Defeating Signed BIOS Enforcement

Corey Kallenberg *John Butterworth* *Xeno Kovah*
Sam Cornwell
ckallenberg@mitre.org, jbutterworth@mitre.org
xkovah@mitre.org, scornwell@mitre.org
The MITRE Corporation

Abstract

In this paper we evaluate the security mechanisms used to implement signed BIOS enforcement on an Intel system. We then analyze 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 BIOS in order to prevent arbitrary writes. It is the responsibility of the BIOS to configure these SPI flash protection registers correctly during power on. Furthermore, the OEM must implement a BIOS update routine in conjunction with the Intel SPI flash protection mechanisms. The BIOS update routine must be able to perform a firmware update in a secure manner at the request of the user. It follows that the primary attack surfaces against signed BIOS enforcement are the Intel protection mechanisms and the OEM implementation of a signed BIOS update routine. In this paper we present an attack on both of these primary attack vectors; an exploit that targets a vulnerability in the Dell BIOS update routine, and a direct attack on the Intel protection mechanisms. Both of these attacks allow arbitrary writes to the BIOS despite the presence of signed BIOS enforcement on certain systems.

1 Introduction

The BIOS is the first code to execute on a platform during power on. BIOS's responsibilities include configuring the platform, initializing critical platform components, and locating and transfering control to an operating system. Due to its early execution, BIOS resident code is positioned to be able to compromise every other component in the system bootup process. BIOS 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 BIOS is able to place arbitrary code into SMM. The BIOS's residence on an SPI flash chip means it will survive operating system reinstallations. These properties make the BIOS a desirable residence for malware.

Although BIOS malware is an old topic, recent results that make use of BIOS manipulations have once again renewed focus on the topic. Brossard showed that implementing a BIOS rootkit may be easier than traditionally believed by making use of opensource firmware projects such as coreboot[3]. Bulygin et al showed that UEFI secure boot can be defeated if an attacker can write to the SPI flash chip containing the system firmware[5]. The Trusted Platform Module (TPM) has recently been adopted as a means for detecting firmware level malware, but Butterworth et al showed that a BIOS rootkit can both subvert TPM measurements and survive BIOS reflash attempts[6].

These results are dependent on an attacker being able to make arbitrary writes to the SPI flash chip. However, most new computers either require BIOS updates to be signed by default, or at least provide an option to enforce this policy. Signed BIOS update enforcement would prevent the aforementioned attacks. Therefore, an examination of the security of signed BIOS enforcement is necessary.

2 Related Work

Invisible Things Lab was the first attack against signed BIOS enforcement[9]. In their attack, an integer overflow in the rendering of a customizable bootup splash screen was exploited to gain control over the boot up process before the BIOS locks were set. This allowed the BIOS to be reflashed with arbitrary contents.

A related result was discovered by Bulygin where it was noticed that on a particular ASUS system, firmware updates were signed, but the Intel flash protection mechanisms were not properly configured[4]. Thus an attacker could bypass the signed BIOS enforcement by skipping the official BIOS update process and just writing to the flash chip directly.

3 Intel Protection Mechanisms

The Intel ICH documentation[7] provides a number of mechanisms for protecting the SPI flash containing the BIOS from arbitrary writes. Chief among these are the BIOS_CNTL register and the Protected Range (PR) registers. Both are specified by the ICH and are typically configured at power on by a BIOS that enforces the signed update requirement. Either (or both) of these can be used to lock down the BIOS.

3.1 BIOS_CNTL

The BIOS_CNTL register contains 2 important bits in this regard. The BIOS Write Enable (BWE) bit is a writeable bit defined as follows. If BWE is set to 0, the SPI flash is readable but not writeable. If BWE is set to 1, the SPI flash is writeable. The BIOS Lock Enable bit (BLE), if set, generates an System Management Interrupt (SMI) if the BWE bit is written from a 0 to 1. The BLE bit can only bet set once, afterwards it is only cleared during a platform reset. It is important to notice that the BIOS_CNTL register is not explicitly protecting the flash chip against writes. Instead, it allows the OEM to establish an SMM routine to run in the event that the BIOS is made writeable by setting the BWE bit. The expected mechanism of this OEM SMM routine is for it to reset the BWE bit to 0 in the event of an illegitimate attempt to write enable the BIOS. *The OEM must provide an SMI handler that prevents setting of the BWE bit in order for BIOS_CNTL to properly write protect the BIOS.*

3.2 Protected Range

Intel specifies a number of Protected Range 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. The HSFS.FLOCKDN bit, when set, prevents changes to the Protected Range registers. Once set, HSFS.FLOCKDN can only be cleared by a platform reset. The Protected Range registers in combination with the HSFS.FLOCKDN bit are sufficient for protecting the flash chip against writes if configured correctly.

3.3 Vendor Compliance

Using our Copernicus tool [1], we surveyed the SPI flash security configuration of systems throughout our organization. Of the 5197 systems in our sample that implemented signed BIOS enforcement, 4779 relied exclusively on the BIOS_CNTL register for protection. In other words, approximately 92% percent of systems did not bother to implement the Protected Range registers.[1]

4 Dell BIOS Update Routine

The BIOS update process is initiated by the operating system. The operating system first writes the new BIOS image to memory. Because the BIOS image may be several megabytes in size, one single contiguous allocation in the physical address space for accomodating the BIOS image may not be possible on systems with limited RAM. Instead, the BIOS image may be broken up into smaller chunks before being written to RAM. These small chunks are referred to as "rbu packets."[2] The rbu packets include structural information such as size and sequence number that later allow the BIOS update routine to reconstruct the complete BIOS image from the individual packets. These rbu packets also include an ASCII signature "$RPK" that the BIOS update routine searches for during a BIOS update.

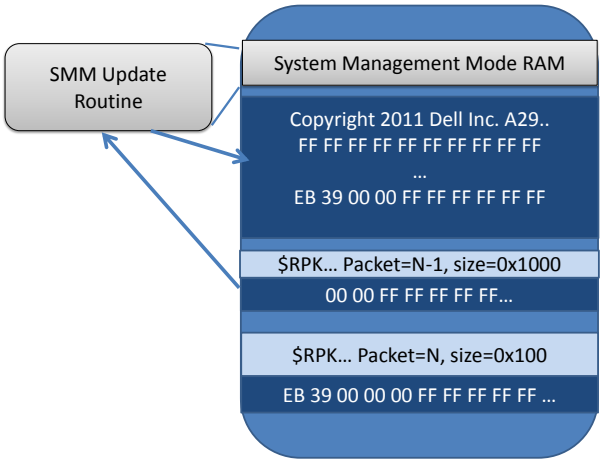

Figure 1: BIOS update image reconstituted from rbu packets

Once the rbu packets have been written to the address space, the operating system sets byte 0x78 in CMOS and initiates a soft reboot. During system startup BIOS

[1] This number is somewhat skewed by the large portion of Dell systems in our sample, which never seem to implement Protected Range registers. However, the issue is not exclusive to Dell.

[2] `http://linux.dell.com/libsmbios/main/RbuLowLevel_8h-source.html`

checks the status of CMOS byte 0x78 and if set, triggers an SMI to execute the SMM BIOS update routine. The BIOS update routine then scans the address space for rbu packets by searcing for the ASCII signature "$RPK." The particular BIOS we analyzed used physical address 0x101000 as the base of the area in RAM where it would reconstruct the incoming BIOS image from the individual rbu packets. Upon discovering each rbu packet in the address space, the update routine uses the rbu packet header information to determine where to place that chunk of the BIOS image described by the current rbu packet in the reconstruction space. Once all rbu packets have been discovered and the new BIOS image has been reconstituted in the reconstruction area, the update routine verifies that the new image is signed with the Dell private key. After the signature has been verified, the new image is written to the flash chip.

5 Attacking Dell BIOS Update

Any vulnerabilities in the BIOS update process that can be exploited before the signature check on the incoming image occurs, can lead to an arbitrary reflash on the BIOS. Importantly, the update process is required to reconstruct the complete update image from the individual rbu packets scattered across the address space before a signature check can occur. Because the rbu packets are generated on the fly by the operating system at runtime, the rbu packets are unsigned.

5.1 Dell BIOS Vulnerability Specifics

After examining the update routine's parsing of rbu packets, a memory corruption vulnerability was identified that stemmed from improper sanity checking on the unsigned rbu packet header. Upon discovery of an rbu packet, select members of the rbu packet's header are written to an SMRAM global data area for use in later reconstruction calculations. The listing below shows this initial packet parsing.

```
mov     eax, [eax] ;eax=rbu pkt
...
movzx   ecx, word ptr [eax+8]
shl     ecx, 4
mov     ds:gHdrSize, ecx
movzx   eax, word ptr [eax+4]
shl     eax, 0Ah
sub     eax, ecx
...
mov     ds:g_pktSizeMinusHdrSize, eax
```

Next, the update routine uses the pktSize and pktNum members of the rbu packet to determine where to write the packet in the reconstruction area. Insufficient sanity checking is done on the pktNum, pktSize and hdrSize members before they are used in the calculations for the inline memcpy parameters below. In fact, a malformed rbu packet header can cause the below memcpy to overwrite SMRAM. If controlled carefully, this can lead to an attacker gaining control of the instruction pointer in the context of the BIOS update routine.

```
xor     edi, edi
mov     di, cx ;di=pktNum
mov     ecx, ds:g_pktSizeMinusHdrSize
dec     edi
imul    edi, ecx
add     edi, 101000h
...
mov     edx, ds:gHdrSize
push    esi
shr     edx, 2
lea     esi, [eax+edx*4]
mov     eax, ecx
shr     ecx, 2
rep movsd
```

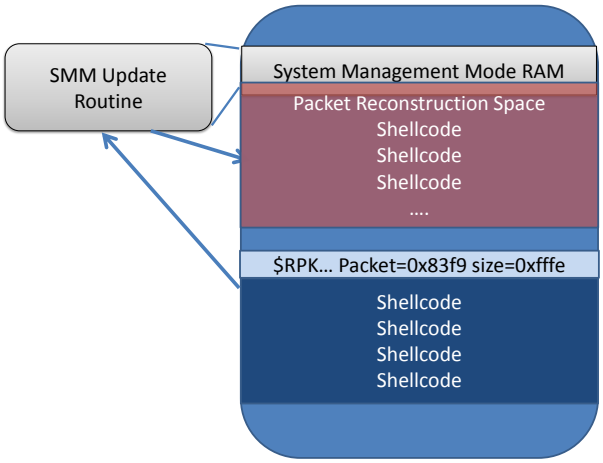

Figure 2: malicious rbu packet causes reconstruction area to overlap with SMRAM

5.2 Exploitation of Dell BIOS Vulnerability

The BIOS update routine executes in the context of SMM which is absent of traditional exploit mitigation technologies such as DEP, stack canaries, ASLR, etc. Because of this the attacker is free to choose any available function pointer for overwriting. However, the attacker must carefully choose how to control the overflow. Overwriting very large amounts of the address space in this super priviledged mode of execution can be problematic. If the attacker overwrites too much, or overwrites the wrong region of code, the system will hang before

he has control of the instruction pointer. In our proof of concept, we chose to overwrite the return address for the update routine itself. We used a brute force search to derive a malicious rbu packet header that would allow us to overwrite this return address without corrupting anything else that would cause the system to hang before the overwritten return address was used. Our brute force search yielded an rbu packet with a pktSize of 0xfffe and a pktNum of 0x83f9, which we verified would successfully exploit the vulnerability.

To store the shellcode we abused the same RAM persistence property of a soft-reboot that is used by the BIOS update process itself. We used a Windows kernel driver to allocate a contiguous portion of the physical address space in which to store both our malicious rbu packet and our shellcode. After poisoning the physical address space with our shellcode and rbu packet, a soft reboot of the system successfully exploits the vulnerability. Our proof of concept shellcode simply writes a message to the screen, but a weaponized payload would be able to reflash the BIOS with a malicious image.

5.3 Dell BIOS Vulnerability Conclusion

The vulnerability described above was discovered on a Dell Latitude E6400 running BIOS revision A29. After coordinating with Dell, the vulnerability was found to effect 22 other Dell systems. This vulnerability has been assigned CVE number CVE-2013-3582[2]. After working with Dell, the vulnerability was patched at revision A34 of the E6400 BIOS.

We can assume that a significant amount of BIOS update code on consumer systems was developed before signed BIOS enforcement became popular. Because of this, it is likely that the code for updating BIOS in a secure manner relies on legacy code that was developed during a time when security of the BIOS was not a high priority. Furthermore, BIOS code is generally propietary and has seen little peer review. Because of these reasons, we suspect that more vulnerabilities like the one presented here are lurking and waiting to be discovered in other vendor's firmware.

6 Attacking Intel Protection Mechanisms

As noted in section 3.3, a majority of the systems we have surveyed opt to rely exclusively on the BIOS_CNTL protection bits to prevent malicious writes to the BIOS. This decision *entangles the security of the BIOS with the security of SMM*. Any vulnerabilities that can be exploited to gain access to SMM can now be leveraged into an arbitrary reflash of the BIOS. To better illustrate this point, we will revisit an old vulnerability.

6.1 Cache Poisoning

In 2009 Duflot et al. and Invisible Things Lab discovered an Intel CPU cache poisoning attack that allowed them to temporarily inject code into SMRAM[8][10]. This attack was originally depicted as a temporary arbitrary code injection in SMRAM that would not persist past a platform reset. However, on the majority of systems that do not employ Protected Range registers, this vulnerability can be used to achieve an arbitrary reflash of the BIOS. Furthermore, because the BIOS is responsible for instantiating SMM, the cache poisoning attack then allows a *permenant* presence in SMM.

The aforemention cache poisoning attack worked by programming the CPU Memory Type Range Register (MTRR) to configure the region of memory containing SMRAM to be Write Back cacheable. Once set to this cache policy, an attacker could pollute cache lines corresponding to SMM code and then immediately generate an SMI. The CPU would then begin executing in SMM and would consume the polluted cache lines instead of fetching the legitimate SMM code from memory. The end result being arbitrary code execution in the context of SMM.

On vulnerable systems, it is straight forward to use this attack to prevent the SMM routine responsible for protecting the BWE bit on the BIOS_CNTL register from running. Once the cache line for this SMM routine is polluted, an attacker can then set the BWE bit and it will stick. Malicious writes can then be made to the BIOS.

We have verified this attack to work against a Dell Latitude D630 running the latest available BIOS revision[3] with signed BIOS enforcement enabled. This particular attack has been largely mitigated by the introduction of SMM Range Registers which, when properly configured, prevent an attacker from arbitrarily changing the cache policy of SMRAM. The particular instantiation of this attack that allows arbitrary BIOS writes was reported to CERT and given tracking number VU#255726. The affected vendors do not plan to release patches for their vulnerable systems due to ending support for BIOS updates on these older systems.

6.2 Other SMM Attacks

Despite the cache poisoning attack being patched on modern systems, the important point is that many signed BIOS enforcement implementations are weakened by failing to implement Protected Range registers and instead relying exclusively on the integrity of SMM for protection. There is a history of SMM break ins including some theoretical proposals by Duflot et al.[8] and another unique attack by Invisible Things Lab[9]. There is

[3] A17 at time of writing

reason to expect this trend to continue.

A cursory analysis of the EFI modules contained in a Dell Latitude E6430 firmware volumes running the latest firmware revision[4] reveals 495 individual EFI modules. 144 of these modules contain the "smm" substring and so presumably contribute at least some code to run in SMM. Despite being of critical importance to the security of the system, the SMM code base on new systems does not appear to be shrinking. This is a disturbing trend. An exploitable vulnerability in any one of these SMM EFI modules could potentially lead to an arbitrary BIOS reflash situation.[5]

We found have another vulnerability that exploits this SMM BIOS_CNTL entanglement and allows for arbitrary BIOS reflashes. This vulnerability affects many new UEFI systems that enforce signed BIOS update by default. This vulnerability has been reported to CERT and been assigned tracking number VU #291102. Because we are still working to contact effected vendors and help them mitigate the vulnerability, we have chosen not to disclose the details of the vulnerability at this time.

7 Conclusion

Signed BIOS enforcement is an important access control that is necessary to prevent malicious actors from gaining a foothold on the platform firmware. Unfortunately, the history of computer security has provided us with many examples of access controls failing. BIOS access controls such as signed firmware updates are no different. Implementing a secure firmware update routine is a complicated software engineering problem that provides plenty of opportunities for missteps. The code that parses the incoming BIOS update must be developed without introducing bugs; a challenge that remains elusive for software developers even today. The platform firmware, including any update routines, are programmed in type unsafe languages.[6] The update code is usually proprietary, complicated and difficult to find and debug as a result of the environment it runs in. This combination of properties makes it highly probable that exploitable vulnerabilities exist in firmware update routines, as we have shown in the case of Dell BIOS.

The SPI flash protection mechanisms that Intel provides to guard the BIOS are complicated and overlapping. A preliminary survey of systems in our enterprise environment reveals that many vendors opt to rely exclusively on the BIOS_CNTL protection of the BIOS. This decision has greatly expanded the attack surface against the BIOS, to include all of the vulnerabilities that SMM may contain. This problem is compounded by an increasingly large SMM code base, a trend present even on new UEFI systems. In our opinion, OEMs should start configuring the Protected Range registers to protect their SPI flash chips as we believe this to be more robust protection than BIOS_CNTL.

As with other facets of computer security, signed BIOS enforcement is a step in the right direction. However, we must continually work to refine the strength of this access control as new weaknesses are presented. Our hope is that the results presented in this paper will contribute towards incremental improvements in vendor BIOS protection, that will ultimately lead to signed BIOS enforcement being a robust and reliable protection.

[4] A12 at time of writing

[5] The Dell Latitude E6430 also fails to implement Protected Range registers.

[6] generally C or handcoded assembly

References

[1] Copernicus: Question your assumptions about bios security. http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about. Accessed: 10/01/2013.

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

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

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

[5] Y. Bulygin, A. Furtak, and O. Bazhaniuk. A tale of one software bypass of windows 8 secure boot. In *BlackHat*, Las Vegas, USA, 2013.

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

[7] Intel Corporation. Intel I/O Controller Hub 9 (ICH9) Family Datasheet. http://www.intel.com/content/www/us/en/io/io-controller-hub-9-datasheet.html. Accessed: 10/01/2013.

[8] Loc Duflot, Olivier Levillain, Benjamin Morin, and Olivier Grumelard. Getting into the smram: Smm reloaded. Presented at CanSec West 2009, http://www.ssi.gouv.fr/IMG/pdf/Cansec_final.pdf. Accessed: 02/01/2011.

[9] R. Wojtczuk and A. Tereshkin. Attacking Intel BIOS. In *BlackHat*, Las Vegas, USA, 2009.

[10] Rafal Wojtczuk and Joanna Rutkowska. Attacking smm memory via intel cpu cache poisoning. http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf. Accessed: 02/01/2011.