IoT Embedded System Vulnerability Analysis: A survey on Embedded System Threats and Firmware Integrity

Latha Sri Vurukuti 40236431, Sarat Sai Bammidi 40231817, Pranathi Akula 40234959
Students in Concordia Institute for Information System Engineering , Concordia University
lathasri1997v@gmail.com, sai.sarat2@gmail.com, akula.pranathi98@gmail.com

***Abstract*—With the spread of Internet of Things (IoT) devices, an age of unparalleled connection has come, transforming countless businesses and areas of daily life. However, due to the fundamental complexity and interconnection of these components inside embedded systems, substantial security holes have been discovered, putting user information, privacy, and system integrity at risk. This assessment comprehensively investigates the vulnerability environment in Internet of Things embedded systems, with a major focus on understanding the risks that these systems face and emphasizing the vital role that firmware integrity plays. This survey also explores and evaluates existing strategies, techniques, and tools for identifying and addressing vulnerabilities in IoT embedded systems. It assesses the efficacy of security measures, identifying their strengths and deficiencies, and recommending potential strengthening techniques.**

***Index Terms*—Vulnerability Analysis, Firmware Integrity, Threats, Cybersecurity, IOT, Embedded Systems, Mitigation Strategies.**

I. INTRODUCTION

IoT has transformed connectivity by connecting hundreds of devices inside embedded systems to improve productivity, efficiency, and usability across a wide range of applications. However, the widespread use of IoT devices in everyday life has highlighted a critical issue: the vulnerability of IoT embedded systems to a variety of threats. Understanding and addressing these flaws is critical for ensuring the security, integrity, and resilience of your IoT embedded systems. This comprehensive survey begins with a deep dive into the complex world of IoT ESS vulnerabilities, with a special emphasis on understanding the multi-layered threats and the critical role firmware integrity plays in protecting these systems. The integration of IoT devices into embedded systems has resulted in significant transformations in a variety of industries.
This integration has facilitated smooth communication and automation in industries such as healthcare, manufacturing, smart homes, transportation, and others. However, the complexity and interconnectedness of these systems have created potential vulnerabilities. This survey aims to define and categorize the diverse threats that plague IoT embedded systems. These threats range from physical tampering to advanced cyber-attacks that exploit flaws in embedded device hardware, software, and communication protocols. Understanding these threats is critical for developing effective security measures in this landscape.
Significance of Firmware Integrity: Firmware serves as the foundation for every IoT device, playing a crucial role in managing device functionality. The security and reliability of IoT embedded systems heavily rely on the integrity of the firmware. Any vulnerabilities present in the firmware, such as weak coding practices or inadequate update mechanisms, can pose significant risks to the overall system integrity.
Purpose and Scope of the Survey: The survey's goal and scope are to provide a comprehensive understanding of the vulnerabilities that plague IoT embedded systems. The survey aims to provide valuable insights for researchers, industry practitioners, and stakeholders by examining various threats and emphasizing the importance of firmware integrity. It intends to assist them in fortifying these systems against emerging threats. The survey is structured to delve into the various dimensions of vulnerabilities within IoT embedded systems. It critically assesses the existing methodologies, tools, and mitigation strategies for analyzing and mitigating these vulnerabilities. Finally, the survey offers a road map for improving the security posture of IoT systems, providing valuable guidance to all parties involved.

II. OBJECTIVE

Because IoT devices transmit vast amounts of valuable and private data, they are extremely vulnerable to cyber-attacks. Every device added to a network increases the network's digital attack surface, which is the number of weak points through which an unauthorized user can gain access to the system. Because of the constant risk of data theft and other intrusions, IoT security solutions are even more critical. The objective of conducting a comprehensive analysis on IoT Embedded System Vulnerability, focusing on Embedded System Threats and Firmware Integrity, aims to achieve multifaceted goals that encompass understanding, evaluating, and proposing strategies to fortify the security of IoT embedded systems. The detailed objectives of this survey are as follows:
Exploration of the Threat Landscape: Conduct a thorough classification of the various threats and attack vectors targeting IoT embedded systems. Identifying vulnerabilities in hardware, software, communication protocols, and firmware to create a detailed threat landscape is part of this process.

Firmware Integrity Investigation: Investigate the vulnerabilities and integrity issues that affect firmware in IoT devices. Analyze the risks associated with insecure firmware, such as insufficient encryption, a lack of secure update mechanisms, vulnerability to reverse engineering, and unauthorized access. Vulnerability Mitigation Strategies Evaluation: Evaluate the efficacy and limitations of existing methodologies, tools, and techniques for vulnerability analysis and mitigation in IoT embedded systems. Assess their suitability for dealing with emerging threats and suggest improvements or alternative approaches as needed.
Future Direction on IOT based Embedded System: Future endeavors in IoT Embedded System Vulnerability Analysis aim to anticipate evolving threats and vulnerabilities, leveraging advanced technologies and proactive strategies to fortify firmware integrity and secure embedded systems. This includes the exploration of innovative threat detection methodologies, robust firmware protection mechanisms, and the integration of AI-driven solutions for real-time threat mitigation, paving the way for resilient and secure IoT ecosystems.

III. CASE STUDIES

Numerous Internet of Things (IoT) gadgets and apps enable consumers to use products comfortably in every aspect of their lives. Examples of devices that are available on the market to draw customers in with their features that will make people's lives more comfortable include electric vehicle chargers that support Android applications and Bluetooth connections ("Kaspersky Lab Security Services"), smart meters for home electricity usage, Fitbits that track personal health information, Google Nest thermostats (G. Hernandez et al. 1–8), Tesla electric vehicles, Chamberlain my garage door access systems, drones for work and play, IP camera systems for home surveillance, and millions more. Nevertheless, the systems and gadgets on this list have been vulnerable to cyberattacks. Any gadget linked to a home or personal network that has its information stolen could have disastrous consequences. The first ChargePoint Inc. instance describes firmware and software vulnerabilities that allow attackers to quickly compromise connectivity. An attacker may circumvent the process by simply changing the debug mode of an EV home charger from ChargePoint Inc. from "branch if equal" (b e q) to "branch if not equal" (b n e). This vulnerability existed during the password authentication phase of the charger. Following a successful modification, attackers might take advantage of a buffer overflow to access both the Android application and the Bluetooth executable process, known as BTclassic. The denial of service (DOS) assault was executed by it. It has been demonstrated that an attacker might physically harm a user by disabling their entire electrical system once they have complete access to the EV charger at home. The second scenario concerns eavesdropping, sometimes known as the Man-in-the-Middle (MITM) attack, which allows attackers to obtain any desired network information. The Fit-bit Aria smart scale, which assists users in recording their own health data, was the target of the hack. Fitbit Aria allows customers to monitor their health by sending their health data to a server. In addition to obtaining user health data, hackers were also able to enter the network by obtaining the pre-shared key (PSK) and service set identification (SSID) from Wireshark log files. The attack was done in simple steps:
a. Set up DHCP server to assign a proper IP address
b. Set up VM to forward IP packets to wlan0 interface
c. Set up "hostapd" as a wireless access point (WAP)
d. Use Wireshark to sniff network traffic
e. Attackers gain full access to the network

Next case is about the device that controls and manages the thermostat from tech-giant, google. Google nest was highly susceptible when it was on device firmware update (DFU) status. When user press the hard-reset button for firmware update, it allows data input with bootable USB stick. Attackers utilized this feature and inserted customized image into the device rom. With x-loader and u-boot included in the customized image, attackers loaded the Linux kernel which has complete control over everything in the system. By executing kernel 7 with Linux inside the attacked nest, attackers gained root access and enabled secure shell (SSH) server installation and Odysseus malware to bypass network address translation (NAT). Nest, the thermostat, now worked as a botnet of home network. It had ability to access every part of the information at home: profiling, illegal surveillance, recording pictures, videos, and voices via connected IoT devices. The thermostat Nest was now operating as a botnet on the household network. It was capable of accessing all of the data in the house, including voice, video, and picture recordings through linked IoT devices, profiling, and unauthorized surveillance.

IV. THREATS AND VULNERABILITIES IN EMBEDDED SYSTEMS

*1) **Software Based Attacks:*** Attacks that target flaws or vulnerabilities in computer software to jeopardize the availability, security, or integrity of systems and data are referred to as software-based attacks. These attacks can be of many different kinds, and they frequently take advantage of holes in-network services, apps, or operating systems. These are a few typical categories of software-based threats and defences.

a) Malware Attack: Malicious software (malware) includes viruses, worms, Trojans, ransomware, and spyware. Malware can infect systems and cause various forms of damage, such as data theft, system disruption, or unauthorized access. Countermeasure: Antivirus software, anti-malware programs, and regular software updates to patch vulnerabilities are essential. User education on safe browsing habits and email practices can also help prevent malware infections.

b) Phishing: Attack: Phishing involves tricking individuals into divulging sensitive information, such as usernames, passwords, or financial details, by pretending to be a trustworthy entity. Countermeasure: User education to recognize phishing attempts, email filtering systems, and multi-factor authentication (MFA) can help mitigate the risks associated with phishing attacks.

c) Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks Attack: These attacks aim to overwhelm a system, network, or service, making it unavailable to users by flooding it with a high volume of traffic. Countermeasure: Network firewalls, intrusion prevention systems (IPS), and DDoS mitigation services can help protect against these attacks. Redundancy and load balancing can also be implemented to distribute traffic and maintain service availability.

d) SQL Injection: Attack: SQL injection involves injecting malicious SQL code into input fields to manipulate a database and potentially gain unauthorized access. Countermeasure: Parameterized queries, input validation, and using prepared statements can help prevent SQL injection attacks. Regular security audits of database systems are also crucial.

e) Cross-Site Scripting (XSS): Attack: XSS attacks inject malicious scripts into web pages that are then viewed by other users, allowing attackers to steal information or perform actions on behalf of the victim. Countermeasure: Input validation, output encoding, and the use of secure coding practices can help prevent XSS attacks. Web application firewalls (WAFs) can also provide an additional layer of protection.

f)Zero-Day Exploits: Attack: Exploiting vulnerabilities in software that are not yet known to the software vendor or for which no patch is available. Countermeasure: Timely software updates and patch management are critical to addressing zero-day vulnerabilities. Network monitoring and intrusion detection systems can help detect and respond to unusual behavior.

g) Insider Threats: Attack: Malicious activities carried out by individuals within an organization, such as employees or contractors, who have inside information concerning the organization's security practices and data. Countermeasure: User access controls, employee training, and monitoring systems for unusual behaviour can help mitigate insider threats. Implementing the principle of least privilege can also minimize the impact of insider attacks.

h) Man-in-the-Middle (MitM) Attacks: Attack: Intercepting and potentially altering communication between two parties without their knowledge. Countermeasure: Encryption protocols, such as HTTPS, can help secure communications and protect against MitM attacks. Public key infrastructure (PKI) and digital certificates can also enhance the authenticity of communication.

i) Ransomware Attack: Encrypting a user's files and demanding a ransom for their release. Countermeasure: Regular data backups, security awareness training, and robust endpoint protection solutions can help prevent and recover from ransomware attacks. Additionally, maintaining up-to-date security software is crucial.

j) Social Engineering: Attack: Manipulating individuals to divulge sensitive information or perform actions that may compromise security. Countermeasure: User education and awareness training are essential for recognizing and resisting social engineering tactics. Implementing strict access controls and multi-factor authentication can also add layers of protection.

2) Network Based Attacks: In the context of embedded systems, malicious activities that target weaknesses in embedded devices network communication are referred to as network-based attacks. Specialized computing systems called embedded systems are made to carry out specific tasks inside bigger systems. These systems can be vulnerable to a variety of network-based attacks because they frequently communicate over networks. The following list of typical network-based threats to embedded systems, along with their defences [6]

a)Man-in-the-Middle (MitM) Attacks:

Attack: Intercepting and potentially altering communication between embedded devices or between an embedded device and a central server.

Countermeasure: Implement secure communication protocols such as Transport Layer Security (TLS) for encryption. Use certificates to authenticate devices and ensure the integrity and confidentiality of data in transit.

b)Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS):

Attack: Overloading the network or specific services on embedded devices to disrupt normal operation.

Countermeasure: Employ network firewalls and intrusion detection/prevention systems. Implement rate limiting and traffic filtering to mitigate the impact of DoS attacks. Additionally, consider using load balancing and redundancy to distribute traffic.

c) Network Sniffing:

Attack: Unauthorized monitoring of network traffic to capture sensitive information.

Countermeasure: Use encryption for sensitive data transmission. Employ secure communication protocols and avoid transmitting sensitive information in plaintext. Regularly monitor and audit network traffic for unusual patterns. **d)Packet Spoofing:**

Attack: Forging or manipulating network packets to gain unauthorized access or disrupt communication. Countermeasure: Implement packet filtering and validation mechanisms. Use strong authentication methods to ensure that only legitimate devices can communicate with the embedded system. Employ network intrusion detection systems to detect and block suspicious activities.

e) Replay Attacks:

Attack: Capturing and later replaying legitimate data to gain unauthorized access.

Countermeasure: Implement timestamping and session management to detect and prevent replay attacks. Use cryptographic techniques to ensure the freshness and integrity of data.

f) DNS Spoofing:

Attack: Manipulating the Domain Name System (DNS) to redirect traffic to malicious servers.

Countermeasure: Use secure DNS protocols, such as DNS Security Extensions (DNSSEC), to prevent DNS spoofing. Regularly monitor DNS records for unauthorized changes.

g) Eavesdropping:

Attack: Unauthorized interception of communication to gather sensitive information.

Countermeasure: Implement encryption for communication channels. Use secure protocols that provide confidentiality to prevent eavesdropping.

h) Unauthorized Access and Exploits:

Attack: Exploiting vulnerabilities in network services to gain unauthorized access to embedded systems.

Countermeasure: Regularly update and patch embedded systems to address known vulnerabilities. Employ network firewalls and intrusion detection/prevention systems. Follow secure coding practices to minimize the risk of software exploits.

i) Traffic Analysis:

Attack: Analysing patterns and characteristics of network traffic to gain insights into the behaviour of embedded systems.

Countermeasure: Use encryption to protect the confidentiality of data. Implement traffic obfuscation techniques to make it more challenging for attackers to gather meaningful information from the network traffic.

j) Secure Configuration Attack: Exploiting misconfigurations in network settings that may expose vulnerabilities. Countermeasure: Follow security best practices for configuring embedded devices and associated network settings. Disable unnecessary services, change default passwords and regularly review and update configurations to align with security requirements.

*3) **Side-Channel Attacks:*** In contrast to direct attacks that target software vulnerabilities or the intended functionality of the system, side-channel attacks take advantage of information that has been compromised through physical implementations of the system. In order to obtain knowledge about a system's internal workings, these attacks concentrate on inadvertent side-channel data, such as power usage, electromagnetic emissions, timing fluctuations, or other observable physical features. To embedded systems, side-channel attacks can be a serious threat. These are a few typical side-channel attack types along with their defenses [6].

a)Power Analysis Attacks:
Attack: Monitoring the power consumption of a device to infer information about its cryptographic operations.
Countermeasure: Implement power analysis-resistant cryptographic algorithms and countermeasures. Use constant-time implementations and consider adding random noise to power consumption profiles to make it harder for attackers to extract meaningful information.

b)Timing Attacks: Attack: Exploiting variations in the execution time of cryptographic algorithms to deduce secret information.
Countermeasure: Implement constant-time algorithms that do not depend on secret data. Use hardware or software countermeasures to ensure consistent execution times, such as adding dummy operations to equalize execution paths.

c) Electromagnetic Analysis (EMA):
Attack: Monitoring electromagnetic emissions from a device to deduce information about its internal operations.
Countermeasure: Employ electromagnetic shielding to reduce emissions. Implement cryptographic algorithms with countermeasures against EMA, such as masking or blinding techniques. Use secure hardware design principles to minimize unintended electromagnetic radiation.

d) Acoustic Attacks:
Attack: Analysing sound emissions produced by a device during cryptographic operations to extract information.
Countermeasure: Implement countermeasures such as acoustic damping materials to reduce sound emissions. Use secure algorithms and implementations that are resistant to acoustic side-channel attacks.

e) Cache-based Attacks:
Attack: Exploiting cache behaviour to deduce information about memory access patterns and gain insights into cryptographic operations.
Countermeasure:Implementcache-timing-resistant algorithms. Use cache partitioning techniques or ensure that sensitive data does not affect the cache state. Employ software or hardware countermeasures to mitigate cache-based side-channel leakage.

f)Fault Injection Attacks:
Attack: Introducing faults, such as voltage glitches or electromagnetic interference, into a system to manipulate its behavior and extract information.
Countermeasure: Implement error-detection and correction mechanisms. Use secure hardware design principles to make the system resilient to faults. Employ techniques like dual modular redundancy (DMR) to detect and recover from faults.

g)Temperature-based Attacks:
Attack: Exploiting temperature variations to gain insights into the internal state of a system.
Countermeasure: Implement temperature-stabilizing measures, such as using thermal insulation or regulation. Ensure that cryptographic algorithms and implementations are resistant to temperature-based side-channel attacks.

V. FIRMWARE AND ITS IMPORTANCE

Firmware is a sort of software that is embedded in physical devices and provides instructions for their operation. Unlike ordinary software, firmware is kept in non-volatile memory and has an unbreakable connection to the hardware on which it works. IoT devices are fundamentally embedded systems with restricted resources like processing power, memory, and energy. Firmware is the native software that is incorporated in device hardware and is stored in non-volatile memory such as ROM or EPROM. Firmware is a critical component that governs how devices work, and updates are required for bug repairs, feature additions, and performance improvements. These devices serve specialized purposes and are linked together to build the IoT ecosystem.
Firmware is crucial for the basic operation of IoT devices, since it provides the instructions needed to control sensors, actuators, and communication modules. To secure the confidentiality and integrity of data sent between devices, firmware frequently contains security procedures and cryptographic

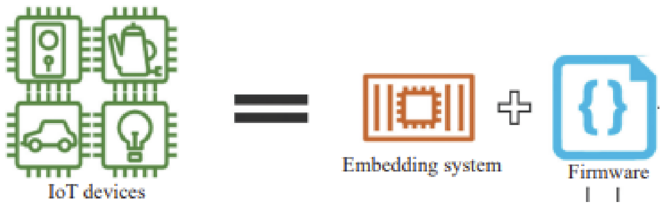


Fig. 1. How Firmware Related to IoT

algorithms. Firmware regulates communication protocols, providing network compatibility and interoperability [1].

1) **The Importance of Firmware Level Identification:** Firmware changes have an influence on the vulnerability of devices. Instead of only device types and suppliers, vulnerabilities are explicitly connected to firmware versions. Understanding the security environment and possible vulnerabilities is aided by fine-grained firmware identification.

2) **Firmware Update Difficulties:** Some older devices are incompatible with the most recent firmware releases. Manufacturers release upgrades after items are distributed, resulting in obsolete firmware in the field. For consumers, firmware upgrading is a complicated procedure that involves obtaining firmware files and reprogramming embedded devices.

3) **Challenges in Firmware Security:** ***a) Difficult Execution Environment:*** In specifically built embedded devices, IoT firmware functions independently. Because of the complicated execution environment, security researchers confront difficulties acquiring full information for firmware analysis. Methods such as side-channel or network-level monitoring are restricted and give limited information.
b) Close Source of Firmware: Firmware acquisition is frequently impossible owing to manufacturer security concerns. Manufacturers are prohibited from open-sourcing firmware and may deactivate Debug mode or JTAG, limiting accessibility. In real-world circumstances, existing technologies that rely on firmware analysis become unfeasible.

VI. VULNERABILITY ANALYSIS METHODOLOGY

Embedded systems that lack effective procedures to mitigate security threats are frequently the weakest link in IoT security. To improve IoT security, vulnerabilities in embedded systems must be addressed efficiently. Designers must take a complete approach, taking into account both new system architecture designs with security as a priority and the integration of extra security measures. Hardware security modules, cryptographic techniques, security protocols, secure setups, and up-to-date software components such as libraries and kernels are all examples of security features. [2]
Traditional testing approaches, such as functional tests and safety-related tests, are important, but they may not identify all vulnerabilities. New testing procedures are being incorporated to supplement old methods, with the goal of discovering more vulnerabilities in less time. Adopting a comprehensive design approach, incorporating robust security measures, and constantly updating testing methodologies are all necessary for effectively safeguarding IoT embedded systems. Balancing security measures with market dynamics is critical for maintaining a competitive advantage in the continually changing IoT world.
Different types of vulnerability analysis methods are as follows.

1) ***FUZZING : UNCOVERING SYSTEM VULNERABILITIES:*** Fuzzing, often known as fuzz-testing, is a technique for identifying vulnerabilities and defects by introducing specially constructed inputs into a target system under test (SUT). This method aids in the detection of problems such as memory violations, assertion violations, null handling mistakes, deadlocks, infinite loops, and resource mismanagement.
Phases of the Fuzzing Process:
The phases of fuzzing process are shown in the picture below [2].
1. Preparation: Attempts to decrease the possibility of gen-

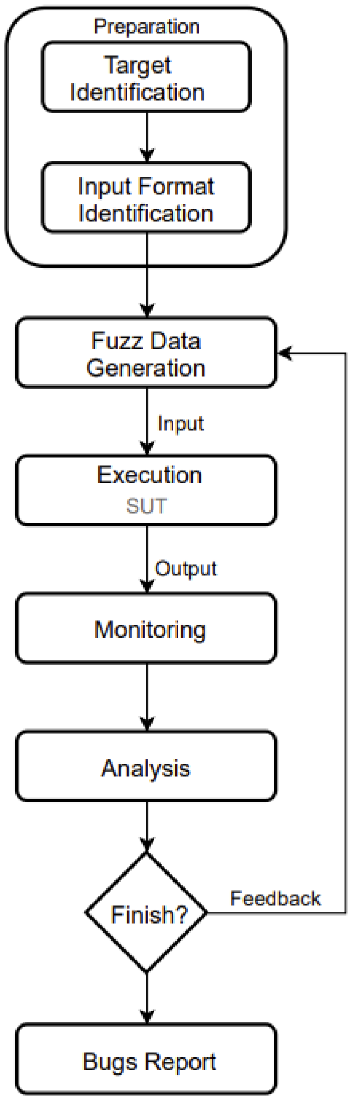


Fig. 2. Phases of Fuzzing Process

erating inaccurate fuzz data by producing precise and valid inputs based on SUT standards.
2. Fuzz Data Generation: Generates input data for the SUT while keeping the format defined during the preparation step in mind. Typically, this entails changing particular data based on an initial input seed. For successful vulnerability search, the quality of generated inputs is critical.
3. Execution: Sends created data to the SUT via the provided medium (communication packet, input file, environment variable, and so on).
4. Monitoring: Monitors the SUT's outputs and behaviour for unusual outputs or crashes that might indicate triggered vulnerabilities.
5. Analysis: Analyzes crashes and monitoring phase inputs to discover which test cases are responsible for vulnerabilities or anomalous behaviour.
6. Bugs Reports: The vulnerabilities discovered during the monitoring phase are reported.
Fuzzing is a method of discovering vulnerabilities by inserting carefully prepared inputs into a target system. Fuzzing efficacy is dependent on reliable data production, meticulous monitoring, and in-depth study of system behaviour and outputs.

*2) **FIRMWARE FINGERPRINTING:*** This section describes the method for creating firmware fingerprints, which are required for identifying and differentiating firmware versions in embedded devices.
Firmware upgrades involve filesystem changes in which engineers edit documents to address faults, add functionality, or improve efficiency. Because of these filesystem changes, there are subtle discrepancies between mixed-version firmware images. Because the filesystem acts as a unique signature for firmware, it is a useful tool for locating online embedded devices. [3]

*3) **DYNAMIC ANALYSIS:*** Dynamic analysis is a cybersecurity method that focuses on studying and monitoring the behaviour of a program while it is executing. This approach gives insights into program execution and aids in the identification of vulnerabilities and unexpected behaviours. There are several dynamic analysis methodologies, each with its own set of advantages and disadvantages.
1. DTA (Dynamic Taint Analysis):
DTA detects relationships between program inputs and logic, assisting in the detection of faulty input validation vulnerabilities and the analysis of information flows in binary code.
Limitations in program analysis, particularly with regard to specific dependencies and restrictions, provide a challenge.
2. Techniques Based on Emulation:
Emulation-based approaches use advanced dynamic analysis to construct partial or full simulations of a certain platform and execute applications within the simulated environment.
Limited support for CPU architectures, generic interrupt management, and peripheral modelling are among the challenges.
Common Dynamic Analysis Technique Limitations:
While dynamic analysis approaches overcome some of the limits of static analysis, they also provide their own set of obstacles, which include:
1. Access to source code is restricted.
2. Limitations when working with unknown code behaviours.
3. Problems with program analysis.
4. Potential challenges in detecting memory corruption.
5. Problems with system resets and wipes.
6. Concerns about speed and efficiency, particularly in emulation.

*4) **FIRMWARE VULNERABILITY IDENTIFICATION USING CODE SIMILARITY DETECTION:*** The identification of code similarity is an important part of discovering vulnerabilities in firmware images. The procedures used by code similarity detection systems differ, taking into account graph architectures, data flows, and distance measures. Evaluations on various firmware images demonstrate their usefulness in detecting vulnerabilities across a wide range of devices and architectures. These methods aid in the proactive detection of possible security vulnerabilities in embedded systems.

*5) **NETWORK SCANNING METHOD:*** The network scanning approach detects vulnerabilities in IoT devices by sending probing packets with payloads to the services operating on these devices. This method is adaptable and serves an important role in improving the security of IoT devices. The strategy involves aggressively probing devices for known vulnerabilities, particularly password problems and other security concerns.
To use the network scanning approach, you must be able to collect vulnerability information, such as Proof of Concept (PoC) for known vulnerabilities. This information aids in the development of appropriate probe packets for identifying specific flaws.
Benefits of Network Scanning:
1. Service Layer Detection: Because the approach concentrates on finding vulnerabilities at the service layer, it does not need a thorough grasp of the device's fundamental structure.
2. Network scanning is quick, effective, and suited for large-scale testing, making it a viable option for discovering vulnerabilities across a wide range of devices.
3. Commercial Vulnerability Detection Systems: The network scanning approach is used by several commercial vulnerability detection systems. These technologies discover susceptible devices by utilizing databases of known vulnerabilities and scanning techniques.
The network scanning method is an effective tool for identifying known vulnerabilities in IoT devices. It identifies flaws in passwords and other security features, so helping to the overall enhancement of IoT security. While there are certain ethical concerns, the method is nevertheless a popular and feasible option for large-scale vulnerability assessment.

*6) **SIMILARITY DETECTION METHOD:*** To find known vulnerabilities in IoT devices, security experts use software code similarity detection tools. These approaches are intended to solve the enormous difficulty given by a huge number of unpatched vulnerabilities in IoT devices. The procedure of detecting similarity entails extracting original information from code and comparing their similarity using algorithms. Similarity detection is divided into two parts:

source code and binary code.
The landscape of similarity detection approaches includes both source code and binary code, with a focus on tackling the issues associated with finding vulnerabilities in IoT devices. The structure of similarity detection is shown below [4]. Cross-

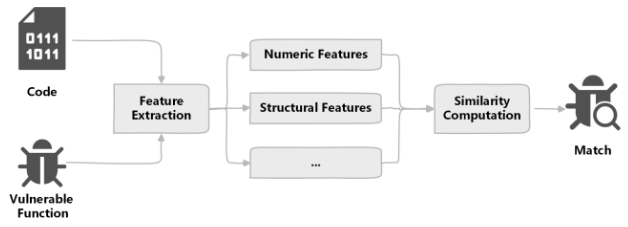
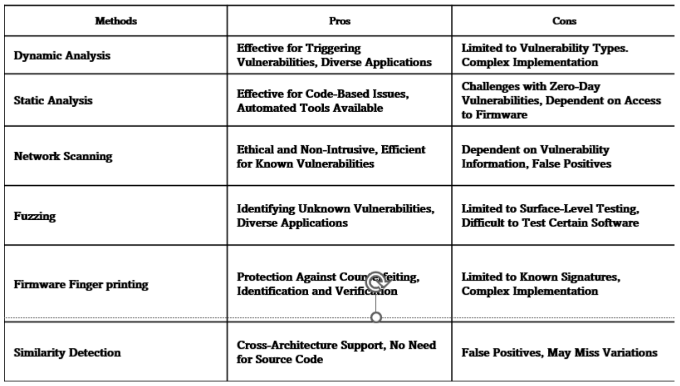


Fig. 3. Structure of Similarity Detection

architecture similarity detection has grown in popularity, and the use of deep learning algorithms has showed promise in boosting vulnerability detection accuracy. Researchers are still investigating and improving these strategies for the varied and diverse environment of IoT devices.
The PROS and CONS of all methodologies are shown below.



Methods	Pros	Cons
Dynamic Analysis	Effective for Triggering Vulnerabilities, Diverse Applications	Limited to Vulnerability Types. Complex Implementation
Static Analysis	Effective for Code-Based Issues, Automated Tools Available	Challenges with Zero-Day Vulnerabilities, Dependent on Access to Firmware
Network Scanning	Ethical and Non-Intrusive, Efficient for Known Vulnerabilities	Dependent on Vulnerability Information, False Positives
Fuzzing	Identifying Unknown Vulnerabilities, Diverse Applications	Limited to Surface-Level Testing, Difficult to Test Certain Software
Firmware Finger printing	Protection Against Counterfeiting, Identification and Verification	Limited to Known Signatures, Complex Implementation
Similarity Detection	Cross-Architecture Support, No Need for Source Code	False Positives, May Miss Variations

Fig. 4. Comparision of all Methods

VII. RESEARCH GAPS AND FUTURE DIRECTIONS

Current research is often concentrated on specific features of IoT embedded systems, such as network security or firmware flaws. Comprehensive cross-domain threat modelling that analyzes interactions between multiple components, protocols, and layers of IoT systems has a research deficit. While vulnerabilities have been uncovered, there has been a dearth of comprehensive research on the real-world consequences of exploiting these vulnerabilities. Future study might look into real scenarios, such as examining the repercussions of successful attacks on IoT devices and systems. Human engagement is frequently required in IoT devices. It is critical to understand the human-centric components of security, such as user behaviour, awareness, and education. Human aspects should be investigated in order to create techniques for increasing user security awareness.

FUTURE DIRECTIONS

Machine Learning for Anomaly Detection: Investigate the use of machine learning methods to detect anomalies in firmware and system behaviour. Create models capable of adapting to new attack vectors and providing early warnings of possible weaknesses.
Blockchain for Firmware Integrity: Look into using blockchain technology to improve firmware integrity. Use distributed ledger technology to securely store firmware upgrades, assuring their legitimacy and preventing unwanted changes.
Investigate privacy-preserving approaches in embedded systems, particularly those that collect sensitive data. Investigate cryptographic algorithms and privacy-enhancing technologies for the protection of user data in IoT contexts.
Quantum-Safe Cryptography: Investigate quantum-safe cryptographic techniques for safeguarding embedded devices to anticipate future attacks. Examine the effects of quantum computing on present security measures and create quantum-resistant alternatives.
User-Centric Security Design: Incorporate user-centric security design concepts into the IoT device development lifecycle. To design more secure and user-friendly embedded systems, understand user behaviours, preferences, and difficulties.

VIII. CONCLUSION

Finally, our study thoroughly investigated a wide range of methodologies and methods for identifying vulnerabilities in IoT embedded device firmware. While IoT security studies has seen a significant rise, it is still in its early stages within the wider information security area. This emphasizes the importance of conducting a comprehensive survey to integrate current information and guide the course of IoT security development. Classification improvement into analytic tools, vulnerability discovery, detection, and mitigation give a systematic lens through which current research activities may be analyzed.Lessons from our survey help us get a deeper understanding of the effectiveness of various techniques and possible problems. In the future, we suggest a quantitative comparison of existing approaches as well as a focused examination into the elements that influence efficiency, accuracy, and scalability. This forward-thinking strategy strives to deliver empirical findings while simultaneously laying the framework.
As we look ahead, the combination of different technologies with developing domains is expected to give rise to large-scale automated vulnerability analysis that overcomes architectural limitations. The future of IoT security is dependent not just on tackling present difficulties, but also on using the potential inherent in these challenges to drive innovation. With more AI integration and a larger acceptance of peripheral system analysis on the horizon, the next phase of IoT security offers a more complex and adaptable approach, enabling a robust and safe environment for networked devices.

REFERENCES

[1] Detecting Vulnerability on IoT Device Firmware: A Survey Xiaotao Feng, Xiaogang Zhu, Member, IEEE, Qing-Long Han, Fellow, IEEE, Wei Zhou, Sheng Wen, Senior Member, IEEE, and Yang Xiang, Fellow, IEEE

[2] M. Eceiza, J. L. Flores and M. Iturbe, ”Fuzzing the Internet of Things: A Review on the Techniques and Challenges for Efficient Vulnerability Discovery in Embedded Systems,” in IEEE Internet of Things Journal, vol. 8, no. 13, pp. 10390-10411, 1 July1, 2021, doi: 10.1109/JIOT.2021.3056179.

[3] D. Papp, Z. Ma and L. Buttyan, ”Embedded systems security: Threats, vulnerabilities, and attack taxonomy,” 2015 13th Annual Conference on Privacy, Security and Trust (PST), Izmir, Turkey, 2015, pp. 145-152, doi: 10.1109/PST.2015.7232966.

[4] Yu, M.; Zhuge, J.; Cao, M.; Shi, Z.; Jiang, L. A Survey of Security Vulnerability Analysis, Discovery, Detection, and Mitigation on IoT Devices. Future Internet 2020, 12, 27. https://doi.org/10.3390/fi12020027

[5] Securing Embedded System from Code Reuse Attacks: A Lightweight Scheme with Hardware Assistance Zhenliang An, Weike Wang,* Wenxin Li, Senyang Li, and Dexue Zhang*

[6] X. Feng, X. Zhu, Q. -L. Han, W. Zhou, S. Wen and Y. Xiang, ”Detecting Vulnerability on IoT Device Firmware: A Survey,” in IEEE/CAA Journal of Automatica Sinica, vol. 10, no. 1, pp. 25-41, January 2023, doi: 10.1109/JAS.2022.105860.

[7] D. Yu, L. Zhang, Y. Chen, Y. Ma and J. Chen, ”Large-Scale IoT Devices Firmware Identification Based on Weak Password,” in IEEE Access, vol. 8, pp. 7981-7992, 2020, doi: 10.1109/ACCESS.2020.2964646.

[8] A. Aldahmani, B. Ouni, T. Lestable and M. Debbah, ”Cyber-Security of Embedded IoTs in Smart Homes: Challenges, Requirements, Countermeasures, and Trends,” in IEEE Open Journal of Vehicular Technology, vol. 4, pp. 281-292, 2023, doi: 10.1109/OJVT.2023.3234069.

[9] J. Hou, T. Li, and C. Chang, “Research for Vulnerability Detection of Embedded System Firmware,” Procedia Computer Science, vol. 107, pp. 814–818, 2017, doi: https://doi.org/10.1016/j.procs.2017.03.181.