Attacking Intel® Trusted Execution Technology

Rafal Wojtczuk
rafal@invisiblethingslab.com

Joanna Rutkowska
joanna@invisiblethingslab.com

---===[Invisible Things Lab]===---

Abstract

In this paper we present the results of our research into security of the Intel® Trusted Execution Technology, part of the vPro™ brand. We describe a practical attack that is capable of bypassing the TXT's trusted boot process, a key building block for Intel's vision of Trusted Computing. As part of the attack we also discuss practical attacks on SMM memory in modern Intel systems.

keywords: Trusted Computing, Trusted Execution Technology, System Management Mode, TXT, SMM, STM, BIOS, security, analysis, attacks.

1. Introduction

Trusted Computing is becoming a part of our lives, whether we want it or not. These days almost every new laptop comes with an on-board Trusted Platform Module (TPM). Some of the Microsoft's Palladium technologies made their way into Vista, and Microsoft BitLocker is, without doubt, the most successful, widely deployed product that is based on the idea of Trusted Computing[8].

On the hardware side, besides the famed TPM, we also have had the LaGrande technology. LaGrande, recently renamed Trusted Execution Technology (TXT)[4], is Intel's response to the Trusted Computing trend. TXT is currently part of the vPro™ brand[5], and for about a year now users can buy a vPro/TXT compatible hardware in regular computer stores (the first one was the DQ35J desktop board[7] that worked with certain Core 2 Duo processors[1]).

TXT is not an alternative to a TPM, in fact TXT heavily relies on the TPM to provide basic services like e.g. secure storage of measurements done by the TXT. Also, Palladium, or whatever it is called these days, is not a competition to TXT. Intel TXT can provide building blocks to e.g. Vista Bitlocker, arguably making it more secure than it is now.

The sole purpose of Intel TXT technology is to provide a trusted way for loading and executing system software, e.g. Operating System kernel or Virtualization Machine Monitor (VMM). This is achieved by performing software measurements and storing them in particular TPM registers. What is extraordinary here is that TXT doesn't make any assumptions about the state of the system before loading the software, thus making it possible for a user to ensure secure load of an OS or VMM, even in a potentially compromised machine.

In other words, our system can be full of boot sector viruses and BIOS rootkits, and god-knows-what-else, and still TXT should allow to load a clean VMM (or OS kernel) in a secure way, immune to all those malware present in the system. This TXT-supported load process is called Late Launch, and is implemented via a special new CPU instruction called SENTER. A good introduction to Intel TXT architecture can be found in the David Grawrock's book[3]. For a detailed technical specification one should consult the Intel TXT documentation[4].

We shall stress that TXT has not been designed to provide *runtime protection*, e.g. against a buffer overflow in a hypervisor code. TXT is supposed to provide only the *launch-time* protection, i.e. ensure that the code we load, at the moment of loading, is what we really intended to load.

It's worth mentioning AMD has its own version of a late launch, implemented via an SKINIT instruction. We haven't looked at the AMD technology thoroughly yet, so we will refrain from commenting on this any further.

The late launch is a pretty amazing technology, when we think about. It promises to effectively

1TXT requires support from both the CPU and the chipset

provide all the benefits of a computer restart without actually restarting it.

It is hard to overemphasize the potential impact that a technology such as TXT could have on computer security. One can immediately see it could provide basic building blocks that could be later used to eliminate all the *system-level* persistent malware[2] — in other words we should be able to easily build systems (VMMs or even standard OSes) that would be immune to attacks that try to compromise system binaries on disk, or attack the system right from the bootloader or BIOS. Combining this with VT-x and VT-d technologies, system developers (for the first time, at least as far as the "PC" platform is considered) have gotten extremely strong tools into their hands that should allow them to create really secure VMMs and OSes…

Let us now describe how we have attacked this new exciting technology…

2. Attacking Intel TXT

TXT's key functionality, the late launch, is often "advertised" as a way to load and start a piece of trusted code (usually a VMM), no matter what is the state of the system, at the moment just before performing the launch. That is, however, not fully correct. In fact there is one piece of system software that should be trusted… This system software is a so called System Management Mode, which we discuss in more detail in the next chapter.

SMM, being the most privileged type of software that ever executes on a CPU (see below), can bypass security protections imposed by the late launch process on a newly loaded VMM. Unfortunately, the assumption that SMM can be always trusted is incorrect, as we demonstrate here with this research.

Indeed, one can imagine the following 2-stage attack on the Intel TXT's late launch functionality:

1. Infecting the system's SMM handler,
2. Compromising the just-securely-loaded code (in our example the Xen hypervisor) from within the infected SMM handler.

We have implemented a proof-of-concept code that demonstrates the above attack scheme against the Xen hypervisor loaded using tboot[6], the Intel's open source implementation of trusted boot. Tboot provides trusted boot for Linux and Xen using Intel TXT's late launch functionality. Tboot is also part of the mainstream opensource Xen hypervisor[12].

Normally tboot should ensure that when a correct, i.e. unmodified, Xen hypervisor is loaded, and only then, correct measurements will be loaded into TPM registers. In practice this means that only a chosen (*trusted*) version of the Xen hypervisor will get access to certain secrets sealed in the TPM, and/or will be able to positively authenticate itself to some peer (e.g. system administrator's laptop) using a special feature of a TPM called Remote Attestation.

With our attack we show that by infecting an SMM handler, we can modify at will the just-loaded (and just-measured) VMM. In other words the attacks completely bypass all the security functionality that is supposed to be provided by the TXT for the purpose of trusted boot. This is a direct result of the SMM code surviving the late launch process in an unmodified form — whether compromised or not.

We describe the details of how to attack an SMM handler in the next chapter.

Intel's remedy to malicious SMM handler is called STM, which stands for SMM Transfer Monitor. The purpose of STM is to sandbox the existing SMM handler by virtualizing it using VT-x and VT-d technologies. STM should be thought of as of a peer hypervisor to the VMM that is being loaded using late launch. STM is supposed to be measured during the late launch process.

Unfortunately STM is, as of today, not available. We discuss this in more detail in the last chapter of this paper.

3. SMM Attacks

SMM aka "Ring -2"

System Management Mode is the most privileged execution mode on x86/x86_64 architectures, even more privileged than ring 0 mode and a hardware hypervisor (VT/AMD-v), often referred to as "ring -1".

One reason for considering this code to be so privileged, is that an SMM code can access the whole system memory, including the kernel and hypervisor memory. Standard OS memory protection mechanisms (Page Tables), as well as hypervisor memory virtualization (Shadow Paging, Nested Paging/EPT, IOMMU/VT-d), do not work against the SMM code.

2 By a system-level malware we mean malware which is resident because of replacing/infecting crucial,privileged, nonvolatile code (e.g. a kernel binary, dll/sys binary, or boot sector, or bios image)

Also, on Intel processors, an SMM code can "preempt" even the VT-x hypervisor, whenever an SMI interrupt is signaled. There exists a way[3] for the hypervisor to establish a special entity, called STM, that would be able to intercept those SMI interrupts, and we will discuss it later in this paper.

Consequently, it will be not much of an exaggeration to name SMM a "ring -2".

Difficulties with finding flaws in SMM

A researcher willing to examine an SMM code for potential security problems faces a non-trivial problem — the SMM memory is not accessible to anyone except… the SMM itself. Thus there is no way for the attacker to get access to the actual image of the SMM code. Without having access to the (binary) code, one cannot, obviously, look for potential flaws there. So, we came to a, somewhat discouraging, conclusion that:

Without having at least one bug in an SMM code that could be exploited to read the SMM memory, one cannot... search for bugs in the SMM memory!

That is indeed a nice example of a vicious circle and partly explains why so little research has been done in the area of SMM attacks (see the discussion later).

One might be tempted to think that an easy short-cut to read the SMM code (and the whole BIOS), would be to de-solder the SPI-flash chip from the motherboard and read its memory using a special, so called, *programmer* device. Unfortunately this approach is not that straightforward as it might seem. It turns out that BIOS code, as seen on the flash chip, is usually heavily packed with custom packing algorithms and its unpacking presents as similar challenge as e.g. unpacking/deobfuscation of modern malware, with the difference that code emulation is extremely difficult to implement in the process of unpacking the BIOS code. After all a BIOS code is supposed to be executed in a very unusual I/O environment, which is hard to emulate properly.

Consequently we have taken a different approach, that is described later. Before we proceed, let's take a quick look at the research done by other authors that involves SMM and security.

Previous SMM-related research

Although there have been several publications, in the recent years, concerning the SMM-related security issues ([2], [10], [1]), they were all concerned about the security implications resulting from an attacker gaining access to the SMM memory, without however focusing on how to bypass system SMM memory protection. That was mostly because until recent years, it was straightforward to gain a read-write access to the SMM memory by only setting appropriate chipset registers, so no special attack method was needed.

Today most modern systems take special steps in order to protect the SMM memory from even a read-only access from the OS, including the kernel and/or the hypervisor.

Example #1: The Q35 remapping bug

During one of our presentations on Xen security at the Black Hat conference in August 2008, we have mentioned a bug we found in a DQ35JO motherboard[4] BIOS that allowed us to e.g. bypass the Xen hypervisor memory protection. A few weeks later Intel has released an updated BIOS that fixed the bug we exploited, and we have published full details about the attack[9], together with a proof-of-concept code[11].

The attack exploits the Intel chipset's memory remapping (AKA memory reclaiming) feature and allows to circumvent certain CPU- or chipset-imposed memory protection mechanisms. This includes ability to bypass an SMM memory protection, allowing the attacker to gain full access to the SMM memory.

This bug, and a resulting attack, turned out to be crucial with our further SMM research. It allowed us to analyze the SMM binary code and find further security problems there, which we describe below.

Example #2: VU#127284

On December 10th, 2008, we have reported the discovery of a few new SMM bugs to Intel Product Security Response Center. All those SMM bugs result from a single design decision to implement certain functionality in an unsafe way. This single design decision has lead to some 40+ places in the SMM handler[5], where each might potentially introduce code execution vulnerability in SMM

3 It is called "dual-monitor mode" by the Intel System Developers Manual.

4 Back then it was the most modern desktop motherboard from Intel available in shops.

5 We have looked at the DQ35JOE's motherboard's BIOS handler, with all the latest patches as of December 2008.

mode. We have successfully exploited only two of those implementation errors. We have seen no point in trying to exploit others. The correct solution to this problem should be based on redesigning the current SMM handlers. One should not confuse this design mistake in the SMM handler with the design problem affecting the security of TXT, that we discuss in the next chapter.

Those recent SMM bugs have still not been patched by Intel. Intel confirmed[6] the issue in "mobile, desktop, and server motherboards", without providing any more details about which exact models are vulnerable. We suspect it might affect all recent Intel motherboards/BIOSes.

Intel estimates the firmware patches to be ready before summer 2009. Intel requested that we withhold the details about the SMM bugs until the patches are available. We currently planning to disclose them at the Black Hat USA 2009 conference, that takes place at the end of July 2009.

Intel told us that they have also notified CERT CC about this problem, because they believed similar SMM bugs might be present in other vendors' BIOSes. CERT CC has assigned the following tracking number to this issue: VU#127284.

Summary of the SMM issues

With our two distinct attacks on the recent SMM handlers, we have shown that even the latest systems don't correctly protect its most privileged software layer, i.e. the System Management Mode. In some aspects, an SMM code is even more privileged than the hardware hypervisor, because it has access to the whole system memory, including the hypervisor memory, but not vice versa.

We should note, however, that our SMM attacks still do not allow to e.g. re-flash the BIOS. Today, most BIOSes are well protected against re-flashing with unsigned images. Also our attacks do not affect the Intel AMT technology, which is independent from the main processor and does not rely on SMM security.

4. The TXT design problem

We have mentioned above that a remedy to malicious SMM handlers is called an STM. We also said that no STM, as of today, is unfortunately available on the market, which yields our attack applicable to all current systems. One aim of our research, besides having fun and all, is to stimulate developers to create an STM.

So, here comes the first question about STM: who should be in charge of creating an STM? We have been told by Intel engineers that STM should be created by OEMs, or BIOS vendors. Apparently STM should be part of a system BIOS, just like an SMM is.

This rises a second question though: if we do not trust OEMs to produce flawless SMM code(s), why should we trust them to create a flawless STM? After all, STM *seems* to us as something much more complex than a typical SMM — STM is a hypervisor, a non-trivial hypervisor that should provide memory, I/O and CPU virtualization to the SMM handler. Moreover STM should work *in parallel* with an existing VMM(s), like e.g. Xen. There should be a way for the two hypervisors to talk to each other, which, in turn, should require a standardized inter-hypervisor protocol. Needless to say, Intel does not provide any publicly available documentation on how to write a working STM[7].

Intel claims[8], however, that STM is not that difficult to write and that Intel will provide detailed specification on how to write one in the near future[9]. Intel argues that STM can be made more secure than a typical SMM because it might not need to be modified as often as SMM handlers are. SMM needs to be "tuned" to each new motherboard/system, while an STM should be fairly generic. This could allow to have just a few mature (and well audited) STMs in existence.

The third question comes to mind however: if indeed STMs were so easy to write, why would Intel still hadn't created one? After all, TXT-capable hardware has been available in shops for over a year now, and also the Intel's tboot project is more than a year old.

---

6 Private communication between ITL (we) and Intel.

7 In fact the term STM is not defined in any of the Intel documents available on intel.com website. The term and its purpose is only introduced in the, already mentioned, David Grawrock's book[3].

8 Private communication between ITL (we) and Intel.

9 Intel claims the specification is already available for select vendors under NDA.

Intel's counter-argument[10] is that there has not been enough "market demand" for an STM as of yet, and consequently OEMs have not been interested in developing an STM.

We cannot agree with this counter-argument, however. First, we shall not forget that Intel is also… an OEM/BIOS vendor — it does sell motherboards and does make BIOSes for them. Second, when it comes to security, one should not use "market demand" as an excuse. If we followed this line of reasoning, we might very well never started using the `snprintf()` function;)

Consequently, we should ask if that was indeed a good idea to design TXT in such a way that it requires additional, probably quite complex, entity called STM to function correctly? Maybe it was a mistake to allow TXT to function without STM? After all the whole point about TXT was to get rid of the Static Root of Trust Measurement, in favor of the Dynamic Trust Measurement. Assuming even that STM itself will not contain any flaws — it is unclear to us whether integration of STM and TXT can be, and will be, done securely; we cannot say more on the topic until we can evaluate an actual STM implementation.

5. Summary

We have described a successful attack against Intel Trusted Execution technology. We have also implemented a working proof-of-concept exploit that works against Xen loaded via tboot — Intel's implementation of TXT-based trusted boot.

Probably equally interesting as the TXT itself is another aspect of our research — the novel research into SMM attacks. SMM attacks have more implications than just attacking TXT… SMM compromises can also be used for creating advanced rootkits, backdoors and trojans. SMM bugs can also be used to compromise hypervisor memory in virtualized systems, even such advanced as Xen.

The details of our new SMM attacks will be made available once Intel patches its firmware, most likely we will present them at the Black Hat USA conference in summer 2009. We will also make the code of our TXT exploit available.

References

[1] BSDaemon, Coideloko, and D0Nand0N. System Management Mode Hack: Using SMM for "Other Purposes". *In Phrack Magazine*, Vol 0x0c, Issue 0x41, 2008.

[2] Loic Duflot. Security Issues Related to Pentium System Management Mode. Presented at CanSecWest 2006, Vancouver, Canada, 2006.

[3] David Grawrock. The Intel Safer Computing Initiative: Building Blocks for Trusted Computing (Computer System Design). Intel Press, 2006.

[4] Intel Corp. Intel® Trusted Execution Technology. http://www.intel.com/technology/security/,

[5] Intel Corp. Intel® vPro™ Technology. http://www.intel.com/technology/vpro/index.htm,

[6] Intel Corp. Trusted Boot (tboot). http://sourceforge.net/projects/tboot, 2007.

[7] Intel Corp. Intel® Desktop Board DQ35JO. http://www.intel.com/products/desktop/motherboards/dq35jo/dq35jo-overview.htm, 2008.

[8] Joanna Rutkowska. Why do I miss Microsoft BitLocker? http://theinvisiblethings.blogspot.com/2009/01/why-do-i-miss-microsoft-bitlocker.html, 2009.

[9] Joanna Rutkowska and Rafal Wojtczuk. Detecting & Preventing the Xen Hypervisor Subversions. Presented at Black Hat USA, Las Vegas, NV, USA, 2008.

[10] Sherri Sparks and Shawn Embleton. SMM Rootkits: A New Breed of OS Independent Malware. Presented at Black Hat USA, Las Vegas, NV, USA, 2008.

[11] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin. Xen 0wning Trilogy: code and demos. http://invisiblethingslab.com/resources/bh08/, 2008.

[12] Xen Hypervisor. http://xen.org/

10 Private communication between ITL (we) and Intel.

http://invisiblethingslab.com