Attacking Intel UEFI by Using Cache Poisoning

Dong Wang* Wei Yu Dong
State Key Laboratory of Mathematical Engineering And Advanced Computing
Zhengzhou, China
*a18844195710@163.com

Abstract—The Unified Extensible Firmware Interface (UEFI) is a software interface between an operating system and platform firmware designed to replace a traditional BIOS. In this paper, we evaluated the security mechanisms used to protected SPI Flash, and then analyzed the attack surface presented by those security mechanisms. Intel provides several registers in its chipset relevant to locking down the SPI Flash chip that contains the UEFI in order to prevent arbitrary writes. Since these registers implement their functions through the system management mode, the main attack surface is concentrated in the system management mode. In this paper, we propose an attack vector for the system management mode, which uses the method of cache poisoning to attack the system management mode and destroy the protection mechanism of SPI Flash. This method can overcome the limitations for the traditional attacks. Experimental results proved that this kind of attack can arbitrarily write to the UEFI.

1. INTRODUCTION
The BIOS is the first code to execute on a platform during power on. It is written in assembly language and is difficulty to extend due to its difficulty in writing. The BIOS is the interface between the operating system and the hardware. However, there is no uniform standard for this interface, which makes the products of each BIOS manufacturer to be different from each other. Moreover, the traditional BIOS has many problems in hardware compatibility and security. In 2003, Intel designed and proposed a new generation of firmware standards: Unified Extensible Firmware Interface (UEFI)[1]. Compared with traditional BIOS, UEFI provides a well interface specification for drivers and applications with favorable scalability.

UEFI’s responsibilities include configuring the platform, initializing critical platform components, and locating and transferring control to an operating system[2]. UEFI is also responsible for configuring and instantiating System Management Mode (SMM), a highly privileged mode of execution on the x86 platform. Thus any malware that controls the UEFI is able to place arbitrary code into SMM, which may generate a huge impact on the system. The UEFI’s residence on an SPI Flash chip means it will survive operating system reinstallations. These properties make the UEFI a desirable residence for malware. Therefore, an examination of the security of UEFI is necessary.

Based on the UEFI open source specification, this paper analysis the protection mechanism of the SPI Flash where UEFI storage. And then, this paper presents an attack method, which attacks the SMM by means of cache poisoning. This attack can bypass the protection mechanism of SPI Flash and reflash the UEFI.
2. UEFI Technology

2.1. UEFI Architecture
UEFI is a new software interface between an operating system and platform firmware designed to replace a traditional BIOS\(^1\). In general, UEFI has many technical advantages over BIOS such as pre-OS environment, boot services and runtime services, CPU-independent drivers and powerful security architecture for OS kernel.

UEFI uses modular technology to divide its functions into two parts\(^3\): Hardware Control and Operating System Software Management. Figure 1 shows the architecture of UEFI.

![UEFI Overall Framework Diagram](image)

The hierarchical relationship between UEFI firmware, operating system and operating system bootloader can be seen in the figure. After the system power on, UEFI is executed first. It verifies the integrity of other firmware and is the root of trusted of firmware execution. After that, UEFI initializes and tests the underlying hardware, loads and executes additional firmware modules. Finally, UEFI choose the boot device and pass control to bootloader to authenticated OS kernel.

2.2. UEFI Security Boot
UEFI boot process pass through several stages\(^5\), each stage has its own role and may potentially pose some security risks, the startup process of UEFI is shown in Figure 2.
SEC. The SEC phase is the first phase executed by the system. It is the root of system security and provides a control node for firmware startup. The SEC stage handles the platform restart event, which uses the processor’s cache as a temporary memory area to perform security checks on the initial code of the platform. After that, the SEC forwards the pointer to the temporary memory and the temporary stack, and transfers control to the next phase.

PEI. The PEI (Pre-EFI Initialization) phase, which is written in C language, and the execution environment needs to be established during execution. The PEI phase is responsible for initializing the CPU and main memory while providing initial configuration space for components such as processors, chipsets, and system mother boards. The PEI phase passes the status information and control to the next phase through the HOBs list.

DXE. The DXE (Driver Execution Environment) phase is where the majority of the system initialization takes place. The DXE phase queries the platform related information through the HOBs and builds a complete running environment for the driver. The DXE core produces a set of Boot and Runtime Services. The DXE enumerates DXE drivers and external device drivers through the boot service, allocating memory and loading executable images. UEFI allows for a variety of sources of driver. In order to ensure the integrity and security of the platform, UEFI must ensure the legitimacy of the firmware by hash encryption and encryption authentication for the firmware. After that, The DXE transfers control to the next phase.

BDS. The Boot Device Selection phase is the final phase in which UEFI has firmware control, and it manages all boot setting through the boot manager. The BDS cooperates with the DXE, provide a user launch control platform, which is responsible for discovering the possible boot devices, selecting one to boot from, loading the Boot Loader and executing it.

3. UEFI SECURITY
The UEFI is the first code to execute on a platform during the startup process. UEFI completes the initialization operation of the system and the loading of the system. Therefore, ensuring the security of the UEFI is very important. This chapter analyzes the UEFI protection mechanism to lay the foundation for follow-up attacks

3.1. SPI Flash Protection Mechanisms
UEFI code is stored in SPI Flash. Once an attacker can maliciously write to SPI Flash, UEFI security will be greatly affected. Intel’s ICH documentation[4] provides a range of SPI Flash protection mechanisms, the most important mechanisms are the BIOS_CNTL register and the Protected Range Register. At boot time, both registers are specified by the ICH [4] and configured by the UEFI.

The protection mechanism of the BIOS_CNTL register is shown in Figure3.

![Figure 3. BIOS_CNTL Register Protection Mechanism](image-url)
The BIOS_CNTL register contains 3 important bits in this regard, BWE, BLE and SMM_BWP[6][8]. The BWE is the write enable bit, if BWE is set to 1, the user can write anything to SPI Flash. If BWE is set to 0, SPI Flash will be locked. The BLE is the lock enable bit, which is used to ensure the security of BWE. Once BLE is set to 1, any modification to BWE will trigger a System Management Interrupt (SMI) and enter SMM. In SMM, the CPU will reset the BWE. The SMM_BWP is the write protection bit in SMM. When SMM_BWP is set to 1, The BWE will only be modified in SMM.

Intel specifies a number of Protected Range (PR) registers that can also protect the flash chip against writes. These 32bit registers specify Protected Range Base and Protected Range Limit fields that sets the relevant regions of the flash chip for the write protection enable and read protection enable bits. When the write protection enable bit is set, the region of the flash chip defined by the Base and Limit fields is protected against writes. Similarly, when the Read Protection Enable bit is set, that same region is protected against read attempts. To ensure that the PR registers are not modified, Intel provides HSFS, FLOCKDN to protects the PR register. Once this bit is set, the PR register will not be modified. Although the PR register has a good protection effect, in practice, some motherboard manufacturers are not correctly configuring PR registers, because of the upgrade and maintenance. After investigation, there were (4779/5197) 92% percent of systems did not bother to implement the PR register.

3.2. System Management Mode
System Management Mode is the x86 architecture operating mode of the CPU[9][12], mainly used for motherboard control and power management. SMM is a special mode of operation, it does not depend on the specific operating system and is completely controlled by firmware. The SMM can only be entered via an SMI and can only be exited via the RSM instruction. The Northbridge chipset will generate an SMI signal by writing to Advanced Management Power Control.

The SMM entry is located in a memory called SMRAM, which is located in RAM. An SMI handler is executed from SMRAM, whenever the SMI handler executes an RSM assembly instruction, the CPU state is restored from the map saved in memory. The operating system does not even notice when it is being interrupted by management software running in SMM. The address of the SMRAM is specified by the SMBASE register of the CPU. In practice, the SMRAM address is usually 0xa0000.

SMM code runs with full privileges on the platform, even more privilege than operating system kernels. There is a need to prevent access to the SMRAM when the system is not in SMM so that only the SMI Handler can modify the content of the SMRAM. The main mechanism to prevent modification of the SMI handler is the D_LCK bit. If the D_LCK bit is set, configuration bits for the SMRAM in the chipset become read only.

In summary, this decision entangles the security of the UEFI with the security of SMM. Any vulnerabilities that can be exploited to gain access to SMM can be leveraged into an arbitrary reflash of the UEFI. So, for the SMM, this paper proposes a UEFI attack method.

4. Cache Poisoning
At the CanSecWest conference in 2006, a privilege escalation scheme was proposed, which is an actual privilege escalation scheme. The 2008 Black Hat Conference Sparks hide the keylogger in the SMI Handler[11][14]. However, these rootkits have strong limitations and will not exist once the platform restarted. Most rootkits can only be designed for a specific platform, making it difficultly to design a generic SMM rootkit. Also, the biggest limitation is that the D_LCK bit cannot be bypassed. This paper adopts the method of cache poisoning, which can overcome the limitations of traditional attacks.

For x86 processors, CPU cache strategy can be divided into Write Back, Write Through and not cacheable[8]. We assume that the memory area of the SMRAM is cached as WB. Once the SMI handler is executed, it will be cached to the CPU’s data and instruction buffer. When the SMI handler passes control to the operating system, then the SMI handler will be reside in the CPU cache for a while.
Based on the above principles, we can use this method set BWE bit as 1, so that we can arbitrarily erase the UEFI. The attack process is shown in Figure 4.

Assume that the rootkit wants to hide some malware in SMRAM. Because of the D_LCK, we cannot access to SMRAM unless in SMM. We can modify the SMRAM cache strategy as Write Back. Then the SMM handler will be temporarily parked in the cache. Then we put an attack SMM handler into a free RAM, we call this area C and the address of this area is Ac. Modify the SMI handler, the SMI handler should modify the SMBASE value in the save state of the CPU so that SMBASE = Ac-0x8000. And then trigger an SMI and the modified SMI handler will executed. The value of the SMBASE register will be set to Ac-0x8000. When the next SMI is triggered, the CPU will determine that the new SMRAM location is SMBASE+0x8000=Ac, and the attack SMI handler will be executed from memory. After that, the attack vector will reside in memory and be located and executed. At the same time, the memory will no longer be protected by D_LCK, so even the D_LCK is set, the attacker can still modify the SMI handler. At this point, we can create an SMI handler which can set BWE = 1. Now, SPI Flash write protection is turned on, UEFI can be arbitrarily erase.

5. CONCLUSIONS
UEFI is proposed to overcome the limitations of the traditional BIOS. It is a new boot platform with many advantages and great development space. However, due to its high scalability, UEFI also has many security risks. The method of cache poisoning in this paper overcomes the limitations of the traditional SMM rootkit and can successfully bypass the D_LCK protection, so that the rootkit code can reside in memory for a long time. Successfully implement the attack of UEFI. The next work has the following two points:

- This method needs to modify the CPU’s cache strategy, so it only applies to kernel-level rootkits.
- On a multicore architecture system, each CPU use a different SMRAM, so the SMI handler may be different. When an SMI occurs, SMI handler cannot be specified.

REFERENCES
[1] Intel Unified Extensible Firmware Interface Specification: Version2.3.1[EB/OL]. 2011.http://www.uefi.org/home/.
[2] Intel Corporation. DQ57TM Development Kit BIOS [EB/OL].2014. http://uefidk.com/develop/development-kit
[3] Intel Corporation. Intel 7 Series Family Platform Controller hub Datasheet[EB/OL]. http://www.intel.com/content/www/us/en/chisets//7-series-chipset-pch-datasheet.html.
[4] Intel Corporation. Intel I/O Controller Hub 10 (ICH10) Family Datasheet.[EB/OL]. http://www.intel.com/content/www/us/en/io-controller-hub-10-family-datasheet.pdf.

[5] C.Kallenberg, J.Butterworth, X.Kovah, C.Cornwell, “Defeating Signed BIOS Enforcement,” In EkoParty. Buenos Aires.2013.

[6] Corey Kallenberg, Sam Cornwell, Xeno Kovah “Setup For Failure: More Ways To Defeat SecureBoot”. In Hack In The Box Amsterdam, Amsterdam. 2014.

[7] R.Wojtczuk, C.Kallenberg, “Attacking UEFI Boot Script.” In Chaos Communicating Congress. Hamburg; Germany. 2014.

[8] Corey Kallenberg, Rafal Wojtczuk. Exploiting an Intel Flash Protection Rase Condition. 2013.

[9] Sherri Sparks, Shawn Embleton, “SMM Rootkits: A New Breed Of OS Independent Malware,” Lasveagas. 2008.

[10] Rod Smith. "Managing EFI Boot Loaders For Linux: Dealing With Secure Boot.". http://www.rodsbooks.com/efi-bootloaders/secureboot.html.

[11] Cve-2013-3582. http://www.kb.cert.org/vuls/id/912156, 2013-01-10.

[12] J. Brossard, Hardware backdooring is practical. In BlackHat, Las Vegas, USA, 2012.

[13] Y. Bulygin. Evil maid just got angrier. In CanSecWest, Vancouver, Canada, 2013.

[14] Y. Bulygin, A Furtak, O. Bazhaniuk. A tale of one software bypass of windows 8 secure boot. In Black Hat, Las Vegas, USA, 2013.

[15] J. Butterworth, C. Kallenberg, X. Kovah. Bios chronomancy: Fixing the core root of trust for measurement. In BlackHat, Las Vegas, USA, 2013.