Do you Trust your Device? Open Challenges in IoT Security Analysis

Lorenzo Binosi[1], Pietro Mazzini[2], Alessandro Sanna[3], Michele Carminati[1], Giorgio Giacinto[3], Riccardo Lazzeretti[2], Stefano Zanero[1], Mario Polino[1], Emilio Coppa[4], and Davide Maiorca[3]

[1]*Politecnico of Milan, Italy*

[2]*Sapienza University, Italy*

[3]*University of Cagliari, Italy*

[4]*LUISS University, Italy*

{*lorenzo.binosi, michele.carminati, mario.polino, stefano.zanero*}*@polimi.it,* {*mazzini, lazzeretti*}*@diag.uniroma1.it,* {*alessandro.sanna96, giorgio.giacinto, davide.maiorca*}*@unica.it, ecoppa@luiss.it*

Keywords: Firmware Analysis, IoT, Security evaluation.

Abstract: Several critical contexts, such as healthcare, smart cities, drones, transportation, and agriculture, nowadays rely on IoT, or more in general embedded, devices that require comprehensive security analysis to ensure their integrity before deployment. Security concerns are often related to vulnerabilities that result from inadequate coding or undocumented features that may create significant privacy issues for users and companies. Current analysis methods, albeit dependent on complex tools, may lead to superficial assessments due to compatibility issues, while authoritative entities struggle with specifying feasible firmware analysis requests for manufacturers within operational contexts. This paper urges the scientific community to collaborate with stakeholders—manufacturers, vendors, security analysts, and experts—to forge a cooperative model that clarifies manufacturer contributions and aligns analysis demands with operational constraints. Aiming at a modular approach, this paper highlights the crucial need to refine security analysis, ensuring more precise requirements, balanced expectations, and stronger partnerships between vendors and analysts. To achieve this, we propose a threat model based on the feasible interactions of actors involved in the security evaluation of a device, with a particular emphasis on the responsibilities and necessities of all entities involved.

1 INTRODUCTION

IoT devices are progressively becoming part of our everyday lives, as they are widely used in various critical contexts, such as healthcare, smart cities, transportation, agriculture, and several others (European Cyber Security Organisation - ECSO, 2022). Consequently, third-party manufacturers working on such devices are significantly larger than in other traditional computing markets (servers, laptops, and desktop workstations).

This variety leads to the development of different hardware and software platforms for devices with the same goals, typically featuring firmware characterized by various levels of complexity (depending on the target application and the architecture). Unfortunately, to keep their products economically competitive, manufacturers need to integrate new features quickly. They may omit rigorous testing of their platforms, thus possibly ignoring well-established software security best practices and leaving undocumented privacy-sensitive functionalities.

This aspect poses a significant security concern: *can we trust devices we did not analyze?* In particular, these devices can either embed *vulnerabilities* that miscreants can exploit to execute, e.g., unauthorized code, or they can even *hide* privacy-breaking functionalities to gather information about users. From this respect, we can imagine two categories of vendors: *(i) benign vendors*, who can unintentionally produce vulnerable code due to the time-the-market pressure; *(ii) malicious vendors*, whose goal is to embed hidden functionalities or vulnerabilities deliberately.

Unsurprisingly, analyzing such devices following standard security evaluation methodologies that rely on popular reverse engineering techniques is troublesome due to the custom platforms used by the devices, making most state-of-the-art tools fail to provide, in some cases, even basic information. Hence, a major question emerges: *what do we need to analyze IoT*

devices effectively?

This paper considers the scenario where an institutional or industrial entity needs to perform a security evaluation of a device before deploying it in production. In particular, we identify two significant aspects:

- The *investigation of the complex dynamics impacting IoT security evaluations*. In particular, the device manufacturer may adopt *anti-analyses techniques* to protect its intellectual property from competitors and complicate the actions of external attackers interested in exploiting the device. However, such strategies can also make traditional security evaluation methodologies ineffective and could be thus seen as hostile practices by a security evaluator. In this complex scenario, a significant research problem is what the manufacturers could reasonably provide to permit effective security evaluations while considering their needs. Reasoning on the minimum requirements for effective security evaluations is inevitably tied with the design of state-of-the-art analysis tools.
- The *development of novel security evaluation approaches*. First, security evaluators should be aware of the possible *anti-analysis* practices a manufacturer may adopt. Hence, we need to devise tools to detect the adoption of such techniques. Second, assuming the partial cooperation of a manufacturer (motivated by the economic incentive of selling its own device), we believe there is a need to rethink existing security evaluation frameworks, making them extensible with minimal building blocks that could be reasonably asked the manufacturer when carrying out an IoT security evaluation.

This paper explores these two significant research directions in more detail. More specifically, our contributions are the following:

- We describe the *actors* involved in a security evaluation of a device, thus considering the vendors, the manufacturers, the attackers, the security evaluator, and the relationship between them.
- We describe a general *workflow* to conduct a security evaluation of a device, pinpointing some anti-analysis techniques that could be adopted by device manufacturers and may impact the effectiveness of the security evaluations.
- We sketch *open challenges* and *future research directions* that impose significant research efforts by the scientific community.

We believe that our work can be helpful to the scientific community in orienting future research works and inspiring new ideas to secure the IoT devices landscape.

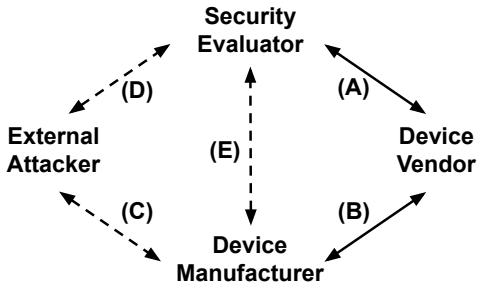

Figure 1: Actors that play a role in a security evaluation of a device. Solid edges (A and B) represent direct interactions, while dotted edges (C, D, and E) are indirect relationships.

2 IOT SECURITY ANALYSIS: CONTEXT

IoT, and more in general embedded devices, pose several security concerns that historically were less perceived when considering traditional systems. Indeed, such devices may come with heavily customized hardware and software platforms developed by different device manufacturers, making it quite hard to assess their security level using standard security methodologies and tools. In this heterogeneous landscape, several institutional and private entities need to verify the security of these devices before deploying them in potentially critical production environments.

In this section, we describe the actors that have a direct or indirect role in this scenario, the workflow that could be followed during the security evaluation of the device, and the several issues that could emerge when carrying out the evaluation.

2.1 Involved Actors

As exemplified by Figure 1, we can identify at least four major actors with a direct or indirect role in the security evaluation of an embedded device.

Security Evaluator. This actor is the entity in charge of performing the security evaluation on behalf of the device buyer. It directly interacts with the device vendor (edge A in Figure 1) and, in several cases, can pose some requirements about the device and the technical material (documentation, software artifacts, etc.) needed by the evaluator to test the device. Indirectly, it also interacts with the device manufacturer (edge E) since the device crucially depends on this actor. Finally, the security evaluator should often verify whether the device may contain vulnerabilities (e.g., outdated and vulnerable libraries) that could be exploited by an external attacker (edge D).

Device Vendor. This entity commercially offers a device in the market. It directly interacts with the manu-

facturer to obtain the device (edge B) and with the security evaluator when selling the device (edge A). Unfortunately, the device vendor does not always have full knowledge about the internals of a device. Hence, its cooperation may not be sufficient to have an accurate view of a device. This situation implies that a vendor could not satisfy some requirements from the security evaluator, even when cooperation is genuinely offered.

Device Manufacturer. This entity produces the device and is expected to have full knowledge of the device's internals. Notice that, in many cases, the device manufacturer may rely on other manufacturers (e.g., the System-On-Chip manufacturer). However, we expect it to be aware of the relevant internal details of the third-party components. The manufacturer may adopt any strategy to protect its own intellectual rights, including making choices or adopting anti-analysis techniques that may hinder, or at least slow down, reverse engineering attacks. Moreover, such anti-analysis techniques could also be adopted to protect end users since they may slow down external attackers (edge C) interested in exploiting the device. However, to keep their product economically competitive, manufacturers often have to quickly integrate innovative features in response to consumer pressure, possibly skipping proper testing due to the need to reduce production costs. Moreover, such vendors may reside in quite different countries with respect to the nationality of the final consumers, possibly inducing a mismatch between regulations and strategic national interests. Therefore, they could rely on anti-analysis techniques to protect poorly tested code and hide undocumented functionalities.

External Attacker. This entity may be interested in finding and exploiting a vulnerability on the device. Such exploitation concerns both the security evaluator (edge D), which would like to detect any easy-to-spot flaw, and the device manufacturer (edge C), which can experience significant reputation damage, potentially harming its business in the long run.

2.2 Security Evaluation Workflow

The security evaluation of an embedded device is a challenging task that may require different approaches and methodologies depending on the device under analysis and the level of cooperation with the device vendor. Various factors significantly influence the evaluation strategy, such as the manufacturer's transparency in sharing firmware and source code or using obfuscation and security features to protect the device. Despite these variations, we identify three major steps to approach the security evaluation of embedded devices (Nadir et al., 2022). For the sake of simplicity, in this paper, we do not consider the steps of analysis aimed at reverse engineering the device hardware internals since such evaluation would most likely be out of reach for most security evaluators. Nonetheless, evaluators indirectly validate the hardware behaviour by checking the device execution through dynamic analyses.

Firmware Extraction. The security analysis begins with acquiring the embedded device's firmware. Notice that this process is performed even when the source code is available since the evaluator may need to check whether the firmware binary is consistent with the released source code. In some instances, manufacturers facilitate the firmware acquisition process by providing the plaintext binary directly on their websites or enabling access via remote interfaces such as SSH or Telnet, allowing complete firmware retrieval. However, as stated in the previous section, this may not always be true. Manufacturers may want to protect intellectual properties and only provide the device, making it necessary to extract the firmware from the device. Typically, this process involves extracting the firmware by dumping the flash memory or disk storage – depending on the firmware's location – via a debug interface like JTAG or UART, which are standard in such scenarios. For instance, a common tool utilized for exploiting this interface is the JTAGulator[1], a hardware tool by Grand Idea Studio used to identify the usually hidden JTAG interface. However, mechanisms like Readout Protection (RDP), Secure Boot, and encryption may complicate the extraction process, often requiring cooperation with the manufacturer to access the firmware successfully. Readout Protection, for instance, may be used to disable the JTAG interface altogether, preventing the device from being reprogrammed without a manual reflash.

Static Analysis. This technique involves the analysis of the source code (when available) and the assembly code of the firmware to identify potential security flaws without running the program itself (Costin et al., 2014; David et al., 2018). Since the availability of the full source code is quite rare in most scenarios, we focus our discussion on techniques that operate on the binary code.

Static analysis of the firmware begins with disassembling the binary image to distinguish between code and data and to identify further assembly references, function boundaries, data structures, etc. This initial step is fundamental for the success of static analysis; a partial or erroneous disassembly process

[1] https://grandideastudio.com/portfolio/security/jtagulator

may lead to an inconclusive analysis. It is then possible to leverage static analysis techniques such as Value-Set Analysis (Balakrishnan and Reps, 2004; Balakrishnan et al., 2005) or Taint analysis (Schwartz et al., 2010) to identify memory corruption vulnerabilities such as buffer overflows. Over the years, many state-of-the-art approaches have adopted heuristics to identify vulnerabilities in the firmware of embedded devices (Neshenko et al., 2019). Some of them are collected in frameworks, such as *FACT* (Fraunhofer FKIE-CAD, 2017), or *EMBA* (Secure Firmware, 2022), to perform several analyses simultaneously. For instance, the *CVE Lookup* analysis tries to match publicly disclosed vulnerabilities with binaries within the firmware. However, this analysis does consider single binaries and does not consider the whole picture of the firmware. Even if there are already known vulnerabilities in a few firmware binaries, they might not represent real vulnerabilities, especially when the vulnerable binaries are never executed or the vulnerable functionalities are never used. This happens often in embedded devices, and simple analysis may produce false positives. Moreover, vendors usually patch vulnerabilities on the spot without updating the entire code base; this choice is mainly due to the adoption of old toolchains, which do not always support updated binaries/libraries (Yu et al., 2022). Hence, more advanced techniques are required. To reduce the number of false positives, *KARONTE* (Redini et al., 2020) leverages multi-binary taint analysis to detect vulnerabilities following the data flow of relevant firmware applications. Moreover, KARONTE considers all the flows generated via Inter-Process Communication (IPC) techniques such as `fork()`, `execve()`, shared `mmap()`, etc. With this approach, KARONTE detected 46 zero-days over 53 firmware, showing that it scales regardless of the firmware size. Albeit promising, taint analysis has well-known limitations, and it cannot detect complex vulnerabilities.

Another approach, not only designed for embedded devices, tries to identify function clones (Hu et al., 2017). This approach is used for both reverse engineering and vulnerability detection; if a known vulnerable function is detected, then the application might be vulnerable.

Finally, other approaches rely on heuristics and Machine Learning models (Nicolao et al., 2018; Remigio et al., 2023; Thomas et al., 2017). ELISA and its extension (Nicolao et al., 2018; Remigio et al., 2023) present a supervised machine learning framework designed for code discovery in header-less binary files, aiding static analysis and reverse engineering by distinguishing executable instructions from firmware without metadata. They use a two-step process: first identifying the Instruction Set Architecture (ISA) using logistic regression, and then delineating code boundaries and identifying data within those boundaries using Conditional Random Fields (CRFs) in (Nicolao et al., 2018) and lstm-based with heuristics in (Remigio et al., 2023). HumIDIfy (Thomas et al., 2017) is a system developed to identify undocumented functionality and authentication bypass vulnerabilities in firmware. It utilizes machine learning by creating profiles of expected firmware behavior using a Binary Functionality Description Language. These profiles are then compared to the actual behavior of the firmware in real-time. Significant deviations suggest the presence of hidden functionalities.

Dynamic Analysis. A significant limitation of static analysis is the inability to test and exploit systems without physical access to the device. Unlike static analysis, which involves dissecting and reading code, dynamic analysis allows for the execution of code and observation of its behavior without detailed knowledge of the program's internals. Testing the firmware on the physical device, from one side, allows the evaluator to validate the actual behavior of the device, but, from the other side, often offers limited inspection capabilities (e.g., to check the internal execution state) and does not scale well when only a limited number of devices is available. Hence, the research community has often suggested relying on device emulators, such as QEMU (Bellard, 2005), to run the firmware images, making it easier to parallelize the analysis and integrate advanced vulnerability detection techniques, such as software fuzzing and symbolic execution. Dynamic analysis, however, requires metadata to emulate firmware accurately, which necessitates some understanding of the firmware's architecture. In addition, dynamic analysis has scalability issues due to the complex and varied configuration setups needed for different devices.

Introduced in 2014, Avatar (Zaddach et al., 2014) is a framework designed for dynamic analysis of embedded devices' firmware. It emulates firmware instructions on an external machine while directing I/O operations to the actual device hardware. Avatar has been utilized in security applications such as reverse engineering, vulnerability discovery, and backdoor detection. It has been tested on diverse hardware types, including a GSM phone, a hard disk bootloader, and a wireless sensor node, using QEMU (Bellard, 2005) for firmware emulation and KLEE (Cadar et al., 2008) to explore firmware paths for vulnerabilities. However, Avatar's reliance on real hardware for vulnerability discovery and the slow execution of its processes from emulation to hardware interaction limit its scalability.

In February 2016, Chen et al. introduced Firmadyne (Chen et al., 2016), an automatic dynamic analysis system targeting Linux-based firmware vulnerabilities in network-connected devices. Firmadyne analyzed 23,035 firmware images, confirming vulnerabilities in 887 of them. The system automatically collects firmware images and metadata from vendor websites using scripts. It employs a customized version of Binwalk for kernel extraction and uses QEMU to emulate a Linux kernel tailored to each firmware image. Firmadyne performs dynamic analyses to detect and exploit vulnerabilities, utilizing the Metasploit framework[2] for exploitation. Potential improvements for Firmadyne include expanding its analysis capabilities to non-Linux systems like RTOS and enhancing firmware extraction methods to handle obfuscated and encrypted images.

In 2016, Costin (Costin et al., 2016) developed a fully automated dynamic firmware analysis framework that employs full system emulation via QEMU, without needing actual hardware. The framework focuses on identifying vulnerabilities in Linux-based systems, especially within embedded web interfaces, using tools like Arachni, Zed Attack Proxy (ZAP), and w3af, with additional capabilities for using scanning and exploitation tools such as Nmap, Nessus, and Metasploit. Despite discovering 225 new vulnerabilities across 45 firmware images, improvements in tool efficacy and emulation accuracy were noted as areas for enhancement due to inherent automation challenges in dynamic analysis.

In 2017, Palavicini (Palavicini Jr. et al., 2017) advanced the analysis of firmware vulnerabilities in Industrial Internet of Things devices, particularly targeting industrial control systems. Employing a suite of tools including binary analysis tools like ANGR (Shoshitaishvili et al., 2016) and American Fuzzy Lop (AFL)[3], and virtualization tools like OpenPLC, Firmadyne (Chen et al., 2016), and QEMU, the analysis process was divided into three phases: extraction of firmware for emulation, emulation of extracted code, and vulnerability analysis using techniques like fuzzing and symbolic execution. This approach uncovered backdoors, information leakage, and botnet-capable code.

In 2019, a framework named FIoT (Zhu et al., 2019) was introduced to address memory corruption issues in constrained IoT devices. FIoT applies symbolic execution to specifically test slices of firmware that may be prone to buffer overflows, highlighting the resource-intensive nature of symbolic execution and its unsuitability for large firmware images. This led to the discovery of 35 zero-day memory corruption vulnerabilities in 115 lightweight IoT devices.

In 2022, the framework OFRAK (Red Balloon Security, 2022) was published by Red Balloon Security. Presented at Summercon 2022, OFRAK facilitates firmware patching. This technique modifies the firmware and observes its behavior when re-flashes inside the device, either via JTAG interface or by desoldering the memory and performing a manual flash. For instance, in Linux-based firmware, an analyzer may introduce custom code inside `/etc/init.d`, which will be run with `sudo` privileges.

2.3 Breaking Security Evaluations

Both static and dynamic analyses can be thwarted by various strategies. In the context of embedded devices, one may for example focus on *custom* or *unknown* architectures that are not recognized by modern disassemblers and decompilers. A typical challenge in this respect is *emulating* the firmware behavior, which would require an in-depth understanding of the processor architecture and memory management. The problem is further worsened by the adoption of unknown formats and undocumented hardware, making the detection of executable sections difficult.

Other security measures can directly target the semantics of the code and are directly inspired by modern anti-analysis techniques in X86-64 or ARM architectures. For example, typical static obfuscation techniques involve removing the symbols from the binary (i.e., binary stripping) or flattening the control flow graph. Other techniques that prevent static analysis include dynamic code loading, where most of the executable content is de-compressed or decrypted at runtime, making static analysis inefficient.

Dynamic analysis can also be hindered by evasive techniques (Galloro et al., 2022), e.g., strategies aimed at (among others) delaying the execution of the binary (a simple example is the use of the `sleep` function) or preventing the execution of debuggers (with the combined use of `fork` and `ptrace`). Other strategies are often seen in Portable Executable (PE) malware, such as the adoption of TLS callbacks.

Anti-fuzzing techniques (Jung et al., 2019) complicate automated vulnerability detection through fuzzing. Methods such as checksum validations ensure only properly formatted inputs are processed, thwarting random or malformed data typical of fuzzing. Timing checks prevent rapid input sequences, halting potential fuzzing attempts. Additionally, adaptive context-aware responses obscure system logic, making it difficult for fuzzers to predict outcomes based on inputs. These strategies collec-

[2]https://www.metasploit.com/

[3]https://github.com/google/AFL

tively hinder the effectiveness of fuzzing tools in discovering software vulnerabilities.

3 IOT SECURITY ANALYSIS: OPEN CHALLENGES

As highlighted in the previous section, IoT and other embedded devices present unique security challenges not as evident in traditional systems due to their customized hardware and software from various manufacturers. Assessing their security with standard tools is difficult. Consequently, both institutional and private entities must rigorously evaluate the security of these devices before using them in critical environments. We now review four research directions that we believe were not already deeply investigated in the past by the community and are worth consideration.

Devising a Threat Model for IoT Security Evaluation Landscape. As detailed in Section 2, different key players have a crucial role when considering the security evaluation of an embedded device, including the manufacturer, the vendor, the security evaluator, and any external attacker. To systematically evaluate the complex dynamics emerging among such players, we believe that the scientific community should propose a threat model able to capture the capabilities of these players, taking into account and putting into perspective their respective constraints and objectives. Indeed, different from traditional systems, several public and private institutions may start to perceive the device manufacturer of an embedded system as an attacker who may have adopted strategies to hinder security analysis to hide scarcely tested code or undocumented functionalities. At the same time, manufacturers have the right to protect their intellectual property from competitors and adopt any defense that could thwart external attackers interested in exploiting the devices. Hence, it would be interesting to model the capabilities of a device manufacturer by identifying in more detail the specific anti-analysis strategies that could be adopted, the levels at which such strategies would operate (hardware, software, a mix of the two), and their consequences from different points of view (economical, practical, technological, strategic). Investigating such research direction could help the community determine the right trade-off between the manufacturers' needs and the security evaluators' needs.

Investigation of Granular but Actionable Security Analysis Requirements. Several public and private institutions may have the commercial or legal power to impose some security analysis requirements when a vendor proposes a device. For instance, such institutions may require that, during a security evaluation, the vendor should cooperate by providing technical documentation, the source code, the compiler toolchain, a disassembler, a device emulator, and several other analysis or reversing tools. However, it could be unlikely, or even completely unrealistic, to believe that any security evaluator can impose any kind of requirement on any vendor. Indeed, manufacturers hardly want to disclose technical information about their devices or provide their internal development tools. Moreover, they are both unable and unwilling to invest effort in devising tools for reversing tasks when their internal workflow does not need them. Finally, since the security evaluators may not directly interact with manufacturers, imposing requests that vendors cannot satisfy could be unrealistic.

In this scenario, we believe that the scientific community should consider three orthogonal aspects:

1. the requirements of the state-of-the-art approaches for security evaluations;
2. the constraints of the vendor and the needs of the manufacturer;
3. the operational context of the device, i.e., how sensitive and critical is the environment where the device will be deployed.

In particular, the community should evaluate and provide some clear insights on what could be asked to the vendor in order to make effective some of the state-of-the-art tools for security analyses while keeping into account the operational context of the device. For instance, if a device is deployed into a critical environment, then it could be reasonable to require the source code of the software executed by the device, since the availability of the source code would allow the use of a large number of powerful state-of-the-art tools. In contrast, when a device is deployed in a less critical environment, the request for the source code may be unrealistic but then it should be clear what other minimum requirements could be imposed by a security evaluator to allow a significant security evaluation.

While devising such guidelines would not typically be the job of the scientific community, it could be argued that security analysis requirements are strictly connected with the state-of-the-art security analysis approaches. Therefore, the scientific community should explore different analysis strategies depending on the requirements that could be realistically imposed in the real world of security evaluations. Hence, the definition of realistic security analysis requirements and the development of security analysis approaches should be carried out simultaneously.

Devising Modular Security Analysis Frameworks.

A crucial limitation of several existing state-of-the-art analysis tools for security evaluations is that they are often not modular and extensible. For instance, it is still quite hard and expensive for a manufacturer to support open source emulators required for running dynamic analyses, such as software fuzzing and taint analysis, or to support heavyweight analysis techniques, such as symbolic execution. This excessive cost results in a barrier that could be exploited by manufacturers to justify their lack of cooperation. We thus believe that the scientific community should investigate how to make existing security analysis frameworks, such as KARONTE (Redini et al., 2020), ELISA (Nicolao et al., 2018), AVATAR (Zaddach et al., 2014), and Firmadyne (Chen et al., 2016), extremely modular and extensible, minimizing as much as possible the effort needed to develop and maintain the low-level building blocks required by such tools, thus lowering the bar for device manufacturers.

Reversible Anti-Analysis Techniques. The manufacturer should be allowed to integrate anti-analysis techniques that are targeted at hindering the reversing tasks carried out by competitors and external attackers. To still allow effective security evaluations from *selected* actors, such as a security evaluator, the research community could explore the idea of *reversible* anti-analysis techniques that, as their name suggests, could allow a selected party to make ineffective the anti-analysis strategies when the manufacturer is willing to (partially) cooperate (possibly, under the request of a vendor). For instance, in a scenario where the manufacturer integrates anti-fuzzing techniques into its code as protection against the fuzzing performed by external attackers, the security evaluator may ask to the vendor to provide the technical bits needed to bypass such anti-analysis approach during the security evaluation. Hence, anti-analysis techniques could be conceived as *trapdoor* functions that are easy to reverse when the trapdoor is provided.

4 CONCLUSIONS

This paper highlighted the challenges that may emerge when carrying out a security evaluation of an IoT device. Specifically, we believe there is a need to systematically analyze security contexts by highlighting the involved actors and the technical security evaluation workflow. Indeed, complex dynamics arise when considering the needs and constraints of the device manufacturers, the external attackers, and the security evaluators. For instance, manufacturers may adopt anti-analyses techniques to protect their own devices from competitors and external attackers. However, at the same time, by adopting such practices, manufacturers may hinder security evaluations, making it unclear whether they are trying to hide poorly tested code, that may lead to easy exploitation from an external attacker, or conceal undocumented privacy-risk functionalities. Unfortunately, existing state-of-the-art security tools do not yet take into the complexity of such a landscape and often fall short by either posing unreasonable requirements (e.g., availability of the source code) or cannot be easily extended with minimal effort, thus imposing a strong barrier for manufacturers that are willing to (partially) cooperate with a security evaluator. We believe that our work can be a starting point to inspire further research in this field, fostering cooperation between the scientific community and device manufacturers to secure our device landscape.

Acknowledgments

This work was partially supported by Project FARE (PNRR M4.C2.1.1 PRIN 2022, Cod. 202225BZJC, CUP D53D23008380006, Avviso D.D 104 02.02.2022) and Project SETA (PNRR M4.C2.1.1 PRIN 2022 PNRR, Cod. P202233M9Z, CUP F53D23009120001, Avviso D.D 1409 14.09.2022). Both projects are under the Italian NRRP MUR program funded by the European Union - NextGenerationEU.

REFERENCES

Balakrishnan, G. and Reps, T. W. (2004). Analyzing memory accesses in x86 executables. In *13th International Conference, CC 2004*, volume 2985 of *Lecture Notes in Computer Science*, pages 5–23. Springer.

Balakrishnan, G., Reps, T. W., Melski, D., and Teitelbaum, T. (2005). WYSINWYX: what you see is not what you execute. In Meyer, B. and Woodcock, J., editors, *Verified Software: Theories, Tools, Experiments, VSTTE 2005*, volume 4171 of *Lecture Notes in Computer Science*, pages 202–213. Springer.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, pages 41–46. USENIX.

Cadar, C., Dunbar, D., and Engler, D. R. (2008). KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008*. USENIX Association.

Chen, D. D., Woo, M., Brumley, D., and Egele, M. (2016). Towards automated dynamic analysis for linux-based embedded firmware. In *23rd Annual Network and*

Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016. The Internet Society.

Costin, A., Zaddach, J., Francillon, A., and Balzarotti, D. (2014). A large-scale analysis of the security of embedded firmwares. In Fu, K. and Jung, J., editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 95–110. USENIX Association.

Costin, A., Zarras, A., and Francillon, A. (2016). Automated dynamic firmware analysis at scale: a case study on embedded web interfaces. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 437–448.

David, Y., Partush, N., and Yahav, E. (2018). Firmup: Precise static detection of common vulnerabilities in firmware. *ACM SIGPLAN Notices*, 53(2):392–404.

European Cyber Security Organisation - ECSO (2022). Technical Paper on Internet of Things (IoT). https://ecs-org.eu/?publications=technical-paper-on-internet-of-things-iot. Accessed: 2024-04-27.

Fraunhofer FKIE-CAD (2017). FACT_core: Firmware Analysis and Comparison Tool. https://github.com/fkie-cad/FACT_core. Accessed: 2024-04-24.

Galloro, N., Polino, M., Carminati, M., Continella, A., and Zanero, S. (2022). A systematical and longitudinal study of evasive behaviors in windows malware. *Comput. Secur.*, 113:102550.

Hu, Y., Zhang, Y., Li, J., and Gu, D. (2017). Binary code clone detection across architectures and compiling configurations. In *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017*, pages 88–98. IEEE Computer Society.

Jung, J., Hu, H., Solodukhin, D., Pagan, D., Lee, K. H., and Kim, T. (2019). Fuzzification: Anti-Fuzzing techniques. In *28th USENIX Security Symposium*. USENIX Association.

Nadir, I., Mahmood, H., and Shah, G. A. (2022). A taxonomy of iot firmware security and principal firmware analysis techniques. *Int. J. Crit. Infrastructure Prot.*, 38:100552.

Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., and Ghani, N. (2019). Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys Tutorials*, 21(3):2702–2733.

Nicolao, P. D., Pogliani, M., Polino, M., Carminati, M., Quarta, D., and Zanero, S. (2018). ELISA: eliciting ISA of raw binaries for fine-grained code and data separation. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018*, volume 10885 of *Lecture Notes in Computer Science*, pages 351–371. Springer.

Palavicini Jr., G., Bryan, J., Sheets, E., Kline, M., and Miguel, J. S. (2017). 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*, pages 470–477. SciTePress.

Red Balloon Security (2022). Open Firmware Reverse Analysis Konsole. https://github.com/redballoonsecurity/ofrak. Accessed: 2024-04-27.

Redini, N., Machiry, A., Wang, R., Spensky, C., Continella, A., Shoshitaishvili, Y., Kruegel, C., and Vigna, G. (2020). Karonte: Detecting insecure multi-binary interactions in embedded firmware. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1544–1561. IEEE.

Remigio, R., Bertani, A., Polino, M., Carminati, M., and Zanero, S. (2023). The good, the bad, and the binary: An lstm-based method for section boundary detection in firmware analysis. In *18th Int. Workshop on Security, IWSEC 2023*, Lecture Notes in Computer Science. Springer.

Schwartz, E. J., Avgerinos, T., and Brumley, D. (2010). All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *31st IEEE Symposium on Security and Privacy, SP 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 317–331. IEEE Computer Society.

Secure Firmware (2022). EMBA - The firmware security analyzer. https://github.com/e-m-b-a/emba. Accessed: 2024-04-27.

Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Krügel, C., and Vigna, G. (2016). SOK: (state of) the art of war: Offensive techniques in binary analysis. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 138–157. IEEE Computer Society.

Thomas, S. L., Garcia, F. D., and Chothia, T. (2017). Humidify: A tool for hidden functionality detection in firmware. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 14th International Conference, DIMVA 2017*, volume 10327 of *Lecture Notes in Computer Science*, pages 279–300. Springer.

Yu, R., Nin, F. D., Zhang, Y., Huang, S., Kaliyar, P., Zakto, S., Conti, M., Portokalidis, G., and Xu, J. (2022). Building embedded systems like it's 1996. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society.

Zaddach, J., Bruno, L., Francillon, A., and Balzarotti, D. (2014). AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society.

Zhu, L., Fu, X., Yao, Y., Zhang, Y., and Wang, H. (2019). Fiot: Detecting the memory corruption in lightweight iot device firmware. In *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 13th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2019*, pages 248–255. IEEE.