Internet of Things Firmware Vulnerabilities and Auditing Techniques

Subjects: Computer Science, Information Systems

Contributor: Taimur Bakhshi , Bogdan Ghita , Ievgeniia Kuzminykh

Internet of Things (IoT) paradigm has been widely applied across a variety of industrial and consumer areas to facilitate greater automation and increase productivity. Higher dependability on connected devices led to a growing range of cyber security threats targeting IoT-enabled platforms, specifically device firmware vulnerabilities, often overlooked during development and deployment.

Internet of Things | firmware auditing | reverse engineering | security testing

1. Introduction

Internet of Things (IoT) devices have become ubiquitous in a wide range of areas, including Industry 4.0, smart homes, smart cities, healthcare systems, the automotive sector, public services, and critical infrastructure [1][2][3][4][5][6][7][8][9]. The anticipated deployment of future generations of mobile access [10] and Low-Power Wide Area Networks (LPWAN) [11] technologies will see greater investment and drive the evolution of the IoT ecosystem [12][13]. Regardless of their popularity, the limited hardware and power capabilities of IoT devices lead to inherent challenges which affect security and device lifespan [14][15]. Many studies over the past decade focused on hardening IoT security at the network and application layers [16][17][18][19][20][21][22]; however, an important and often overlooked facet of secure IoT infrastructure is maintaining the integrity of IoT firmware.

A 2021 Microsoft review of the security landscape indicated that an increasing number of attacks focus on the IoT device firmware and BIOS (basic input/output system) due to a significant lapse and support for firmware security primitives [23][24]. Various categories of IoT vulnerabilities are directly linked to the firmware content and device capabilities. Firstly, due to their typical disposable nature, some devices cannot be updated or modified, which renders them vulnerable to issues discovered after their release. From the perspective of hardware capabilities, their fit-for-purpose design encompasses reduced storage and processing power, hence additional protection mechanisms may impede their functionality; further related to their design, their actual implementation cycle is not iterative and unlikely to be supportive of eliminating vulnerabilities identified after market release [24][25]. Due to all these inherent challenges, many traditional cybersecurity solutions cannot run on IoT hardware. Any vulnerable firmware present on IoT devices, coupled with their Internet-readiness, can therefore be exploited in a more streamlined and straightforward fashion; subsequently, any such device can be used as a bot, cause disruption, or be the starting point for other attacks. Some leading security companies, such as Checkpoint, offer products that investigate the security level provided by the firmware through runtime, weak credentials, and code checks against high-severity vulnerabilities from the Common Vulnerabilities and Exposures (CVE) list [26]. Potential attackers

often consider alternative attack vectors, using domains and network endpoints that an IoT device firmware connects to, as infected firmware files might contain malicious payload as part of a more sophisticated attack.

Several studies catalogued IoT firmware security issues by aligning them with higher operational layers.

- Interface security: Vulnerable interface identification in hardware, software, network, and application domains of IoT-ware represented the focality of studies in [27][28]. Some of the work in this area focuses on interface security and vulnerability solutions of consumer devices, detailing mechanisms for remote hijacking and control of IoT-ware, including surveillance nodes and general threats posed by IoT-specific malware [27][29][30][31]. Additionally, [28] provided a classification of existing solutions to detect IoT firmware threats, albeit without discussing corresponding solutions.

- Auditing techniques: Solutions describing the challenges in static [32][33][34][35] and dynamic auditing methods [36][37][38][39][40][41][42][43] have been proposed for IoT firmware vulnerability detection. Furthermore, to describing these fundamental vulnerability auditing techniques, some studies also highlighted the use of fuzzing technology and symbolic system execution to identify susceptibility in IoT-ware [39][44][45][46][47][48]. The primary efforts have been focused on assessing the effectiveness of different existing auditing methods and recommendations for developers/testers.

- Reverse engineering: Reverse engineering evaluation has been carried out on several commodity IoT devices to understand firmware vulnerabilities [49][50]. Employing fault injection, researchers have sought to identify the shortcomings of several vulnerabilities including weak authentication (password, PIN, etc.), device capability, and backdoors in IoT-ware [51][52]. System emulation schemes have also been the subject of research with a view to understand common challenges faced by developers and testers [37][42][53]. The tools and techniques employed for reverse engineering have been discussed in [19][25][49][50][51][54][55], providing basic discussion of pre-processing, de-compiling, unpacking, and evaluation techniques.

- Emerging applications: Ongoing advances in blockchain technology and machine learning technologies have also been topical areas of research in IoT-ware. Firmware data transmitted to IoT devices connected to a blockchain network is cryptographically proofed and signed by the true sender holding a unique public key, ensuring authentication and integrity of firmware [56][57][58][59][60][61]. When an IoT device needs to be updated, a smart contract [62] sends the hash or metadata file to that IoT device to obtain a copy of the update through peer-to-peer exchange with other nodes [57][58], or it is directly downloaded from the manufacturer's server [63]. Bitcoin technology can also be employed to verify a firmware version before the update begins and to acknowledge a transaction before the IoT device can download and install it [56][60]. The studies [64][65] proposed direct and indirect firmware update distribution based on Ethereum blockchain. Similarly, Skipchain blockchain technology has also been proposed for secure trusted firmware updates using smart contracts [66].

Firmware identification is vital in preventing spoofed firmware packages. Machine learning algorithms are used for identification and classification of IoT image fingerprinting [67], according to vendor or device type [68]. Greater ML-

based automation significantly reduces the latency involved in reverse engineering maneuvers such as firmware decompression [69][70].

- Commercial developments: In the commercial realm, TrustZone [71] by ARM has provided users a hardware-based security extension establishing a root of trust (RoT) and cryptographic services to securely store critical (firmware) data, which is an improvement over conventional trusted platform modules (TPM). TrustZone allows a wider set of hosted sensitive services driven by (hardware-based) isolation; however, an ever-expanding set of threats from secure-mode operation is not uncommon [72]. Similarly, Intel Turstlite [73], a generic security architecture suited to low-power embedded devices, allows remote management, authentication, and over the air (OTA) updating as well as remote attestation [74]. Among low-cost solutions, IoT-ware memory access control can also be implemented using SMART [75], using a ROM measurement routine with a secret key to provide remote attestation. However, SMART does not specifically deal with memory access violations or provide provisions for updating the attestation code [76] and, as discussed in [77][78], the verifier can also be malicious while the prover is benign, a significant limitation of remote attestation.

2. Firmware Vulnerability: Influences and Challenges

2.1. System Properties

System software exploitation remains one of the fundamental avenues to target and exploit IoT firmware vulnerabilities.

- Software corruption: IoT firmware is inherently susceptible to software corruption, such as coding bugs introduced at service initiation or operation or during upgrades [27]. Coding bugs may introduce pointer violations, and type/format confusion, while programming related issues can also lead to malicious code injections, running of privileged commands and system failures. Tainted data and unexpected input can alter device behavior and further expose it to firmware threats.

- Memory management: Inefficient or corrupt coding can also lead to integer and buffer overflow, a common cause of security vulnerabilities further exacerbated by memory constraints inherent in IoT-ware [79]. Application requirements also may dictate implementing safety critical services in separate hardware chips [48][80]. While hardware-based trust management (HTM) is considered an optimal solution, the spatial and financial cost again might render it unfeasible for IoT-ware. Adopting HTM is also limited by the typical absence of dedicated Memory Management Units (MMU) in IoT systems, leading to frequent memory violations. Service isolation can also be offered solely in software, utilizing virtual memory and enabling monitoring of device sub-systems, allowing wider cryptographic support despite code-sharing on a single processor [75][80].

- Misconfiguration: Domain limitations including limited memory, power efficiency and device heterogeneity need to be recognized during system configuration to mitigate some of the system vulnerability exploits discussed earlier. Misconfiguration of the system may lead to a successful exploitation.

2.2. Access Mechanisms

Access, authentication, and credential management all play an essential role in patching IoT nodes, as devices can be located in remote environments where manual (local console) updates are economically infeasible, requiring over-the-air-update mechanisms.

- Access control: IoT firmware access requires well-defined policies and suitable encryption to mitigate against password, certificate, or encryption key threats. The device manifest, containing author information and firmware update policy, if left un-encrypted, can lead to accessing, altering, or deleting vital metadata required for future authentication and upgrades to device firmware. Similarly, public certificate servers utilizing SSL (Secure Sockets Layer) certificates for provision of IoT-ware security may lead to man-in-the-middle attacks if repeatedly reused for a range of devices [34]. While vendors may also incorporate backdoor channels or push mechanisms to access devices for regular updates, such channels, if not protected by adequate credential management, may result in compromising device firmware or device data [40][81].

- Authentication: IoT-ware attacks due to weak authentication mechanisms are rather common [28]. Misconfigured and erroneous authentication routes allow control and jeopardizing of normal operation [81]. Weak authentication is usually due to resource constraints, allowing limited authentication schemes to conserve memory and processing power.

2.3. Component Re-Use

- Hardware and Software Re-use: Hardware and software components re-use, including off-the-shelf boards, circuitry, sensors, bootloaders, or software libraries, is prevalent among vendors to reduce development time and associated costs in the IoT domain [43], while inadvertently overlooking vulnerabilities arising due to heterogeneous cross-connectivity. In multi-controller systems, firmware from different manufacturers requires comprehensive security analysis and testing of each individual component. Firmware vulnerabilities in one controller or in exploitable software can lead to cascaded threats disrupting the entire operation and to the mass production of a range of insecure IoT devices [18].

- Development Resource: An ever-evolving set of IoT applications has also generally led to vendors frequently employing developers with limited expertise in developing high-quality firmware [16][82]. In addition, vendors also tend to overlook firmware vulnerabilities in favor of overall device usability and performance.

2.4. Network Interfacing

IoT devices interact with other heterogeneous systems over several interface types and networking protocols. However, this may translate into application programming interfaces (APIs) and protocol susceptibilities, presenting potential attackers with opportunities to compromise device functionality as well as accessibility. In order to understand the impact at each level, it is worth observing how each communication level may introduce its own attack opportunities.

- Web Services: IoT devices communicate with cloud, fog, edge computing and monitoring systems over a range of web APIs. Insecure, poorly designed web services remain one of the leading causes of device exploitation, allowing service interruption via application-level and firmware-based attacks [36][55][83]. Prominent malware such as IoTReaper have successfully exploited IoT web interfacing to launch wide variety of attacks on device-ware [84]. Limited resources again hamper the adoption of multi-factor authentication incorporation in IoT-web interactions [85].

- Network Protocols: Vendors use a wide range of standardized and proprietary network protocols that, when combined with reusable hardware and software components, may lead to propagation of existing security issues in IoT-ware. Poor management of device network configuration, such as leaving open unused ports, may lead to security issues. The security firm Kaspersky reported that, in the first half of 2023, honeypots recorded that nearly 98% of the network-related attacks on IoT-ware occurred on the unsecure Telnet interface [86]. Over-the-air updates need to employ standardized and tested protocols that offer greater protection against man-in-the-middle and spoofing attacks while patching firmware.

- Tainted Data: The sensor and actuation services process incoming data that may require acquisition, perusal, validation, processing, and sanitization through associated fog and cloud nodes. Data acquired from sensory or actuator portals, if tainted or malformed, can overwhelm device operability and expose the device firmware to security risks [87].

2.5. Image Management

Firmware image management is vital for the longevity and secure operation of IoT-ware. Inefficient non-redundant firmware storage and upgradation schedules coupled with suboptimal configuration parameters, will negatively influence IoT firmware integrity.

- Storage Integrity: IoT device firmware requires image storage integrity as well as secure distribution and updating to mitigate the threat exposure. Despite improvements in OTA mechanisms, device developers are generally reluctant to provide security patching as a continual maintenance service [88]. Given the significant lifespan of IoT operations, devices may be running obsolete firmware several years old that has several discovered, widely acknowledged flaws. Where OTA updates and encryption mechanisms are available, the network protocols also need to be tested for security compliance and suitable encryption.

- Update Delivery: The process of firmware update, where available, can be used as an attack delivery option, as it can be initiated by the customer, pushed from a server, or follow a hybrid approach; in addition, vendors may introduce provisions for backdoor updating of device firmware [65][66][89]. While not an intrinsic vulnerability, having firmware downloads available publicly may also offer an insight into the libraries, settings, and functionality to launch sophisticated attacks. Lack of coordination between the operating parties, server and network downtime, and device outages can also lead to inconsistencies in update tracking, causing unnecessary delays to firmware updating.

2.6. User Awareness

Firmware updates frequently involve complex decision-making processes, such as the re-certification of tested code, and once the device has been deployed, consumer consent in upgrading to any of the new functionalities.

- Automation and Intervention: An efficient device update process requires a balance between automation and human intervention, whereby large-scale updates should be performed using minimal manual intervention. To optimize decision making, necessary provisions for manual intervention can be kept, while maximizing de-facto upgrade policies using dynamic updates to be applied as released. Users can also be incentivized to update by flagging the risks that they expose themselves to in case of non-compliance.

- Optimization: Incorrect operational settings such as disabling or reducing event logging to conserve energy makes post-incident analysis difficult and prone to errors. A significant number of IoT vendors provide devices without any user guidelines for updating configuration parameters based on usage. Opting for default settings, ranging from generic authentication passwords, switched off update notifications, or outdated web applications, vendors pass the responsibility and burden of device firmware updates to the end-user. However, as widely acknowledged in the literature, firmware adjustments are rarely considered or applied by everyday users [23][29][89]. A general improvement in the set of guidelines to provide the user with sufficient information to secure their devices is nonetheless vital and consortiums such as IoT Alliance Australia issued specific user guidelines on the maintenance and operation of IoT firmware updating and help in identifying firmware hijacking [90].

2.7. Regulatory Compliance

The IoT paradigm is still an emerging technology subject to ongoing standardization.

- Standardization: Existing IoT-ware regulations have been introduced by commercial and governmental organizations including OWASP (Open Web Application Security Project), IoT Security Foundation, and NIST (National Institute of Standards and Technology). Standardization bodies have provided operational guidelines as well as best-practice mechanisms to provide secure IoT systems; however, these have not been widely adopted due to limited regulation. On a similar note, inadequate and inefficient compliance resulted in insecure booting, minimal or no encryption, and outdated firmware. Enforcing security compliance as part of IoT-related products engineering, development frameworks, and business policies requires greater regulatory oversight by governmental and non-governmental bodies which are usually beyond the scope of standardization organizations.

- Development Oversight: Vendors with inadequate experience in the IoT domain have been mass producing devices without adequate security inclusion [49][91][92]. A separate category of oversight challenges is linked to the design and manufacturing process. Hardware device manufacturing and software provision tend to be rather independent processes and coordination issues between original device manufacturers (ODM) and original equipment manufacturers (OEM) may result in overlooking firmware flaws. Code developed and

supplied by ODMs may contain security loopholes that, when used and implemented by OEMs, may result in replication across thousands of commercial devices [16][83].

2.8. Adversarial Vector

It is important to consider adversarial models when documenting the vulnerability triggers of IoT-ware.

- Local and Remote Vectors: Remote or over the network adversarial factors can infect systems via malware, while local adversaries can eavesdrop and interfere with device communication [93]. Stealth-based adversaries can attack either from closer physical proximity or remotely, masquerading as an authentic entity and gain unwarranted access to the IoT ecosystem [75].

- Side-channeling: Similarly, side-channel attacks can be carried out by a physical non-intrusive entity, while an intrusive adversary can completely overtake an authentication mechanism to prove its identity to an IoT device aiming to solicit information or exploit device behavior through hardware-software modification [75].

- Hybrid Designs: Dedicated hardware and software security have associated cost implications; the inherent spatial, financial, and power efficiency compromises for IoT-ware require careful trading off. A combinatorial approach using a mix of hardware and software-based controls to address adversarial threats is often considered to be a more viable option compared to purely hardware-based security or an entirely software-oriented security primitive [93].

2.9. Domain Limitations and Associated Impact

As mentioned, IoT devices are inherently limited devices, but these limitations span across multiple areas. From the hardware perspective, they have limited memory (lm) and limited storage space (ls) due to their reduced manufacturing cost; a significant direct impact of these characteristics is that such devices cannot typically employ additional security monitoring processes. Also under this heading are their limited encryption (le) capabilities, which directly impact protection mechanisms thar are computationally intensive or require specialized hardware. Given their wide deployment, each IoT device must also benefit from a low operational cost (oc) and must deliver excellent power efficiency (pe), both components also having a direct impact on any security mechanisms that users may wish to deploy but, in addition, also severely impacting any support, update, or monitoring infrastructure or associated management costs aiming to keep them up to date. The limited hardware also impacts their ability to perform any additional security and update functions beyond their primary purpose, which has specific quality of service (QoS) associated constraints. All above listed points relate to individual devices; expanding to the overall IoT ecosystem, there is a wide range of devices from a variety of manufacturers, which leads to significant heterogeneity (ht) and cross-connectivity (cc) to allow them to operate. While beneficial from a market competitiveness perspective, these two constraints have a direct impact on harmonizing defense mechanisms and they also make regulatory compliance virtually impossible to achieve.

3. Vulnerability Auditing

Firmware auditing is a manually intensive task, requiring assessor expertise in reverse engineering (RE) and a multitude of static (SA) and dynamic analysis (DA) techniques [87]. Prior to vulnerability analysis, the respective firmware needs to be systematically processed to ensure its compatibility with the chosen auditing method. Once processed or re-hosted, the firmware is subjected to vulnerability auditing testing processes for accurate determination of inherent weaknesses. The completeness and accuracy of vulnerability auditing is subject to several associated challenges in reverse engineering tasks, as well as the adequacy of state-of-the-art vulnerability analysis mechanisms.

3.1. Reverse Engineering

Firmware source code is usually not readily available for vulnerability auditing. As part of the firmware examination process, the first and foremost step is to perform a series of reverse engineering tasks involving binary file acquisition, unpacking, and de-compilation to access the source code [16][55][81][94].

3.1.1. Firmware Acquisition

Firmware can be acquired from a vendor repository, locally extracted from a device [49], or intercepted and saved during OTA updating [19]. Firmware acquisition automation using web-crawling and scripting techniques is also possible [34][81], although dedicated FTP-based image servers remain the preferred option [38].

Code can also be acquired from devices through JTAG and UART ports or by using forensic analysis techniques [40][49][95]. Device manifests and update servers may schedule regular upgrades of device firmware using OTA updates [19]. Depending on encryption, firmware data (update) transfer mechanisms can allow vulnerability analyzers to record and store data during the update process through packet sniffing or mirroring [96]. Establishing central repositories that aggregate firmware code from multiple vendors to expedite and scale auditing procedures remains a long-standing tester requirement [91].

3.1.2. Firmware Unpacking

The criteria and scheme for binary packing is usually vendor-specific and considered proprietary [34][36][81][97]. Some of the common challenges faced by testers during unpacking include file encryption, obfuscation [98], compression using non-standard schemes, or a monolithic multi-feature systems containing kernel, OS and IoT applications bundled together [36]. Each of the unpacking concerns require independent selection and application of tools, the foremost being entropy analysis to determine the encryption or obfuscation techniques. The overall confidence in the output generated is, however, minimal, requiring repeated analysis by domain experts for unpacking [68]. Some of the other prominent tools used for unpacking include Binwalk [99] and BANG [100] using recursive unpacking, while FMK [101] and FACT [102] focus on Linux-based platforms.

3.1.3. Decompiling

Decompiling machine code is needed for greater human readability in a higher-level language and comprises disassembly, data flow, control flow analysis and data type inspection [103][104].

Machine code is first converted to a low-level assembly equivalent. Modern compilers are capable of separating executables from data; however, if data are placed in the executable section, it may result in inefficient execution code and data isolation.

After disassembly, during lifting and data flow assembly processing, the code is translated to a higher level internal representation. Control flow analysis can also employ control flow graphs, allowing data type identification in the code. Debugging is sometimes also used to analyze sections of particular security interest [105]. Popular de-compilation tools include Radare2 [106] and Binary Ninja [107] and provide binary analysis capabilities with (optional) GUI support. IDA Pro [108] and Ghidra [109] have multiple features including interactive disassembly and multi-architecture support. KLEE [110] uses symbolic VM processing (LLVM) compiler with relatively heavy resource consumption.

3.1.4. Challenges

The impact of the issues relating to the acquisition, unpacking and de-compilation process is amplified by a number of additional challenges highlighted as follows.

- Packing logic: packers do not modify the code functionality, making presentation of the code sequential and not readily human-legible for security analysis. Therefore, use of automated dynamic analysis as opposed to manual perusal can yield better results, providing auditing scalability for a multitude of firmware solutions [70]. Testing frameworks, including FAT [111] and QEMU [112], simplify the analysis by incorporating several vulnerability assessments tools and emulation.

- Mitigation techniques: In addition to cryptic packing, vendors may resort to de-compilation mitigation, adding to firmware source inspection obstacles.

- Metadata unavailability: Masquerading device meta-data to avoid hardware-based hacking can inadvertently complicate the security auditing process [49][97] by limiting information on product release, update log and version number, and hardware architecture for de-compiler selection [113]. Intuitively assuming protocols, OS and libraries and other data inputs are used to analyze the device for security vulnerabilities is therefore common, as is brute-force fuzzing using genetic algorithms such as the American Fuzzy Lop (AFL) fuzzer [114] that aids when randomizing input testing. The scope, applicability, and operational capability of auditing techniques remains vital to firmware vulnerability assessment and device protection.

3.2. Auditing Techniques

3.2.1. Static Analysis

Static analysis involves manual and intensive scanning of the source code against ruleset patterns to identify coding errors [115][116][117]. Static analysis, therefore, does not involve the actual execution or emulation of firmware and does not require the auditor to have physical access to IoT devices for scrutiny [98]. Typical vulnerabilities determined using static analysis include invalid references, buffer overflows and memory corruption flaws [118], segmentation faults, and uninitialized variables [119]. To reduce cost and time, auditors can use tools to automate sub-processes, sometimes at the risk of greater false positives.

A typical example of a manual analysis tool is woodpecker, introduced in 2012 for Android applications [120]. Although the tool itself was not intended to find firmware vulnerabilities, it did find permission leaks in pre-loaded applications.

Firmalice, another binary analysis tool proposed in [81], used an automation and parallelism approach that slices a program and uses a symbolic execution engine to execute parallel functions for recording vulnerabilities. The tool has the ability to understand security policies as well as identify privileged instructions.

A parsing-based analysis group was introduced by Parser Identification in Embedded Systems (PIE) [33], which is a tool for detecting functions while parsing components and complex code. Before any classification can be performed on the parsed components, the firmware binary code is converted to an intermediate language via LLVM, thereby allowing PIE to analyze the firmware of embedded systems without any documentation or source code.

As a follow up to Firmalice, Shoshitaishvili et al. proposed ANGR [121], enabling both static and dynamic analysis as briefly described earlier on; ANGR remains popular among many other tool frameworks for carrying out firmware analysis using binary control flow graphs (CFG). Following a different approach, FirmUp [35] performs static vulnerability analysis of firmware images using CFGs and, additionally, firmware slicing to find the exact location of vulnerable procedures.

Machine learning has been used to enable greater automation by incorporating pattern recognition in existing static analysis techniques. In 2016, Feng [122] introduced an algorithm called Genius to solve the scalability problem with control flow graphs using a combination of machine learning and computer vision techniques.

3.2.2. Dynamic Analysis

The dynamic schemes execute firmware code allowing auditors to observe system behavior without requiring access to the program internals information. Dynamic analysis requires metadata information to optimize firmware emulation. However, images cannot always be emulated without knowledge of the underlying architecture, therefore dynamic analysis does not scale well when automated emulation is not possible, as it would require repeated customization of emulation and configuration setup. Typically, dynamic analysis is employed when source code is unavailable or de-compilation is unsuccessful.

FIE [47] was developed to scrutinize memory locations of peripherals using invocation of interrupt handlers to observe behavior. FIE was built using KLEE symbolic execution engine [110] and is micro-controller specific. FIE keeps records of all previously analyzed states, filtered using state pruning and memory smudging. State pruning helps remove redundant state executions for even small firmware images, while memory smudging allows FIE to recognize loop counters and replace them with symbolic variables to help with greater code coverage.

Symbolic execution is a rather slow yet powerful technique to determine equations capable of defining as well as fully describing the stagnant and operational state of firmware in real-time. Using symbolic execution, peripherals are emulated, and input is generated for execution and testing in real-time. Tools such as Laelaps [53], μEmu [123], or Gerbil [43] can run various embedded device software without coding any specific device related information into the emulator. Unknown peripheral registers are considered as symbols and the input firmware image is symbolically executed to infer rules responding to unknown peripheral access types. The rules are further stored in a database that can be referred to during firmware analysis.

Several multi-utility frameworks were developed to execute the dynamic analysis supported by full system emulation via QEMU [112], which emulates the I/O and kernel operations. One such framework is Avatar [40], able to perform dynamic analysis of embedded device firmware and having equal applicability in the IoT domain; however, it requires real hardware to discover vulnerabilities slowing execution of the entire procedure, which is adding to its scalability concerns.

Expanding to industrial IoT-ware, the dynamic framework proposed by Palavicini et al. [54] uses a combination of methods, including binary analysis tools, cyber reasoning system, fuzzer, as well as security analysis virtualization solutions such as OpenPLC, Firmadyne, and QEMU. The study proposes a three-stage approach, starting with the extraction of the firmware blob to extract code for emulation, further emulating the code, and analyze the results for vulnerabilities using a number of techniques such as fuzzing and symbolic execution. This analysis results in finding backdoors, information leakage and code for creating botnets.

3.2.3. Hybrid Proposals

Hybrid approaches, amalgamating static binary analysis with dynamic real-time investigations are a valuable option for greater auditing coverage. A hybrid combination of auditing techniques can be used to increase unknown vulnerability detection efficacy. From a practical perspective, multiple systems can be considered hybrid; Costin and Zaddach's work [29], described earlier, is a combination of dynamic and static analysis that aims to achieve full automation.

Hybrid techniques can also rely on fuzzing, using malicious input patterns to trigger unexpected device operation, essentially stress-testing system security. IoTFuzzer [97] is one such framework that uses a black-box approach to detect possible memory corruption vulnerabilities. It sends probing messages to the IoT device and, when crashing, collects the generated error messages.

3.3. Discussion of Auditing Techniques

Dynamic analysis is preferred by practitioners over static analysis, despite the inherently high vulnerability determination efficiency of the later, because complex reverse engineering and tight software–hardware coupling raise additional challenges for static auditing [37].

There are several classes of vulnerabilities that need to be audited and analyzed, including system properties, access mechanisms and networking, code reusage, and user awareness. While some vulnerability triggers such as authentication bypassing, hard-coded credentials, and memory corruption have been the subject of interest due to their ubiquity, less frequent alternatives are often overlooked during auditing. Misconfiguration, user-awareness, lack of regulatory compliance and standardization, as well as tainted data input and essential image management have received lesser attention due to the complexity of a potential investigation and variability across the spectrum of IoT-ware in use.

Due to heterogeneity of IoT-ware, auditing tools vastly focus on identifying and replicating recognized attacks, while only a few solutions focus on finding zero-day vulnerabilities. The versatility of existing tools is questionable, as most of them only cover a particular class or subclass of vulnerabilities and may not be easily extended to cover others. In hindsight, some solutions such as ANGR [121], Genius [122], Gemini [124], Avatar [40], and DICE [37] can detect numerous vulnerabilities due to their underlying methodology, but may also exhibit a high rate of false positives unless used by domain experts. Several solutions discussed earlier employ a wide range of methods for firmware analysis, including function profiling, program slicing, inter-relating shell scripts, code snippet emulation, and augmented process emulation. While beneficial for individual tools, establishing and developing a similar critical mass of equivalent human expertise in such a wide variety of techniques is very challenging. The complexity involved and the associated human expertise required can directly impact auditing results. In terms of future trends, research is increasingly focusing on machine learning and blockchain technology. ML and blockchain can, to an extent, bring further flexibility, adaptability, and automation to firmware auditing. However, harnessing the full spectrum of potential applications of these technologies remains an open initiative.

References

1. Trasvi-Moreno, C.A.; Blasco, R.; Casas, R.; Marco, A. Autonomous WiFi Sensor for Heating Systems in the Internet of Things. J. Sensors 2016, 2016, 7235984.
2. AlLifah, N.M.; Zualkernan, I. Ranking Security of IoT-based Smart Home Consumer Devices. IEEE Access 2022, 10, 18352–18369.
3. Das, A.; Sharma, S.C.M.; Ratha, B.K. The new era of smart cities, from the perspective of the internet of things. In Smart Cities Cybersecurity and Privacy; Elsevier: Amsterdam, The Netherlands, 2018; pp. 1–9.
4. Jeyaraj, P.R.; Nadar, E.R.S. Smart-Monitor: Patient Monitoring System for IoT-Based Healthcare System Using Deep Learning. IETE J. Res. 2022, 68, 1435–1442.

5. TajDini, M.; Sokolov, V.; Kuzminykh, I.; Shiaeles, S.; Ghita, B. Wireless Sensors for Brain Activity-A Survey. Electronics 2020, 9, 2092.
6. Pradha, S.E.; Moshika, A.; Natarajan, B.; Andal, K.; Sambasivam, G.; Shanmugam, M. Scheduled Access Strategy for Improving Sensor Node Battery Lifetime and Delay Analysis of Wireless Body Area Network. IEEE Access 2022, 10, 3459–3468.
7. Ni, Y.; Cai, L.; He, J.; Vinel, A.; Li, Y.; Mosavat-Jahromi, H.; Pan, J. Toward Reliable and Scalable Internet of Vehicles: Performance Analysis and Resource Management. Proc. IEEE 2020, 108, 324–340.
8. Kuzminykh, I. Development of traffic light control algorithm in smart municipal network. In Proceedings of the 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), Lviv, Ukraine, 23–26 February 2016; pp. 896–898.
9. Hunzinger, R. Scada fundamentals and applications in the IoT. In Internet of Things and Data Analytics Handbook; Wiley: Hoboken, NJ, USA, 2017; pp. 283–293.
10. Liberg, O.; Wang, Y.P.E.; Sachs, J.; Sundberg, M.; Bergman, J. Cellular Internet of Things—Technologies, Standards and Performance; Academic Press: Cambridge, MA, USA, 2017; Chapter 9; pp. 327–360.
11. Chaudhari, B.S.; Zennaro, M. LPWAN Technologies for IoT and M2M Applications; Academic Press: Cambridge, MA, USA, 2020.
12. Kshetri, N. The evolution of the internet of things industry and market in China: An interplay of institutions, demands and supply. Telecommun. Policy 2017, 41, 49–67.
13. Kuzminykh, I.; Ghita, B.; Such, J.M. The Challenges with Internet of Things Security for Business. In Internet of Things, Smart Spaces, and Next Generation Networks and Systems; Koucheryavy, Y., Balandin, S., Andreev, S., Eds.; Springer: Cham, Switzerland, 2022; LNCS; Volume 13158, pp. 46–58.
14. Sørensen, A.; Wang, H.; Remy, M.J.; Kjettrup, N.; Sørensen, R.B.; Nielsen, J.J.; Popovski, P.; Madueño, G.C. Modelling and Experimental Validation for Battery Lifetime Estimation in NB-IoT and LTE-M. IEEE Internet Things J. 2022, 9, 9804–9819.
15. Kuzminykh, I.; Yevdokymenko, M.; Sokolov, V. Encryption Algorithms in IoT: Security vs. Lifetime. Available online: https://ssrn.com/abstract=4636161 (accessed on 29 November 2023).
16. Gupta, A.; Guzman, A. IoT Penetration Testing Cookbook: Identify Vulnerabilities and Secure Your Smart Devices; Packt Publishing: Birmingham, UK, 2017; ISBN 9781787280571.
17. Abdul-Ghani, H.A.; Konstantas, D.; Mahyoub, M. A comprehensive IoT attacks survey based on a building-blocked reference model. Int. J. Adv. Comput. Sci. Appl. 2018, 9, 355–373.

18. Adat, V.; Gupta, B.B. Security in Internet of Things: Issues, challenges, taxonomy, and architecture. Telecommun. Syst. 2018, 67, 423–441.
19. Ammar, M.; Russello, G.; Crispo, B. Internet of Things: A survey on the security of IoT frameworks. J. Inf. Secur. Appl. 2018, 38, 8–27.
20. Kuzminykh, I.; Carlsson, A.; Yevdokymenko, M.; Sokolov, V. Investigation of the IoT Device Lifetime with Secure Data Transmission. In Internet of Things, Smart Spaces, and Next Generation Networks and Systems (NEW2AN/ruSMART 2019); Galinina, O., Andreev, S., Balandin, S., Koucheryavy, Y., Eds.; Springer: Cham, Switzerland, 2019; LNCS; Volume 11660, pp. 16–27.
21. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and other botnets. Computer 2017, 50, 80–84.
22. Ling, Z.; Liu, K.; Xu, Y.; Jin, Y.; Fu, X. An End-to-End View of IoT Security and Privacy. In Proceedings of the GLOBECOM 2017—2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–7.
23. Microsoft. Security Signals March 2021. Available online: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWPStZ (accessed on 29 November 2023).
24. Rothman, M.; Zimmer, V. Understanding UEFI Firmware Update and Its Vital Role in Keeping Computing Systems Secure. Available online: https://embeddedcomputing.com/technology/security/software-security/understanding-uefi-firmware-update-and-its-vital-role-in-keeping-computing-systems-secure (accessed on 15 October 2023).
25. Vasile, S.; Oswald, D.; Chothia, T. Breaking All the Things-A Systematic Survey of Firmware Extraction Techniques for IoT Devices. In Smart Card Research and Advanced Applications; Springer: Cham, Switzerland, 2019; pp. 171–185.
26. Quantum IoT Protect Firmwar—Security Risk Assessment. Available online: https://pages.checkpoint.com/iot-firmware-risk-assessment.html (accessed on 29 November 2023).
27. 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.
28. Xie, W.; Jiang, Y.; Tang, Y.; Ding, N.; Gao, Y. Vulnerability Detection in IoT Firmware: A Survey. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017; pp. 769–772.
29. Costin, A.; Zaddach, J. IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies. In Proceedings of the Black Hat USA 2018, Las Vegas, NV, USA, 4–9 August 2018; pp.

1–7. Available online: http://firmware.re/malw/bh18us_costin.pdf (accessed on 29 November 2023).

30. Mohanty, A.; Obaidat, I.; Yilmaz, F.; Sridhar, M. Control-hijacking vulnerabilities in IoT firmware: A brief survey. In Proceedings of the 1st International Workshop on Security and Privacy for the Internet-of-Things, Orlando, FL, USA, 17–20 April 2020; pp. 1–42018.
31. Costin, A. Security of CCTV and Video Surveillance Systems: Threats, Vulnerabilities, Attacks, and Mitigations. In Proceedings of the 6th International Workshop on Trustworthy Embedded Devices, Vienna, Austria, 28 October 2016; pp. 45–54.
32. Thomas, S.L.; Garcia, F.D.; Chothia, T. HumIDIFy: A Tool for Hidden Functionality Detection in Firmware. In Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2017); Polychronakis, M., Meier, M., Eds.; Springer: Cham, Switzerland, 2017; Volume 10327, pp. 279–300.
33. Cojocar, L.; Zaddach, J.; Verdult, R.; Bos, H.; Francillon, A.; Balzarotti, D. PIE: Parser Identification in Embedded Systems. In Proceedings of the 31st Annual Computer Security Applications Conference, New York, NY, USA, 7–11 December 2015; pp. 251–260.
34. Costin, A.; Zaddach, J.; Francillon, A.; Balzarotti, D. Large Scale Security Analysis of Embedded Devices' Firmware. In Proceedings of the 23rd USENIX Conference on Security Symposium, San Diego, CA, USA, 20–22 August 2014; pp. 95–110.
35. David, Y.; Partush, N.; Yahav, E. FirmUp: Precise Static Detection of Common Vulnerabilities in Firmware. ACM SIGPLAN Not. 2018, 53, 392–404.
36. Costin, A.; Zarras, A.; Francillon, A. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. In Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIA CCS), Xi'an, China, 30 May–3 June 2016; pp. 437–448.
37. Mera, A.; Feng, B.; Lu, L.; Kirda, E. DICE: Automatic Emulation of DMA Input Channels for Dynamic Firmware Analysis. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; pp. 1938–1954.
38. Chen, D.D.; Woo, M.; Brumley, D.; Egele, M. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In Proceedings of the NDSS Symposium 2016, San Diego, CA, USA, 21–24 February 2016.
39. Gui, Z.; Shu, H.; Kang, F.; Xiong, X. FIRMCORN: Vulnerability-Oriented Fuzzing of IoT Firmware via Optimized Virtual Execution. IEEE Access 2020, 8, 29826–29841.
40. Zaddach, J.; Bruno, L.; Francillon, A.; Balzarotti, D. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. In Proceedings of the NDSS Symposium 2014, San Diego, CA, USA, 23–24 February 2014.

41. Feng, B.; Mera, A.; Lu, L. P2IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling. In Proceedings of the 29th USENIX Conference on Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 1237–1254.
42. Clements, A.A.; Gustafson, E.; Scharnowski, T.; Grosen, P.; Fritz, D.; Kruegel, C.; Vigna, G.; Bagchi, S.; Payer, M. HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation. In Proceedings of the 29th USENIX Conference on Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 1201–1218.
43. Gustafson, E.; Muench, M.; Spensky, C.; Redini, N.; Machiry, A.; Fratantonio, Y.; Balzarotti, D.; Francillon, A.; Choe, Y.R.; Kruegel, C.; et al. Toward the Analysis of Embedded Firmware through Automated Re-hosting. In Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Beijing, China, 23–25 September 2019; pp. 135–150.
44. Liu, X.; Cui, B.; Fu, J.; Ma, J. HFuzz: Towards automatic fuzzing testing of NB-IoT core network protocols implementations. Future Gener. Comput. Syst. 2020, 108, 390–400.
45. Maier, D.; Radtke, B.; Harren, B. Unicorefuzz: On the viability of emulation for kernel space fuzzing. In Proceedings of the 13th USENIX Workshop on Offensive Technologies, Santa Clara, CA, USA, 12–13 August 2019; p. 8.
46. Wang, D.; Zhang, X.; Chen, T.; Li, J. Discovering Vulnerabilities in COTS IoT Devices through Blackbox Fuzzing Web Management Interface. Secur. Commun. Netw. 2019, 2019, 5076324.
47. Davidson, D.; Moench, B.; Jha, S.; Ristenpart, T. FIE on firmware: Finding vulnerabilities in embedded systems using symbolic execution. In Proceedings of the 22nd USENIX Conference on Security Symposium, Washington, DC, USA, 14–16 August 2013; pp. 463–478.
48. Yao, Y.; Zhou, W.; Jia, Y.; Zhu, L.; Liu, P.; Zhang, Y. Identifying Privilege Separation Vulnerabilities in IoT Firmware with Symbolic Execution. In Computer Security—ESORICS 2019; Sako, K., Schneider, S., Ryan, P., Eds.; Springer: Cham, Switzerland, 2019; Volume 11735, pp. 638–657.
49. Shwartz, O.; Mathov, Y.; Bohadana, M.; Elovici, Y.; Oren, Y. Reverse Engineering IoT Devices: Effective Techniques and Methods. IEEE Internet Things J. 2018, 5, 4965–4976.
50. Zaddach, J.; Costin, A. Embedded Devices Security and Firmware Reverse Engineering. In Proceedings of the Black Hat USA 2013, Las Vegas, NV, USA, 31 July–1 August 2013.
51. Neshenko, N.; Bou-Harb, E.; Crichigno, J.; Kaddoum, G.; Ghani, N. Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. IEEE Commun. Surv. Tutor. 2019, 21, 2702–2733.
52. Makhdoom, I.; Abolhasan, M.; Lipman, J.; Liu, R.P.; Ni, W. Anatomy of Threats to the Internet of Things. IEEE Commun. Surv. Tutor. 2019, 21, 1636–1675.

53. Cao, C.; Guan, L.; Ming, J.; Liu, P. Device-agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation. In Proceedings of the 36th Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December 2020; pp. 746–759.
54. Palavicini, G., Jr.; Bryan, J.; Sheets, E.; Kline, M.; San Miguel, J. Towards Firmware Analysis of Industrial Internet of Things (IIoT)–Applying Symbolic Analysis to IIoT Firmware Vetting. In Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017), Porto, Portugal, 24–26 April 2017; pp. 470–477.
55. Wang, Z.; Zhang, Y.; Tian, Z.; Ruan, Q.; Liu, T.; Wang, H.; Liu, Z.; Lin, J.; Fang, B.; Shi, W. Automated Vulnerability Discovery and Exploitation in the Internet of Things. Sensors 2019, 19, 3362.
56. Boudguiga, A.; Bouzerna, N.; Granboulan, L.; Olivereau, A.; Quesnel, A.; Roger, A.; Sirdey, R. Towards Better Availability and Accountability for IoT Updates by Means of a Blockchain. In Proceedings of the 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Paris, France, 26–28 April 2017; pp. 50–58.
57. Choi, S.; Lee, J.H. Blockchain-Based Distributed Firmware Update Architecture for IoT Devices. IEEE Access 2020, 8, 37518–37525.
58. Fukuda, T.; Omote, K. Efficient Blockchain-based IoT Firmware Update Considering Distribution Incentives. In Proceedings of the 2021 IEEE Conference on Dependable and Secure Computing (DSC), Aizuwakamatsu, Japan, 30 January–2 February 2021; pp. 1–8.
59. Cao, B.; Liu, W.; Peng, M. Blockchain Driven Internet of Things. In Wireless Blockchain: Principles, Technologies and Applications; Imran, M.A., Cao, B., Zhang, L., Peng, M., Eds.; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2021; pp. 93–115.
60. Lee, B.; Lee, J.H. Blockchain-based secure firmware update for embedded devices in an Internet of Things environment. J. Supercomput. 2017, 73, 1152–1167.
61. Lee, B.; Malik, S.; Wi, S.; Lee, J.H. Firmware Verification of Embedded Devices Based on a Blockchain. In Quality, Reliability, Security and Robustness in Heterogeneous Networks; Lee, J.H., Pack, S., Eds.; Springer: Cham, Switzerland, 2017; LNICSSITE; Volume 199, pp. 52–61.
62. Ream, J.; Chu, Y.; Schatsky, D. Upgrading Blockchains: Smart Contract Use Cases in Industry; Deloitte University Press: Westlake, TX, USA, 2016; Available online: https://www2.deloitte.com/us/en/insights/focus/signals-for-strategists/using-blockchain-for-smart-contracts.html (accessed on 1 December 2023).
63. Witanto, E.N.; Oktian, Y.E.; Lee, S.-G.; Lee, J.-H. A Blockchain-Based OCF Firmware Update for IoT Devices. Appl. Sci. 2020, 10, 6744.
64. Yohan, A.; Lo, N.-W. An Over-the-Blockchain Firmware Update Framework for IoT Devices. In Proceedings of the 2018 IEEE Conference on Dependable and Secure Computing (DSC),

Kaohsiung, Taiwan, 10–13 December 2018; pp. 1–8.

65. Yohan, A.; Lo, N.-W. FOTB: A secure blockchain-based firmware update framework for IoT environment. Int. J. Inf. Secur. 2020, 19, 257–278.
66. Yohan, A.; Lo, N.-W.; Santoso, L.P. Secure and Lightweight Firmware Update Framework for IoT Environment. In Proceedings of the 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE), Osaka, Japan, 5–18 October 2019; pp. 684–685.
67. Miettinen, M.; Marchal, S.; Hafeez, I.; Frassetto, T.; Asokan, N.; Sadeghi, A.-R.; Tarkoma, S. IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2511–2514.
68. Costin, A.; Zarras, A.; Francillon, A. Towards Automated Classification of Firmware Images and Identification of Embedded Devices. In ICT Systems Security and Privacy Protection (SEC 2017); De Capitani di Vimercati, S., Martinelli, F., Eds.; Springer: Cham, Switzerland, 2017; IFIP AICT; Volume 502, pp. 233–247.
69. Lee, S.; Paik, J.-Y.; Jin, R.; Cho, E.-S. Toward Machine Learning Based Analyses on Compressed Firmware. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; pp. 586–591.
70. Yu, D.; Zhang, L.; Chen, Y.; Ma, Y.; Chen, J. Large-Scale IoT Devices Firmware Identification Based on Weak Password. IEEE Access 2020, 8, 7981–7992.
71. Pinto, S.; Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. ACM Comput. Surv. 2019, 51, 130.
72. Koutroumpouchos, N.; Ntantogian, C.; Xenakis, C. Building Trust for Smart Connected Devices: The Challenges and Pitfalls of TrustZone. Sensors 2021, 21, 520.
73. Koeberl, P.; Schulz, S.; Sadeghi, A.-R.; Varadharajan, V. TrustLite: A security architecture for tiny embedded devices. In Proceedings of the 9th European Conference on Computer Systems, Amsterdam, The Netherlands, 14–16 April 2014; pp. 1–14.
74. Dushku, E.; Østergaard, J.H.; Dragoni, N. Memory Offloading for Remote Attestation of Multi-Service IoT Devices. Sensors 2022, 22, 4340.
75. Eldefrawy, K.; Tsudik, G.; Francillon, A.; Perito, D. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In Proceedings of the NDSS Symposium 2019, San Diego, CA, USA, 24–27 February 2019.
76. Brasser, F.; Rasmussen, K.B.; Sadeghi, A.-R.; Tsudik, G. Remote attestation for low-end embedded devices: The prover’s perspective. In Proceedings of the 53rd Annual Design Automation Conference, Austin, TX, USA, 5–9 June 2016; pp. 1–6.

77. Hristozov, S.; Heyszl, J.; Wagner, S.; Sigl, G. Practical Runtime Attestation for Tiny IoT Devices. In Proceedings of the NDSS Workshop on Decentralized IoT Security and Standards (DISS), San Diego, CA, USA, 18 February 2018.
78. Conti, M.; Dushku, E.; Mancini, L.V.; Rabbani, M.; Ranise, S. Remote Attestation as a Service for IoT. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 320–325.
79. Tsoutsos, N.G.; Maniatakos, M. Anatomy of Memory Corruption Attacks and Mitigations in Embedded Systems. IEEE Embed. Syst. Lett. 2018, 10, 95–98.
80. Strackx, R.; Piessens, F.; Preneel, B. Efficient Isolation of Trusted Subsystems in Embedded Systems. In Security and Privacy in Communication Networks; Jajodia, S., Zhou, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; LNICSSITE; Volume 50, pp. 344–361.
81. Shoshitaishvili, Y.; Wang, R.; Hauser, C.; Kruegel, C.; Vigna, G. Firmalice–Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. In Proceedings of the NDSS Symposium 2015, San Diego, CA, USA, 8–11 February 2015.
82. Gupta, A. The IoT Hacker’s Handbook—A Practical Guide to Hacking the Internet of Things; Apress: Berkeley, CA, USA, 2019.
83. Hamada, R.; Kuzminykh, I. Exploitation Techniques of IoST Vulnerabilities in Air-Gapped Networks and Security Measures—A Systematic Review. Signals 2023, 4, 687–707.
84. Greenberg, A. The Reaper IoT Botnet Has Already Infected a Million Networks. Available online: https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/ (accessed on 4 December 2023).
85. Barcena, M.; Wueest, C. Insecurity in the Internet of Things, Symantec Report. Available online: https://docs.broadcom.com/doc/insecurity-in-the-internet-of-things-en (accessed on 1 December 2023).
86. Kaspersky Unveils an Overview of IoT-Related Threats in 2023. Available online: https://www.kaspersky.com/about/press-releases/2023_kaspersky-unveils-an-overview-of-iot-related-threats-in-2023 (accessed on 15 October 2023).
87. Mandal, A.; Ferrara, P.; Khlyebnikov, Y.; Cortesi, A.; Spoto, F. Cross-program taint analysis for IoT systems. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; pp. 1944–1952.
88. Lezzi, M.; Lazoi, M.; Corallo, A. Cybersecurity for Industry 4.0 in the current literature: A reference framework. Comput. Ind. 2018, 103, 97–110.
89. Wazzan, M.; Algazzawi, D.; Bamasaq, O.; Albeshri, A.; Cheng, L. Internet of Things Botnet Detection Approaches: Analysis and Recommendations for Future Research. Appl. Sci. 2021, 11,

5713.

90. IoT Alliance Australia. Available online: https://iot.org.au/ (accessed on 4 December 2023).
91. Fagan, M.; Megas, K.; Scarfone, K.; Smith, M. NIST IR 8259; Foundational Cybersecurity Activities for IoT Device Manufacturers. NIST: Gaithersburg, MD, USA, 2020.
92. Regenscheid, A. NIST SP 800-193; Platform Firmware Resiliency Guidelines. NIST: Gaithersburg, MD, USA, 2018.
93. Abera, T.; Asokan, N.; Davi, L.; Koushanfar, F.; Paverd, A.; Sadeghi, A.-R.; Tsudik, G. Invited: Things, trouble, trust: On building trust in IoT systems. In Proceedings of the 53rd Annual Design Automation Conference, Austin, TX, USA, 5–9 June 2016; pp. 1–6.
94. Sutherland, I.; Kalb, G.E.; Blyth, A.; Mulley, G. An empirical examination of the reverse engineering process for binary files. Comput. Secur. 2006, 25, 221–228.
95. Manske, A. Conducting a Vulnerability Assessment of an IP Camera. Master's Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2019.
96. Mansfield-Devine, S. Ransomware: Taking businesses hostage. Netw. Secur. 2016, 2016, 8–17.
97. Chen, J.; Diao, W.; Zhao, Q.; Zuo, C.; Lin, Z.; Wang, X.F.; Lau, W.C.; Sun, M.; Yang, R.; Zhang, K. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing. In Proceedings of the NDSS Symposium 2018, San Diego, CA, USA, 18–21 February 2018; pp. 1–15.
98. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. 2012, 44, 6.
99. Binwalk: Firmware Analysis Tool. Available online: https://github.com/ReFirmLabs/binwalk (accessed on 4 December 2023).
100. Hemel, A. binaryanalysis-ng: Binary Analysis Next Generation (BANG). Available online: https://github.com/armijnhemel/binaryanalysis-ng (accessed on 4 December 2023).
101. FMK: Firmware Mod Kit. Available online: https://github.com/rampageX/firmware-mod-kit/ (accessed on 4 December 2023).
102. The Firmware Analysis and Comparison Tool (FACT). Available online: https://github.com/fkie-cad/FACT_core (accessed on 4 December 2023).
103. Popa, M. Binary Code Disassembly for Reverse Engineering. J. Mob. Embed. Distrib. Syst. 2012, IV, 233–248.
104. Serrano, M. Lecture Notes on Decompilation; Lecture 20; Carnegie Mellon School of Computer Science: Pittsburgh, PA, USA, 2013.
105. gdb(1)—Linux Man Page. Available online: https://linux.die.net/man/1/gdb (accessed on 1 December 2023).

106. Radare2: Libre Reversing Framework for Unix Geeks. Available online: https://github.com/radareorg/radare2 (accessed on 4 December 2023).
107. Vector 35, Binary Ninja. Available online: https://binary.ninja/features/ (accessed on 4 December 2023).
108. IDA Pro: A Powerful Disassembler and a Versatile Debugger. Available online: https://hex-rays.com/ida-pro/ (accessed on 4 December 2023).
109. Ghidra Firmware Utilities. Available online: https://github.com/al3xtjames/ghidra-firmware-utils (accessed on 4 December 2023).
110. Cadar, C.; Dunbar, D.; Engler, D. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI), San Diego, CA, USA, 8–10 December 2008; pp. 209–224.
111. FAT: Firmware Analysis Toolkit. Available online: https://github.com/attify/firmware-analysis-toolkit (accessed on 1 December 2023).
112. Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of the USENIX Annual Technical Conference, Anaheim, CA, USA, 10–15 April 2005; pp. 41–46.
113. Zandberg, K.; Schleiser, K.; Acosta, F.; Tschofenig, H.; Baccelli, E. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. IEEE Access 2019, 7, 71907–71920.
114. Zalewski, M. AFL: American Fuzzy Lop. Available online: https://github.com/google/AFL (accessed on 4 December 2023).
115. Hicken, A. How Does Static Analysis Prevent Defects & Accelerate Delivery? Available online: https://www.parasoft.com/blog/how-does-static-analysis-prevent-defects-and-accelerate-delivery/ (accessed on 1 December 2023).
116. Chess, B.; Mcgraw, G. Static analysis for security. IEEE Secur. Priv. 2004, 2, 76–79.
117. Johnson, S.C. Lint, a C Program Checker. Comp Sci Tech. Rep. 1978, 65, 1–11.
118. Chen, H.; Dean, D.; Wagner, D. Model Checking One Million Lines of C Code. In Proceedings of the NDSS Symposium 2004, San Diego, CA, USA, 5 February 2004; pp. 171–185.
119. Fagbuyiro, D. Benefits of Using Static Code Analysis Tools for Software Testing. Available online: https://www.stickyminds.com/article/benefits-using-static-code-analysis-tools-software-testing (accessed on 1 December 2023).
120. Grace, M.; Zhou, Y.; Wang, Z.; Jiang, X.; Drive, O. Systematic Detection of Capability Leaks in Stock Android Smartphones. In Proceedings of the NDSS Symposium 2012, San Diego, CA,

USA, 5–8 February 2012; pp. 1–15.

121. Angr: Platform-Agnostic Binary Analysis Framework. Available online: https://github.com/angr/angr (accessed on 4 December 2023).
122. Feng, Q.; Zhou, R.; Xu, C.; Cheng, Y.; Testa, B.; Yin, H. Scalable graph-based bug search for firmware images. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 480–491.
123. Zhou, W.; Guan, L.; Liu, P.; Zhang, Y. Automatic Firmware Emulation through Invalidity-guided Knowledge Inference. In Proceedings of the 30th USENIX Conference on Security Symposium, Virtual, 11–13 August 2021; pp. 2007–2024.
124. Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; Song, D. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 363–376.