Idols with Feet of Clay: On the Security of Bootloaders and Firmware Updaters for the IoT

Lionel Morel and Damien Couroussé
Univ. Grenoble Alpes, CEA, List
F-38000 Grenoble, France
Email: {lionel.morel, damien.courousse}@cea.fr

***Abstract*—IoT devices are generally implemented with low-cost embedded solutions, and connectivity and communication capabilities are the *raison d'être* of such devices. But this is a double-edged sword, since connectivity also implies (1) to open the door to more attack possibilities, and (2) the targeted system, once breached, can be the support for attacks at a larger scale, possibly involving many connected systems.**

Our observation is that such devices lack proper hardware and software security protections. Bootloader and Firmware Update (BFU) mechanisms are critical components in the software stack of IoT devices. BFUs are a target of choice since they use the highest privileges and are executed before the system's security policy is set up. An attacker able to compromise the BFU can gain full control over the target system. Moreover, the update mechanism often supported by the BFU is essential to ensure devices can be upgraded and maintained for a long time. If not properly secured, the BFU allows an attacker to gain control over a system throughout its whole lifetime, including future upgrades.

In this paper, we provide an overview of the threats targeting BFUs, and existing protections. We cover the hardware and software attacks that are known to the scientific literature. Also, we argue that vulnerabilities to physical attacks, in particular to fault injection attacks, are mostly left un-attended.

I. Introduction

IoT devices are deployed in many everyday life situations such as home appliances (fluid metering), leisure enhancement (fitness trackers) or even very sensitive applications such as healthcare with monitoring of vital signs or automated medication devices such as insulin pumps or cardiac defibrillators. Their security is thus becoming a major challenge, with an impact on infrastructures, privacy and user safety.

One of the goals of a secured bootloader is to setup a chain-of-trust (CoT). Starting from the boot of the platform, the system uses a component recognized as secure, identified as root-of-trust (RoT). Each element in the CoT is then checked using its predecessor in the CoT. This init phase manipulates the RoT and is thus particularly sensitive. This phase is also in charge of setting up hardware and software components, access rights, system-wide data structures, and so on, which require high privileges. Thus, the init stages of the boot process are particularly sensitive, and may threaten the whole system's security if not properly secured.

This work was supported by the French program "Programme Investissements d'Avenir IRT Nanoelec" ANR-10-AIRT-05, and the European project SERENE-IoT (16 004) within the framework of PENTA, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

The firmware update procedure is usually associated with the boot step, because the device is then at the right level of protection to allow code replacement and hardware configuration often required by updates. Also, during boot, no other system services are running, making the upgrade procedure less context-sensitive. An attacker gaining access over this process may end up with full access to the platform. Also, controlling the firmware update stage makes the attacker powerful in time: she can control the whole upgrade process and thus the future behaviour of the compromised node.

A firmware update procedure also requires a proper security network infrastructure, e.g. for the management of security keys, for ensuring secured communications between updates servers and devices, etc. In this paper, we focus on the security aspects of the IoT device itself, and the security aspects of IoT networks are not discussed.

In this paper, we claim that, to secure IoT nodes and infrastructures, one needs to tackle the security of the Bootloader and Firmware Update (BFU). We report on the security of the BFU of IoT devices, both showing successful attacks and protection mechanisms that are effective. We highlight the limits of currently adopted approaches, with examples of devices, applications and attacks performed on them (section II). We show recent progress made by device manufacturers to enhance security of BFUs by providing dedicated frameworks and software stacks to secure BFUs (section III). Finally, we try an orthogonal look at the security of IoT BFUs, supporting the idea that physical attacks are currently too poorly considered on IoT devices (section IV).

II. Existing Attacks on IoT BFUs

The security of IoT nodes has been an important subject in the literature since the appearance of wireless sensor networks. Many surveys have been published, covering for example various security aspects of IoT devices [1], [2], or specific application domains such as fitness trackers [3] or Implantable Medical Devices (IMDs) [4]. These studies show that many deployed devices lack the very basic security mechanisms. For example, firmware images are not mapped to protected memory nor checked for authenticity [4]. The same vulnerabilities were demonstrated on the firmware update process [5]. An attacker can thus reverse engineer a valid firmware image, then craft a brand new malevolent firmware and update the

device with it. Often, data is not encrypted at all, or encrypted only when received/sent on external communication channels, which leaves them accessible to an attacker with a physical access to the device. When cryptographic primitives are used, implementations are often bugged so that an attacker can bypass these mechanisms quite easily.

Even when some security elements are present, these can be attacked and breached. Most of IoT devices have little value for themselves or for the information they process, but can be used as a vehicle for attacks at larger scales. In the worst cases, a device vulnerability was demonstrated as a path to breach into other hosts connected on the same network [6], to take over a large network of IoT devices [7], or to launch a Mirai-style botnet attack [8]. Yet, some devices are very sensitive to safety or security issues, such as IMDs [4]. In this case a security breach will lead to serious safety issues, since a successful attacker can not only access sensitive data but also provoke a malfunction, leading to injury or even death of a patient. This strongly supports our idea that the BFU is the system's Achilles heel.

III. Secure BFUs

The literature above makes it clear that designing more secure BFUs is paramount. We now outline a generic architecture for secure BFUs. We then make a tour of recent solutions implementing these generic principles. Finally, we show that this approach is still prone to attacks.

A. A Generic BFU

This model is directly inspired by the specification described by Atmel in [9]. It supports integrity of the firmware code present on the device, and confidentiality, integrity and authenticity of new firmwares for updates. Our description here leaves out platform-dependent specificities, such as: page sizes, validity criteria, update triggering conditions, presence of HW or SW cryptographic components, key management. The BFU is responsible for 1) setting up HW and SW components upon boot and 2) upon request, updating the whole software of the platform.

To perform upgrades, a BFU interacts with a host, which delivers new firmware instances and may trigger their installation. The host is part of the attacker model: an attacker can corrupt it and use it to install malicious updates to devices. During normal operation, firmware instances are encrypted using public or private key cryptography, depending on resources available on the node (CPU, memory).

Figure 1 depicts the typical workflow of a secured BFU. Control points are drawn as green diamonds, non-cryptographic and cryptographic functions as white and red boxes respectively, and start/end points in dark gray.

Upon boot, the platform enters a setup phase where memories and peripherals are initialized. This requires code to be executed with high permission levels, e.g., with read/write access to most of the memory space. Memory protections are then switched appropriately: some memory regions are made read-only, some access ports on peripherals are disabled.

When initialization is over, availability of a firmware update is checked. If an update is available, the update process itself is started. In the opposite case, the BFU moves to the verification stage, checking validity of the pre-existing firmware instance.

The update is an iterative process to cope with the low memory budget of IoT devices: the new firmware is downloaded page by page, each page being decrypted upon reception.

The verification step checks both integrity and authenticity, using state-of-the-art hash functions, digital signature or message authentication codes. This verification also occurs in case no upgrade was triggered, so the already present firmware image is verified for unwanted modifications. If verification is successful, the firmware is loaded and executed. Otherwise, it means that a malicious (or erroneous) firmware has been loaded. Protective countermeasures can be triggered: firmware deletion, device suicide or bricking, etc.

B. Existing Secure BFUs

The ideas of the secure BFU model of section III-A are deployed in several works of the literature. Atmel's application note [9] discusses the pros and cons of different techniques for ensuring integrity and authentication (hash functions, signatures, MACs) as well as data and code confidentiality (encryption). A similar workflow is found at the heart of the SecFOTA [10]. Integrity checks and encryption/decryption are also deployed in this solution. ST-Microelectronics's Secure Boot and Secure Firmware Update [11] and ARM's Trusted Firmware [12] are so far the most complete solutions publicly available. They integrate various SW security mechanisms and can leverage HW mechanisms as well, when the underlying platform provides them.

C. Identified Vulnerabilities on BFUs

For each element in the BFU, we identify vulnerabilities, attacks as well as possible countermeasures. Works presented target either IoT platforms or larger platforms (PCs and smartphones) when they could be applied to IoT-grade devices.

1) At System Initialization: Two recent contributions appear most representative on the subject of securing the init phase. [13] focuses on smartphone platforms, where the notions of secure boot and RoT have been deployed thoroughly. The authors show that, albeit these BFU-like mechanisms, many vulnerabilities persist. Boot-related meta-data can be corrupted so as to permit execution of maliciously-crafted code, possibly granting unlimited access over the platform to the attacker. Integrity checking mechanisms included in the boot process can also be pushed to brick the device. In [14], an attack on the init phase of an IoT-grade platform is deployed. This attack relies on the capacity to iteratively execute, freeze and observe the behaviour of the system, upon boot, using a debug capability often left unattended even in deployed systems. This cold-boot stepping technique is used to access read-protected memories and ultimately dump the whole firmware code residing in these memories.

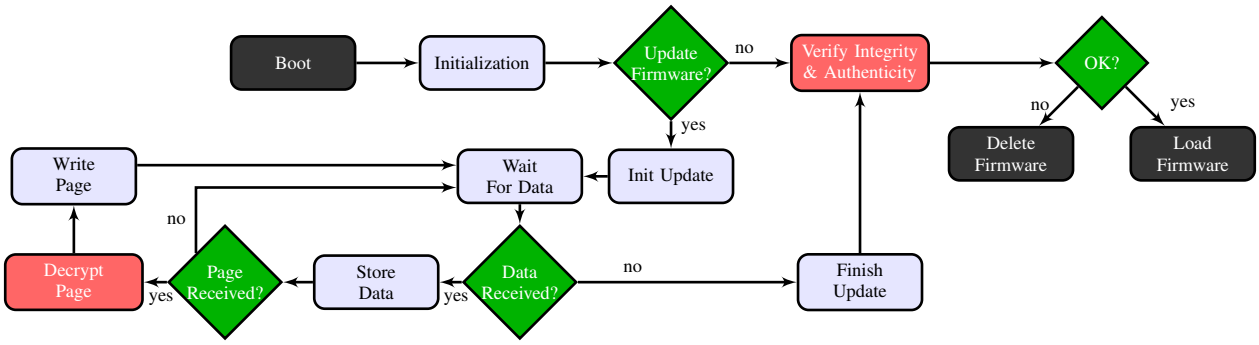

Figure 1. Flowchart of a generic BFU.

2) Attacks on the CFG: The CFG itself can be the target of various types of attacks, identified as control-flow hijacking. Some of these attacks can be used to force a conditional instruction to take a particular branch of a test. More sophisticated attacks compromise the stack in order to deviate the control-flow during call/return sequences. Return-oriented Programming [15] relies on buffer overflows. With judiciously chosen overflows, one can overwrite return addresses and deviate the course of execution to an attacker-chosen location. Sigreturn-ROP [16] deviates the program control to signal handlers that encapsulate the malign code; these attacks are more portable, as signal-handlers do not depend on the layout of binary instructions in memory. Heap Spraying [17] uses assumptions made on the implementation of heap management: a buffer overflow achieved on a heap element is used to overwrite one of the victim's block of memory, bypassing memory protections.

Many countermeasures can be used against such attacks. Control-flow integrity [18] consists in dynamically checking that a branch taken corresponds to the one expected. This requires modifications to the protected program [19], possibly in combination with hardware mechanisms [20]. With Control-Flow Attestation [21], the application's CFG is not modified but attested by a tracer that compares the actual execution paths with statically built references. Stack Canaries [22] are secret values placed on the call stack, that change regularly. Whenever a call occurs, the canary is checked and the program is exited if the observed value differs from the expected one. Address-Space Layout Randomization [23] has also been proposed to counteract ROPs. Randomization is usually performed in the OS memory management procedure, or at compile-time so as to make it application specific [24].

3) Cryptographic Primitives: Cryptographic primitives, even though their algorithms are considered secure from the point of view of cryptanalysis, need their implementation to be secured against logical attacks and physical attacks. We discuss here the security issues related to software attacks, and physical attacks is discussed below in section IV.

Timing and cache-timing attacks are closely related to physical attacks, but we discuss them here as then can be exploited with a logical access to the target. Timing attacks exploit the dependency of the executed code to the data processed, in order to reveal secret information such as a cipher key or an authentication code. Such attacks are effective if the attacker gets a logical access to the device, e.g. via a bootloader console, but have also been shown effective over network accesses [25]. Many other attacks may exploit the micro-architecture [26], such as the variability in execution time incurred by cache accesses, e.g. to recover secret cryptographic keys.

IV. The Case of Physical Attacks

Side-channel attacks [27], [28] and fault injection attacks [29], [30] are the two sides of the same threat, where the attacker exploits a physical access to the target device. Historically, the use of such attacks was mostly restricted to devices embedding microcontrollers such as Smart Cards, and considered expensive because they would require expert skills and expensive attack benches (e.g., laser injection). However, many research works have recently demonstrated that attack benches can be built at low cost [7], [31], or are even commercially available [32]. Plus, IoT devices usually present a lower system complexity as compared to high-end computer or mobile platforms, which makes them easy targets for these attacks. Hence, in the context of IoT security, the threat posed by these attacks is increasing. In a network of interconnected devices, the impact of such attacks at large scale could be even more devastating [7].

Cryptographic primitives, which ground the security properties supported by BFUs, are known vulnerable against side-channel [27] and fault attacks [29]. Similarly, bootloaders have been demonstrated vulnerable to these attacks. Ronen et al. illustrate how a side-channel attack is used to recover a master cipher key, statically stored into the device, and then to force the upgrade of a malicious firmware [7]; they also illustrate how the malicious firmware can then spread over a large network of devices. Fault injection attacks could be similarly used to recover a secret cipher key, or break an authentication or integrity check.

Outside of cryptographic primitives, other parts of BFUs are sensitive to physical attacks. Side-channel attacks have been used for example to help with the identification of the executed instructions on small processor architectures [33], [34]. Other techniques allow to recover branching information from PC-

style architectures [35]. Using fault injection, Timmers et al. illustrate how to bypass a conditional branch [36], which is typically used after a security check in a boot procedure. They also demonstrate the modification of load instructions to perform arbitrary modifications of the program counter [37], which can constitute a first step towards arbitrary code execution. Similar attacks have also been successfully used to exploit software vulnerabilities during boot [31].

The research on this topic is vivid. Due to lack of space, we cite only a few research works that could be applied to BFUs: de Clerq and Verbauwhede provide a comprehensive survey on control flow integrity, covering fault injection attacks and existing countermeasures [20]. Belleville et al. propose a generic software countermeasure against side-channel attacks [38]. Werner et al. propose a systematic approach for code confidentiality, authenticity and integrity, against logical and fault injection attacks [39].

V. Conclusion

In this paper, we have drawn a panorama on the security of BFUs on IoT platforms. First, we have shown that the security of BFUs is not sufficiently addressed in IoT devices. We have sketched a typical workflow for a BFU supporting integrity of the existing firmware, and authenticity, integrity and confidentiality of firmware upgrades. We have outlined the capabilities offered by security-oriented frameworks recently proposed. These should be deployed as widely as possible, in order to provide a minimal security layer. Finally, we have shown that despite these improvements, much work remains to be done, in particular to address physical attacks which represent a significant threat on IoT devices.

References

[1] O. Arias, J. Wurm, K. Hoang, and Y. Jin, "Privacy and security in internet of things and wearable devices," *IEEE TMSCS*, 2015.

[2] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE TETC*, 2017.

[3] H. Fereidooni, T. Frassetto, M. Miettinen, A.-R. Sadeghi, and M. Conti, "Fitness trackers: Fit for health but unfit for security and privacy," in *CHASE*, 2017.

[4] B. Rios and J. Butts, "Security evaluation of the implantable cardiac device ecosystem architecture and implementation interdependencies," 2017, white paper, WhiteScope.

[5] J. Rieck, "Attacks on Fitness Trackers Revisited: A Case-Study of Unfit Firmware Security," *arXiv:1604.03313 [cs]*, 2016.

[6] A. Cui, M. Costello, and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation," in *NDSS*, 2013.

[7] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," in *S&P*. IEEE, 2017.

[8] O. Shwartz, Y. Mathov, M. Bohadana, Y. Oren, and Y. Elovici, "Reverse engineering iot devices: Effective techniques and methods," *IEEE Internet of Things Journal*, 2018.

[9] Atmel, "At02333: Safe and secure bootloader implementation for sam3/4," Application Note, 2013.

[10] S. Schmidt, M. Tausig, M. Hudler, and G. Simhandl, "Secure firmware update over the air in the internet of things focusing on flexibility and feasibility," in *IoTSU*, 2016.

[11] ST-Microelectronics, "X-CUBE-SBSFU - secure boot and secure firmware update," User Manual, 04 2018.

[12] A. Ltd, "The arm trusted firmware," online, last accessed: 27th November 2017. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware

[13] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Bootstomp: On the security of bootloaders in mobile devices," in *USENIX Security*, 2017.

[14] J. Obermaier and S. Tatschner, "Shedding too much light on a microcontroller's firmware protection," in *WOOT*, 2017.

[15] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *Proceedings of CCS 2008*. ACM Press, Oct. 2008.

[16] E. Bosman and H. Bos, "Framing Signals - A Return to Portable Shellcode," in *IEEE SSP*, May 2014.

[17] J. Pincus and B. Baker, "Beyond stack smashing: recent advances in exploiting buffer overruns," *IEEE S&P*, Jul. 2004.

[18] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow Integrity Principles, Implementations, and Applications," *ACM TISS*, 2009.

[19] J.-F. Lalande, K. Heydemann, and P. Berthomé, "Software countermeasures for control flow integrity of smart card C codes," in *ESORICS*, 2014.

[20] R. de Clercq and I. Verbauwhede, "A survey of Hardware-based Control Flow Integrity (CFI)," *arXiv:1706.07257 [cs.CR]*, 2017.

[21] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-Flow Attestation for Embedded Systems Software," in *CCS*. ACM, 2016.

[22] A. van de Ven, "Execshield: New security enhancements in red hat enterprise linux v.3, update 3: Execshield and support for nx technology," 2004. [Online]. Available: https://static.redhat.com/legacy/f/pdf/rhel/WHP0006US_Execshield.pdf

[23] PaX-Team, "Pax aslr (address space layout randomization)," 2003. [Online]. Available: http://pax.grsecurity.net/docs/aslr.txt

[24] H. Koo, Y. Chen, L. Lu, V. P. Kemerlis, and M. Polychronakis, "Compiler-assisted Code Randomization," in *S&P*, 2018.

[25] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, 2005.

[26] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware," Cryptology ePrint Archive 2016/613, 2016.

[27] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2008.

[28] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Systematic classification of side-channel attacks: A case study for mobile devices," *IEEE Communications Surveys & Tutorials*, 2017.

[29] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE*, pp. 3056–3076, 2012.

[30] B. Yuce, P. Schaumont, and M. Witteman, "Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation," *Journal of Hardware and Systems Security*, 2018.

[31] A. Cui and R. Housley, "BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection," in *WOOT*, 2017.

[32] "ChipWhisperer® – NewAE Technology Inc." https://newae.com/tools/chipwhisperer, 2019.

[33] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a Side Channel Based Disassembler," in *Transactions on Computational Science X*, ser. LNCS. Springer, 2010.

[34] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *DAC*. ACM, 2018.

[35] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A New Side-Channel Attack on Directional Branch Predictor," in *ASPLOS*, 2018.

[36] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," in *FDTC*. IEEE, 2016.

[37] N. Timmers and A. Spruyt, "Bypassing Secure Boot using Fault Injection," 2016, black Hat Europe.

[38] N. Belleville, D. Couroussé, K. Heydemann, and H.-P. Charles, "Automated Software Protection for the Masses Against Side-Channel Attacks," *ACM Trans. Archit. Code Optim.*, 2019.

[39] M. Werner, T. Unterluggauer, D. Schaffenrath, and S. Mangard, "Sponge-Based Control-Flow Protection for IoT Devices," *arXiv:1802.06691 [cs.CR]*, 2018.