SoK: A Study of Using Hardware-assisted Isolated Execution Environments for Security

Fengwei Zhang
Department of Computer Science
Wayne State University
fengwei@wayne.edu

Hongwei Zhang
Department of Computer Science
Wayne State University
hongwei@wayne.edu

ABSTRACT

Hardware-assisted Isolated Execution Environments (HIEEs) have been widely adopted to build effective and efficient defensive tools for securing systems. Hardware vendors have introduced a variety of HIEEs including system management mode, Intel management engine, ARM TrustZone, and Intel software guard extensions. This SoK paper presents a comprehensive study of existing HIEEs and compares their features from the security perspective. Additionally, we explore both defensive and offensive use scenarios of HIEEs and discuss the attacks against HIEE-based systems. Overall, this paper aims to give an essential checkpoint of the state-of-the-art systems that use HIEEs for trustworthy computing.

Keywords

Isolated execution environments, hardware, security

1. INTRODUCTION

Isolating code execution is one of the fundamental approaches to achieving security. Researchers use virtualization technology to create an isolated execution environment for running defensive tools. Moreover, Virtual machine introspection [21] has been widely adopted for attacks detection and malware analysis. However, existing virtualization-based approaches have limitations including: 1) Dependence on hypervisors that may have a large Trusted Computing Base (TCB). For instance, the latest Xen hypervisor has 532K lines of source code obtained from [11]. 2) Failure to deal with hypervisor or firmware rootkits. Virtualization-based approaches rely on hypervisors so they cannot analyze the same or higher privilege-level rootkits. And 3) suffering from system performance overhead (e.g., context switches from a VM to a hypervisor).

In light of these problems, researchers proposed to use Hardware-assisted Isolated Execution Environments (HIEEs) for securing systems. This approach combines the isolated execution concept with hardware-assisted technologies. Both are crucial to secure computer systems: The isolated execution concept provides a Trusted Execution Environment (TEE) for running defensive tools on a compromised system. Using hardware-assisted technologies excludes the hypervisors from TCB, achieves a high level of privilege (i.e., hardware-level privilege), and reduces performance overhead giving that context switches are performed faster in hardware.

In this SoK paper, we survey the state-of-the-art systems that leverage HIEEs for security. We first study six hardware-level computing environments (i.e., HIEEs) that have been used for building security tools. Based on the timelines they introduced, we categorize them as follows. 1) Legacy HIEEs: System Management Mode (SMM) and Dynamic Root of Trust for Measurements (DRTM); 2) recent HIEEs: Intel Management Engine (ME), AMD Platform Security Processor (PSP), and ARM TrustZone; 3) the latest HIEE: Intel Software Guard Extensions (SGX). We discuss the security usage scenarios of HIEEs from both defensive and offensive points of view, and further describe these usage scenarios along with the existing HIEE-based systems. Additionally, we identify attacks and security concerns to HIEE-based systems from two aspects: 1) the isolated computing environments (i.e., HIEEs themselves) and 2) the approach of using HIEEs for security. For each HIEE itself, we enumerate potential attacks against it and describe corresponding mitigations. In terms of the approach of using HIEEs for security in general, we raise security concerns such as ensuring trusted path and verifying trustworthy hardware. Note that the concept of HIEE needs to be distinguished from TEE. On one hand, a TEE may not be a HIEE such as software-based TEEs (e.g., virtualization technology); on the other hand, a HIEE does not have to be a TEE (e.g., SMM is a HIEE but may not be a TEE because it is not designed for security).

The main contributions of this SoK paper are:

- We present a thorough study of six HIEEs including SMM, Intel ME, AMD PSP, DRTM, Intel SGX, and ARM TrustZone, and compare their hardware features for trustworthy computing.
- We explore both the defensive and offensive use scenarios of HIEEs and describe them with the state-of-the-art HIEE-based systems.
- We discuss all attacks against the computing environment of each HIEE (e.g., bypassing the isolation) and some mitigations.

HASP 2016, June 18 2016, ,

DOI: http://dx.doi.org/10.1145/2948618.2948621

- We raise the concerns about the approach of using HIEEs for security including ensuring the trusted switching path and verifying trustworthiness of hardware technologies.

The rest of the paper is organized as follows. Section 2 explains different HIEEs including SMM, Intel ME, AMD PSP, DRTM, Intel SGX, and ARM TrustZone. Section 3 presents the security use cases of HIEEs. Section 4 presents all attacks against the HIEEs. Section 5 discusses the security concerns of the HIEE-based approach. Lastly, Section 6 concludes the SoK paper with our expectations.

2. HIEE

In this section, we first explain Hardware-assisted Isolated Execution Environments (HIEEs) including system management mode, Intel management engine, AMD secure processor, dynamic root of trust for measurement, Intel software guard extension, and ARM TrustZone. Then, we briefly compare these HIEEs.

2.1 System Management Mode

System Management Mode (SMM) [24] is a mode of execution similar to Real and Protected modes available on x86 platforms (Intel started to use SMM in its Pentium processors since early 90s). It provides a hardware-assisted isolated execution environment for implementing platform-specific system control functions such as power management. It is initialized by the Basic Input/Output System (BIOS).

SMM is triggered by asserting the System Management Interrupt (SMI) pin on the CPU. This pin can be asserted in a variety of ways, which include writing to a hardware port or generating Message Signaled Interrupts with a PCI device. Next, the CPU saves its state to a special region of memory called System Management RAM (SMRAM). Then, it atomically executes the SMI handler stored in SMRAM. SMRAM cannot be addressed by the other modes of execution. The requests for addresses in SMRAM are instead forwarded to video memory by default. This caveat therefore allows SMRAM to be used as a secure storage. The SMI handler is loaded into SMRAM by the BIOS at boot time. The SMI handler has unrestricted access to the physical address space and can run privileged instructions (For this reason, SMM is often referred to as *ring -2*.) The `RSM` instruction forces the CPU to exit from SMM and resume execution in the previous mode.

In general, there are software- and hardware-based methods to trigger an SMI. In software, we can write to an ACPI port to raise an SMI. For example, Intel chipsets use port `0x2b` as specified by the Southbridge datasheet; AMD K8 chipset with a VIA VT8237r Southbridge uses `0x52f` as the SMI triggering port [53]. In terms of hardware-based methods, there are many hardware devices that can be used to raise an SMI, including keyboards, network cards, and hardware timers. Figure 1 shows the process of triggering an SMI from an OS in Protected Mode. Note that other CPU modes (e.g., Real Mode) can also switch into SMM by triggering an SMI.

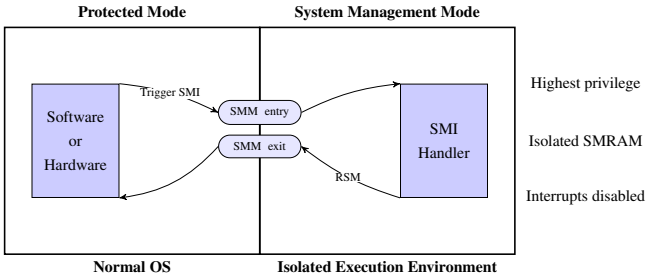

Figure 1: SMI Triggering Process

2.2 Intel Management Engine

The Intel Management Engine (ME) is a micro-computer embedded inside of all recent Intel processors, and it exists on Intel products including servers, workstations, desktops, tablets, and smart phones [36]. Intel introduced ME as an embedded processor in 2007. At that time, its main function was to support Intel Active Management Technology (AMT), and Intel AMT is the first application running in the ME. Recently, Intel started to use ME as a Trusted Execution Environmental (TEE) for executing security-sensitive applications. According to the latest ME book [36] written by an Intel Architect working on ME, a few security applications have been or will be implemented in ME including enhanced privacy identification, protected audio video path, identity protection technology, and boot guard.

Figure 2 shows the hardware architecture of ME. From the figure we can see that ME is like a computer; it contains a processor, cryptography engine, Direct Memory Access (DMA) engine, Host-Embedded Communication Interface (HECI) engine, Read-Only Memory (ROM), internal Static Random-Access Memory (SRAM), a timer, and other I/O devices. ME executes the instructions on the processor, and it has code and data caches to reduce the number of accesses to the internal SRAM. The internal SRAM is used to store the firmware code and runtime data. Besides the internal SRAM, ME also uses some Dynamic Random-Access Memory (DRAM) from the main system's memory (i.e., host memory). This DRAM serves a role as the disk; the memory pages of code/data that are not currently used by ME processor will be evicted from SRAM and swapped out to DRAM in the host memory. The region of DRAM is reserved by the BIOS when system boots. This DRAM is dedicated for ME use and the operating system cannot access it. However, the design of ME does not trust the BIOS and it assumes the host can access the reserved DRAM region.

Since the embedded processor (i.e., ME processor) cannot address the host memory, two engines (i.e., DAM engine and HECI engine) are introduced for the data transmission between the ME memory and the main system's memory. DMA engine is used to move large amounts of data between the ME memory and the host memory. Note that the DMA engine can only understand the physical memory addresses when accessing host memory. The cryptography engine is used to execute the expensive cryptography algorithms so that it offloads the commutation from the ME processor. The cryptography engine includes many algorithms including AES, SHA, DRING, and big number arithmetic [36], and the ME firmware can use them by invoking their APIs.

The ME firmware is stored on two types of media: ROM and SPI flash memory. As shown in the figure 2, the ROM is located in the ME. The code in the ROM is burned in the manufacture stage and it cannot be modified. The ROM stores a boot loader and serves as the root of trust of the

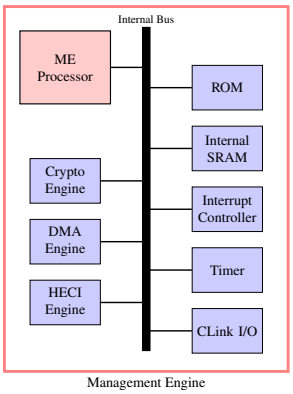

Figure 2: Architecture of Management Engine

Table 1: Main Hardware Components of ME

Hardware	Description
ME processor	Main master device that executes the firmware
ROM	Boot loader; cannot be modified; as the root of trust of ME
Internal SRAM	Storing the code and data at runtime
Crypto engine	Executing crypto algorithm to save the processor's cycles
DMA engine	Transmitting large amounts of data between host and ME
HECI engine	Moving small amounts of data; host can program it

ME. The majority of the ME firmware is stored on SPI flash memory, and it includes a custom OS and applications. The flash is divided into multiple regions. Depending on the applications implemented in ME, the flash firmware located differently on the host motherboard (e.g., BIOS and NIC). The flash normally are locked by the OEMs to prevent malicious modifications. However, researchers have demonstrated bypassing these lock mechanisms to inject code into ME [50]. Table 1 lists the main hardware components of ME.

2.3 AMD Embedded Processors

Though ME is for Intel processors, we can find similar technologies on AMD platforms. AMD Secure Processor [4] (also called Platform Security Processor or PSP) is a dedicated processor embedded inside of the main AMD CPU. It works with ARM TrustZone technology and software-based Trusted Execution Environment (TEE) to enable running third-party trusted applications. AMD Secure Processor is a hardware-based technology which enables secure boot up from BIOS level into the TEE. Trusted third-party applications are able to leverage industry-standard APIs to take advantage of the TEE's secure execution environment. Another example is System Management Unit (SMU) [30]. The SMU is a subcomponent of the Northbridge that is responsible for a variety of system and power management tasks during boot and runtime. The SMU contains a processor to assist [3]. Since AMD integrated Northbridge into the CPU, the SMU processor is an embedded processor inside of the CPU, which is same as Intel ME.

2.4 Dynamic Root of Trust for Measurement

Trust Computing Group (TCG) introduced Dynamic Root of Trust for Measurement (DRTM) [52], also called "late launch", in the TPM v1.2 specification [51] in 2005. It is an alternative to the Static Root of Trust for Measurement (SRTM). Unlike SRTM which operates at boot time, DRTM allows the root of trust for measurement to be initialized

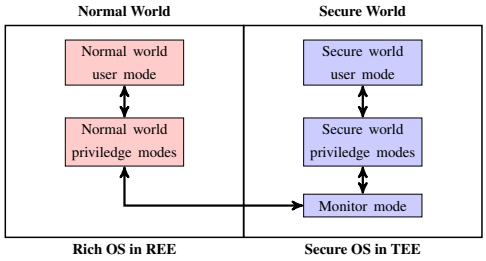

Figure 3: Processor Modes in TrustZone-enabled ARM Architecture

at any point. To implement this technology, Intel developed Trusted eXecution Technology (TXT) [25], providing a trusted way to load and execute system software (e.g., OS or VMM). TXT uses a new CPU instruction, `SENTER`, to control the secure environment. Intel TXT does not make any assumptions about the system state, and it provides a dynamic root of trust for late launch. Thus, TXT can be viewed as a hardware-assisted isolated execution environment to run security sensitive tasks. AMD has a similar technology called Secure Virtual Machine [2], and it uses the `SKINIT` instruction to enter the secure environment. Note that both TXT and SVM introduce a significant overhead on the late launch operation (e.g., the `SKINIT` instruction in [31]).

2.5 Intel Software Guard Extensions

In 2013, Intel presented three introduction papers on Software Guard Extensions (SGX) [34, 5, 23]; SGX is a set of instructions and mechanisms for memory accesses added to Intel architecture processors. These extensions allow an application to instantiate a protected container, referred to as an enclave. An enclave could be used as a TEE, which provides confidentiality and integrity even without trusting the BIOS, firmware, hypervisors, and OSes. Some of researchers consider SGX as a new generation of TXT [39, 15]. Jain et al. [27] developed OpenSGX, an open-source platform that emulates Intel SGX hardware components at the instruction level by modifying QEMU. It demonstrated OpenSGX can be used to protect sensitive information of Tor nodes.

2.6 ARM TrustZone

ARM TrustZone technology [6] is a hardware feature that creates an isolated execution environment since ARMv6 around 2002 [12]. Similar to other hardware isolation technologies, it provides two environments or worlds. The Trust Execution Environment (TEE) is called the secure world, and the Rich Execution Environment (REE) is called the normal world. To ensure the complete isolation between the secure world and the normal world, TrustZone provides security extensions for hardware components including CPU, memory, and peripherals.

The CPU on a TrustZone-enabled ARM platform has two security modes: secure world and normal world. Figure 3 shows the processor modes in a TrustZone-enabled ARM platform. Each processor mode has its own memory access region and privilege. The code running in the normal world cannot access the memory in the secure world, while the program executed in the secure world can access the memory in normal world. The secure and normal worlds can be identified by reading the NS bit in the Secure Configuration

Table 2: Summary of HIEEs

	SMM	ME	PSP	DRTM	SGX	TrustZone
Timelines	~1993	~2007	~2013	~2005	~2013	~2002
Supported hardware	x86	Intel	AMD	Intel/AMD	Intel	ARM
Sharing main CPU	✓			✓	✓	✓
High privilege	✓	✓	✓			✓
Zero overhead		✓	✓			
Designed for security		✓	✓	✓	✓	✓

Register (SCR), which can only be modified in the secure world. As shown in Figure 3, TrustZone uses Monitor mode that only runs in the secure world to serve as a gatekeeper managing the switches between the two worlds. The normal world can call a special instruction called the Secure Monitor Call (`smc`) to enter the Monitor mode and modify the NS bit to switch into the secure world.

TrustZone uses Memory Management Unit mechanism to support virtual memory address spaces in both the secure and normal worlds. The same virtual address space in the two worlds is mapped to different physical regions. There are two types of hardware interrupts: Interrupt Request (IRQ) and Fast Interrupt Request (FIQ). The secure world can assert FIQ or IRQ while the normal world can only assert IRQ.

2.7 Summary of HIEEs

Table 2 summarizes the features of HIEEs. SMM is available on all x86 architecture including Intel and AMD processors; ME is only for Intel-specific processors, while PSP is a hardware feature for AMD; DRTM is introduced by TCG; both Intel and AMD processors have a corresponding implementation (i.e., VT-x/SVM); TrustZone is a security extension on ARM processors. Except for Intel ME and AMD PSP, all other HIEEs share the main CPU in a time-sliced fashion. SMM, ME, PSP, and TrustZone have a high privilege; for instance, they are able to access all host physical memory. Additionally, ME- or PSP-based systems have zero performance overhead to the main CPU because they run on an independent processor. Except for SMM, all other HIEEs are originally designed for security purposes.

3. USE CASES

In this section, we survey the use cases of HIEE for both defense and offensive purposes and describe the state-of-the-art HIEE-based systems for each scenario.

3.1 System Introspection

System introspection has been widely adopted for ensuring security. There are an array of tools that use HIEE for system introspection, integrity checking, and malware detection. HyperGuard [41] suggests using SMM to monitor hypervisor integrity by taking snapshots of a VM guest and checking it in SMM. HyperCheck [65] has similar goals, but outsource the snapshot to an external server for OS/hypervisor integrity checking, since it can reduce the computation overhead on the protected machine. HyperSentry [8] uses an out-of-band channel, specifically the Intelligent Platform Management Interface, to trigger SMM to check the integrity of base code operating on critical data. While HyperGuard, HyperCheck, and HyperSentry focus on enforcing OS or hypervisor integrity checking, IOCheck [64] is a framework to enhance the security of I/O devices at runtime. It leverages SMM to quickly check the integrity of I/O configurations and firmware. Spectre [62] is another SMM-based system that introspects the host memory for malware detection. It periodically checks the host memory for heap overflow, heap spray, and rootkit attacks. Ge et al. [22] propose SPROBES that leverages ARM TrustZone technology as a HIEE to restrict the normal world's kernel execution to approved kernel code memory. It enables the secure world to cause the normal world to trap on any normal world instruction and provides an unforgeable view of the normal world's processor state. TZ-RKP [7] is a similar TrustZone-based system that aims to enforce kernel code integrity on ARM platforms. It further improves SPROBES by adding defense mechanism including data integrity protection and performance enhancement. Flicker [31] and Trustvisor [32] employ Dynamic Root of Trust Measurement (DRTM) to provide a HIEE for running security code. One particular usage is to run a rootkit detector for OS integrity checking.

3.2 Memory Forensics

Jiang et al. [54] propose a firmware-assisted memory acquisition and analysis tool for digital forensics. It leverages SMM to reliably perform acquisition of volatile memory of a target system, and then transmits the memory contents to a remote machine for analysis by using a network card. SMMDumper [35] implements the proposed system from paper [54] on QEMU, and it further enables to dump and transmit physical memory extending over 4 GB. TrustDump [48] is a TrustZone-based memory acquisition mechanism that is capable of reliably obtaining the RAM memory and CPU registers of the mobile OS even if the OS has crashed or has been compromised. It uses the secure domain of TrustZone as a HIEE and runs the memory acquisition module in it for memory forensics.

3.3 Transparent Malware Analysis

Zhang et al. [61] propose MalT, a bare-metal debugging tool for malware analysis. Its core idea is to use SMM to increase the debugging transparency. Specifically, it leverages SMM as a HIEE that leaves a minimal footprint on the debugging system and provides a more transparent execution environment for debuggers. We believe other HIEEs including ARM TrustZone and Intel ME can also be used for transparent malware analysis due to their high privilege and stealthiness. Furthermore, Intel ME can achieve a higher level of transparency because it executes on a co-processor and does not introduce any performance overhead on the main CPU.

3.4 Executing Sensitive Workloads

Flicker [31] and TrustVisor [32] employ DRTM with a small trusted computing base to create a HIEE. Flicker creates an on-demand secure environment using DRTM, while TrustVisor uses DRTM to securely initialize a light-weight hypervisor that uses hardware virtualization (VT-x/SVM) to protect the applications running in the secure environments. The two systems use the TPM to provide remote attestations and to securely store data for executing sensitive workloads. Bumpy [33] is a Flicker-based system for securing sensitive network input. It handles inputs in a special code module that is executed in an isolated environment using the Flicker. Sun et al. presents TrustICE [49], a TrustZone-based isolation framework to provie isolated computing environments (ICEs) on mobile devices. The main

idea of TrustICE is to create ICEs in the normal world rather than in the secure world. It leverages the TrustZone extensions to isolate the sensitive workloads in an ICE from an untrusted OS in the normal world. TrustOTP [47] aims to transform mobile phones to display one-time password tokens. It leverages TrustZone technology and can securely display the password even if the mobile OS is malicious or crashed. SICE [9] is a framework to provide hardware-level isolation and protection for sensitive workloads running on x86 platforms in the cloud. It uses SMM as a HIEE and runs on multi-core processors to allow the isolated environments to concurrently run security sensitive workloads and the normal OS. It does not rely on any software component in the host environment and supports up to 4GB of isolated memory. TrustLogin [63] is another SMM-based system that securely performs login operations on commodity operating systems. Even if the operating system and applications are compromised, an attacker is not able to reveal the login password from the host. TrustLogin leverages SMM to transparently protect the login credentials from keyloggers. In addition, Microsoft Research presents two systems, Haven [10] and VC3 [43], that use Intel SGX as a HIEE and protect the confidentiality and integrity of applications in the cloud. They rely on SGX processors to isolate memory regions and keep the OS and hypervisor out of the TCB. Kim et al. [28] adopts SGX to secure network applications such as Tor.

3.5 Rootkits and Keyloggers

Though researchers have used hardware-assisted isolated execution environments for implementing defensive tools, attackers can also use it for malicious purposes due to their `high privilege` and `stealthiness`. **SMM Rootkits**: Security researchers have proposed to use SMM to implement attacks. In 2004, Duflot [17] demonstrated the first SMM-based attack to bypass the protection mechanism in OpenBSD. Embleton et al. [20] use SMM to implement a chipset level keylogger and a network backdoor capable of directly interacting with the network card to send logged keystrokes to a remote machine via network packets. Schiffman and Kaplana [42] further demonstrated that with USB keyboards instead of PS/2 ones. Other SMM-based attacks focus on achieving stealthy rootkits [13, 1]. For instance, the National Security Agency (NSA) uses SMM to build an array of rootkits including DEITYBOUNCE for Dell and IRONCHEF for HP Proliant servers [1]. However, these attacks require bypassing or unlocking SMRAM protection. We discuss how to bypass SMRAM protection mechanism in Section 4.1. **ME Rootkits**: Serveral attacks [50, 46] have been demostrated using ME to implement advanced stealthy rootkits. Tereshkin and Wojtczuk [50] injects malicious code in to the Intel Active Management Technology (AMT) to implement ME ring -3 rootkits. DAGGER [46] is a DMA-based keylogger implemented in ME, and it captures keystrokes very early in the platform boot process, which enables DAGGER to capture harddisk encryption passwords. **DRTM, SGX, and TrustZone Rootkits**: To the best of our knowledge, we have not seen any publicly available examples of DRTM, SGX, and TrustZone rootkits. However, similar to SMM or ME rootkits, attackers have the motivation to implement rootkits in them due to their `stealthiness`. In particular, researchers [37, 16, 27] raise concerns of misusing SGX for malware or rootkits, so malicious programs running in an enclave cannot be analyzed due to the hardware protection; normal security tools like anti-virus and rootkits-detectors cannot know if an enclave has been compromised or not.

One one hand, hardware-assisted isolated execution environments are powerful trusted computing environments for executing security functions (e.g., system introspection for malware/attacks defense as described); on the other hand, it creates an ideal environment or infrastructure that attracts attackers to implement super-powerful rootkits. We believe the security of HIEEs themselves needs to be explored more and we discuss the attacks against HIEEs in Section 4.

Table 3: Summary of SMM Attacks and Solutions

SMM Attacks	Solutions
Unlocked SMRAM [17, 20, 13]	Set D_LCK bit
SMRAM reclaiming [41]	Lock remapping and TOLUD registers
Cache poisoning [58, 19]	SMRR
Graphics aperture [18]	Lock TOLUD
TSEG location [18]	Lock TSEG base
Call/fetch outside of SMRAM [18, 60]	No call/fetch outside of SMRAM

4. HIEE ATTACKS

In this section, we present the attacks against HIEEs (i.e., the hardware computing environments). We show the attacks that can bypass the isolation of these hardware environments.

4.1 SMM Attacks

Before 2006, computers did not lock their SMRAM in the BIOS [20], and researchers used this flaw to implement SMM-based rootkits [17, 20, 13]. Modern computers lock the SMRAM in the BIOS so that SMRAM is inaccessible from any other CPU modes after booting. Wojtczuk and Rutkowska demonstrated bypassing the SMRAM lock through memory reclaiming [41] or cache poisoning [58]. The memory reclaiming attack can be addressed by locking the remapping registers and Top of Low Usable DRAM (TOLUD) register. The cache poisoning attack forces the CPU to execute instructions from the cache instead of SMRAM by manipulating the Memory Type Range Register (MTRR). Duflot also independently discovered this architectural vulnerability [19], but it has been fixed by Intel adding SMRR [24]. Furthermore, Duflot et al. [18] listed some design issues of SMM, but they can be fixed by correct configurations in BIOS and careful implementation of the SMI handler. Table 3 shows a summary of attacks against SMM and their corresponding solutions. Wojtczuk and Kallenberg [55] recently presented an SMM attack by manipulating UEFI boot script that allows attackers to bypass the SMM lock and modify the SMI handler with ring 0 privilege. The UEFI boot script is a data structure interpreted by UEFI firmware during S3 resume. When the boot script executes, system registers like BIOS_NTL (SPI flash write protection) or TSEG (SMM protection from DMA) are not set so that attackers can force an S3 sleep to take control of SMM. Fortunately, as stated in the paper [55], the BIOS update around the end of 2014 fixed this vulnerability.

Butterworth et al. [14] demonstrated a buffer overflow vulnerability in the BIOS updating process in SMM, but this is not an architectural vulnerability and is specific to that particular BIOS version. Recently, Intel introduced SMM-Transfer Monitor (STM), which virtualizes the SMM code [24]. It is also the answer to attacks against Intel

TXT [57].

4.2 ME Attacks

Intel uses ME as a TEE to execute security sensitive operations. However, several attacks have been demonstrated to bypass the hardware protection mechanism and implement rootkits in it. In 2009, Tereshkin and Wojtczuk [50] demonstrated that they can implement ring -3 rootkits in ME by injecting the malicious code into the Intel Active Management Technology (AMT), and this is the first attack against Intel ME. DAGGER [46] bypasses the ME isolation using a similar technique in [50], but it hooks the ME firmware function `memset` because it is invoked more often. Skochinsky [45] discovers that the ME firmware on the SPI flash uses Huffman encoding to prevent reverse engineering for implementing rootkits.

4.3 DRTM Attacks

One implementation of Dynamic Root of Trust for Measurement (DRTM) is the Intel Trusted Execution Technology (TXT). Wojtczuk and Rutkowska from Invisible Things Lab demonstrate several attacks [57, 56, 59] against Intel TXT. In February 2009, Invisible Things Lab presented an attack against Intel TXT because SMM is not measured and able to interfere with TXT launch [57]. In December same year, they demonstrated another way to circumvent Intel TXT by tricking `SENTER` into mis-configurating VT-d setup, so that attackers can compromise the newly loaded hypervisor using DMA attacks [56]. In 2011, they presented another attack [59] that exploits a bug in `SINIT` module, an internal part of the Intel TXT. Based on the latest talk from them [40] in December 2015, SMM-Transfer Monitor (STM) is still not present in Intel products. From this point of view, Intel TXT must trust the SMM code due to the absence of STM.

4.4 SGX Attacks

Intel SGX is the latest iteration for trustworthy computing, and all future Intel processors will have this feature and use it as a TEE for addressing security problems. However, researchers raised security concerns about it. Recently, Costan and Devadas [15] published an extensive study on SGX. They analyzed the security features of SGX and raised concerns such as cache timing attacks and software side-channel attacks. Additionally, SGX tutorial slides from ISCA 2015 [26] mentioned that SGX does not protect against software side-channel attacks including using performance counters. Moreover, SGX for desktop-like environments needs to establish a secure channel between I/O devices (e.g., keyboard and video display) and an enclave to prevent sensitive data leakage [38, 27]. Fortunately, Intel Protected Audio Video Path (PVAP) technology can securely display video frames and play audio to users; Intel Identity Protection Technology (IPT) provides security features including Protected Transaction Display (e.g., entering a PIN by an user). According to the ME book [36], both PVAP and IPT are realized by ME. SGX needs Enhanced Privacy Identification (EPID) support for remote attestation [27]. The EPID is a security mechanism exclusively built in ME and serves as the hardware root of trust [36]. During the manufacturing stage, a unique EPID private key is programmed in ME, and the system uses the EPID private key to provide to the local host or a remote server that it is a genuine intel platform. The design of SGX assumes that the firmware could be malicious, it becomes unclear if the ME firmware is malicious since SGX relies on many hardware features (IPT, PVAP, and EPID) implemented by ME.

4.5 TrustZone Attacks

With the proliferation of mobile computing and wide adoption of TrustZone technology, this attracts attackers to study TrustZone and bypass the isolation mechanism for stealing sensitive information. Di [44] found vulnerabilities that are able to execute arbitrarily code in secure world using a user-level application in normal world. A proof-of-concept demonstration has been shown on a Huawei HiSilicon device. Additionally, researchers demonstrate that a TrustZone kernel privilege escalation vulnerability exists on Qualcomm implementation [29].

5. DISCUSSION

Section 4 explains the specific attacks against each execution environment. In this section, we discuss the concerns on the approach of using HIEEs for security.

One challenge of using HIEEs for security is to ensure the trusted switching path. HIEE-based systems assume attackers have ring 0 privilege; attackers can intercept the switching from the normal environment to TEE and provide a fake switching process to deceive users (i.e., spoofing attack). Additionally, the attackers can perform a Denial of Service (DoS) attack against a system by simply disabling the switching due to their ring 0 privilege. Fortunately, Intel ME and AMD PSP do not have this problem because they are running independently from the main CPU so that there is no switching process for them. However, other HIEEs like SMM, DRTM, SGX, and TrustZone have this problem due to sharing the main CPU in a time-sliced fashion. To address this problem, several ad-hoc solutions are implemented in HIEE-based systems. Bumpy [33], a DRTM-based system, uses an external smartphone as the trusted monitor to acknowledge the switching. TrustLogin [63], an SMM-based system, uses the keyboard LED lights to show a user-defined sequence and the PC speaker to play a melody to ensure the switching. TrustICE [49], a TrustZone-based system, uses the LED lights to prevent spoofing attacks. However, these solutions that ensures the trusted switching path are not user-friendly. We believe building a generic and user-friendly trusted path mechanism for HIEE-based systems is still an open research problem.

The approach of using HIEEs for security heavily depends on the hardware vendors. It assumes hardware vendors are trustworthy and provided hardware features are bug-free (e.g., isolation is guaranteed). Unfortunately, there is no clear way of verifying these assumptions. Moreover, hardware vendors tend not to release the details of their implementations due to various reasons (e.g., commercial secrecy). For instance, Intel ME is a secret land that only the hardware vendor knows, though Intel uses it for many security features as explained in Section 2. We would like to draw attention to the community on how to reliably evaluate the trustworthiness of these mysterious hardware security technologies.

6. CONCLUSIONS

Hardware-assisted isolated execution environments play

critical roles for trustworthy computing. Moreover, hardware vendors are actively developing hardware-assisted technologies to address security issues. By surveying the existing hardware-assisted isolated execution environments and their state-of-the-art systems, we expect our observations in this SoK paper provide helpful guidelines for future HIEEs or HIEE-based systems.

7. ACKNOWLEDGEMENTS

We would like to thank our shepherd, Larry Shi, and the anonymous reviewers for their insightful comments that improved the paper. This work is supported by the National Science Foundation Grant No. CNS-1054634. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government.

8. REFERENCES

[1] NSA's ANT Division Catalog of Exploits for Nearly Every Major Software/Hardware/Firmware. http://Leaksource.wordpress.com.

[2] Advanced Micro Devices, Inc. AMD64 ArchitectureProgrammer's Manual Volume 2: System Programming. /urlhttp://support.amd.com/TechDocs/24593.pdf, June 2015.

[3] Advanced Micro Devices, Inc. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 30h-3Fh Processors. http://support.amd.com/TechDocs/52740_16h_Models_30h-3Fh_BKDG.pdf, March 2015.

[4] AMD TATS BIOS Development Group. AMD Security and Server Innovation. http://www.uefi.org/sites/default/files/resources/UEFI_PlugFest_AMD_Security_and_Server_innovation_AMD_March_2013.pdf, 2013.

[5] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13)*, 2013.

[6] ARM. ARM Security Technology - Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009.

[7] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*, 2014.

[8] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. HyperSentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, 2010.

[9] A. M. Azab, P. Ning, and X. Zhang. SICE: A Hardware-level Strongly Isolated Computing Environment for x86 Multi-core Platforms. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*, 2011.

[10] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, 2014.

[11] Black Duck Open Hub. Black Duck Software, Inc. https://www.openhub.net/p?ref=homepage&query=xen. Access time: 05/03/2016.

[12] D. Brash. ARM White Paper, The ARM Architecture Version 6 (ARMv6). http://lars.nocrew.org/computers/processors/ARM/ARMv6.pdf, January 2002.

[13] BSDaemon, coideloko, and D0nAnd0n. System Management Mode Hack: Using SMM for 'Other Purposes'. *Phrack Magazine*, 2008.

[14] J. Butterworth, C. Kallenberg, and X. Kovah. BIOS Chronomancy: Fixing the Core Root of Trust for Measurement. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13)*, 2013.

[15] V. Costan and S. Devadas. Intel SGX Explained. https://eprint.iacr.org/2016/086.pdf, 2016.

[16] S. Davenport and R. Ford. SGX: the good, the bad and the downright ugly. https://www.virusbulletin.com/virusbulletin/2014/01/sgx-good-bad-and-downright-ugly, January 2014.

[17] L. Duflot, D. Etiemble, and O. Grumelard. Using CPU System Management Mode to Circumvent Operating System Security Functions. In *Proceedings of the 7th CanSecWest Conference (CanSecWest'04)*, 2004.

[18] L. Duflot, O. Levillain, B. Morin, and O. Grumelard. System Management Mode Design and Security Issues. http://www.ssi.gouv.fr/IMG/pdf/IT_Defense_2010_final.pdf.

[19] L. Duflot, O. Levillain, B. Morin, and O. Grumelard. Getting into the SMRAM: SMM Reloaded. In *Proceedings of the 12th CanSecWest Conference (CanSecWest'09)*, 2009.

[20] S. Embleton, S. Sparks, and C. Zou. SMM rootkits: A New Breed of OS Independent Malware. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm'08)*, 2008.

[21] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the 10th Annual Network and Distributed Systems Security Symposium (NDSS'03)*, 2003.

[22] X. Ge, H. Vijayakumar, and T. Jaeger. SPROBES: Enforcing Kernel Code Integrity on the TrustZone Architecture. In *Proceedings of The 3rd IEEE Mobile Security Technologies Workshop (MoST)*, 2013.

[23] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas, V. Phegade, and J. del Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2nd Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13)*, 2013.

[24] Intel. 64 and IA-32 Architectures Software Developer's Manual. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[25] Intel. Trusted Execution Technology. http://www.intel.com/content/www/us/en/trusted-execution-technology/trusted-execution-technology-security-paper.html.

[26] Intel. ISCA 2015 SGX Tutorial. https://software.intel.com/sites/default/files/332680-002.pdf, 2015.

[27] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han. OpenSGX: An Open Platform for SGX Research. In *Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS'16)*, San Diego, CA, Feb. 2016.

[28] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han. A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets)*, Philadelphia, PA, Nov. 2015.

[29] laginimaineb. Exploring Qualcomm's TrustZone implementation. http://bits-please.blogspot.com/2015/08/exploring-qualcomms-trustzone.html, April 2015.

[30] R. Marek. AMD x86 SMU firmware analysis - Do you care about Matroshka processors? https://events.ccc.de/congress/2014/Fahrplan/system/

attachments/2503/original/ccc-final.pdf, 2014.

[31] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2008.

[32] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[33] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *NDSS*, 2009.

[34] F. Mckeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13)*, 2013.

[35] A. Reina, A. Fattori, F. Pagani, L. Cavallaro, and D. Bruschi. When Hardware Meets Software: A Bulletproof Solution to Forensic Memory Acquisition. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'12)*, 2012.

[36] X. Ruan. *Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine*. Apress, 2014.

[37] J. Rutkowska. Thoughts on Intel's upcoming Software Guard Extensions (Part 2). http://blog.invisiblethings.org/2013/09/23/thoughts-on-intels-upcoming-software.html.

[38] J. Rutkowska. Thoughts on Intel's upcoming Software Guard Extensions (Part 1). http://blog.invisiblethings.org/2013/08/30/thoughts-on-intels-upcoming-software.html, August 2013.

[39] J. Rutkowska. Intel x86 Considered Harmful. http://blog.invisiblethings.org/papers/2015/x86_harmful.pdf, October 2015.

[40] J. Rutkowska. Towards (reasonably)easonably) Trustworthy x86 Laptops. 32st Chaos Communication Congress (32C3), https://events.ccc.de/congress/2015/Fahrplan/events/7352.html, December 2015.

[41] J. Rutkowska and R. Wojtczuk. Preventing and Detecting Xen Hypervisor Subversions. http://www.invisiblethingslab.com/resources/bh08/part2-full.pdf, 2008.

[42] J. Schiffman and D. Kaplan. The SMM Rootkit Revisited: Fun with USB. In *Proceedings of 9th International Conference on Availability, Reliability and Security (ARES'14)*, 2014.

[43] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15)*, 2015.

[44] D. Shen. Exploiting Trustzone on Android. Black Hat USA Briefings, https://www.blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android-wp.pdf.

[45] I. Skochinsky. Intel ME Secrets: Hidden code in your chipset and how to discover what exactly it does. In RECON, https://recon.cx/2014/slides/Recon%202014%20Skochinsky.pdf, 2014.

[46] P. Stewin and I. Bystrov. Understanding DMA Malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12)*. 2012.

[47] H. Sun, K. Sun, Y. Wang, and J. Jing. TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS'15)*, 2015.

[48] H. Sun, K. Sun, Y. Wang, J. Jing, and S. Jajodia. TrustDump: Reliable Memory Acquisition on Smartphones. In *Proceedings of The 18th European Symposium on Research in Computer Security (ESORICS'13).*, 2013.

[49] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang. TrustICE: Hardware-assisted Isolated Computing Environments on Mobile Devices. In *Proceedings of The 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*, 2015.

[50] A. Tereshkin and R. Wojtczuk. Introducing Ring -3 Rootkits. http://invisiblethingslab.com/itl/Resources.html, 2009.

[51] Trusted Computing Group. TCG PC Client Specific Implementation Specification For Conventional BIOS, Version 1.20, Revision 1.00, For TPM Family 1.2. http://www.trustedcomputinggroup.org/files/temp/64505409-1D09-3519-AD5C611FAD3F799B/PCClientImplementationforBIOS.pdf, July 2005.

[52] Trusted Computing Group. TCG D-RTM Architecture Document Version 1.0.0. http://www.trustedcomputinggroup.org/resources/drtm_architecture_specification, June 2013.

[53] VIA. VT8237R Southbridge. http://www.via.com.tw/.

[54] J. Wang, F. Zhang, K. Sun, and A. Stavrou. Firmware-assisted Memory Acquisition and Analysis Tools for Digital Forensic. In *Proceedings of the 6th International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE '11)*, 2011.

[55] R. Wojtczuk and C. Kallenberg. Attacking UEFI Boot Script. 31st Chaos Communication Congress (31C3), http://events.ccc.de/congress/2014/Fahrplan/system/attachments/2566/original/venamis_whitepaper.pdf, 2014.

[56] R. Wojtczuk and J. Rutkowska. Another Way to Circumvent Intelő Trusted Execution Technology. http://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf, December 2009.

[57] R. Wojtczuk and J. Rutkowska. Attacking Intel Trusted Execution Technology. http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf, February 2009.

[58] R. Wojtczuk and J. Rutkowska. Attacking SMM Memory via Intel CPU Cache Poisoning, 2009.

[59] R. Wojtczuk and J. Rutkowska. Attacking Intel TXT via SINIT Code Execution Hijacking. http://www.invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf, November 2011.

[60] R. Wojtczuk and A. Tereshkin. Attacking Intelő BIOS. https://www.blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf.

[61] F. Zhang, K. Leach, A. Stavrou, H. Wang, and K. Sun. Using Hardware Features for Increased Debugging Transparency. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15)*, May 2015.

[62] F. Zhang, K. Leach, K. Sun, and A. Stavrou. SPECTRE: A Dependable Introspection Framework via System Management Mode. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*, 2013.

[63] F. Zhang, K. Leach, H. Wang, and A. Stavrou. TrustLogin: Securing Password-Login on Commodity Operating Systems. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (AsiaCCS'15)*, 2015.

[64] F. Zhang, H. Wang, K. Leach, and A. Stavrou. A Framework to Secure Peripherals at Runtime. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14)*, 2014.

[65] F. Zhang, J. Wang, K. Sun, and A. Stavrou. HyperCheck: A Hardware-assisted Integrity Monitor. In *IEEE Transactions on Dependable and Secure Computing (TDSC'14)*, 2014.