Appendix D:

IoT Vulnerabilities

1 Introduction

There are an estimated 31 billion Internet of Things devices currently in use. As these devices are being adopted in all aspects of life - from home devices, medical devices, industrial devices - there are valid concerns raised about the security capabilities of these devices. The requirement for improvements to the resilience of these devices to attack is clear following a number of large scale botnet attacks, including the Mirai attack.

The Mirai attack was able to generate a massive 1Tbps of data, and targeted the Dyn DNS service provider. It is estimated that over 600,000 devices were a part of the Mirai botnet. The Mirai code exploits poor user credential implementations on a variety of consumer IoT devices. Mirai uses 60 hard-coded username and password pairs to gain access to the devices. These credentials are a combination between factory-set, and common user-set username and password pairs.

The ability for a botnet to grow this large, and to launch such a large scale flooding attack leads to the questioning of the security implementations on any connected embedded system, and of the other vulnerabilities which may be exploited.

The aim of this evaluation is to identify the vulnerable areas of IoT security, whether this is user authetication, transport protocol, or encryption based.

2 IoT Device Resources

Internet of Things devices are typically low-power devices and sensors. These devices are network connected, and often interact with a users personal information. As an affect of the low power resources used on these devices, the devices critical functions take precedence in terms of memory or processing performance. This leaves little availability in terms of available resources for aspects such as security.

3 Attack Vectors

There are a number of ways in which an attacker can gain acces to, or completely shut down a device. These are known as attack vectors. Attack vectors in terms of the Internet of things can be vulnerabilities due to the encyrption used, the transport protocol, or the user authentication. Mohamad Noor et al. breaks the IoT network architecture into three layers, the Application Layer, the Network Layer, and the Perception Layer[1]. The perception layer deals with the device hardware. The network layer deals with the protocols used for device communications. Finally, the applcation layer deals with the services connected to the devices, such as a cloud computing platform. Each layer can be linked to one of the aforementioned attack vectors.

3.1 User Authentication

User authentication methods within IoT, as shown by the Mirai botnets, is an area of clear exposure for devices. Due to the overhead which is associated with implementing complex authentication systems, such as the LKA protocol[1], the devices manufacturer may choose to forgo these implementations in order to reserve resources for the devices critical functionality.

At the application layer, Ammar et. al present how Amazon Web Services' (AWS) IoT

framework provides authentication using the Identity and Acess Management service of the AWS cloud computing platform[2]. AWS IAM is a policy based authentication service. Using this service, any IoT device connected to the AWS IoT cloud is given access to resources within the cloud based on a user-defined access policy.

3.2 Encryption

Encryption tends to suffer from the same issues as user authentication protocols, in that manufacturers often choose to forgo a more complex implementation to reserve devices resources.

3.3 Communication Protocols

4 OS Security Features

Non-resource constrained, "fully fledged" operating systems which run on consumer PCs and enterprise server hardware have a number of security features which can be implemented to protect against attack. These range from firewall and packet filtering software, to firmware verification software.

4.1 Secure Boot

Secure boot is a security mechanism used to verify that software being run on a system is from a trusted source. The vendors whose software has been granted permission to run on the system are stored in firmware[3]. This allows for any software being loaded onto the device at boot time can be verified by comparing the vendor signature against the keys which are stored in firmware.

4.1.1 UEFI Secure Boot

UEFI found the need to implement the secure boot mechanism as protections for post-boot code injection had become competent enough that attackers were beginning to focus their efforts on pre-boot firmware injection. Pre-boot malware injection includes rootkit and bootkits, which, before secure boot, proved extremely difficult to detect.

There are a number of intended results for pre-boot exploits, ranging from control over the operating system, or user identification credential snooping, to control over ROM and PCIe peripherals.

Vendor signatures are stored in databases in the devices firmware. The two types of keys used for accessing these databases are Platform Keys, and Key Exchange Keys[4]. Platform keys are the vendor defined key, which prevents modifications to the Key Exchange Key. The Key Exchange Key prevents modifications to the signature database. This mechanism allows vendors with valid KEK's to modify the signature database, such as inserting or deleting a signature. By comparing each code module's signature to the database at boot time, a decision can be made on whether to load this module and proceed with the boot process, or not to load the module and terminate the boot process.

The device manufacturer controls the signatures which are included from the factory. Vendors such as Microsoft offer programs for vendors to submit modules for authorization, to have their signatures added to the signatures database. This system has been mirrored by a number of Linux distributions.

Secure boot is compatible with the x64 platform, and is enabled by default on Windows and most Linux distributions[5]. By enabling this mechanism by default, the user must disable the feature to open themselves to vulnerability.

4.1.2 Secure Boot in IoT Systems

While IoT devices typically have a one-time setup, they can be rebooted multiple times over there lifetime, especially in the case of smart home IoT devices, as they are relocated or power cycled. By initiating their boot process, these devices become vulnerable to the injection of malicious code modules. There is a recognition for the need of a secure boot type mechanism for Internet of Things devices, with implementations such as MCUboot in development for a number of microcontroller based operating systems such as RIOT and Zephyr[6]. There are, however, a number of limitations which have to be considered, namely the lack of available storage on these constrained devices. As a workaround for this constraint the boot code can be written to non-modifiable ROM, although this could raise issues in the devices future, as patches required to the devices firmware to mitigate other vulnerabilities could not be implemented[7].

4.2 Device Level Firewalls

Firewalls define rules used for the purpose of allowing or blocking traffic entering a system.

4.2.1 Linux Iptables and Netfilter

Iptables is a software package for filtering traffic at the Network Layer (layer 3 of the OSI model). This package interacts with the "netfilter" kernel hooks, which are five hooks that can be triggered at different stages of packet processing. They work by applying a set of rules to incoming or outgoing packets. These rules can be pre-applied, i.e. defined by the manufacturer, or user-defined.

Iptables are broken down into a number of layers. The first of these is the table. The table is used to separate rules by function. The default tables are the "filter" table, the "NAT" table, the "mangle" table, the "raw" table, and the "security" table. Tables

contain a number of chains, which are associated with the five netfilter hooks, and are triggered when a packet invokes a rule.

The built in iptable chains are triggered by each of the five netfilter hooks as follows:

- NF_IP_PRE_ROUTING -> PREROUTING
- NF_IP_LOCAL_IN -> INPUT
- NF_IP_FORWARD -> FORWARD
- NF_IP_LOCAL_OUT -> OUTPUT
- NF_IP_POST_ROUTING -> POSTROUTING

5 IoT Vulnerabilities

Neshenko et al.[8] define 9 classes of IoT vulnerabilities. These are deficient physical security, insufficient energy harvesting, inadequate authentication, improper encryption, unnecessary open ports, insufficient access control, improper patch management capabilities, weak programming practices, and insufficient audit mechanisms. While a number of these are not relevant to the susceptibility of these devices to malicious attack for their future use in a large scale botnet, for example physical security - an attack is not going to physically connect their PC to thousands of IoT devices to build their botnet - issues such as improper and encryption authentication methods, open ports, and an inability to update flaws in firmware certainly leave these devices vulnerable to attack on a large scale.

5.1 IoT Operating Systems

There are a number of commonly used Internet of Things focused operating systems, The following sections will look at the known vulnerabilities for four of these operating

systems, namely RIOT-OS, Contiki-OS, Amazon FreeRTOS, and Zephyr OS.

5.1.1 RIOT-OS

RIOT is a free, open-source operating system for low power microcontrollers, from 8 to 32 bits. The operating systen uses a micro-kernel architecture, with very low initial memory usage[9]. The micro-kernel architecture offers advantages such as the availability of third-party modules for functions such as networking and encryption. RIOT uses the "GNRC" (generic) network stack, which aimed to cater to the low power, low resource availability contraints which are associated with Internet of Thigns devices[10].

5.1.2 Contiki-OS

Contiki, or "Contiki-NG" is another open-source operating system for low power microcontrollers. The operating systems code base is only 100kB, and can have memory usage of as low as 10 kB. This operating system focuses primarily on 32 bit microcontrollers. Contiki-NG aims to implemenet a low-power IPv6 networking stack.

5.1.3 AWS FreeRTOS

Amazon Web Services FreeRTOS is a real-time operating system for microcontrollers, which has the benefit of being closely tied with AWS cloud computing platform.

5.1.4 Zephyr OS

Zephyr OS is a project by the Linux Foundation which is focused on IoT device security. Much like the Linux operating system, Zephyr is uses a monolithic architecture. It is intended for constrained resource IoT devices, and supports devices which are 32

and 64 bit systems. The security focus of the Zephyr project has the benefits of providing systems such as Secure Boot, as discussed earlier, and over the air (OTA) update mechanisms.

5.2 Common Vulnerabilities and Exposures

CVE, or Common Vulnerabilities and Exposures, is a publicly accessible list of cybersecurity vulnerabilities[11]. This list is comprised of reported vulnerabilities from user and engineers, with details of the vulnerability and level of exposure to the system due to the vulnerability. Each CVE entry contains a CVE ID, details of the vulnerability, and Common Vulnerability Scoring System (CVSS) score. The CVSS score is calculated using a number of metrics, including at its base level the attack vector and complexity, the requirement for user interaction, and the impact of the vulnerability on a users confidentiality or the availability of the service[12]. The score defines a severity rating from Low to High for the listed vulnerability. With the CVSS 3.0 rating system, scores from 0.1 to 3.9 are low severity, 4.0 to 6.9 is medium severity, 7.0 to 8.9 are high severity, and 9.0 to 10 are critical severity.

Listed in Table 1 are the currently listed CVE reports for the IoT operating systems RIOT, Contiki, Amazon FreeRTOS and Zephyr OS.

Table 1: IoT Operating System CVE Logs

Operating System	CVE ID	CVE Score
RIOT-OS	CVE-2019-16754	5.0
	CVE-2019-15702	5.0
	CVE-2019-15134	7.8
Contiki-OS	CVE-2017-7296	4.3
	CVE-2017-7295	7.8
Amazon FreeRTOS	CVE-2018-16603	4.3
	CVE-2018-16602	4.3
	CVE-2018-16601	6.8
	CVE-2018-16600	4.3
	CVE-2018-16599	4.3
	CVE-2018-16598	4.3
	CVE-2018-16527	4.3
	CVE-2018-16526	6.8
	CVE-2018-16525	6.8
	CVE-2018-16524	4.3
	CVE-2018-16523	5.8
ZephyrOS	CVE-2018-1000800	7.5
	CVE-2017-14202	4.6
	CVE-2017-14201	4.6
	CVE-2017-14199	7.5

5.2.1 RIOT OS Vulnerabilities

The listed RIOT OS vulnerabilities range from medium to high severity. The two medium severity vulnerabilities are protocol vulnerabilities. The first of these is an issue with the OS's implementation of the MQTT messaging protocol[13]. Due to the implementation, null pointer dereferencing results in a segmentation fault, creashing

the system[14]. The second of these protocol vulnerabilities is a TCP parsing error[15], whereby the system can be places in an infinite loop while attempting to parse a packet with an unknown option specified, and an option length of zero[16].

The highest severity vulnerability listed for RIOT OS is another protocol vulnerability. This is cited as exposing the devices running this operating system to Denial of Service attacks. An error with the implementation of the TCP handshake, in which a memory leak occurs when an ACK is received by the device as the first packet in the handshake instead of a SYN packet[17]. This memory leak can lead to an exhaustion of memory resources on the device.

5.2.2 Contiki-OS

Both the first and second listed CVE reports for Contiki-OS refer to the same Cross-Site Scripting vulnerability in the mqtt.html configuration page. The first of these reports is a medium severity vulnerability[18], while the second is listed as a high severity vulnerability[19]. This difference in severity is due to the level of detail given in the CVE reports. This vulnerability in the operating systems included MQTT configuration implementation causes a null pointer dereference error, which can be initiated via a HTTP POST request, and can cause the host device to crash, causing a denial of service[20].

5.2.3 Amazon FreeRTOS

There are 11 listed CVE reports for Amazons FreeRTOS[21], all of which are based on vulnerabilities in the operating systems TCP/IP implementation. The higher scored medium severity reports are vulnerable to DoS attack and remote code execution. Report CVE-2018-16526[22] outlines how a buffer overflow occuring during the generation of a checksum can give an attacker access to execute code on the device. The vulnerability described in report CVE-2018-16601[23] can lead to a denial of service. All other listed vulnerabilities are either due to buffer overflows or TCP packet parsing error.

5.2.4 ZephyrOS

The listed vulnerabilities for Zephyr OS range from medium to high severity[24]. The first of these allows an attacker connected via telnet to cause a system crash, and therefore a denial of service attack. The crash could be caused by executing the "history" command, causing a buffer overflow**zepCVe1**. The second medium severity report is a duplicate of this issue.

The first of the high severity reports[25] details how an incorrect implementation of type checking within the sys_ring_bug_get() and sys_ring_buf_put() kernel API calls caused the kernel to spin, which again could be used to initiate a denial of service[26].

The final report is of high severity[27], and details how a buffer overflow can be initiated due to the kernel getaddrinfo() call, which is used during DNS resolution[28].

6 Detection and Mitigation Tecniques

6.1 Detection

6.2 Mitigation

7 Conclusion

References

[1] M. b. Mohamad Noor and W. H. Hassan, "Current research on internet of things (iot) security: A survey," *Computer Networks*, DOI: 10.1016/j.comnet.2018.11.025.

[2] M. Ammar, G. Russello and B. Crispo, "Internet of things: A survey on the security of iot frameworks," *Journal of Information Security and Applications*, vol. 38, pp. 8–27, 2018. DOI: 10.1016/j.jisa.2017.11.002.

[3] *Secure boot*. [Online]. Available: https://wiki.ubuntu.com/UEFI/SecureBoot.

[4] R. Wilkins and B. Richardson, *UEFI Secure Boot in Modern Computer Security Solutions*, [Online]. Available: https://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf.

[5] C. Hoffman, *How secure boot works on windows 8 and 10, and what it means for linux*, Jul. 2017. [Online]. Available: https://www.howtogeek.com/116569/htg-explains-how-windows-8s-secure-boot-feature-works-what-it-means-for-linux/.

[6] JuulLabs-OSS, *Juullabs-oss/mcuboot*. [Online]. Available: https://github.com/JuulLabs-OSS/mcuboot.

[7] I. S. Foundation, *Device secure boot*. [Online]. Available: https://www.iotsecurityfoundation.org/best-practice-guide-articles/device-secure-boot/.

[8] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019. DOI: 10.1109/comst.2019.2910750.

[9] *Riot in a nutshell*. [Online]. Available: https://doc.riot-os.org/.

[10] M. Lenders, "Analysis and comparison of embedded network stacks,"

[11] *Common vulnerabilities and exposures (cve)*. [Online]. Available: https://cve.mitre.org/index.html.

[12] *Common vulnerability scoring system calculator*. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[13] *Vulnerability details : Cve-2019-16754*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-16754/.

[14] Riot-Os, *Asymcute: Fix null pointer dereference by nmeum · pull request #12293 · riot-os/riot*. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/12293.

[15] *Vulnerability details : Cve-2019-15702*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-15702/.

[16] ——, $Gnrc_tcp : Optionparsingdoesn't terminateonallinputs, potentialdosissue$#12086$riot-os/riot$. [Online]. Available: https://github.com/RIOT-OS/RIOT/issues/12086.

[17] *Vulnerability details : Cve-2019-15134*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-15134/.

[18] *Vulnerability details : Cve-2017-7296*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-7296/.

[19] *Vulnerability details : Cve-2017-7295*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-7295/.

[20] 262588213843476, *Contiki mqtt and xss*. [Online]. Available: https://gist.github.com/jackmcbride/c9328627f1ee104ce84f3fb7eff42f1e.

[21] *Amazon " freertos : Security vulnerabilities*. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-12126/product_id-51624/Amazon-Freertos.html.

[22] *Vulnerability details : Cve-2018-16526*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2018-16526/.

[23] *Vulnerability details : Cve-2018-16601*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2018-16601/.

[24] *Zephyrproject " zephyr : Security vulnerabilities*. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-19255/product_id-50119/Zephyrproject-Zephyr.html.

[25] *Vulnerability details : Cve-2018-1000800*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2018-1000800/.

[26] Zephyrproject-Rtos, *Get fault when fuzzing sys_ring_buf_put and sys_ring_bug_get api sis sue* #7638 *zephyr project – rtos/z* [Online]. Available: https://github.com/zephyrproject-rtos/zephyr/issues/7638.

[27] *Vulnerability details : Cve-2017-14199*. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-14199/.

[28] ——, *Net: Sockets: Getaddrinfo() buffer overflow, etc. fixes by pfalcon · pull request 6158 · zephyrproject-rtos/zephyr*. [Online]. Available: https://github.com/zephyrproject-rtos/zephyr/pull/6158.