sensitive data does not undermine fundamental rights.
Another important point is the design of explainable AI models, allowing both experts and non-experts to understand the reasoning behind automated decisions. This cannot be done if organization do not start to collaborate and facilitate the study of already compromised models. On the ethical side, AI should not be considered a substitute for human knowledge and expertise, but a tool that enhances analytical capacity. Developing explainable and transparent models remains essential. Ethical and legal considerations, ranging from fairness and inclusivity in training datasets to accountability for automated decisions, must not be viewed as suggestion but as foundational elements.
In summary, the future of AI in cybersecurity requires progress along multiple axes: technical robustness, scalable deployment, human-centered design, and ethical governance. Only by aligning these dimensions can AI truly strengthen cybersecurity and provide resilient, trustworthy defenses against an increasingly complex threat landscape.

Chapter 11

Conclusion

11.1 Summary of Findings

This thesis has explored the security of embedded IoT systems, using a Yocto-based gateway device as a real-world device. The assessment confirmed that the device is securely designed, with no critical vulnerabilities detected except for the potential brute-force attack on the dashboard password.
All other findings are considered minor issues related to configuration choices and implementation practices, such as password management, filesystem permissions, firewall zoning, and VPN settings. These issues were either remediated or addressed with clear recommendations to further strengthen the device's security posture.
Through a structured validation and some penetration testing campaign, it was demonstrated that integrating security mechanisms such as secure boot, encrypted storage, fine-grained access control, and hardened OTA update procedures significantly improves system resilience. CVE mapping and regular monitoring revealed that at the time of test vulnerabilities were patched while also highlighting the importance of continuous maintenance. The Yocto framework allowed the practical application of these security practices, showing that a flexible and modular environment is essential for embedding security at every layer of the system.

11.2 Main Contributions

The thesis makes several key contributions to the field of embedded cybersecurity:

1. **Integration of Standards and Best Practices** By mapping international standards (IEC 62443, ETSI EN 303 645, EN 18031-1 and more) to practical development workflows, the research provides a structured methodology for manufacturers and developers to align design and operational processes with

recognized security requirements.

2. **Empirical Validation on a Real Device** By using a working Yocto-based embedded platform, the research demonstrates how theoretical principles can be applied in practice, revealing both strengths and limitations of existing mechanisms, highlighting the importance of a layered security and providing quantitative assessments of residual risk.

3. **Forward-looking Analysis** Examination of AI applications illustrates dual-use scenarios: defensive uses such as anomaly detection and predictive threat analysis, and potential misuse for automated attacks, phishing attacks and deepfakes, highlighting challenges for future IoT security.

4. **Methodological Framework** The combination of standards review, compliance assessment, CVE mapping, penetration testing, and Yocto-based implementation provides a comprehensive and replicable methodology for embedded security evaluation.

11.3 Limitations

Despite its contributions, the study is subject to certain limitations due to restrictions set by the company that commissioned and retains ownership of the solution. Penetration testing was limited to non-destructive techniques, excluding advanced physical attacks such as fault injection, side channel, or invasive chip decapsulation. Resource constraints prevented evaluation across large-scale device fleets, and some performance-security trade-offs remain partially quantified.
EN 18031-1 compliance was largely achieved and long-term monitoring has been demonstrated to be implemented correctly. However, supply chain risks and the management of heterogeneous hardware configurations remain open challenges. Furthermore, the Yocto-based test environment, while realistic and modular, represents a single development scenario; replicating these findings across diverse hardware or software stacks would require additional effort. Fortunately, Yocto-based solutions are the most common for embedded devices.

11.4 Comparison with Related Work

Existing literature shows that most studies focus on theoretical threat models or isolated security mechanisms such as secure boot, encryption, or lightweight authentication. This thesis distinguishes itself by combining multiple layers of defense, validating them on a real embedded platform, and systematically linking them to regulatory and compliance requirements. The practical integration of

Yocto-based customization, CVE monitoring, and real-world penetration testing bridges the gap between theoretical models and industrial application.

11.5 Implications for Industry and Research

From an industrial perspective, this thesis aims to highlight the need for *security-by-design*. Manufacturers should not rely solely on checklists or isolated compliance audits. Security must be integrated continuously into every step of the development process up until deployment, and then maintenance workflows. The modularity and flexibility of frameworks like Yocto provide an effective means to implement, validate, and scale security practices across heterogeneous embedded platforms while also proving the possibility to easily implement layered security.
For the research community, the thesis highlights open questions such as efficient cryptographic algorithms for resource-constrained devices, scalable patch management systems, lightweight intrusion detection tailored to IoT, and methodologies to evaluate multilayered defenses in real-world scenarios.

11.6 Ethical and Societal Implications

IoT devices increasingly influence both private and public spaces, blurring the line between individual and collective security. Poorly secured devices can be exploited for large-scale attacks, underscoring the responsibility of manufacturers, regulators, and developers.
Artificial intelligence adds another layer of ethical concern. AI can enhance security through automation and predictive analysis, but also poses risks such as reduced transparency, accountability challenges, and potential workforce displacement. Moreover, malicious use of AI can be easier and more impactful than its deployment for cybersecurity defense. Addressing these implications requires collaboration between technical experts and policy makers but especially ethics experts.

11.7 Future Developments

Several promising directions emerge from this study:

- **Hardware-level Security** Investigation of side-channel resistance, tamper-proof modules, and trusted execution environments.
- **Lifecycle Security Management** Frameworks for secure provisioning, updates, and decommissioning at scale.

- **AI-driven Security** Real-time anomaly detection, automatic CVEs check, adaptive access control, and predictive vulnerability analysis.
- **Ethical and Societal Considerations** Responsible deployment of AI, transparency, and accountability.
- **Standards Evolution** Empirical evidence to refine EN 18031 and similar frameworks.
- **Platform-focused Research** Leveraging frameworks like Yocto for practical, scalable security across heterogeneous devices.

11.8 Closing Remarks

Securing embedded systems is an ongoing process that evolves with technology and attackers capabilities. This thesis has demonstrated that careful design, rigorous testing, and adherence to standards significantly improve device resilience to attacks. However, no system is perfectly secure; vigilance, adaptability, and collaboration between academia, industry, and regulatory bodies remain essential.
By combining theoretical foundations, real-world experimentation on a device deployed in a real-world environment, CVE analysis, and regulatory compliance evaluation, this research provides a roadmap for embedding security into IoT devices. It illustrates how bridging theory and practice is both feasible and necessary to advance the security of embedded systems in real-world contexts.

Bibliography

[1] OWASP Foundation. *OWASP Top Ten 2021*. https://owasp.org/Top10/. 2021 (cit. on pp. 6, 54, 55).

[2] *ISA/IEC 62443: Security for Industrial Automation and Control Systems*. 2024. URL: https://www.isa.org/isa62443/ (cit. on p. 8).

[3] *ETSI EN 303 645: Cyber Security for Consumer Internet of Things*. 2020. URL: https://www.etsi.org/standards/303645 (cit. on pp. 9, 12, 23, 24, 27, 40, 41, 43).

[4] IoT Security Foundation. *IoTSF Secure Design Best Practice Guides*. IoT Security Foundation. 2024. URL: https://iotsecurityfoundation.org/best-practice-guide-articles/ (cit. on pp. 9, 12, 17, 20, 24, 25, 29, 39, 43).

[5] National Institute of Standards and Technology (NIST). *IoT Device Cybersecurity Baseline*. Tech. rep. NIST SP 8259. NIST, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.8259.pdf (cit. on pp. 10, 24).

[6] National Institute of Standards and Technology (NIST). *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*. Tech. rep. NIST SP 8228. NIST, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.8228.pdf (cit. on pp. 10, 24).

[7] Gregg Lindemulder and Matt Kosinski. *What is Role-Based Access Control (RBAC)?* https://www.ibm.com/think/topics/rbac. Accessed: 2025-09-11. 2024 (cit. on p. 12).

[8] Medha Mehta. *Context-Aware Authentication: Meaning, Tools, Examples*. https://www.pomerium.com/blog/context-aware-authentication-meaning-tools-examples. Accessed: 2025-09-11. Sept. 2024 (cit. on p. 13).

[9] Amazon Web Services. *AWS Identity and Access Management User Guide: Introduzione al controllo accessi basato su attributi (ABAC)*. Technical Report IAM-UG-ABAC. Accessed: 2025-09-11. Amazon Web Services, 2024. URL: https://docs.aws.amazon.com/pdfs/IAM/latest/UserGuide/iam-ug.pdf (cit. on p. 13).

[10] F5. *What is Secure Remote Access?* https://www.f5.com/glossary/secure-remote-access. 2024 (cit. on p. 14).

[11] Fortinet. *What is Remote Access? How Does It Work?* https://www.fortinet.com/uk/resources/cyberglossary/remote-access. Accessed: 2025-09-11. 2024 (cit. on p. 14).

[12] Aliaksandr Kavalchuk. *Diving into JTAG — Security (Part 6)*. https://interrupt.memfault.com/blog/diving-into-jtag-part-6. Aug. 2024 (cit. on p. 16).

[13] Yogin Savani. *Understanding JTAG Security in Embedded Systems: Risks and Best Practices*. https://yoginsavani.com/understanding-jtag-security-in-embedded-systems-risks-and-best-practices/. Sept. 2024 (cit. on p. 16).

[14] OnLogic. *Secure Boot 101: A Guide to Protecting Your Edge Devices*. OnLogic, 2024. URL: https://www.onlogic.com/blog/secure-boot/ (cit. on pp. 17, 21).

[15] Mender. *Over-the-air (OTA) update best practices for industrial IoT and embedded devices*. Mender. 2025. URL: https://mender.io/resources/reports-and-guides/ota-updates-best-practices (cit. on pp. 24, 25).

[16] Pure Storage. *La relazione tra IoT e Big Data*. 2025. URL: https://www.purestorage.com/it/knowledge/big-data/internet-of-things-and-big-data.html (cit. on p. 27).

[17] SAEP ICT. *La sicurezza IoT per i tuoi dati aziendali industriali*. 2025. URL: https://www.saep-ict.it/magazine/sicurezza-iot-per-i-tuoi-dati-aziendali-industriali/#gref (cit. on pp. 27, 28).

[18] Talend. *Che cos'è l'Internet of Things?* 2025. URL: https://www.talend.com/it/resources/internet-of-things/ (cit. on pp. 27, 28).

[19] Joint Task Force. *Security and Privacy Controls for Information Systems and Organizations*. Tech. rep. NIST Special Publication (SP) 800-53, Rev. 5. Gaithersburg, MD: National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-53r5 (cit. on pp. 30, 43).

[20] Goran Milenkovic, Dr. Marnix Dekker, and ENISA. *Guideline on Security Measures under the EECC, 4th Edition*. Tech. rep. European Union Agency for Cybersecurity (ENISA), July 2021. URL: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20-%20Guideline%20on%20Security%20Measures%20under%20the%20EECC-%204th%20edition.pdf (cit. on p. 30).

[21] Kerry J. Kimberlin, Robert L. Martin, and Peter E. Mell. *Developing Cyber Resilient Systems: A Systems Security Engineering Approach, Volume 2*. Special Publication 800-160, Volume 2. National Institute of Standards and Technology (NIST), 2021. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2.pdf (cit. on pp. 30, 38, 40, 41).

[22] Stellarix. *Tecniche di mitigazione degli attacchi a canale laterale*. Tech. rep. Stellarix, 2025. URL: https://stellarix.com/insights/articles/mitigation-techniques-of-side-channel-attacks/ (cit. on p. 31).

[23] Paul Kocher, Joshua Jaffe, and Benjamin Jun. «Differential Power Analysis». In: *Advances in Cryptology—CRYPTO'99*. Springer, Berlin, Heidelberg, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25. URL: https://paulkocher.com/doc/DifferentialPowerAnalysis.pdf (cit. on p. 31).

[24] NCC Group. *Approcci alternativi per le contromisure contro l'iniezione di guasti*. Tech. rep. NCC Group, 2025. URL: https://www.nccgroup.com/research-blog/alternative-approaches-for-fault-injection-countermeasures-part-33/ (cit. on p. 32).

[25] R. Tietema. «4.21 - Large-Scale Industrial Coating Applications and Systems». In: *Comprehensive Materials Processing*. Ed. by Saleem Hashmi, Gilmar Ferreira Batalha, Chester J. Van Tyne, and Bekir Yilbas. Oxford: Elsevier, 2014, pp. 519–561. ISBN: 978-0-08-096533-8. DOI: https://doi.org/10.1016/B978-0-08-096532-1.00430-1. URL: https://www.sciencedirect.com/science/article/pii/B9780080965321004301 (cit. on p. 33).

[26] Katie Wohletz. *Ruggedized Solutions to Protect Your Electronics*. Accessed: 2025-10-03. 2022. URL: https://riversideintegratedsolutions.com/n/ruggedized-solutions-protect-electronics (cit. on p. 33).

[27] Holoware. *Sicurezza hardware: misure essenziali di cybersecurity*. Tech. rep. Holoware, 2025. URL: https://holoware.co/computer-hardware-security-essential-cybersecurity-measures/#:~:text=Secure%20Hardware%20Design%20and%20Manufacturing%20Tamper%2DEvident%20Packaging:,any%20unauthorized%20access%20can%20be%20easily%20detected. (cit. on p. 33).

[28] Elaine Barker and John Kelsey. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. NIST Special Publication 800-90A Revision 1. National Institute of Standards and Technology, 2015. DOI: 10.6028/NIST.SP.800-90Ar1. URL: https://doi.org/10.6028/NIST.SP.800-90Ar1 (cit. on p. 34).

[29] Namirial. *HSM Hardware Security Module: guida introduttiva*. https://focus.namirial.com/it/hsm-hardware-security-module-guida-introduttiva/. Oct. 2014 (cit. on p. 35).

[30] IBM. *VLANs in IBM Storage Fusion 2.4.0*. URL: https://www.ibm.com/docs/it/storage-fusion/storage/2.4.0?topic=connections-vlans (cit. on p. 38).

[31] Cloudflare. *What is a firewall?* URL: https://www.cloudflare.com/learning/security/what-is-a-firewall/ (cit. on p. 38).

[32] Cyber Security Centre. *Cloud Network Security Zones - ITSP.80.023*. URL: https://www.cyber.gc.ca/en/guidance/cloud-network-security-zones-itsp80023 (cit. on p. 38).

[33] IoT Security Foundation. *Vulnerability Disclosure Best Practice Guidelines, Release 2.0*. Tech. rep. Accessed: YYYY-MM-DD. IoT Security Foundation, Sept. 2021. URL: https://www.iotsecurityfoundation.org/wp-content/uploads/2021/09/IoTSF-Vulnerability-Disclosure-Best-Practice-Guidelines-Release-2.0.pdf (cit. on pp. 38, 41, 43).

[34] OWASP Cheat Sheet Series. *Input Validation Cheat Sheet*. 2025. URL: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html (cit. on p. 39).

[35] *ETSI 2024: Cybersecurity Standard for IoT*. 2024. URL: https://www.etsi.org/standards/2024 (cit. on pp. 40, 41, 43).

[36] Simon Chu, Justin Koe, David Garlan, and Eunsuk Kang. «Integrating Graceful Degradation and Recovery through Requirement-driven Adaptation». In: *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24)*. ACM, 2024, pp. 122–132. DOI: 10.1145/3643915.3644090. URL: https://dl.acm.org/doi/10.1145/3643915.3644090 (cit. on p. 40).

[37] Bugcrowd. *Application-Level Denial-of-Service (DoS)*. https://www.bugcrowd.com/glossary/application-level-denial-of-service-dos/ (cit. on pp. 40, 41).

[38] OWASP. *API Security Top 10*. 2023. URL: https://owasp.org/www-project-api-security/ (cit. on p. 41).

[39] OWASP Cheat Sheet Series. *Authentication Cheat Sheet*. 2023. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (cit. on p. 41).

[40] NIST. *Zero Trust Architecture*. Special Publication 800-207. National Institute of Standards and Technology, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (cit. on p. 41).

[41] International Electrotechnical Commission (IEC). *IEC 62443-4-2: Security for Industrial Automation and Control Systems – Technical Security Requirements for IACS Components*. Tech. rep. 2019 (cit. on p. 41).

[42] CVE Program. *Common Vulnerabilities and Exposures (CVE)*. 2025. URL: https://www.cve.org/ (cit. on p. 42).

[43] Tasmiha Khan and Michael Goodwin. *Che cosa sono le CVE (Vulnerabilità ed esposizioni comuni)?* IBM, July 2024. URL: https://www.ibm.com/it-it/think/topics/cve (cit. on pp. 42, 43).

[44] Kim Schaffer, Peter Mell, Hung Trinh, and Isabel Van Wyk. *Recommendations for Federal Vulnerability Disclosure Guidelines*. Tech. rep. Special Publication 800-216. National Institute of Standards and Technology (NIST), May 2023. DOI: 10.6028/NIST.SP.800-216. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-216.pdf (cit. on p. 43).

[45] Karen Scarfone and Peter Mell. *Technical Guide to Information Security Testing and Assessment (SP 800-115)*. Tech. rep. National Institute of Standards and Technology, 2008 (cit. on pp. 43, 53–55).

[46] Yuhao Wu, Jinwen Wang, Yujie Wang, Shixuan Zhai, Zihan Li, Yi He, Kun Sun, Qi Li, and Ning Zhang. «Your Firmware Has Arrived: A Study of Firmware Update Vulnerabilities». In: *Proceedings of the 32nd USENIX Security Symposium*. 2023. URL: https://www.usenix.org/system/files/sec23winter-prepub-484-wu-yuhao.pdf (cit. on p. 45).

[47] Ang Cui, Michael Costello, and Salvatore J. Stolfo. «When Firmware Modifications Attack: A Case Study of Embedded Exploitation». In: *Proceedings of the Network and Distributed System Security Symposium (NDSS 2017)*. 2017. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/03_4_0.pdf (cit. on p. 45).

[48] Seyed Ali Alavi, Hamed Pourvali Moghadam, and Amir Hossein Jahangir. «Beyond botnets: Autonomous Firmware Zombie Attack in industrial control systems». In: *International Journal of Critical Infrastructure Protection* 48 (2025), p. 100729. ISSN: 1874-5482. DOI: https://doi.org/10.1016/j.ijcip.2024.100729. URL: https://www.sciencedirect.com/science/article/pii/S1874548224000702 (cit. on p. 47).

[49] Eclypsium. *The Top Firmware and Hardware Attack Vectors*. 2025. URL: https://eclypsium.com/blog/the-top-5-firmware-and-hardware-attack-vectors/ (cit. on pp. 47, 48, 57).

[50] Jetpack. *How Weak Passwords Expose You to Serious Security Risks*. https://jetpack.com/resources/weak-passwords/. 2024 (cit. on pp. 49, 50).

[51] Jen Swisher. *Password Attacks: Types and How to Prevent Them*. https://www.sailpoint.com/identity-library/8-types-of-password-attacks. 2025 (cit. on pp. 50–52).

[52] Jetpack. *Password Attacks: The 9 Most Common Types & How to Prevent Them*. 2024. URL: https://jetpack.com/resources/password-attacks/ (cit. on pp. 50–52).

[53] TuringSecure. *Unencrypted Communications*. 2025. URL: https://turingsecure.com/knowledge-base/issues/unencrypted-communications/ (cit. on p. 53).

[54] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Prentice Hall, 2010 (cit. on pp. 53, 55).

[55] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. «A Survey of Man-In-The-Middle Attacks». In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 2027–2051 (cit. on pp. 53, 55).

[56] IBM. *Che cos'è un attacco man-in-the-middle (MITM)?* IBM, 2025. URL: https://www.ibm.com/it-it/think/topics/man-in-the-middle (cit. on p. 53).

[57] William Stallings. *Cryptography and Network Security: Principles and Practice*. 7th. Pearson, 2017 (cit. on pp. 53, 55).

[58] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World*. 2nd. Prentice Hall, 2002 (cit. on pp. 54, 55).

[59] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2002 (cit. on pp. 54, 55).

[60] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. «Security and Privacy Challenges in Industrial Internet of Things». In: *Proceedings of the 52nd Annual Design Automation Conference (DAC)*. ACM. 2015, pp. 1–6 (cit. on p. 55).

[61] ReasonLabs. *Understanding Memory-Based Attacks*. https://cyberpedia.reasonlabs.com/EN/memory-based.html. 2025 (cit. on p. 56).

[62] Contrast Security. *Buffer Overflow: Types and Prevention*. https://www.contrastsecurity.com/glossary/buffer-overflow. 2025 (cit. on p. 56).

[63] Imperva. *Che cos'è un buffer overflow: Tipi e prevenzione*. https://www.imperva.com/learn/application-security/buffer-overflow/. 2025 (cit. on p. 56).

[64] Fortinet. *Che cos'è un buffer overflow? Attacchi, tipi e vulnerabilità*. Fortinet. 2025. URL: https://www.fortinet.com/uk/resources/cyberglossary/buffer-overflow (cit. on p. 56).

[65] OWASP. *Format String Attack*. https://owasp.org/www-community/attacks/Format_string_attack. Accessed: 2025-09-11 (cit. on p. 57).

[66] Jayson E. Street, Kent Nabors, Brian Baskin, and Marcus J. Carey. *Dissecting the Hack: The F0rb1dd3n Network*. Syngress, 2010. ISBN: 978-1597495684 (cit. on p. 57).

[67] Nir Nissim, Ran Yahalom, and Yuval Elovici. «USB-based attacks». In: *Computers & Security* 70 (2017), pp. 1–15. DOI: 10.1016/j.cose.2017.05.002. URL: https://www.sciencedirect.com/science/article/abs/pii/S0167404817301578 (cit. on p. 58).

[68] NordVPN. *Cos'è un attacco cold boot*. https://nordvpn.com/it/cybersecurity/glossary/cold-boot-attack/. 2025 (cit. on p. 58).

[69] Dayeon Kim, Hyungdong Park, Inguk Yeo, Youn Kyu Lee, Youngmin Kim, Hyung-Min Lee, and Kon-Woo Kwon. «Rowhammer Attacks in Dynamic Random-Access Memory and Defense Methods». In: *Sensors* 24.2 (2024), p. 592. DOI: 10.3390/s24020592. URL: https://doi.org/10.3390/s24020592 (cit. on p. 58).

[70] PixelQA. *Common Network Protocol Vulnerabilities & How to Secure Your Network*. https://www.pixelqa.com/blog/post/network-protocol-vulnerabilities-security. 2025 (cit. on p. 59).

[71] Cloudflare. *Cos'è il DNS Cache Poisoning?* https://www.cloudflare.com/it-it/learning/dns/dns-cache-poisoning/. 2025 (cit. on p. 59).

[72] Osama Alsad and Qasem Abu Al-Haija. «DNS Cache Poisoning Attack Detection: A Systematic Review». In: *IET Conference Proceedings*. Vol. 2023. 44. 2024, pp. 426–432. DOI: 10.1049/icp.2024.0962. URL: https://doi.org/10.1049/icp.2024.0962 (cit. on p. 60).

[73] K. Chandrasekaran, Usha Divakarla, and C. K. Srinivasan. «DNS Cache Poisoning: Investigating Server and Client-Side Attacks and Mitigation Methods». In: *2023 Cyber Research Conference - Ireland (Cyber-RCI)*. 2023, pp. 1–8. DOI: 10.1109/Cyber-RCI59474.2023.10671556 (cit. on p. 60).

[74] OWASP. *HTTP Response Splitting*. https://owasp.org/www-community/attacks/HTTP_Response_Splitting. 2025 (cit. on p. 61).

[75] MITRE. *CAPEC-34: HTTP Response Splitting*. https://capec.mitre.org/data/definitions/34.html. 2025 (cit. on p. 61).

[76] Eman Salem Alashwali and Kasper Rasmussen. «What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS». In: *IET Conference Proceedings*. Vol. 44. 2023, pp. 426–432. DOI: 10.1049/icp.2024.0962. URL: https://doi.org/10.1049/icp.2024.0962 (cit. on p. 63).

[77] Cloudflare. *Attacco DDoS di tipo SYN Flood*. https://www.cloudflare.com/it-it/learning/ddos/syn-flood-ddos-attack/. 2025 (cit. on p. 63).

[78] L. Kavisankar and C. Chellappan. «A mitigation model for TCP SYN flooding with IP spoofing». In: *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*. 2011, pp. 251–256. DOI: 10.1109/ICRTIT.2011.5972435 (cit. on p. 64).

[79] Randall R. Stewart, Mitesh Dalal, and Anantha Ramaiah. *Improving TCP's Robustness to Blind In-Window Attacks*. RFC 5961. Aug. 2010. DOI: 10.17487/RFC5961. URL: https://www.rfc-editor.org/info/rfc5961 (cit. on p. 64).

[80] «Chapter 16 - IDS Evasion». In: *Hack Proofing Your Network (Second Edition)*. Ed. by David R. Mirza Ahmad et al. Second Edition. Burlington: Syngress, 2002, pp. 689–717. ISBN: 978-1-928994-70-1. DOI: https://doi.org/10.1016/B978-192899470-1/50019-7. URL: https://www.sciencedirect.com/science/article/pii/B9781928994701500197 (cit. on p. 65).

[81] Brad Woodberg, Mohan Krishnamurthy Madwachar, Mike Swarm, Neil R. Wyler, Matthew Albers, and Ralph Bonnell. «Chapter 10 - Attack Detection and Defense». In: *Configuring Juniper Networks NetScreen SSG Firewalls*. Ed. by Brad Woodberg, Mohan Krishnamurthy Madwachar, Mike Swarm, Neil R. Wyler, Matthew Albers, and Ralph Bonnell. Burlington: Syngress, 2007, pp. 479–549. ISBN: 978-1-59749-118-1. DOI: https://doi.org/10.1016/B978-159749118-1/50012-5. URL: https://www.sciencedirect.com/science/article/pii/B9781597491181500125 (cit. on p. 65).

[82] «Chapter 4 - Layer 3: The Network Layer». In: *Hack the Stack*. Ed. by Michael Gregg. Burlington: Syngress, 2006, pp. 103–150. ISBN: 978-1-59749-109-9. DOI: https://doi.org/10.1016/B978-159749109-9/50008-3. URL: https://www.sciencedirect.com/science/article/pii/B9781597491099500083 (cit. on p. 65).

[83] Abhishek Singh, Ola Nordström, Chenghuai Lu, and Andre L. M. dos Santos. «Malicious ICMP Tunneling: Defense against the Vulnerability». In: *Information Security and Privacy*. Ed. by Rei Safavi-Naini and Jennifer Seberry. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 226–236. ISBN: 978-3-540-45067-2 (cit. on p. 66).

[84] Akamai. *Che cos'è un attacco DDoS di tipo ICMP Flood?* `https://www.akamai.com/glossary/what-is-icmp-flood-ddos-attack`. 2025 (cit. on p. 66).

[85] Twingate Team. *What is DHCP Spoofing? How It Works & Examples*. 2024. URL: `https://www.twingate.com/blog/glossary/dhcp%20spoofing` (cit. on p. 66).

[86] CrowdStrike. *ARP Spoofing*. `https://www.crowdstrike.com/en-us/cybersecurity-101/social-engineering/arp-spoofing/`. 2025 (cit. on p. 68).

[87] MITRE ATT&CK. *T1557.002: ARP Cache Poisoning*. `https://attack.mitre.org/techniques/T1557/002/`. 2025 (cit. on p. 68).

[88] Cybersecurity and Infrastructure Security Agency (CISA). *NTP Amplification DDoS Attack*. `https://www.cisa.gov/news-events/alerts/2014/01/13/ntp-amplification-attacks-using-cve-2013-5211`. 2025 (cit. on p. 69).

[89] Cloudflare. *NTP Amplification DDoS Attack*. `https://www.cloudflare.com/it-it/learning/ddos/ntp-amplification-ddos-attack/`. 2025 (cit. on p. 69).

[90] Marie Baezner and Patrice Robin. «Stuxnet». In: (Feb. 2018) (cit. on p. 72).

[91] Forescout. *AMNESIA:33 – Identificare e Mitigare i Rischi da Vulnerabilità in Dispositivi IoT, OT e IT*. Tech. rep. Forescout Technologies, 2025. URL: `https://www.forescout.com/resources/amnesia33-identify-and-mitigate-the-risk-from-vulnerabilities-lurking-in-millions-of-iot-ot-and-it-devices/` (cit. on p. 73).

[92] Forescout. *AMNESIA:33 – Come gli Stack TCP/IP Generano Vulnerabilità Critiche in Dispositivi IoT, OT e IT*. Tech. rep. Forescout Technologies, 2025. URL: `https://www.forescout.com/resources/amnesia33-how-tcp-ip-stacks-breed-critical-vulnerabilities-in-iot-ot-and-it-devices/?utm_medium=amnesia33-webpage&utm_source=amnesia33-executive-summary&utm_campaign=amnesia33-research-report&utm_content=amnesia33-research-report` (cit. on p. 73).

[93] Manos Antonakakis et al. «Understanding the Mirai Botnet Understanding the Mirai Botnet». In: (Sept. 2022) (cit. on p. 74).

[94] Xiaolu Zhang, Oren Upton, Nicole Lang Beebe, and Kim-Kwang Raymond Choo. «IoT Botnet Forensics: A Comprehensive Digital Forensic Case Study on Mirai Botnet Servers». In: *Forensic Science International: Digital Investigation* 32 (2020), p. 300926. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2020.300926. URL: https://www.sciencedirect.com/science/article/pii/S2666281720300214 (cit. on p. 74).

[95] The Linux Foundation / Yocto Project. *Yocto Project Documentation*. 2025. URL: https://docs.yoctoproject.org/index.html (cit. on pp. 75–78, 82).

[96] The Yocto Project / The Linux Foundation. *Checking for Vulnerabilities — Yocto Project Development Manual*. 2025. URL: https://docs.yoctoproject.org/dev/dev-manual/vulnerabilities.html (cit. on pp. 79, 81).

[97] CompuLab Ltd. *IOT-GATE-iMX8 Industrial ARM IoT Gateway*. 2025. URL: https://www.compulab.com/products/iot-gateways/iot-gate-imx8-industrial-arm-iot-gateway/ (cit. on p. 84).

[98] Android Open Source Project. *Implementare dm-verity*. 2025. URL: https://source.android.com/docs/security/features/verifiedboot/dm-verity?hl=it (cit. on pp. 86, 87).

[99] Arch Linux. *Dm-crypt*. 2025. URL: https://wiki.archlinux.org/title/Dm-crypt (cit. on p. 89).

[100] Arch Linux. *Dm-integrity*. 2025. URL: https://wiki.archlinux.org/title/Dm-integrity (cit. on p. 89).

[101] wolfSSL. *CAAM Supported wolfSSL*. Documentazione tecnica. 2025. URL: https://www.wolfssl.com/caam-supported-wolfssl/ (cit. on p. 93).

[102] NXP Semiconductors. *MCUXpresso SDK API Reference Manual, Rev. 0*. Manuale API SDK. 2025 (cit. on p. 93).

[103] firewalld.org. *firewalld.conf Configuration Documentation*. 2025. URL: https://firewalld.org/documentation/configuration/firewalld-conf.html (cit. on p. 95).

[104] Red Hat. *Managing Firewall Using the Web Console*. 2025. URL: https://docs.redhat.com/it/documentation/red_hat_enterprise_linux/8/html/managing_systems_using_the_rhel_8_web_console/firewalld_managing-firewall-using-the-web-console (cit. on p. 95).

[105] NethServer. *DNSMasq Configuration*. 2025. URL: https://docs.nethserver.org/projects/ns8/it/latest/dnsmasq.html (cit. on pp. 96, 97).

[106] OpenVPN. *Which OpenVPN Product is Right for You?* 2025. URL: https://openvpn.net/community-docs/which-openvpn-product-is-right-for-you-.html (cit. on p. 98).

[107] Palo Alto Networks. *What is OpenVPN?* 2025. URL: https://www.paloaltonetworks.com/cyberpedia/what-is-openvpn (cit. on p. 98).

[108] Mender Hub. *How to Run CVE Checks Using the Yocto Project*. 2025. URL: https://hub.mender.io/t/how-to-run-cve-checks-using-the-yocto-project/1142 (cit. on pp. 100, 101).

[109] EdgeX Foundry. *EdgeX Foundry 2.1 Documentation*. 2025. URL: https://docs.edgexfoundry.org/2.1/ (cit. on p. 102).

[110] Redis Labs. *Recommended Security Practices*. 2025. URL: https://redis.io/docs/latest/operate/rs/security/recommended-security-practices/ (cit. on p. 102).

[111] Mender. *Mender Overview and Introduction*. 2025. URL: https://docs.mender.io/overview/introduction (cit. on p. 104).

[112] Grafana Labs. *Grafana Agent Documentation*. 2025. URL: https://grafana.com/docs/agent/latest/ (cit. on p. 106).

[113] Grafana Labs. *Configure Security Hardening in Grafana*. 2025. URL: https://grafana.com/docs/grafana/latest/setup-grafana/configure-security/configure-security-hardening/ (cit. on p. 106).

[114] Red Hat. *Writing a custom SELinux policy*. 2025. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/html/using_selinux/writing-a-custom-selinux-policy (cit. on p. 124).

[115] *EN 18031: Requisiti comuni di sicurezza per dispositivi radio connessi a Internet*. 2024. URL: https://www.cen.eu (cit. on pp. 129, 130, 132).

[116] IMQ. *Direttiva RED: Requisiti di Cybersecurity*. Tech. rep. IMQ, 2025. URL: https://www.imq.it/storage/documents/it/IMQ%20Accreditato%20Direttiva%20RED_Cyber.pdf (cit. on p. 130).

[117] Ramanpreet Kaur, Dušan Gabrijelčič, and Tomaž Klobučar. «Artificial intelligence for cybersecurity: Literature review and future research directions». In: *Information Fusion* 97 (2023), p. 101804. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2023.101804. URL: https://www.sciencedirect.com/science/article/pii/S1566253523001136 (cit. on pp. 134, 135).

[118] Syed Adnan Jawaid. «Artificial Intelligence with Respect to Cyber Security». In: *Journal of Advances in Artificial Intelligence* 1 (Jan. 2023), pp. 96–102. DOI: 10.18178/JAAI.2023.1.2.96-102 (cit. on p. 136).

[119] Lucia Stanham. *AI-Powered Cyberattacks*. 2025. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/ai-powered-cyberattacks/ (cit. on pp. 137–140, 142).

[120] Mixmode AI. *What is AI-Generated Attacks?* 2024. URL: https://mixmode.ai/what-is/ai-generated-attacks/ (cit. on pp. 138–140, 142).

[121] Morgan Stanley. *AI and Cybersecurity: The New Era*. 2024. URL: https://www.morganstanley.com/articles/ai-cybersecurity-new-era (cit. on p. 145).

[122] IBM. *Navigating the Ethics of AI in Cybersecurity*. https://www.ibm.com/think/insights/navigating-ethics-ai-cybersecurity. 2025 (cit. on p. 145).

[123] (ISC)². *The Ethical Dilemmas of AI in Cybersecurity*. https://www.isc2.org/Insights/2024/01/The-Ethical-Dilemmas-of-AI-in-Cybersecurity. 2025 (cit. on p. 145).