Journal of
Information Technology & Software Engineering

Echard, J Inform Tech Softw Eng 2017, 7:5
DOI: 10.4172/2175-7866.1000217

Review Article Open Access

Ensuring Software Integrity in IoT Devices

Chris Echard*

Department of Computer Science, East Carolina University, Greenville, North Carolina, USA

Abstract

This paper will review the available literature covering topics relevant to ensuring software integrity of the boot image and running code in networked devices. The paper will focus on hardware devices defined as IoT (Internet of Things) in the consumer space. Software-only (virtual) devices will also be discussed. Topics reviewed will include trusted anchor concepts and technologies, chain of trust and validation of code integrity, as well as the technologies which support them, such as PKI (public key infrastructure), secure storage and mutable and immutable identities.

Keywords: IoT; Internet of Things; TOC; TOU; Chain of trust; Code integrity

Introduction

IoT is defined as the Internet of Things. The Internet of Things refers to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other Internet-enabled devices and systems [1]. According to Electronic Design, a recent report by Gartner predicts that there will be 20.4 billion connected Internet of Things (IoT) devices by 2020, with 5.5 million new things getting connected every day. Furthermore, more than half of major new business processes and systems will include an IoT component by 2020 [2]. With so many devices currently in service, and many more expected to come online, it seems a little late to be discussing the security aspect of the design lifecycle. Yet, that is exactly what is happening. Manufacturers are rushing products to market with little or no thought to security, often including hardcoded passwords or known vulnerable software libraries. While this problem is most obvious in the consumer space (which gets the most news coverage), vulnerable IoT devices are present in every business sector as well. This paper will review the available literature on the subject of ensuring software integrity for IoT devices, focusing on detecting and preventing modification of the original software, so that the device cannot be used for unintended purposes.

Review of Literature

Secure boot

The startup of every device begins with powering on, verifying hardware components, then loading one or more software modules. NIST special publication 800-147 defines "booting" as a 5 step process: 1) Execute Core Root of Trust, 2) Initialize and Test Low-Level Hardware, 3) Load and Execute Additional Firmware Modules, 4) Select Boot Device, and 5) Load Operating System. Each of these steps is an opportunity to introduce new code by an attacker. For example, the system BIOS is a potentially attractive target for attack. Malicious code running at the BIOS level could have a great deal of control over a computer system. It could be used to compromise any components that are loaded later in the boot process, including the SMM code, boot loader, hypervisor, and operating system [3]. Therefore, code at each step of the boot process must be verified, beginning with the integrity of the platform itself. The trusted computing approach to solving this issue entails adding a separate chip called the trusted platform module (TPM) to the system. A trusted platform module enhances the security of general purpose computer systems by authenticating the platform at boot time. The TPM is the root of trust for a computing system. It manages the three roots of trust that lie at the core of a trusted platform: (i) A root of trust for measurement (RTM) to measure the platform integrity; (ii) a root of trust for storage (RTS) to securely store different integrity measurements, secrets, and keys; and (iii) a root of trust for reporting (RTR) that reliably and securely reports the platform information stored in the RTS [4]. The TPM stores secret keys, passwords, and digital certificates in its secure internal storage protecting them from software and physical attacks. The TPM acts as a root of trust for checking platform integrity at boot time (i.e., check against any malicious change). A cryptographic hash value of the platform configuration is calculated and compared against the precomputed hash value of the platform. Access to the platform is denied if the integrity check fails [4]. This is the beginning of the "chain-of-trust" for software modules that are subsequently initiated. This transitive trust mechanism is one of the important security features in trust computing. It uses the trust root as a starting point to establish a chain of trust model, in the order of trust root, boot loader, OS, and Application. The verification is taken step by step, to extend the trust boundary to the entire platform [5]. Also, since the TPM can store multiple hashes, software modules can authenticate themselves during the boot process. Secure boot provides the foundation for Trusted Boot, which extends the trust boundary to the boot process and eventually the operating system.

Using a TPM chip to verify system integrity is an example of hardware attestation. For devices that are constrained (usually because of price) and do not have hardware security capabilities, software attestation methods are available. Software attestation is a trust establishment mechanism that allows a system, the verifier, to check the integrity of the program memory content of another system, the prover, against modification, e.g., by malicious code.

Software attestation follows a radically different approach than most conventional security mechanisms: It exploits the intrinsic physical constraints of the underlying hardware and side-channel information, typically the computation time required by the prover to complete the attestation protocol [6]. Put another way, software attestation attempts to achieve a dynamic root of trust without specific hardware support. This method has the advantage of not requiring any stored secrets (cryptographic keys or passwords) and allows applications or modules

***Corresponding author:** Chris Echard, Department of Computer Science, East Carolina University, Greenville, NC 27858, USA, Tel: +1 252-328-6131; E-mail: echardc16@students.ecu.edu

Received November 15, 2017; **Accepted** November 30, 2017; **Published** December 08, 2017

Citation: Echard C (2017) Ensuring Software Integrity in IoT Devices. J Inform Tech Softw Eng 7: 217. doi: 10.4172/2175-7866.1000217

to be updated, which may not be possible if hash values are stored in immutable formats, such as a TPM chip [7].

There are several methods available for computing and verifying hash values with software based attestation. For example, a simple hash of all stored memory could be computed and then compared with a stored value. The disadvantage of this approach is that malicious code could have run this function and stored the result before making its own modifications to the system. Only the stored result would be returned to the verifier. A slightly better approach would be to include other variables, such as computation time, into the verifiable result. This approach can work if an attacker cannot modify the verification code and the stored values. Also, the verification routine must be data dependent; running a data independent function such as a checksum will always give the same time result. This approach is also useful for software only devices (software in virtual environments, abstracted from underlying hardware) assuming that an adequately precise time source is available.

Trusted boot

While Secure Boot validates the platform and firmware, Trusted Boot is generally defined as verifying each software module before execution and extending the chain-of-trust to the entire operating system. During the boot sequence, the digest of each executing program is recorded before it executes. A TPM (Trusted Platform Module) is used to store all these records and then report on them securely. For example, before the BIOS hands over control to a boot loader, it hashes the boot loader and extends the resultant values into a PCR (platform Configuration Registers) in a TPM. However, the TPM itself does not prohibit booting into an insecure OS or using an insecure boot loader [8]. If the values stored in the TPM do not match the calculated values, the BIOS has the option to give or deny control to the boot loader, depending on its level of configurability. It is important to note that Trusted Boot requires a TPM chip so the operating system can see the chain of execution, thus it may not be an option for some IoT devices. Lack of trusted boot support would allow an attacker with physical access, or using a software vulnerability during run time, to potentially modify the stored code and compromise the device. Software only TPM implementations based on the public domain TPM emulator have been explored, but have limitations. Although not completely equivalent to a conventional TPM chip in terms of protection against physical and hardware attacks, SW-TPM can be executed within protected or isolated execution domains that are increasingly provided by embedded CPUs (e.g., ARM TrustZone) and can utilize on-chip storage in order to provide a reasonable degree of tamper-resistance [9]. Implementation of the TPM registers and the trust anchor code are stored in the processors' on-board memory, providing some physical security to these important functions.

Runtime integrity

There are many other attack possibilities to consider with IoT devices. For example, existing TPM architectures do not support runtime integrity checking and this allows attackers to exploit vulnerabilities to modify the program after it has been verified (at time of check or TOC) but before the time of its use (at time of use or TOU) to trigger unintended program behavior, such as the execution of malicious code or the leaking of sensitive data [4]. Since Iot devices are designed to run continuously, barring power outages or other unpredictable events, code may be compromised at run time through a vulnerability and never be discovered. A good example of this is the Mirai malware. Mirai finds devices with known vulnerabilities, infects the device, but does not modify stored code. The device can be "fixed" simply by rebooting, but since most IoT devices are designed for little or no management, there is no indication of compromise unless the unwanted behavior is noticed.

Physical attack is a viable method of compromising the integrity of a device. Modifying and replacing firmware by an attacker may be worth the effort, depending on the perceived value of the device or the data it may access (such as a video camera or ATM). Attackers can go as far as removing memory and reading its contents. In 2008, a team of researchers demonstrated that disk encryption keys could be recovered from DDR and DDR2 DRAMs by transferring memory modules from a locked machine into an attacker's machines [10]. A hard or flash drive, if present, is an easy target for physical removal and cloning. If the operating system and data are not encrypted in some way, the entire system can be read or modified. Encryption is the easiest way to mitigate this risk, but does require hardware storage of keys.

Side channel attacks, which take advantage of information leakage from a device, present a credible threat to any physical computing device. Information can be leaked by detecting changes in processing behavior (usually by monitoring power usage) to determine specific processing routines, such as encryption or decryption. For example, in low-end microcontrollers common in IoT devices, during processing of an algorithm such as RSA decryption, a multiplication will only be performed if the exponent bit being processed is 1. The attacker can simply measure changes in current to derive the key one bit at a time [11]. Mitigation methods addressing side channel attacks can include randomizing execution of routines, coding routines to ensure the timing differences are small for the different paths a routine may take, and reducing the effects of cache hit and miss speed differences. Leakage can also be classified as responding to invalid login attempts by informing the attacker if a username is or isn't in the device's database.

Conclusion

A quote from an IoT security whitepaper written by WindRiver sums up the issue nicely: "Knowing no one single control is going to adequately protect a device, how do we apply what we have learned over the past 25 years to implement security in a variety of scenarios? We do so through a multi-layered approach to security that starts at the beginning when power is applied, establishes a trusted computing baseline, and anchors that trust in something immutable that cannot be tampered with." But security is also an evolution [12]. Software in IoT devices may have been declared "bug-free" when they were manufactured, but flaws could be discovered years later which could be easily exploited. Coupled with poor security design, (e.g., hardcoded passwords) and no patching capabilities, a device built today can become a major problem tomorrow. And, given that these devices (especially in the consumer world) are "set and forget", the internet will have literally billions of obsolete hosts in the coming years. Some researchers are looking ahead to a possible scenario of "losing the war" of security versus hacking, and are investigating how to change the game to mitigate the damage or economic value of breaching new systems by asking questions such as 'Can we even in the presence of a malicious attacker - offer some limited form of security for the most valuable transactions (such as e-banking) or assets?' and 'Can we make the 'business' of the attackers less attractive by applying security technologies that are particularly tailored towards destroying the business model of the attackers?' [13].

References

1. https://www.webopedia.com/TERM/I/internet_of_things.html
2. Blyler J (2017) 8 Critical IoT Security Technologies. Electronic Design.
3. Cooper D, Polk T, Regenscheid A, Souppaya M (2011) BIOS Protection

Guidelines. National Institute of Standards and Technology, USA.

4. Kanuparthi A, Zahran M, Karri R (2012) Architecture Support for Dynamic Integrity Checking. IEEE Trans Inf Forensics Security 7: 321-332.
5. Kai T, Xin X, Guo C (2012) The Secure Boot of Embedded System Based on Mobile Trusted Module. Second International Conference on Intelligent System Design and Engineering Application.
6. Armknecht F, Sadeghi AR, Schulz S, Wachsmann C (2013) A Security Framework for the Analysis and Design of Software Attestation. Proceedings of the ACM SIGSAC conference on Computer & Communications Security.
7. Perrig (2012) A Software Based Attestation Software Root of Trust. Carnegie Mellon University, USA.
8. Lin KJ, Wang CY (2012) Using TPM to improve boot security at BIOS layer. IEEE International Conference on Consumer Electronics.
9. Aaraj N, Raghunathan A, Ravi S, Jha N (2007) Energy and execution time analysis of a software-based trusted platform module. DATE '07 Proceedings of the conference on Design, automation and test in Europe, Nice, France. pp: 1128-1133.
10. Yitbarek SF, Aga MT, Das R, Austin T (2017) Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. IEEE International Symposium on High Performance Computer Architecture (HPCA). pp: 313-324.
11. http://www.techdesignforums.com/practice/guides/side-channel-analysis-attacks/
12. https://www.windriver.com/whitepapers/security-in-the-internet-of-things/wr_security-in-the-internet-of-things.pdf
13. Kursawe K, Katzenbeisser S (2007) Computing under occupation. NSPW '07 Proceedings of the Workshop on New Security Paradigms, New Hampshire. pp: 81-88.