Speed Racer: Exploiting an Intel Flash Protection Race Condition

Corey Kallenberg Rafal Wojtczuk

Abstract

In this paper we describe a race condition that allows an attacker to subvert a component of the firmware flash protection mechanisms provided by Intel chipsets. Although the impact of this attack is mitigated by additional chipset flash protection features, we discuss how these additional features can also be overcome in practice.

1 Introduction

The platform firmware[1] is primarily responsible for initializing the hardware and then transfering control to an operating system. Due to the earliness of its execution, the firmware is positioned to compromise the other components of the software stack. Furthermore, the firmware lives on a flash chip that is persistent across operating system reinstallations. These properties dictate that maintaining firmware integrity is of critical importance to the security of the platform. However, it is also desirable to provide an update capability by which an OEM can update the contents of the firmware for the purposes of patching bugs or adding additional features. Thus a potential conflict exists between the need to maintain the integrity of the firmware and the need to update the contents of the firmware. To resolve this conflict, the Intel chipset provides both flash programming and flash protection features. These complimentary features are used by OEMs to implement a firmware update routine that writes authentic OEM updates to the flash and then protects the firmware against other arbitrary update attempts.

2 Flash Programming Mechanisms

On modern hardware, programming the firmware is accomplished through a memory mapped programming interface exposed by the chipset[3]. The Serial Peripheral Interface Memory Mapped Configuration (SPIBAR) Registers provides an interface to control flash programming operations. Among these registers, Hardware Sequencing Flash Control (HSFC), Flash Address (FADDR), and Flash Data (FDATA) are directly relevant for initiating flash programming operations. FADDR determines the target location for the programming operation. HSFC controls the size and type (read,write,erase) of the programming operation. The FDATA registers contain the data to write during a write operation, and the data read during a read operation. Once these registers have been configured, setting the FGO bit in HSFC initiates the flash programming operation. Figure 1 depicts the process of a kernel driver programming a sector of the firmware.

3 Flash Protection Mechanisms

Due to the importance of maintaining firmware integrity, the chipset also provides mechanisms for protecting the firmware from arbitrary flash programming operations. The primary defense against malicious or accidental firmware modification is provided by the BIOS Control register (BIOS_CNTL) and the Protected Range (PR) registers. The PR registers can be programmed to mask off specified regions of the flash as unprogrammable. The PR registers are themselves protected against modification by the Flash Configuration Lock-Down (FLOCKDN) bit in the Hardware Sequencing Flash Status (HSFS) register. Once set, FLOCKDN can only be cleared by a system reset.

[1] BIOS on older systems, UEFI on newer systems.

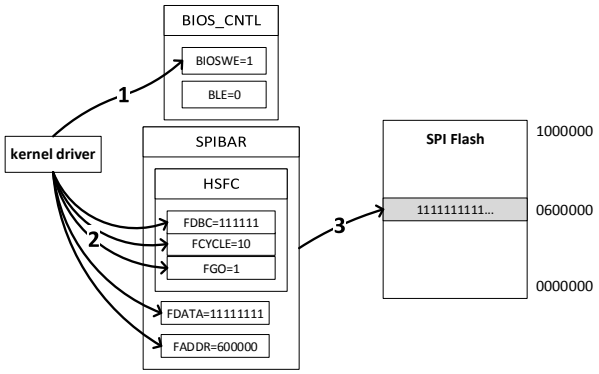

Figure 1: Initiating SPI Programming Operation

The BIOS_CNTL register contains the BIOS Lock Enable (BLE) and the BIOS Write Enable (BIOSWE) bits. Flash programming operations are only permitted when BIOSWE is set. Setting BLE forces a System Management Interrupt (SMI) to execute whenever BIOSWE is set. In this way, the SMI handler can evaluate whether or not the attempt to write enable the flash is legitimate, and if not, unset BIOSWE. Once BLE is set, it can only be cleared through a platform reset. The important distinction between PR flash protection and BIOS_CNTL flash protection, is that the former blocks all programming access to the flash, while the latter puts a privileged arbitirator[2] in a position to allow or disallow programming attempts.

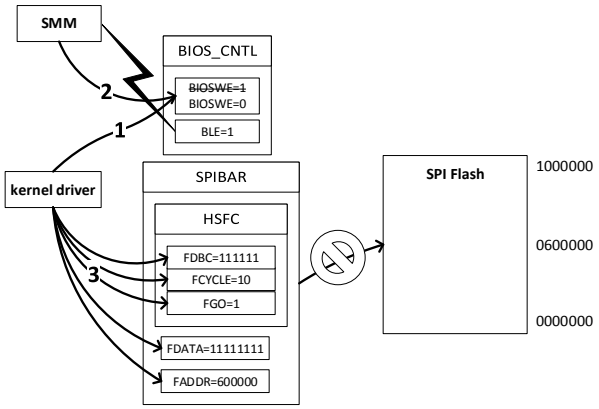

Figure 2: SPI Programming Blocked By SMM

The transition to the Platform Controller Hub (PCH) chipset architecture introduced a new bit to the BIOS_CNTL register: SMM BIOS Write Protect Disable (SMM_BWP)[4]. The description of the SMM_BWP bit gives pause: *(when set) The BIOS Region is not writable unless all processors are in SMM.* The attentive reader may wonder what the SMM_BWP description implies about the robustness of the BIOS_CNTL flash protection in the absence of SMM_BWP.

4 Race Condition

The authors identified a race condition in the BIOS_CNTL flash protections when SMM_BWP is unset or unavailable. On a multicore system, an attacker can program one core to set BIOSWE in a tight loop, and program another core to repeatedly attempt a flash programming operation. Occasionally, the flash

[2]An SMI Handler.

programming operation will succeed before the SMI handler invoked by the BLE protection unsets BIOSWE. The authors implemented this attack with a pair of Windows kernel drivers: a driver for setting BIOSWE upon receiving an IOCTL (input/output control), and a driver for attempting a flash programming operation upon receiving an IOCTL. A corresponding pair of userland agents would repeatedly send IOCTL requests to their respective kernel driver components. The exploitation process is depicted visually in figure 3. This

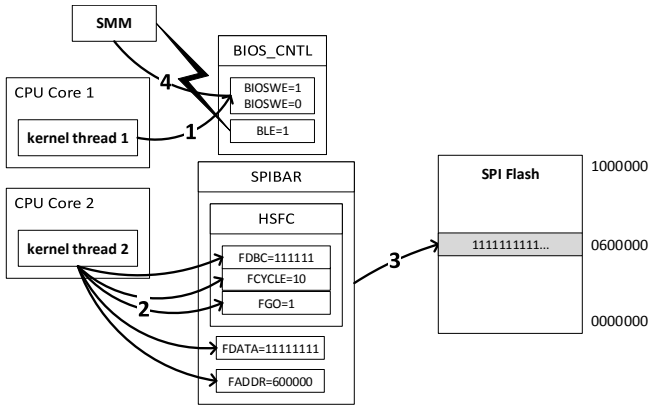

Figure 3: BIOS_CNTL Race Condition Exploited

attack was verified to succeed on a single core system with hyper threading enabled. On single core systems with hyper threading disabled, the SMI handler executes immediately after setting BIOSWE, and no other instructions are given an opportunity to run while BIOSWE remains set. This vulnerability was reported to CERT and Intel in May 2014 and assigned tracking number CERT VU#766164.

The result of successful exploitation of the race condition is an attacker is allowed to make arbitrary changes to any portion of the BIOS firmware not protected by Protected Range masks.

5 Bypassing Protected Range Masks

Although the race condition allows the circumvention of BIOS_CNTL flash protection, it remains to be shown how PR protection can also be defeated on modern systems. Most modern computers ship with UEFI firmware as opposed to legacy BIOS firmware. Interestingly, portions of the UEFI region of the flash are expected to change throughout the runtime of the system. For instance, UEFI non-volatile variables can be updated by the operating system and the contents of these non-volatile variables are written to the flash. Due to the need for runtime updateability, this region of the firmware must be left unprotected by PR masks. Instead, the variable region is reliant on the SMM protection provided by BIOS_CNTL. Thus, by leveraging the race condition, an attacker can make arbitrary changes to the variable region.

Contained within the variable region is data that is only controllable by SMM and is thus treated as trusted by the firmware. For instance, stored in the variable region are the authenticated variables which are normally not arbitrarily updateable, and which control UEFI Secure Boot functionality. The data within the variable region is also encapsulated in a complex filesystem format. Using the race condition, an attacker can maliciously modify the filesystem metadata in an effort to instantiate a memory corruption vulnerability during the the firmware's parsing of the variable region. If this vulnerability can be instantiated during platform boot and before PR masks are applied, an attacker can hijack control of the boot process and successfully perform flash programming operations. Listings 1 and 2 show examples of such vulnerabilities discovered by the authors in the UEFI reference implementation's parsing of the variable region.

```
// file: SecurityPkg/VariableAuthenticated/RuntimeDxe/AuthService.c
// function: AutenticatedVariableServiceInitialize
// Check "AuthVarKeyDatabase" variable's existence.
// If it doesn't exist, create a new one with initial value
// of 0 and EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS set.

Status = FindVariable (
           AUTHVAR_KEYDB_NAME,
           &gEfiAuthenticatedVariableGuid,
           &Variable,
           &mVariableModuleGlobal->VariableGlobal,
           FALSE
           );

if (Variable.CurrPtr == NULL) {
  ...
} else {
  //
  // Load database in global variable for cache.
  //
  DataSize  = DataSizeOfVariable (Variable.CurrPtr);
  Data      = GetVariableDataPtr (Variable.CurrPtr);
  ASSERT ((DataSize != 0) && (Data != NULL));
  CopyMem (mPubKeyStore, (UINT8 *) Data, DataSize);
```

Listing 1: UDK2014 Authenticated Variable Parsing Vulnerability

Listing 1 describes a variable region parsing vulnerability the authors discovered in UDK2014[5]. Because it is an authenticated variable, the contents of **AuthVarKeyDatabase** are not normally attacker controlled, and hence its contents are treated as trusted by the UEFI code. However, by exploiting the race condition, an attacker can arbitrarily control the contents of the variable. Included in the contents of this otherwise trusted variable is the size parameter. By specifying a large enough size, an attacker can induce a buffer overflow in the CopyMem operation and hijack control flow.

```
//file: Sample/Universal/Variable/RuntimeDxe/FS/FSVariable.c
//function: Reclaim
Variable = (VARIABLE_HEADER *) (VariableStoreHeader + 1);
...
ValidBufferSize = sizeof (VARIABLE_STORE_HEADER);
while (IsValidVariableHeader (Variable)) {
  NextVariable = GetNextVariablePtr (Variable);
  ...
  if (Variable->State == VAR_ADDED) {
    VariableSize = (UINTN) NextVariable - (UINTN) Variable;
    EfiCopyMem (CurrPtr, (UINT8 *) Variable, VariableSize);
    ValidBufferSize += VariableSize;
    CurrPtr += VariableSize;
```

Listing 2: EDK Variable Region Reclaim Vulnerability

Listing 2 describes another variable region parsing vulnerability the authors discovered in EDK1[2]. Using the race condition, an attacker can control the contents of the Variable structure. By supplying a large VariableSize, CurrPtr can potentially extend beyond the acceptable limits of the variable region. Then during the next loop iteration an out of bounds write will occur when EfiCopyMem is called, resulting in a memory corruption. Depending on the UEFI implementation, an attacker could exploit the vulnerabilities described in listing 1 or 2 to bypass PR mask protection and reflash the entire contents of the UEFI region. These vulnerabilities were reported to CERT and Intel in September 2014 and assigned tracking number VU #533140.

In fact, it may not be necessary to resort to such memory corruption attacks. In the UEFI reference implementation, firmware updates are validated by authenticated variables. An attacker can thus insert his

own evil certificate into the relevant authenticated variable, sign a malicious update with the evil private key, and then use the legitimate update process to reflash the firmware with a malicious image. However, many OEMs choose not to use the UEFI reference implementation's version of firmware updates, and instead implement their own custom solution. The next section discusses further why bypassing PR protection may not be necessary.

6 Vulnerable Systems

It has previously been pointed out that many OEMs fail to utilize PR registers[6]. Furthermore, although use of SMM_BWP negates the race condition vulnerability, use of the SMM_BWP bit also appears sporadic[7]. Systems that fail to use both PR registers and SMM_BWP are vulnerable to having their firmware trivially reflashed. UEFI systems that use PR registers to protect parts of their firmware code can likely still be reflashed through more sophisticated exploitation methods, such as those described in section 5. The following is a list of systems in posession of the authors which were determined to be vulnerable.

1. Dell Latitude E6400 at BIOS Revision A31: Does not support SMM_BWP[3] and does not use PR registers.
2. Dell Latitude E6430 at BIOS Revision A16: Does not use SMM_BWP and does not use PR registers.
3. Dell Latitude E6520 at BIOS Revision A19: Does not use SMM_BWP and does not use PR registers.
4. Intel Desktop Board DQ57TML with UEFI Development Kit firmware SDV13: Does not use SMM_BWP[1].

   Variable region is not protected by PR masks.
5. HP EliteBook 2540p at BIOS Revision F22: Does not use SMM_BWP.

   Variable region is not protected by PR masks.

7 Conclusion

We have described how systems that fail to use or do not support SMM_BWP are subject to a race condition that allows reflashing of the firmware. Although the firmware may be further protected by PR masks, we have described scenarios in which an attacker can circumvent these protections as well. The authors strongly encourage OEMs to make use of SMM_BWP on systems where it is available. Use of SMM_BWP negates the race condition attack and forces an attacker to break into SMM before the firmware contents can be directly targetted. Although breaking into SMM is a substantial barrier which necessitates an additional privilege escalation vulnerability, the authors describe such an SMM runtime vulnerability prevalent in UEFI firmware in a related paper[8].

8 Acknowledgments

Thanks to the Intel Product Security Incident Response Team (PSIRT) for helping coordinate these vulnerabilities with affected OEMs and IBVs.

References

[1] Intel Corporation. DQ57TM Development Kit BIOS. `http://uefidk.com/develop/development-kit`. Accessed: 12/01/2014.

[2] Intel Corporation. EFI Development Kit. `http://tianocore.sourceforge.net/wiki/EDK`. Accessed: 12/1/2014.

[3] E6400 uses ICH9 chipset which does not support SMM_BWP.

[3] Intel Corporation. Intel I/O Controller Hub 10 (ICH10) Family Datasheet. http://www.intel.com/content/www/us/en/io/io-controller-hub-10-family-datasheet.pdf. Accessed: 12/03/2014.

[4] Intel Corporation. Intel Platform Controller Hub 5 (PCH5) Family Datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/5-chipset-3400-chipset-datasheet.pdf. Accessed: 12/03/2014.

[5] Intel Corporation. UEFI Development Kit 2010. http://sourceforge.net/projects/edk2/files/UDK2014_Releases/UDK2014/. Accessed: 06/13/2014.

[6] C. Kallenberg, J. Butterworth, X. Kovah, and C. Cornwell. Defeating Signed BIOS Enforcement. In *EkoParty*, Buenos Aires, 2013.

[7] C. Kallenberg, C. Cornwell, X. Kovah, and J. Butterworth. Setup For Failure: More Ways to Defeat SecureBoot. In *Hack In The Box Amsterdam*, Amsterdam, 2014.

[8] R. Wojtczuk and C. Kallenberg. Attacking UEFI Boot Script. In *Chaos Communication Congress*, Hamburg, Germany, 2014.