SENTER Sandman: Using Intel TXT to Attack BIOSes

Xeno Kovah *Corey Kallenberg* *John Butterworth*
Sam Cornwell
`xkovah@mitre.org @xenokovah`
`ckallenberg@mitre.org @coreykal`
`jbutterworth@mitre.org @jwbutterworth3`
`scornwell@mitre.org @ssc0rnwell`
The MITRE Corporation

Abstract

A comparatively large number of security problems at the PC BIOS level have been found in the last 2 years relative to the preceding years [7]. In this paper we will discuss the interplay between the behavior of Intel Trusted Execution Technology (TXT) and existing attacks, and how this leads to Intel TXT being useful for an attacker. We will also discuss how Intel TXT behavior has changed in "newer CPUs" (not defined by the Intel documentation), and how that removes the possibility of these initial attacks, while simultaneously removing the trustworthiness of technologies like Copernicus 2[8] for combatting attacks like the "Smite'em" SMM MitM[9]. This leads to a situation where if TXT behaves in the old way, a system is vulnerable to certain attacks, and if it behaves in the new way, it's vulnerable to others.

1 Introduction

The Basic Input/Output System (BIOS) is the first code external to the CPU that is executed upon startup on Intel x86-based PCs. On modern systems it is stored in a non-volatile flash memory chip accessible via the Serial Peripheral Interface (SPI) bus, which is in turn accessible via the Platform Control Hub (PCH) or I/O Control Hub (ICH). The CPU makes a request for address 0xFFFFFFF0, which based on the reset state of hardware on the system, is directed to the SPI chip.

Because the code from the BIOS executes first on the system, it is necessarily the most powerful code. It is responsible for configuring hardware, and loading subsequent software. But as has been seen with typical Master Boot Record-based boot kits, subverted boot code can compromise and downgrade the security of code that runs after it. Beyond this, the BIOS instantiates System Management RAM (SMRAM), which is the special protect memory region where code executes from when the CPU transitions to System Management Mode (SMM). SMM was originally intended to provide a location with confidentiality and integrity from which platform vendors could run hardware management and other code. However SMM execution is an attractive target for attackers because it is functionally the most privileged execution domain on the CPU. This is because SMM code cannot be read (or integrity checked) once it is locked down; but when it is running it can read and write all memory for all other code (e.g. hypervisors, OSes, and applications.)

With this in mind, security of the BIOS is paramount to the security of the overall system. In this paper we discuss a novel way to write into the BIOS (e.g. to brick it or backdoor it) when it would otherwise be deemed properly secured.

2 Related Work

A recent defensive talk, "Copernicus 2: SENTER the Dragon"[9], and a recent vulnerability talk, "Setup for Failure: Defeating SecureBoot"[6] are highly related to the vulnerability that is discussed in this paper. And specifically it is the "Charizard" portion, which had been tacked on to an existing short presentation, beginning on slide 49, which is the most relevant. As whitepapers do not exist for either of these talks, the relevant portions will be re-described here.

2.1 Copernicus 2

Copernicus 2 is an improvement upon MITRE's Copernicus[1] tool which was released in 2013. It was understood at the time of release that Copernicus is a "best effort" system for performing BIOS vulnerability & integrity checking. This is because as a simple Windows kernel driver, it is vulnerable to attack via all the typical rootkit-type means such as hooking the OS filesystem writing routines, hooking it's IO via a VMM-based rootkit, or simply targeting it directly. Just like all other security software, with the exception of our previous Checkmate work[10], it does not provide adequate self-protection against attackers at the same privilege level (ring 0).

But as if typical kernel-mode subversion was not enough, in [9] a new generic attack against all software-based SPI firmware capture mechanisms was shown. Two variants of such attacks were shown, relying on either the FSMIE (Flash System Management Interrupt Enable) bit being set, or on the attacker racing the defender. The FSMIE bit based method was particularly subversive (at least against software that didn't know about it), because it allowed an attacker to reside in SMM, where defenders can not directly inspect, and Man in the Middle (MitM) the defender's SPI flash acquisition attempts. This SMM MitM attacker was named "Smite'em".

In the presence of an attacker like Smite'em, it's desirable to significantly strengthen the defensive software through the application of trusted computing technology. To this end, the improved Copernicus 2 made use of Intel TXT. More specifically, it was built as a Piece of Application Logic (PAL) for the open source Flicker project[11][1]. In Intel terminology, the code which is launched by TXT in a trustworthy way is called the Measured Launch Environment (MLE), and it is started with the SENTER instruction. Flicker builds upon the MLE by essentially creating a framework by which a basic piece of an application (but probably not the entire thing), that needs to be trustworthy, can effectively be statically compiled into a standalone PAL for execution within the MLE. PALs must be fully self-sufficient, as they can not expect that they have OS services to rely on, as any other code outside of the PAL should be treated as untrustworthy.

Porting Copernicus to a Flicker PAL was relatively straightforward as it is the type of application which mostly uses direct assembly-based access to hardware registers, rather than highly relying on API calls. This is a property that it shares with the Sandman attacker. Ultimately, while Flicker provides a robust measurement & attestation starting point, the core reason for starting to use TXT was actually because of a side-effect. In the Intel TXT Software Development Guide[2] it states, "The ILP [Initial Launch Processor] must re-enable SMIs that were disabled as part of the SENTER process; most systems will not function properly if SMIs are disabled for any length of time." This property, that SMIs would be suppressed on entry to the MLE, would theoretically mean that any Smite'em-like attacker would be functionally suppressed for the duration of the Copernicus 2 runtime, as it need not re-enable SMIs until it was done running. This is the core property that Copernicus 2, and Sandman, take advantage of.

[1]http://flickertcb.sf.net. Code was contributed back by the authors to fix support for Windows 7 32 bit so that it would properly handle Physical Address Extensions-based paging, the default paging type.

2.2 SMI Suppression Attacks

Both the attack code-named "The Sicilian" [5] (after the chess opening) and "Charizard" [6] (after the pokemon :P) showed ways to defeat one BIOS protection mechanism, if System Management Interrupts (SMIs) could be prevented from executing their intended handler. But they go about achieving SMI suppression in two very different ways. The Sicilian utilizes SMM cache poisoning [**?**][13] in order to insert a RSM (resume) instruction at the SMM entry point. This makes SMM return immediately, doing nothing. The insertion can be removed once the attacker has written to the BIOS. Charizard, on the other hand, finds a SMI enable bit for the ICH/PCH, and temporarily disables SMIs generated by those pieces of hardware.

The value from disabling SMIs is as follows. One of the oldest and most frequently used protection mechanisms on the BIOS is the interaction between the BIOS_CNTL.BIOSWE (BIOS Write Enable) and BIOS_CNTL.BLE (BIOS Lock Enable) bits found via the LPC interface of the ICH/PCH. When BLE is set to 1, if BIOSWE is set to 1, it generates an SMI. The SMI causes the CPU to stop whatever code is currently running, and transition to SMM. The SMI handler is responsible for determining whether the BIOS is currently supposed to be writable (e.g. it may have a policy to only allow setting BIOSWE to 1 in the event that it's initiated by the SMI handler itself.) If the SMI handler determines that BIOSWE should not be allowed to be set to 1, then it resets the value to 0, and exits SMM. From the perspective of the interrupted software, reading back the value of BIOSWE after having set it, it would look like the write of BIOSWE to be 1 had never occurred at all.

Therefore the attacker's approach in both [5] and [6] was to suppress SMIs, write BIOSWE = 1, and then write directly to the flash chip via the SPI programming registers found in the ICH/PCH.

3 SENTER Sandman

As should be obvious when presented in this step-wise fashion, the Sandman attacker described in this paper is a new 3rd way to perform SMI suppression style attacks. It is possible to write *malicious* Flicker PAL code that will execute within the context of the MLE. With SMIs enabled upon entry to the MLE, the Sandman attacker can set BIOSWE = 1, and proceed to write to the BIOS to brick or backdoor it.

4 Caveats

4.1 Differing behavior for "newer CPUs"

There is another important line of text that was missed by the authors of Copernicus 2 shortly after the motivating "The ILP must re-enable SMIs that were disabled as part of the SENTER process; most systems will not function properly if SMIs are disabled for any length of time." That line of text is "Newer CPUs may automatically enable SMIs on entry to the MLE;" This is a *pivotal* line. If SMIs are not suppressed upon MLE entry, then Copernicus 2 is not trustworthy, and Sandman is not a valid attacker. It turns out we had only one test system where Flicker Win 7 32 bit would function AND where the BIOS was protected primarily by BIOS_CNTL.BLE[2]. On this system, a "1st generation" Core i5-540M, we could see that indeed, when the Sandman PoC PAL set BIOSWE = 1 and read it back and printed it, the value was reported as 0. This indicates that on such systems, SMIs are not suppressed on entry to the MLE.

An attempt to collect a list, based on the documentation, of which processors exhibited this behavior proved fruitless. The furthest we got in our search was to determine that CPUs which support "dual monitor treatment" (which can be tested by checking for bit 49 = 1 in the IA32_VMX_BASIC MSR) should also be suppressing SMIs. While we can of course test this on all systems that we have access to, it would be nice if a list existed somewhere. Private discussions with Intel employees suggested that this will likely be supported on all "Nehalem" microarchitecture (released Q3 2009) and newer CPUs.

Importantly the TXT developer guide says that SMIs "*may* automatically enable SMIs". But it does not say under what condition they may still be suppressed. This begs the question of whether there are conditions that the Sandman attacker can control which will allow him to continue to be relevant on such systems. Here again, we went looking in the documentation. While the public documentation does not contain any information, when we saw that the support was tied to "dual monitor treatment", we went looking for private docs. This is because "dual monitor treatment" is another name for the Intel SMM Transfer Monitor (STM). The STM is effectively a second virtual machine monitor (hence, "dual monitor") which is meant to virtualize and jail the SMM code. This is Intel's solution for the problem of SMM code run amuck, capable of subverting normal VMMs. If the system supports it, when the MLE is launched, it can indicate that it supports an STM. This is public information documented both in the Intel TXT developer guide, and the normal Intel software developer guide Volume 3[4].

What's not public is the exact logic whereby the decision is made to suppress SMIs or not. Intel has a STM Developers Guide which is still not public, more than 5 and a half years after the last time you might have heard it mentioned, by ITL in their attack against TXT[12]. However the guide is available to those with an NDA with Intel (search for Intel document ID 415068). In that guide there is a helpful picture that describes the logic. Intel has not granted us permission to excerpt and reproduce only that table. However, we can say that there exist paths by which the Sandman attack, and Copernicus 2 can retain relevance. Unfortunately they are not the default path, as exhibited by the test on our own system.

[2]An HP Elitebook 2540p at revision F09. Revision F22 added additional protections beyond BLE.

5 Defense

In this section we examine how we can defend against the Sandman attack.

5.1 BIOS Access Controls

As BIOS programmers have been hearing from Intel, us, and others, it is imperative that they properly set the access control on the BIOS. Unfortunately what exactly is "proper" access control on the BIOS depends both on the architectural design decisions that have been made, as well as the moving target of what's currently known for BIOS vulnerabilities. E.g. while it would be tempting to think that vendors need only set SMI_LOCK, in reality we have already found and disclosed an *architectural* flaw in the protection of the BIOS_CNTL.BLE bit. This will be described in a future publication.

Therefore we will re-state in this publication what we have stated in past [5] publications [6]: BIOS programmers need to make use of Protected Range Registers (PRRs) for any portion of the BIOS flash chip that they do not want being manipulated after the system has booted. A lack of the use of PRRs (e.g. as shown by Copernicus), is a good indication that a system will likely be vulnerable to one or more attacks.

Another access control method that can potentially help protect against SMI suppression attacks is BIOS_CNTL.SMM_BWP. While this should be used where it's supported, we are also aware of mechanisms to bypass this bit as well. However those are implementation flaws, and not architectural flaws. And PRRs are currently the only BIOS access control mechanism for which no architectural bypasses are known (although documentation suggests that Intel AMT may have this ability.)

5.2 TXT Access Controls

Properly configured BIOS access controls are not always available from OEMs. Luckily it is still possible for the end user to configure the system to prevent TXT-born attackers like Sandman, or other unknown future attackers. This is possible through the use of Launch Control Policies (LCPs). LCPs provide a way to allow some whitelisted MLEs to run, while denying everything else. In this way something like Copernicus 2 could be the sole allowed MLE, while blocking any possible future attackers such as Sandman from running.

LCPs are stored in the non-volatile RAM (NVRAM) in the TPM. Modification of NVRAM data requires authorization of the TPM owner, which takes the form of a password. Under the assumption that the TPM is being securely provisioned in order to provide remote attestation capabilities for a tool like Copernicus 2, it is reasonable to expect that the TPM owner authorization could be used without fear of compromise at provisioning time, and then would not need to be exposed to future LCP-restricted attackers thereafter.

6 Conclusion

In this paper we have described a novel new attack that could compromise the BIOS of some system which otherwise appear to be fully protected. We call this attacker the Sandman, because it uses SMI suppression to effectively put the SMI handler to sleep, allowing it to bypass the BIOS_CNTL.BLE protection bit, and write to portions of the BIOS not protected by PRRs or other mechanisms.

The SMI suppression behavior of Intel TXT has changed over time. However we would argue that the behavior on the newest CPUs of *not* suppressing SMIs is highly detrimental. While it does negate the Sandman attack, this benefit is far outweighed by the negation of tools like Copernicus 2 in order to obtain trustworthy BIOS measurements. That leaves us in a situation where currently there is *no* trustworthy software way to perform BIOS measurements to check for the presence of attackers on currently shipping systems. We plan to address this in the future by applying our work in Timing-Based Attestation[10][3] to Copernicus 2, creating Copernicus 3. However we would urge Intel to change the behavior of the SINIT module to retain SMI suppression upon entry to the MLE. It is entirely within their power to fix this behavior on all shipping systems with a simple software fix. We would also urge all Intel customers who want to have trustworthy BIOS measurement capabilities to contact us, so that we can provide you Intel contact information where you can also convey your desire for Intel to provide this fix.

7 Acknowledgements

Thanks to an anonymous Intel employee who, having seen the Copernicus 2 presentation, made it clear that we were not properly taking into consideration "newer CPU" TXT behavior, and the damaging security implications that has.

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] Intel TXT software developer's guide. `http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf`. Accessed: 10/01/2014.

[3] J. Butterworth, C. Kallenberg, X. Kovah, and A. Herzog. BIOS chronomancy: Fixing the static core root of trust for measurement. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*.

[4] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer Manual, Vol. 3b, Part 2. `http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-software_developer-vol-3b-part-2-manual.pdf`. Accessed: 11/01/2012.

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

[6] 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.

[7] X. Kovah. Low level PC attack papers timeline. `http://timeglider.com/timeline/5ca2daa6078caaf4`. Accessed: 10/01/2014.

[8] X. Kovah. Playing hide and seek with BIOS implants. `http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/playing-hide-and-seek-with-bios-implants.` Accessed: 10/01/2014.

[9] X. Kovah, J. Butterworth, C. Kallenberg, and S. Cornwell. Copernicus 2: SENTER the dragon. In *CanSecWest*, Vancouver, Canada, 2013.

[10] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth. New results for timing-based attestation. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*.

[11] J. M. McCune, B. Parno, A. Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proceedings of the ACM European Conference in Computer Systems (EuroSys)*, April 2008.

[12] R. Wojtczuk and J. Rutkowska. Attacking Intel TXT. In *BlackHat Federal*, Washington D.C., USA, 2009.

[13] 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.