Protecting Data In-Use from Firmware and Physical Attacks

Stephen Weis

PrivateCore
Palo Alto, CA

ABSTRACT

Defending computers from unauthorized physical access, malicious hardware devices, or other low-level attacks has proven extremely challenging. The risks from these attacks are exacerbated in cloud-computing environments, where users lack physical control over servers executing their workloads.

This paper reviews several firmware and physical attacks against x86 platforms, including bootkits, ”cold booting”, and malicious devices. We discuss several existing tools and technologies that can mitigate these risk such as Trusted Execution Technology (TXT) and main memory encryption. We will also discuss upcoming technologies that may help protect against firmware and physical threats.

1. INTRODUCTION

In 2013, journalists revealed that the United States National Security Agency's (NSA) Tailored Access Operations unit engaged in low-level attacks targeting platform firmware and utilizing hardware implants [3]. These attacks were detailed in an internal catalog of tools, exploits, and devices referred to as the ANT catalog.

The NSA ANT catalog contained programs with codenames such as DIETYBOUNCE, GOURMETTROUGH, and IRATEMONK targeted the BIOS, system management mode (SMM), and device firmware on platforms from vendors such as Dell, Cisco, Huawei, and Juniper. Other programs, such as IRONCHEF, GINSU, and COTTONMOUTH involved hardware implants integrated in peripheral interfaces or on the PCIe bus. Some of these hardware implants contained two-way radios capable of bridging air-gapped systems.

The types of attacks illustrated by the NSA ANT revelations were not new developments; similar attacks against modern platforms have been publicly discussed among the security community for at least 15 years. However, the NSA ANT revelations do highlight that these attacks are not only being used in practice, but are also low-cost and feasible for even an individual attacker [25].

While low-level attacks are relatively easy to conduct, there are limited defensive technologies available on x86 platforms. The defensive technologies that do exist have not been widely adopted. This paper will review several categories of attacks against x86 systems, discuss the pros and cons of existing defensive technologies, and review several technologies in the pipeline that may benefit platform security.

To appear in BlackHat 2014. Generated June 25, 2014.

2. PHYSICAL ATTACKS

It is generally understood that physical access to an x86 platform can completely compromise software security. Historically, physical security controls such as cages, cameras, and locks have been employed to prevent or detect physical access. Yet with adoption of outsourced infrastructure and cloud computing, x86 platforms are increasingly run outside the physical control of the software owner.

This section briefly summarizes several well-known physical attack vectors against x86 platforms, including DMA and physical memory extraction.

2.1 Direct Memory Access

By design, x86 architectures provide direct memory access (DMA) from hardware subsystems to main memory without invoking the CPU. DMA is generally used to improve performance. For example, DMA allows disk, network, and graphics devices to read and write data directly to memory without incurring CPU cycles.

Yet, without proper controls, devices with DMA may access arbitrary regions of memory. Access to runtime memory may compromise system security by exposing secrets or allowing an attacker to modify running software in place. Secrets from captured memory can be extracted easily with forensics tools like Volatility [69].

Tribble [11, 28] is an early example of a device designed for exfiltrating data via DMA, based on an off-the-shelf Intel evaluation platform. Copilot [57] also used DMA with a PCI device for the purpose of monitoring kernel integrity. The Maux attacks [66, 67] exploited remote vulnerabilities in a standard network interface device and accessed memory via DMA. Off-the-shelf intelligent network adapters, such as those made by Cavium, are able to exfiltrate DMA memory over a network connection [35].

The IEEE 1394 Firewire interface also provides DMA by design. This led to several demonstrations of memory extraction to steal data or for forensics [5, 16, 17, 71]. The Thunderbolt interface actually extends the PCIe bus, providing an easy way to use DMA attack devices through an external interface [50, 59].

2.2 Physical Memory Extraction

While DMA may be mitigated by software-based countermeasures or a hardware I/O memory management unit (IOMMU) such as Intel VT-d [38], such countermeasures do not mitigate physical extraction of system memory. For example, memory bus analyzers can interdict memory traffic. However, off-the-shelf bus analyzers are unwieldy. They must be installed ahead of time, tend to be relatively expensive, and are physically large.

A "cold boot attack" is a low-cost memory extraction attack that involves literally freezing system memory modules with an aerosol freeze spray [30]. The frozen memory contents are preserved long enough to boot to a "scraper" image such as *bios-memimage* or *msramdump* which can copy the memory contents to persistent storage.

Cold booting disrupts a running system and data must be recovered before the memory module thaws. Furthermore, it does not reliably capture all memory contents, as there is some degradation over time. Conducting the attack may be further complicated by error-correcting memory which is cleared on a reset or by data scrambling for power supply noise suppression [52].

Persistent or non-volatile memory (referred to as NV-RAM), designed to persist data after a power loss. NV-RAM is now available in DIMM form-factors used by standard x86 servers. An attacker installing NV-RAM modules in a server may remove them once the server is in use by a victim. Since contents are preserved like a disk, attackers can recover all data in memory without loss at a later time. If a memory mirroring mode is configured, an attacker could remove an NV-RAM module from a running system and replace it without disrupting service.

Persistent memory will likely see increased adoption in the near future as production systems move toward fully in-memory architectures. As memory is essentially used like a disk, there will be an increased risk of physical memory extraction.

3. BOOT INTEGRITY ATTACKS

Modern x86 platforms depend on multiple pieces of firmware to load prior to or during the execution of the operating system. Attackers with either logical or physical access may be able to compromise boot integrity with bootkits or platform malware. We use the term "platform malware" to distinguish from malware functioning in the operating system or hypervisor level. Since platform malware persists outside the operating system, it may re-infect new operating system installations. Platform malware may also run higher levels of privilege and be invisible to an OS, such as within system management mode (SMM).

Attacking boot integrity is well-trod research territory. Firmware dependencies are typically vendor- or hardware-specific, so offer many fragmented targets. Consequently, researchers have discovered numerous attacks against nearly every piece of firmware used in the boot process.

As a brief sample, researchers have found and exploited vulnerabilities in the BIOS and associated data structures [8, 49, 61], UEFI [42], master boot records (MBR) [43], NIC firmware [14, 15, 20, 21, 66, 67], hard drive firmware [74], PCI device option ROMs (OptROMs) [13, 31, 48], keyboard controllers [27], CPU management engines [60, 62, 65] or System Management Controllers (SMC) [40], ACPI [19, 32], and SINIT authenticated code modules [72]. This list is by no means exhaustive and largely represents more recent work.

These many, varied attack vectors illustrate the difficulty of securing an execution environment on x86 platforms. Any piece of code that is executed during the lifetime of a system must be measured, verified, or otherwise isolated to establish trust in a system.

4. DIAGNOSTICS TOOLS

Diagnostics tools are available for assessing BIOS and platform security, or performing forensics on a potentially compromised system. For example, Flashrom [33] is a general purpose tool for reading and modifying a large variety of devices, but does not have any security-specific functionality.

Intel's CHIPSEC [47] provides a set of utilities and modules for conducting firmware forensics and detecting known vulnerabilities. CHIPSEC's modules address issues such as BIOS protection, SMRAM locking, and SMRR configuration.

MITRE's Copernicus [10] provides similar functionality. Copernicus can dump the BIOS of a system to be compared against a known, clean copy. Copernicus also checks the status of the system configuration to determine whether a BIOS can be modified. MITRE's researchers have developed attacks to evade Copernicus, and integrated countermeasures into a new version, Copernicus 2 [44].

5. DEFENDING THE BOOT PROCESS

5.1 Verified Boot

One approach to ensuring that a boot process has not been compromised is to verify a chain of signatures on each component as it is loaded. This chain-of-signatures approach is used for Windows 8 *secure boot* and ChromeOS *verified boot*. We'll use the latter term for convenience.

Verified boot typically involves a root public key that resides in some non-volatile and tamper-resistant component. This key may be referred as a *platform key* and reside in SPI flash, read-only firmware, or a trusted platform module (TPM). For flexibility in updates, the root platform key typically will be used to verify a sub-key, which might be referred to as a "key exchange key", "firmware data key", or "kernel data key". These sub-keys will be used to verify signatures on the actual firmware and kernels which are loaded.

Verified boot does raise the bar against bootkit and platform malware, but as with all software, may itself be susceptible to vulnerabilities [9]. In practice, verified boot may be disabled to allow users to boot to arbitrary operating systems. Attackers with physical access may also replace or modify whatever component the root public keys reside in, allowing them to circumvent the verified boot process.

5.2 Measured Boot and Attestation

Verified boot doesn't provide any mechanism to know what actually booted. Users must trust that the boot process completed as expected, but do not have an independent measurement of what code executed. Providing this mechanism is the core concept behind a *measured boot*.

For x86 platforms, the Trusted Computing Group (TCG) [29] specified the predominant measured boot technology, which employs a trusted platform module (TPM) as an independent auditor. The TPM contains special platform configuration registers (PCRs) which are used to record measurements of firmware and configuration loaded during the boot process. These measurements are intended to measure every piece of firmware or software required to boot an operating system. PCRs may only be updated in specific conditions, so they cannot be arbitrarily overwritten by malicious software.

Following a boot, a remote agent can interrogate the TPM via a challenge-response protocol and recover a signed set of measurements called a *quote*. This process is referred to as *remote attestation*. The remote attestor will verify the TPM's signature on the quote, then evaluate whether the quoted values abide by a known policy or whitelist.

Originally, the TCG specifications relied on a *static root of trust measurements* (SRTM). SRTM relies on a static root of firmware to initially measure other boot modules into TPM PCRs. Besides having to trust that initial firmware, in practice SRTM is inflexible to manage.

Changes like upgrading the BIOS or installing a new device would require updating corresponding policies. For SRTM, another issue is that there was also no authoritative source of the provenance of firmware. Users did not know that a given firmware was "good". They could only accept it as-is and monitor for unexpected changes.

An alternate approach is a *dynamic root of trust measurement* (DRTM), of which Intel Trusted Execution Technology (TXT) [26] is one implementation. The concept behind TXT's implementation of DRTM is that after platform firmware has executed, a special SENTER instruction can bring the system into a known, clean state.

At that point, the operating system and its configuration can be measured into PCRs, then "late launched". The idea is to remove dependencies on the initially loaded firmware, so that only the operating system level software would need to be measured. An advantage is that users can know the provenance of their operating system and can derive the expected measurements on their own.

In theory, the late launched OS should be isolated from the prior executed firmware. In practice, system management mode (SMM) code remains resident after SENTER and can be a target for malware [72]. This makes it necessary to check SRTM measurements of the SMM code, even when using DRTM. That brings the management inflexibility and provenance questions back into scope.

Another issue is that TXT still ultimately relies on software in the form of a signed, authenticated code module (ACM) from Intel called SINIT. This module may contain exploitable flaws that could subvert the measured boot process. At least one buffer overflow attack was demonstrated against SINIT [73], although it has since been fixed.

When it comes to physical attacks, the TPM was not designed to resist physical attackers. It is connected to the CPU on a low pin count (LPC) bus which can be interposed [46, 70]. Attackers could subvert TXT by modifying measurements of malicious firmware with good measurements.

Additionally, the certificates and signing keys from within a TPM can be compromised by a physical attacker [64]. Once the signing keys are exposed, an attacker can emulate a TPM to the remote attestation protocol and spoof measurements.

6. PHYSICAL MEMORY DEFENSES

The risks of exposing plaintext memory highlighted in Section 2.2 are well understood. While there do exist secure processors such as the Dallas Semiconductor DS5002FP that encrypt all data on the memory bus, x86 systems currently do not support full memory encryption. As persistent memory technologies like non-volatile RAM, MRAM, or phase-change memory come to market, this problem will become more serious. Memory is becoming the new disk – and it's in plaintext.

Researchers have proposed numerous architectures with encrypted, authenticated, and oblivious memory models to address the issue [12, 18, 23, 22, 24, 37, 63]. These specific proposals have not been adopted by either x86 or ARM architectures, although there is some initial progress toward hardware-based memory encryption discussed in Section 7.

In the absence of hardware-encrypted memory, there have been several software-based proposals. Perhaps the most well known, TRESOR [53] was designed with cold boot attacks in mind. TRESOR works by using a CPU debug register to store AES key material. Attackers able to obtain memory contents would not recover the actual key. Unfortunately, TRESOR only protects key material and is vulnerable to attacks able to modify memory, such as DMA attacks [4].

Cryptkeeper [56] is another approach intended to minimize plaintext memory. It essentially keeps a small unencrypted portion of memory, while encrypting the rest of memory. The issue of where to store keys is not adequately addressed by Cryptkeeper; the authors allude to using TPMs or exposure-resilient functions, but do not offer a conclusive solution to where the keys will be kept.

The approach taken by FrozenCache [54] is to encrypt sensitive memory and to keep the keys in the CPU cache. To keep the keys resident in cache, the cache is then put into non-evict mode (NEM), also known as Cache-as-RAM mode. Using NEM resulted in significant performance degradation in practice and is not feasible for general purpose use.

CARMA [68] is another approach that keeps keys and trusted code entirely within the CPU cache. However, CARMA is intended for a small trusted computing base that does not support main memory, and thus is not suited for general purpose use.

A Software Cryptoprocessor [36] is an approach that supports full-memory encryption on generic x86 platforms. It is reminiscent of Cryptkeeper and CARMA in that it keeps a small portion of unencrypted memory entirely within the CPU last-level (L3) cache, while keeping main memory fully encrypted. All key material and kernel code remains resident in the L3 cache and is never exposed in memory as plaintext. By using an authenticated mode of encryption like AES-GCM, full-memory encryption can resist replay or substitution attacks that modify memory contents.

7. UPCOMING TECHNOLOGIES

The boot integrity techniques discussed in Section 5 and the physical memory defenses discussed in Section 6 do raise the bar for attackers, but are not complete solutions.

Besides being exposed to SMM code, TXT-based attestation relies on a physical TPM connected to the LPC bus, which are both vulnerable to physical attack. An improvement would be to reduce the trust perimeter to a single component. Fortunately, systems on a chip with TPMs or TPM-like functionality within a single package help reduce the exposure considerably and have started emerging on the market.

As for software full-memory encryption technology, the challenges are primarily around performance. Running a software cryptoprocessor resident within the L3 cache effectively reduces the cache size, while encrypting memory access introduces a new bottleneck on the critical memory path.

While performance is not severely impacted for many applications with small or sequential memory access patterns, applications with large or random-access memory can perform relatively poorly. One positive trend is that the cache overhead is fixed, but cache sizes are growing rapidly, meaning the relative cost is rapidly decreasing.

7.1 Software-based Attestation

Software-based attestation is an alternative approach to using a separate, TPM-like device [10, 41, 45, 55, 58]. A common approach to software-based attestation is to rely on some performance or timing measurements which the presence of malware would negatively impact. The idea is that the only way that a system can achieve the expected performance measurements is if it's running exactly the expected code. This requires designing a metric such that even a minor code change will have a large, remotely measurable impact.

One weakness of software-based attestation is that steps must also be taken to ensure that a device is not being emulated by a faster, more powerful device. That would typically require some hardware-rooted key material to authenticate that a device is legitimate.

7.2 Enhanced Privacy ID

Intel Enhanced Privacy Identification (EPID) [7] is a forthcoming technology that could address the problem of authenticating a device. EPID is a successor to the Direct Anonymous Attestation (DAA) protocol [6] currently supported by TPM 1.2. DAA was designed to support attestation without uniquely identifying a piece of TPM hardware. Previous to DAA, it was expected that a "Privacy Certificate Authority" (Privacy CA) would emerge as a trusted third party to obscure TPM identifying material. In practice, neither privacy CAs or DAA were widely adopted.

EPID offers similar functionality to DAA, except that the device key material resides in the CPU package and not an external TPM. The presence of unique, per-CPU keys opens up the possibility of remotely authenticating that a CPU is legitimate.

Unfortunately, in the first iteration of EPID, the key material will be written at manufacture time and is not user-configurable. This means the manufacturer could retain keys which may later be used to spoof or identify CPUs.

By design, EPID does not expose key material to software. However, one concern is that if there is a flaw in the enforcement mechanism, the existence of unique identifying keys could compromise user privacy. Concerns over this type of functionality arose around the Pentium III processor serial number. Allowing CPU keys to be provisioned post-manufacture, ideally by an end user, would mitigate this concern.

7.3 Software Guard Extensions

Perhaps the most notable security development on the horizon are the Intel Software Guard Extensions (SGX) [2, 34, 39, 51]. At a high level, SGX provides isolated "secure enclaves" that protect small, user-level programs from malware running outside the enclave. This functionality is somewhat similar to ARM TrustZones [1], which provides a "secure world" region of memory that is inaccessible to the "insecure world".

Code loaded into SGX secure enclaves is attested using key material within the CPU; perhaps with the same keys as EPID. This attestation gives a remote user a means to ensure that an enclave loaded with the code they expected. Once loaded, access to secure enclaves from other software is restricted through hardware controls, making it inaccessible even to code running at higher privileges. Furthermore, the enclave itself is backed by hardware-implemented encrypted and authenticated memory.

These features of SGX address several of the issues brought up with today's defenses. Attestation functionality is entirely within the CPU and does not rely on other components. Meanwhile, memory encryption is implemented in hardware, which reduces the performance overhead of encryption.

Unfortunately, SGX is not a full solution and requires rewriting applications to take advantage of new security functionality. In the first generation, enclaves will be limited in size to perhaps 128 megabytes. They also cannot run any privileged instructions, thus would need to depend on an external kernel to make syscalls. This limits what types of work can be performed within an enclave in practice.

What enclaves will be suited for loading a small application, attesting it, establishing a secure transport, and provisioning a secret key. That key would not be exposed to any other code on the system or someone with physical access. From that point, the enclave could offer very similar functionality to a hardware security module; acting as a secure place to perform cryptographic operations.

One of the applications that could easily make use of this functionality is digital rights management (DRM). Encrypted content could be remotely streamed into an enclave, decrypted, then re-encrypted for use by a display technology like Intel Protected Transaction Display.

8. FUTURE WORK

Regardless of the initial uses of SGX, it does hold significant potential to reduce the risk of physical and firmware attacks. By embedding key material within the CPU and providing hardware support for full memory encryption, it can reduce the trusted components to solely the CPU.

Moving forward, several suggested developments could potentially improve the firmware and physical security on x86 platforms:

- Provide a mature SMM transfer monitor (STM) or some other means of isolating the SMM.
- Add a means to support for privileged instructions in an SGX secure enclave.
- Extend support for hardware-based memory encryption, potentially for arbitrary regions of memory.
- Provide fine-grained L3 cache controls to lock lines in the L3 cache or otherwise support cache coloring.
- Provide end users or at least vendors the ability to write their own CPU-specific key material.

9. ACKNOWLEDGEMENTS

Thanks to Xeno Kovah for sharing a detailed low-level attack bibliography.

10. REFERENCES

[1] T. Alves and D. Felton. TrustZone: Integrated hardware and software security. *ARM White Paper*, 3(4), 2004.

[2] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*, June 2013.

[3] J. Appelbaum, J. Horchert, and C. Stöcker. Shopping for Spy Gear: Catalog Advertises NSA Toolbox. *Der Spiegel*, December 2013.

[4] E.-O. Blass and W. Robertson. TRESOR-HUNT: attacking CPU-bound encryption. In *ACSAC '12: Proceedings of the 28th Annual Computer Security Applications Conference*. ACM Request Permissions, Dec. 2012.

[5] A. Boileau. Hit by a Bus: Physical Access Attacks with Firewire. In *RUXCON*, Jan. 2006.

[6] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *ACM Computer and Communications Security*, pages 132–145. ACM, 2004.

[7] E. Brickell and J. Li. Enhanced Privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. *Dependable and Secure Computing, IEEE Transactions on*, 9(3):345–360, 2012.

[8] J. Brossard. Bypassing pre-boot authentication passwords. In *Defcon 16*, 2008.

[9] Y. Bulygin, A. Furtak, and O. Bazhaniuk. A Tale of One Software Bypass of Windows 8 Secure Boot. In *Black Hat USA*, 2013.

[10] J. Butterworth, C. Kallenberg, X. Kovah, and A. Herzog. BIOS chronomancy: Fixing the core root of trust for measurement. In *ACM Computer and Communications Security*, pages 25–36. MITRE, 2013.

[11] B. D. Carrier and J. Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1(1):50–60, 2004.

[12] S. Chhabra and D. Solihin. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *International Symposium on Computer Architecture (ISCA)*, pages 177–188. IEEE, 2011.

[13] P. Chifflier. UEFI and PCI bootkits. In *PacSec*, June 2013.

[14] G. Delugré. Closer to metal: reverse-engineering the Broadcom NetExtreme's firmware. In *Hack. lu*, pages 27–29, 2010.

[15] G. Delugré. How to develop a rootkit for Broadcom NetExtreme network cards. Technical report, Sogeti ESEC Lab, 2011.

[16] M. Dornseif. 0wned by an ipod. In *PacSec*, 2004.

[17] M. Dornseif. Firewire – all your memory are belong to us. In *CanSecWest*, 2005.

[18] G. Duc and R. Keryell. CryptoPage: an efficient secure architecture with memory encryption, integrity and information leakage protection. *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 483–492, 2006.

[19] L. Duflot, O. Levillain, and B. Morin. ACPI: Design Principles and Concerns. In *Trust '09: Proceedings of the 2nd International Conference on Trusted Computing*. Springer-Verlag, Feb. 2009.

[20] L. Duflot, Y. Perez, G. Valadon, and O. Levillain. Can you still trust your network card. *CanSecWest/core10*, pages 24–26, 2010.

[21] L. Duflot, Y.-A. Perez, and B. Morin. What if you canâĂŹt trust your network card? In *Recent Advances in Intrusion Detection*, pages 378–397. Springer, 2011.

[22] R. Elbaz, D. Champagne, C. Gebotys, R. Lee, N. Potlapally, and L. Torres. Hardware mechanisms for memory authentication: A survey of existing techniques and engines. *Transactions on Computational Science IV*, pages 1–22, 2009.

[23] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, M. Bardouillet, and A. Martinez. A parallelized way to provide data encryption and integrity checking on a processor-memory bus. *Proceedings of the 43rd annual Design Automation Conference*, pages 506–509, 2006.

[24] W. Enck, K. Butler, T. Richardson, and P. McDaniel. Securing Non-Volatile Main Memory. Technical report, Pennsylvania State University, 2008.

[25] J. Fitzpatrick. NSA Playset: PCIe. In *Defcon 22*, August 2014.

[26] W. Futral and J. Greene. *Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters, 1st edition*. ApressOpen, Sept. 2013.

[27] A. Gazet. Sticky fingers & KBC Custom Shop. In *RECON*, pages 180–193, June 2011.

[28] J. Grand. Patent US7181560 - Method and apparatus for preserving computer memory. US Patent Office, 2007.

[29] T. C. Group. TCG Specification Architecture Overview. *TCG Specification Revision*, 1, 2007.

[30] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: cold boot attacks on encryption keys. In *SS'08: Proceedings of the 17th conference on Security symposium*. USENIX Association, July 2008.

[31] J. Heasman. Implementing and detecting a PCI rootkit. In *Black Hat DC*, page 3, 2006.

[32] J. Heasman. Implementing and Detecting an ACPI BIOS Rootkit. In *Black Hat Federal*, 2006.

[33] U. Hermann and C.-D. Hailfinger. FLashrom. http://flashrom.org/Flashrom, 2012.

[34] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*, June 2013.

[35] O. Horovitz and S. A. Weis. Physical Privilege Escalation and Mitigation in the x86 World. In *CanSecWest*, 2013.

[36] O. Horovitz, S. A. Weis, C. A. Waldspurger, and S. Rihan. Software Cryptoprocessor. US Patent App. 13/614,935, 2013.

[37] Y. Hu, G. Hammouri, and B. Sunar. A fast real-time memory authentication protocol. *Proceedings of the 3rd ACM workshop on Scalable trusted computing*,

pages 31–40, 2008.

[38] Intel Corporation. *Intel Virtualization Technology for Directed I/O*, September 2013. Order number D51397-006.

[39] Intel Corporation. *Software Guard Extensions Programming Reference*, September 2013. Order number 329298-001US.

[40] A. Ionescu. Apple SMC, The place to be definitely! (For an implant). In *Recon*, 2013.

[41] M. Jakobsson and K.-A. Johansson. Practical and secure software-based attestation. In *Lightweight Security & Privacy (LightSec)*, pages 1–9. IEEE, 2011.

[42] S. Kaczmarek. UEFI and Dreamboot. In *Hack in the Box*, June 2013.

[43] P. Kleissner. Stoned bootkit. In *Black Hat USA*, 2009.

[44] X. Kovah, J. Butterworth, C. Kallenberg, and S. Cornwell. Copernicus 2: SENTER the Dragon. Technical report, MITRE, 2014.

[45] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth. New Results for Timing-Based Attestation. *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 239–253, 2012.

[46] K. Kursawe, D. Schellekens, and B. Preneel. Analyzing Trusted Platform Communication. In *CRASH–CRyptographic Advances in Secure Hardware*, 2005.

[47] J. Loucaides and Y. Bulygin. Platform Firmware Security Assessment with CHIPSEC. In *CanSecWest*, 2014.

[48] K. Loukas. De Mysteriis Dom Jobsivs–Mac EFI Rootkits. In *Black Hat USA*, 2012.

[49] A. L. Luksenberg and N. A. Economou. Deep Boot. In *CanSecWest*, pages 1–54, Mar. 2012.

[50] C. Maartmann-Moe. Inception. http://www.breaknenter.org/projects/inception/, June 2011.

[51] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*, June 2013.

[52] C. P. Mozak. Patent US7945050 - Suppressing power supply noise using data scrambling in double data rate memory systems. US Patent Office, 2011.

[53] T. Müller, F. C. Freiling, and A. Dewald. TRESOR runs encryption securely outside RAM. In *SEC'11: Proceedings of the 20th USENIX Conference on Security*. USENIX Association, Aug. 2011.

[54] J. Pabel. FrozenCache Mitigating cold-boot attacks for Full-Disk-Encryption software. In *27th Chaos Communication Congress*, 2010.

[55] C. Peiqiang, J. Bøegh, and Y. Yuyu. Software behavior based trusted attestation. In *Measuring Technology and Mechatronics Automation (ICMTMA)*, volume 3, pages 298–301. IEEE, 2011.

[56] P. Peterson. Cryptkeeper: Improving security with encrypted RAM. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 120–126, 2010.

[57] N. L. Petroni Jr, T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - A Coprocessor-based Kernel Runtime Integrity Monitor. In *USENIX Security Symposium*, pages 179–194, 2004.

[58] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. *ACM SIGOPS Operating Systems Review*, 39(5):1–16, 2005.

[59] R. Sevinsky. Funderbolt: Adventures in Thunderbolt DMA Attacks. In *Black Hat USA*, Aug. 2013.

[60] I. Skochinsky. Intel ME Secrets. In *Code Blue*, 2014.

[61] D. Soeder and R. Permeh. eEye BootRoot. In *BlackHat USA*, 2005.

[62] P. Stewin and I. Bystrov. Understanding DMA malware. In *DIMVA'12: Proceedings of the 9th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer-Verlag, July 2012.

[63] G. Suh, D. Clarke, B. Gasend, M. Van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 339–350, 2003.

[64] C. Tarnovsky. Deconstructing a 'secure'processor. *Black Hat DC*, 2010, 2010.

[65] A. Tereshkin and R. Wojtczuk. Introducing ring-3 rootkits. *Black Hat USA*, 2009.

[66] A. Triulzi. Project Maux Mk. II, I Own the NIC, now I want a shell. *The 8th annual PacSec conference*, 2008.

[67] A. Triulzi. The Jedi Packet Trick takes over the Deathstar (or:"Taking NIC Backdoors to the Next Level"),". *CanSecWest*, pages 24–26, 2010.

[68] A. Vasudevan, J. McCune, J. Newsome, and A. Perrig. CARMA: A Hardware Tamper-Resistant Isolated Execution Environment on Commodity x86 Platforms. In *AsiaCCS*, 2012.

[69] A. Walters. http://www.forensicswiki.org/wiki/Volatility_Framework, 2014.

[70] J. Winter. Eavesdropping trusted platform module communication. *4th European Trusted Infrastructure Summerschool, ETISS*, 2009.

[71] F. Witherden. Memory Forensics over the IEEE 1394 Interface. *Tech Report*, Sept. 2010.

[72] R. Wojtczuk and J. Rutkowska. Attacking Intel Trusted Execution Technology. *Black Hat DC*, 2009.

[73] R. Wojtczuk and J. Rutkowska. Attacking Intel TXT via SINIT code execution hijacking. Technical report, Invisible Things Labs, 2011.

[74] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltsidas. Implementation and implications of a stealth hard-drive backdoor. In *Computer Security Applications Conference*, pages 279–288, 2013.