SoK: 20 Years of Power, Privilege, and Peril in x86 System Management Mode

Antonis Louka and Jo Van Bulck

DistriNet, KU Leuven, 3001 Leuven, Belgium

Abstract

System Management Mode (SMM) is a highly privileged execution mode present in x86 processors since the early 1990s. With full control over hardware and system memory, SMM has long been a prime target for powerful attacks and persistent rootkits, including cases linked to nation-state adversaries. However, despite widespread deployment and unlike other x86 isolation mechanisms such as Intel SGX and AMD SEV, SMM has received relatively little academic attention, with insights primarily scattered across industry disclosures and hacker community contributions.

We present the first comprehensive systematization of SMM attack research, covering over two decades of work. We consolidate architectural details, trace the evolution of hardware and the firmware ecosystem, and introduce an extensible taxonomy of attack vectors. Organizing the landscape into three eras, we highlight a shift from configuration-based exploits to sophisticated software vulnerabilities and the growing importance of automated analysis tools. Finally, we identify knowledge gaps, outline research priorities, and distill lessons transferable to privileged firmware beyond x86.

1 Introduction

System Management Mode (SMM) is a specialized, highly privileged x86 operating mode designed to handle low-level system functions such as power management, thermal control, and hardware monitoring transparent to the operating system. SMM was first introduced in the early 1990s and remains a standard feature in modern Intel and AMD x86 processors [16, 86]. This mode is colloquially referred to as "ring -2," operating completely invisibly from the operating system (ring 0) or hypervisor (ring -1). Entry occurs via a dedicated System Management Interrupt (SMI), triggered by hardware or software events and handled outside normal interrupt mechanisms. The SMI-handler code runs from a protected memory region, System Management RAM (SMRAM), saving and restoring architectural state to resume the interrupted context transparently. Underscoring SMM's role as a foundational x86 technology, it serves as a critical security substrate in Unified Extensible Firmware Interface (UEFI) implementations, supporting widely deployed mechanisms such as secure boot [211], suspend to memory [213], and authenticated variables [214]. Academic designs have further leveraged SMM to provide security services to higher layers, including secure I/O [131,221] and live kernel patching [227], and early Intel research even explored using SMM as the basis for a trusted execution environment prior to Intel SGX [47].

Given its broad platform authority and opacity to system software, SMM has been an attractive target for offensive research over the past decades. Classified intelligence documents, leaked as part of the 2013 Snowden revelations, provide clear evidence that the NSA employed SMM-based rootkits for stealthy, below-the-OS monitoring as early as 2007 [149, 150]. Independent research from both the hacker community and academia has further demonstrated invisible, SMM-resident implants (e.g., rootkits and keyloggers) capable of monitoring or manipulating execution and influencing OS-visible state [67, 72, 124, 125, 176]. Researchers have also shown that persistent compromise can be achieved through firmware modification, including SPI flash rewriting and subversion of boot-time trust mechanisms [51, 120, 121, 204].

Despite over two decades of documented attacks, however, the literature on offensive SMM research remains highly fragmented. A substantial portion of contributions has appeared in the hacker community and industry (in the form of talks, blog posts, and proof-of-concept code repositories), with comparatively few formal academic publications. Moreover, public vulnerability disclosures and advisories often provide minimal technical detail, complicating reproduction and thorough root-cause analysis. As a result, it is challenging to reason systematically about longitudinal trends, recurring failure modes, and which defenses meaningfully address the SMM threat surface. Remarkably, compared to more recent x86 isolation extensions like Intel SGX/TDX and AMD SEV, SMM has received far less attention and scrutiny from academia. This dis-

parity can be attributed, in part, to the largely opaque, closed-source firmware ecosystem and dispersed documentation of SMM execution environments across processor generations, hindering the development of systematic evaluation infrastructure and creating a high entry barrier for new researchers.

In this paper, we systematically review two decades of academic publications, hacker community contributions, and industry advisories to synthesize a structured and comprehensive view of offensive research targeting SMM. As a first contribution, we demystify SMM by presenting a concise, security-oriented view of its execution model, isolation primitives, and feature evolution. We further clarify the opaque firmware ecosystem by overviewing modern hardening mechanisms drawn from Intel and AMD documentation and public engineering practice, while also highlighting real-world deployment and use cases of SMM in contemporary platforms.

As a second contribution, we systematically review 69 attack studies and 57 security advisories, synthesizing an extensible taxonomy of vulnerability root causes and organizing the offensive research landscape into three distinct eras. The first era (2006–2014) is dominated by platform initialization weaknesses, which have since largely been mitigated through the introduction of automated platform configuration assessment tools [134]. The second era (2014–2020) marks a shift toward low-level bugs in the SMI handler code, including traditional memory-safety vulnerabilities as well as more subtle confused-deputy shielding bugs arising from SMM's shared address space design. In the third era (2020–present), we observe a continued presence of low-level memory-safety and shielding bugs, alongside the growing adoption of automated analysis tools such as fuzzers.

Finally, as a third contribution, we generalize insights from two decades of research to identify knowledge gaps and highlight research priorities, including hardening and containment strategies for SMM, reproducible evaluation stacks with low entry barriers, open-source development frameworks and firmware datasets, and a call to increased attention for microarchitectural attack surfaces. We conclude by distilling transferable lessons for secure design of highly privileged firmware in the x86 ecosystem and beyond.

Contributions. In summary, our main contributions are:

- We demystify the design and operation of SMM, focusing on its execution model, isolation properties, and practical use in modern platforms.
- We systematize two decades of offensive SMM research, encompassing over 69 studies and 57 advisories, to develop a structured taxonomy of root causes and examine longitudinal trends across three evolutionary eras.
- We identify gaps and research directions, and we discuss transferable lessons for privileged firmware design.

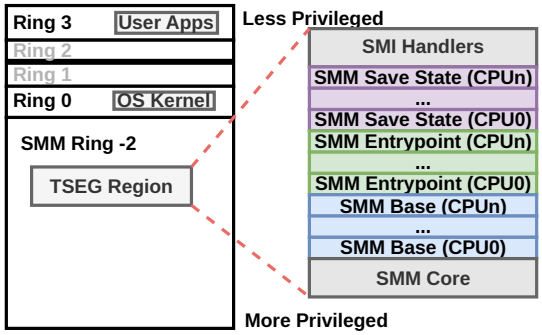

Figure 1: Position of SMM in x86's privilege hierarchy.

Open Science. To support reproducibility of our work and to facilitate future research, we open source the code to recreate our advisory and CVE datasets at https://github.com/antonislouca/SoK-SMM.

Paper Outline. The remainder of the paper is organized as follows. Section 2 demystifies SMM's execution model, isolation properties, and practical use in modern firmware stacks. Section 3 systematizes two decades of offensive SMM research, including an extensible taxonomy. Section 4 analyzes era-based trends and longitudinal findings. Finally, Section 5 distills gaps and research directions for strengthening the SMM ecosystem.

2 Demystifying SMM

Inspired by previous demystification analyses [55, 63, 154, 165] for other processor security features, this section aims to clarify SMM by positioning it within the system software stack, detailing its isolation model and evolution, and outlining its role in contemporary platforms.

2.1 SMM Isolation Properties

System Management Mode is a highly privileged x86 execution mode that operates outside the conventional privilege hierarchy (Fig. 1). It is invoked via an SMI, independently of any software-visible transitions: upon SMI delivery, the processor suspends the currently executing context (user, kernel, hypervisor, or guest) and transfers control to firmware-defined SMM handler code. Because this transition is entirely opaque to the interrupted software, SMM executes invisibly even to ring 0 and ring -1 code, resuming the preempted context only when the handler issues a dedicated `RSM` instruction.

Table 1 situates this behavior among other x86 hardware-assisted isolation mechanisms. Unlike software-managed isolation (e.g., the OS kernel) and hardware-enforced trusted

Table 1: Comparison of SMM versus other x86 memory isolation mechanisms.

Feature \ Mechanism	*x86 privilege levels*				*x86 trusted execution environments*			
	ME/ASP [36, 85]	**SMM**	**VMM**	**Kernel**	**Intel SGX [63]**	**Intel TDX [98]**	**AMD SEV [84]**	**Intel TXT [99]**
Release year	~2008/2013	~1990	~2005	1982	2015	2021	2017-2020	2008
Execution context	"Ring −3" (co-processor)	"Ring −2"	Ring −1	Ring 0	Ring 3 (enclave)	Ring 0–3 (guest)	Ring 0–3 (guest)	Early boot
Granularity	System	System	System	OS / VM	Enclave	VM (TD)	VM	Launch
Memory protection	Internal SRAM	SMRAM	Ext page tables	Page tables	PRM / EPC	PRM + TDX Module	Reverse Map Table	(TXT heap, launch data)
Memory setup	Platform Init	Boot	Boot	OS	OS	FW + TDX Module	FW + ASP + VMM	FW + Chipset
Entry	MEI/Mailbox protocol	SMI	VMEXIT	`SYSCALL` / IRQ	`EENTER`	`SEAMCALL`	`VMRUN`	`SENTER`
Exit	MEI/Mailbox protocol	`RSM`	`VMRUN/VMLAUNCH`	`SYSRET/IRET`	`EEXIT`	`SEAMRET`	VMEXIT	`SEXIT`
Code origin	Intel/AMD FW	FW	VMM	OS	User App	Guest	Guest	Authenticated SW
Trust	Root of Trust	CPU	SMM	SMM + VMM	CPU	CPU + TDX Module	CPU + ASP	CPU + Chipset + TPM

execution environments (TEEs) such as Intel Software Guard Extensions (SGX), Intel Trust Domain Extensions (TDX), and AMD Secure Encrypted Virtualization (SEV), SMM is provided with system-wide authority and can preempt any execution context. SMM is designed to perform privileged system management functions such as thermal control, power state transitions, and legacy hardware emulation (e.g., PS/2 devices), underscoring its orthogonal role relative to software- and virtualization-based isolation models.

2.1.1 Memory Isolation

SMM executes from a dedicated physical memory region, called SMRAM, which is isolated from *all* other x86 operating modes. SMRAM protection combines chipset-enforced controls, imposed by the memory-controller complex, with CPU-enforced isolation primitives (see Section 2.2). SMRAM contains the SMI handler code and its associated data, as well as the processor save-state area. Architectural state is saved on SMI entry and restored on exit via `RSM`, enabling transparent context switch. Beyond handler state, firmware may also store system-management metadata in SMRAM (e.g., platform configuration variables, S3 bootscript, or OEM-specific data).

SMRAM Initial Configuration. By default, SMRAM is a 64 KB region located at a physical base address specified by the processor's internal `SMBASE` register. Firmware may reprogram `SMBASE` during early initialization (or from within an SMI handler) to relocate the SMRAM region [84, 86]. A larger SMRAM area can also be used, if needed, varying from 32 KB to 4 GB. On SMM entry, the processor uses `SMBASE` to locate the SMI handler entry point and the save-state area within SMRAM. SMIs target the current `SMBASE` location; a relocated `SMBASE` resets to its default value on system reset. Once in SMM, firmware executes with system-wide privilege and can access platform resources beyond the OS and hypervisor, including main memory and device state.

SMRAM can be placed in different reserved memory regions depending on the platform, commonly the compatible, high, and top-of-memory segments. Historically, legacy BIOS implementations favored the compatible segment, whereas modern UEFI firmware typically allocates SMRAM from the high or top-of-memory segments to obtain a larger, cacheable region suitable for more complex SMM drivers. Once initialized, the firmware locks the allocated SMRAM region during boot, rendering it inaccessible to all execution contexts outside SMM.

SMRAM Protection and Configuration. Firmware must lock the platform memory-map configuration to prevent remapping attacks that could expose or overlap SMRAM [66, 68, 147, 174]. This includes registers that define usable DRAM boundaries and remap windows, which should be programmed early and then locked. Firmware should also configure and lock DMA-protection settings for the SMRAM region to prevent peripherals from targeting protected memory [203, 207]. When SMRAM is allocated from cacheable regions such as the high or top-of-memory regions, the CPU's System Management Range Registers (SMRRs) should be configured to define and protect the SMRAM range, mitigating SMRAM cache poisoning attacks [205].

2.1.2 Controlled Entry Points

Entering SMM. Entry into SMM occurs exclusively via delivery of an SMI, triggered through the `SMI#` pin or the Advanced Programmable Interrupt Controller (APIC). SMIs are non-maskable and operate outside the processor's normal interrupt and exception handling. SMM is non-reentrant, and additional SMIs are disabled while executing in SMM. Outside of SMM an SMI takes priority over both maskable interrupts and non-maskable interrupts (NMIs). On SMI delivery, the processor waits for in-flight instructions to retire and pending stores to complete, saves architectural state in SMRAM, transitions into SMM, and begins executing the firmware-defined SMI handler.

SMM entry is rooted in legacy x86 execution semantics: early SMM implementations operated entirely in 16-bit real mode, leaving the SMI handler responsible for any necessary

mode transitions. Modern Intel platforms address this through a dedicated *SMM Entry Code* binary. This trusted code stub, executed on every SMM entry, is responsible for transitioning the processor to protected mode, enabling paging, activating long mode, and configuring Intel-specific runtime defense mechanisms before handing off to the SMI handler.

Exiting SMM. SMM can be exited only via the `RSM` instruction, which is valid exclusively in SMM; executing `RSM` outside of SMM raises an invalid-opcode exception. On execution, `RSM` restores the processor state saved in SMRAM and resumes the previously interrupted context.

2.2 Evolution of SMM

System Management Mode was introduced by Intel in the early 1990s, and as x86 platforms grew in complexity and security relevance, SMM evolved from a minimal compatibility mechanism into a hardened, hardware-assisted firmware execution environment. Table 2 summarizes this progression, highlighting incremental architectural and firmware evolution of SMM emphasizing security additions and how they may be related to attack primitives. For UEFI, the SMM services are initialized during boot, by a driver execution environment that loads the different SMM services into SMRAM.

2.2.1 SMRAM Access Control

Baseline SMM. Early SMM implementations, first appearing in Intel i386SL processors, established a firmware-controlled execution mode entered via an SMI and exited through `RSM`. SMM code executed from SMRAM, with the processor saving and restoring architectural state on entry and exit to preserve transparency. At this stage, SMM primarily served platform management tasks, without any explicit isolation primitives or standardization over them.

System Management RAM Control. As SMM became more widely deployed, early hardening efforts focused on adding explicit, chipset-mediated controls over SMRAM visibility and placement. Chipsets introduced the System Management RAM Control (`SMRAMC`) register, exposing control bits allowing firmware to configure when SMRAM is inaccessible outside SMM from less privileged execution modes. During platform initialization, firmware temporarily enables access to initialize SMM state and seals the configuration by making the `SMRAMC` register read-only until system reset. These controls are mutually exclusive and must be programmed in a strict order. While these mechanisms reduced accidental exposure and improved configurability, they are dependent on correct firmware boot-time initialization.

CPU-Enforced Isolation. The introduction of System Management Range Registers shifted SMRAM protection from chipset-level to CPU-enforced isolation. On processors that support SMRRs, firmware can configure the CPU to treat the SMRAM address range as inaccessible outside SMM, independent of normal memory-type settings or remapping behavior. SMRRs are hardware registers that define the memory range used by the SMRAM. They are implemented using two model-specific registers (MSRs) to define and protect SMRAM, establishing its base address and memory protected range. The values of the MSRs are configured during early boot, superseding the configuration of standard MSRs, and are locked to permit modifications only from within SMM.

SMRRs, once properly configured, prevent even ring-0 or hypervisor-level code from reading or writing SMRAM. However, their effectiveness depends on correct early-boot configuration, motivating firmware hardening measures such as early SMRR programming, pre-boot SMRAM lock-down, initial configuration locking, and cache-safe initialization to mitigate remapping and cache-poisoning attacks.

2.2.2 SMM Software Hardening

Static Paging. Modern platforms increasingly harden SMM by enforcing a *static* paging configuration that prohibits runtime page-table modifications. In this model, the SMM page table is established once during boot and remains fixed until the next reboot. Page-table pages are marked read-only to prevent tampering; code pages are read-only and confined to SMRAM; data pages are non-executable, with read-only protection applied to pages that require no runtime modification. Memory outside SMRAM that is not required by the SMI handler is left unmapped, reducing SMM's exposure to OS-owned memory and limiting the impact of SMI handler vulnerabilities. Intel reports contributing this *Runtime BIOS Resilience* paging approach to open-source UEFI implementations [62], aiming to raise the baseline for SMM security.

SMM Virtualization and Runtime Containment. A recurring challenge in SMM security is that traditional designs implicitly trust SMI handlers despite their history of exploitable vulnerabilities. Intel's SMI Transfer Monitor (STM) [65, 103, 212] introduced an early containment model motivated by attacks [124, 126, 204] showing that compromised SMM can undermine Intel Trusted Execution Technology (TXT) and measured-launch guarantees [99]. During boot, firmware reserves a dedicated memory region within SMRAM, the *Monitor Segment*, to host the STM image. Under a Measured Launch Environment, STM establishes a two-layer setup in which the hypervisor virtualizes the OS while the STM contains and virtualizes SMI handler execution [65].

While STM saw limited adoption, modern platforms increasingly pursue related runtime restriction mechanisms that deprivilege SMI handlers and enforce resource-access policies. On AMD platforms [36, 73], an SMM Supervisor is loaded during UEFI boot and authenticated by the AMD Plat-

Table 2: Evolution of SMM hardware (HW) and firmware (FW) defense mechanisms.

SMM Change	Era	Description	HW	FW
Baseline SMM	1990s (Intel i386SL)	SMI/RSM, SMRAM, and state save/restore	●	○
SMRAMC & SMM locking	Mid–late 1990s (i486)	Chipset-controlled SMRAM & relocatable SMBASE	●	○
CPU SMRAM isolation	Early–mid 2000s	SMRR SMRAM protection (CPU-enforced)	●	○
Expanded state save map	Mid-2000s (x86-64 era)	Extended SMRAM save-state formats for 64-bit CPUs	●	○
SMM-SMEP	2014 (Intel 4th Gen)	Mitigations against SMM callouts	●	○
Intel STM	∼2015	Deprivilege SMI handlers	●	●
SMM paging in EDK II	2016	Static SMM page tables with defensive memory attributes	○	●
Runtime BIOS Resilience	2018 (Intel 8th Gen)	Paging setup, long mode and lockdown	○	●
System Security Report	2019 (Intel 9th Gen)	OS-visible reporting of below-the-OS security properties	○	●
System Resources Defense	2020 (Intel 10th Gen)	Policy-based containment and deprivileging SMI handlers	●	●
AMD SMM Supervisor	2022	Deprivileging and policy mediation for SMI handlers	●	●

form Security Processor, after which SMI delivery transitions into a small ring-0 core that deprivileges handlers to ring 3. Deprivileged handlers request privileged operations via a system-call interface, and the supervisor mediates access to MSRs, I/O ports, and memory regions according to a platform-defined policy. In contrast, Intel Hardware Shield [61] enforces SMM access policies using CPU and firmware mechanisms and, on newer systems, supports attestation of SMM memory configuration via a signed SMM module integrated with Intel TXT.

2.3 SMM in Practice

SMM was initially designed to support platform-management tasks that require firmware-controlled and OS-transparent access to hardware. Typical uses include power and thermal management, low-level hardware control, and additional OEM-specific management logic. SMM handlers may also provide firmware information interfaces (e.g., SMBIOS), legacy device emulation (PS/2 keyboard and mouse), and mediation of platform I/O events during boot and at runtime, including handling selected USB events for policy enforcement or compatibility.

Modern x86 platforms employ a layered firmware stack in which UEFI [191] provides the primary boot and runtime framework (with TianoCore EDK II [188, 190] as its open-source reference implementation), alongside vendor-specific co-processors such as Intel's Management Engine (ME) [85] and the AMD Platform Security Processor (ASP) [36]. This subsection explores how SMM is used and hardened in practice to support security-critical UEFI services, as well as secure boot and update mechanisms.

2.3.1 SMM Usage in UEFI

SMM Hardening Practices. SMM poses an attractive attack target due to its high privilege and system-wide authority, and prior work demonstrates that misconfigured or overly permissive SMM components can introduce serious vulnerabilities. UEFI firmware stacks such as EDK II, therefore, provide hardening mechanisms that allow developers to strengthen SMM beyond baseline architectural guarantees.

EDK II applies SMM memory protections that reduce code-injection and data-corruption risks, including read-only and non-executable page attributes, static page tables, and controlled memory layouts. Further measures include a dedicated SMM *communication buffer* API [215] to avoid confused-deputy pointer dereferences, intra-SMM ASLR to increase the difficulty of control-flow exploitation, and guard pages to enable detection of stack and heap overflows at runtime [216]. Finally, *SMI handler profiling* assists developers in auditing and reducing the set of required handlers, thereby minimizing the active SMM attack surface.

Authenticated Variables. Authenticated variables protect security-critical firmware settings through digital signatures, requiring an execution environment isolated from the OS to prevent malicious software from bypassing verification or tampering with variable storage directly. On x86 platforms, SMM provides this isolation [214], enabling signature verification and controlled updates of UEFI authenticated variables.

S3 Save State. In the S3 sleep state (suspend-to-RAM), OS and firmware context remain resident in DRAM and must be restored on wake, making the resume path sensitive to memory tampering. An attacker who can modify the S3 boot script can compromise firmware execution during resume, e.g., by restoring incorrect register values or leaving critical hardware protections unlocked. To mitigate this, UEFI platforms rely on SMM to protect firmware context during S3 resume [213]: in EDK II, SMM provides "LockBox" services to the S3 bootscript to preserve the integrity of resume-critical state across suspend and resume, ensuring correct platform restoration.

Profiling Features. UEFI provides profiling capabilities to help developers analyze and validate SMM behavior [217]. Memory profiling tracks per-module allocation during boot to detect memory leaks in UEFI and SMM drivers; perfor-

mance profiling records execution-time metrics used to populate ACPI data structures; and SMM profiling monitors code and data accesses from SMM to regions outside SMRAM, enabling detection of isolation violations.

2.3.2 Secure Boot and Firmware Updates

On x86 platforms, establishing firmware authenticity before OS execution relies on an initial trusted computing base. Intel Boot Guard [209] uses a processor-validated Authenticated Code Module to verify early-boot firmware before transferring control to the UEFI stack, where UEFI Secure Boot [211] then authenticates the OS loader. To preserve integrity across these flows, critical verification logic and policy databases are staged in SMRAM.

UEFI further leverages SMM as an isolated and privileged environment to perform secure verification and flashing of standardized Capsule firmware updates applied by third-party code (e.g., the OS). Intel BIOS Guard [210] further hardens this architecture by employing an SMM-based agent that authenticates OEM-signed update packages using platform-provisioned keys and mediates all writes to the SPI Flash, ensuring that potentially compromised code in the update path cannot bypass flash protection.

2.4 Summary

In summary, this section positioned SMM within the x86 software stack and clarified its core execution semantics, memory model, and isolation properties. We highlighted how SMM has evolved from a low-level platform-management mechanism into a security-critical firmware component, and explored how modern systems running in the UEFI ecosystem both rely on and harden SMM in practice.

3 A 20-Year Retrospective of SMM Security

In this section we provide a systematization outlining the evolution of the SMM attack landscape over the past two to three decades. Although dedicated security research into SMM was sparse prior to 2006, early evidence of its impact appeared in a 2004 Linux kernel report, which documented SMM implementation bugs responsible for system crashes [164]. Declassified intelligence documents also demonstrate SMM as a realistic target, specifically the NSA Advanced Network Technology Catalog and the Tailored Access Operations projects [37, 149, 150], reveal that sophisticated state actors had already weaponized the mode for platform subversion. Implants in these documents, demonstrate how the NSA weaponized SMM between 2004 and 2008 to maintain persistent surveillance on Dell servers, and Juniper network routers. These examples prove that using SMM as a real-world attack vector for stealthy monitoring has already existed before it became a major focus by the security research community.

The remainder of this section reviews the public literature on SMM attacks over the past two decades, organizes recurring attack surfaces into an extensible taxonomy, and surveys public CVE records and security advisories.

3.1 Literature Review and Methodology

System Management Mode has been a recurring target in offensive firmware research, yet the corresponding body of knowledge remains fragmented and difficult to navigate. Unlike many systems security topics that are documented through manuals and peer-reviewed academic venues, a substantial portion of SMM-related work appears in practitioner talks, blog posts, and informal media such as recorded presentations, often with uneven levels of detail, varying technical depth, and inconsistent terminology. The tight coupling between SMM behavior, platform-specific firmware, and vendor documentation further steepens the learning curve for reproducing results and comparing techniques across publications.

To systematize the offensive SMM landscape, we consolidate the available public record into a structured literature review and normalized encoding scheme. Table 3 on page 8 presents our systematization, where each entry corresponds to a distinct offensive contribution (e.g., an attack, vulnerability analysis, proof-of-concept, framework, or analysis tool) and is annotated with a standardized set of labels to support longitudinal comparison across time, research communities, vulnerability classes, artifact availability and the degree of automation used.

Inclusion Criteria. We curated a set of public works from 2006–2026 spanning peer-reviewed papers, hacker conference presentations (e.g., Black Hat, CanSecWest, REcon), industry technical reports, and practitioner writeups on SMM and related firmware attack surfaces. Academic works were identified via Google Scholar; conference materials and proof-of-concept artifacts were collected from public repositories and community-curated resources including DarkMentor [123] and the CasualHacking UEFI compendium [79], supplemented by their citation networks.

We exclude generic UEFI-only attacks that do not materially involve SMM, except where SMM is required for exploitation, persistence, or bypass of platform trust mechanisms. We include works that *(i)* directly target SMM or leverage it to compromise other platform components (e.g., boot–time misconfiguration of protection bits or runtime SMI-handler flaws); *(ii)* exploit platform mechanisms whose failure enables SMM compromise (e.g., SMRAM remapping, S3 boot-script flows, TXT INIT); or *(iii)* provide proof-of-concept demonstrations of SMM-adversary capabilities.

Labeling Scheme. We encode each work along four dimensions: (1) *era*, (2) *vulnerability/attack class*, (3) *artifact availability and exploit completeness*, and (4) *automation*.

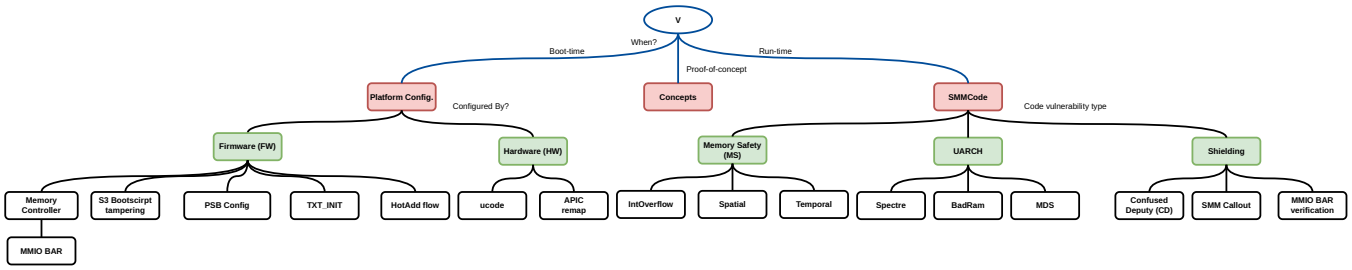

Figure 2: SMM attack classification tree used for the systematization in Table 3.

The *vulnerability* label captures the primary attack surface using a compact taxonomy derived from a classification tree (Fig. 2) developed during our literature review. This tree abstracts specific exploitation techniques into underlying failure modes (e.g., configuration, isolation/shielding, memory safety, microarchitectural leakage), enabling consistent comparison across heterogeneous sources, while supporting trend and gap analysis. Section 3.2 summarizes the resulting top-level categories, which are designed to be extensible so future work can be classified within the same framework.

To capture reproducibility and practical maturity, Table 3 includes three additional indicators. The *src* column denotes artifact availability (●: open source; ◐: partial release; ○: not available; citation: availability in an external repository). The *E2E* column indicates whether the work demonstrates an end-to-end exploit on real hardware (from vulnerability to successful exploitation). The *auto* column captures whether discovery or validation relied on automated tooling analysis rather than purely manual reverse engineering effort.

Era Boundaries. We partition the literature into three eras that reflect shifts in dominant vulnerability classes and tooling maturity. Era 1 (2006–2014) is dominated by platform configuration weaknesses and manual exploitation, culminating in the introduction of CHIPSEC [134] as a systematic assessment framework. Era 2 (2014–2020) exhibits a surge in SMM code vulnerabilities and increased reproducibility, with code analysis tooling such as efiXplorer [138] emerging as a community standard. Era 3 (2020–2026) is characterized by persistent SMM code vulnerability classes, growth in automated analysis through fuzzing, and the emergence of AMD platform configuration issues as tooling improves.

3.2 Attack Taxonomy

Table 3 labels each offensive work using a compact taxonomy that captures *where* the weakness resides and *what* exploitation surface it targets. The taxonomy is descriptive where many works span multiple categories, assigning the primary classes of vulnerabilities per entry. The taxonomy below summarizes the most important nodes of our classification tree of Fig. 2 used to label Table 3.

Platform Configuration. This category captures vulnerabilities caused by insecure or inconsistent platform configuration, typically set during boot and enforced by firmware or hardware. Representative issues include incorrect SMRAM/SMM access-control and locking (e.g., memory-controller configuration and lockdown bits) and incomplete protection during sensitive firmware flows such as S3 boot-script state restoration. These weaknesses can enable SMM compromise without a flaw in SMI handler code; instead they rely on exploiting misconfigured policies and incomplete lockdown sequences.

SMM Code Vulnerabilities. This category covers vulnerabilities within SMI handlers and other SMM-resident code. They range from memory-safety bugs (e.g., buffer overflows and use-after-free) to unsafe handling of attacker-controlled inputs and missing shielding checks. Many such flaws manifest as *confused-deputy* behavior, where unprivileged software coerces SMM into performing security-sensitive actions on its behalf. Common sub-patterns include attacker-controlled pointer dereferences in the shared address space and callouts from SMM into untrusted memory, resulting in spatial or temporal memory corruption. These vulnerabilities increasingly dominate as UEFI-era SMM codebases grow and become more modular, motivating stronger engineering practices, and exploration of memory-safe implementations (e.g., Rust-based UEFI components [163]).

Microarchitectural Attacks. This category captures attacks that exploit microarchitectural effects in or across SMM, such as transient-execution style leakage. While comparatively rare in the public literature, these works are important because they test whether CPU-wide mitigations and isolation assumptions remain intact during SMM transitions and under its highly privileged execution context.

3.3 CVE and Vendor Advisory Review

CVE review. Before presenting our era analysis, we provide a preliminary view using publicly available Common Vulnerabilities and Exposures (CVE) data. We download the

Table 3: Systematization of the SMM offensive landscape over the last ~20 years, split into 3 distinct eras. Columns list year and publication venue (including contributions from academia, industry, and the hacker community); vulnerability classification according to the taxonomy in Fig. 2; whether public exploit code is fully (●) or partially (◐) available in the original or a subsequent publication; whether a full (●) or partial (◐) end-to-end exploit was demonstrated; and whether automated tools were used to discover this vulnerability.

	Name	Year	Venue	Vulnerability	Src	E2E	Auto
Era 1	SMM OS Security Bypass [67]	2006	CanSecWest	*Plat-config→FW→Memory-Controller*	◐	○	○
	SMM Rootkits [72]	2008	SecureComm	*Plat-config→FW→Memory-Controller*	○	●	○
	SMM hacks [48]	2008	Phrack	*Plat-config→FW→Memory-Controller*	●	●	○
	Xen Hypervisor Subversion [174]	2008	Black Hat	*Plat-config→FW→Memory-Controller*	◐	○	○
	SMRAM Access Techniques [136]	2009	CanSecWest	*Plat-config→FW→Memory-Controller*	○	○	○
	BIOS SMI Rootkit [57]	2009	Phrack	*Concept*	●	◐	○
	Attacking Intel TXT [204]	2009	Black Hat	*Plat-config→FW→Memory-Controller*	○	●	○
	Attacking Intel BIOS [202]	2009	Black Hat	*SMM-code→Shielding→CD*	○	○	○
	SMM CPU Cache Poisoning [205]	2009	White Paper	*Plat-config→FW→Memory-Controller*	○	●	○
	Attacking SMM using SINIT [206]	2011	White Paper	*Plat-config→FW→TXT SINIT*	○	●	○
	Flea-reflash hopping [51]	2013	ACM CCS	*Concept*	○	●	○
	Insyde SMM Vulnerabilities [77]	2013	NCC Blog	*SMM-code→[Shielding→CD | MS→Spatial]*	○	○	○
	Sandman [125]	2014	Mitre	*Plat-config→FW→Memory-Controller*	○	◐	○
	Chipsec [134]	2014	CanSecWest	*[Plat-config | SMM-code]*	●	○	●
Era 2	SMM Rootkits Revisited [176]	2014	ARES	*Concept*	○	○	○
	Defeating Secure Boot [121]	2014	Syscan	*Plat-config→FW→Memory-Controller*	○	◐	○
	Defeating Signed BIOS [120]	2014	Blog	*SMM-code→MS→Spatial*	○	●	○
	The Watcher [122]	2014	Mitre	*SMM-code→MS→IntOverflow*	[179]	●	○
	Smite'em MiTM and Copernicus 2 [124]	2014	Mitre	*[Concept | SMM-code]*	[80]	◐	●
	UEFI Security Attacks [203]	2014	31C3	*[Plat-config→FW→S3BootScript | SMM-code→[Shielding→SMMCallout]]*	○	●	○
	Venamis misery [207]	2014	White Paper	*Plat-config→FW→S3BootScript*	[156]	●	○
	Excite [43]	2015	WOOT	*SMM-code→Shielding→SMMCallout*	○	○	●
	SMM Backdoor [157]	2015	GitHub	*Concept*	●	○	○
	Attacking VMM through SMM [76]	2015	Black Hat	*SMM-code→Shielding→CD*	○	○	●
	SMM Memory Sinkhole [66]	2015	Black Hat	*Plat-config→HW→APICRemap*	◐	●	○
	A Million BIOSes Infected (Lighteater) [60]	2015	HITBSecConf	*SMM-code→[Shielding→SMMCallout | MS→Spatial]*	○	○	○
	Breaking UEFI Security Using DMA [155]	2015	Blog	*SMM-code→Shielding→CD*	○	●	●
	SMI Handler Vulnerabilities [41]	2015	CanSecWest	*SMM-code→Shielding→CD*	○	○	●
	Aptiocalypsis [158]	2016	Blog	*SMM-code→Shielding→CD*	[159]	●	●
	EDK2 communication [215]	2016	White Paper	*SMM-code→Shielding→[CD | SMMCallout]*	○	○	○
	ThinkPwn/SmmRuntime [161]	2016	Blog	*SMM-code→Shielding→[CD | SMMCallout]*	[162]	●	●
	Exploiting SMM callout vulnerabilities [160]	2016	Blog	*SMM-code→Shielding→SMMCallout*	○	●	●
	SMM Data Centers [93]	2017	CanSecWest	*Plat-config→FW→HotAddFlowDMA*	○	●	○
	LONGKIT Framework [170]	2017	ICISSP	*Concept*	○	○	○
	BARing the System [50]	2017	RECON	*SMM-code→Shielding→CD→BAR*	○	●	●
	BIOS holes. Don't Come Inside [70]	2017	Black Hat	*SMM-code→Shielding→SMMCallout*	[71]	●	●
	Digging Into the Core of Boot [49]	2017	RECON	*[Plat-config→FW→Memory-Controller | SMM-code→Shielding→CD]*	○	○	●
	Exploring Your System Deeper [42]	2017	CanSecWest	*[Plat-config→[FW→Memory-Controller | S3BootScript] | SMM-code→[Shielding→CD]]*	●	○	●
	UEFI firmware rootkits [137]	2017	Black Hat	*SMM-code→Shielding→CD*	○	●	○
	SMM Spectre v1 [69]	2018	Blog	*SMM-code→Uarch→Spectre*	●	●	●
	Code Check(mate) [167]	2018	Blog	*SMM-code→Shielding→SMMCallout*	◐	◐	●
	efiXplorer [138]	2020	Black Hat	*SMM-code→[Shielding→[CD | SMMCallout] | MS→[Temporal | Spatial]]*	[44]	○	●
Era 3	Attacking the golden ring [151]	2020	Blog	*SMM-code→Shielding→CD*	●	●	○
	Firmware Fuzzing using Simics [208]	2020	DAC	*SMM-code→[Shielding→[CD] | MS→Spatial]*	○	○	●
	SMRAM Callout Vulnerability [168]	2020	Blog	*SMM-code→Shielding→SMMCallout*	◐	◐	●
	SPENDER [219]	2022	IEEE S&P	*SMM-code→Shielding→CD*	○	○	●
	SPI Write Protections [38]	2022	Blog	*[Plat-config→[FW→Memory-Controller]]*	○	○	●
	HP Repeatable Failures [185]	2022	Blog	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	●	●	●
	AMI Repeatable Failures [184]	2022	Blog	*SMM-code→Shielding→[SMMCallout | CD]*	○	●	○
	23 High-Impact Vulnerabilities [183]	2022	Blog	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	●	○	●
	UDK-Based SMM Rootkits [81]	2022	Blog	*Concept*	[82]	◐	○
	SMM Supervisor Security [193]	2022	Hardwear.io	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	○	◐	○
	AMD platform Security [144]	2022	IOActive Blog	*Plat-config→FW→Memory-Controller*	●	●	●
	BIOS Past, Present,Future [45]	2022	Black Hat	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	●	●	●
	Breaking firmware trust [139]	2022	Black Hat	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	○	●	●
	Lenovo High-impact Vulnerabilities [173]	2022	ESET Blog	*SMM-code→[Shielding→CD]*	◐	●	●
	Simple SMM Rootkit [181]	2022	icWCSN	*Concept*	●	○	○
	The art of SMM bug hunting [53]	2022	Blog	*SMM-code→[Shielding→[CD | SMMCallout] | MS→Spatial]*	○	○	●
	Ringhopper [220]	2023	DEF CON	*[SMM-code→[Shielding→CD]]*	○	●	○
	RSFuzzer [218]	2023	IEEE S&P	*SMM-code→[MS→[Spatial | Temporal] | Shielding→[CD | SMMCallout]]*	○	○	●
	Ice Lake TOCTOU [78]	2023	NCC Blog	*SMM-code→[MS→Spatial | Shielding→CD]*	○	○	○
	Back to the Future [146]	2023	IOActive Blog	*Plat-config→FW→Memory-Controller*	[119]	○	●
	Platform Security Disclosure [145]	2023	IOActive Blog	*SMM-code→[Shielding→CD]*	[119]	●	●
	Stase [180]	2024	ASE	*SMM-code*	●	○	●
	AMD Sinkclose [147]	2024	DEF CON	*SMM-code→Shielding→CD*	[119]	●	●
	AMD Platform Secure Boot [152]	2024	IOActive Blog	*Plat-config→FW→PSBStatus*	[119]	●	●
	FuzzUEr [75]	2025	NDSS	*SMM-code→[MS→[Spatial | Temporal] | Shielding→CD]*	●	○	●
	Memory corruption in SMM [199]	2025	Black Hat	*SMM-code→[MS→[Spatial | Temporal] | Shielding→[CD | SMMCallout]]*	○	○	●
	SmuFuzz [201]	2026	IEEE S&P	*SMM-code→[MS→[Spatial | Temporal] | Shielding→[SMMCallout]]*	◐	○	●

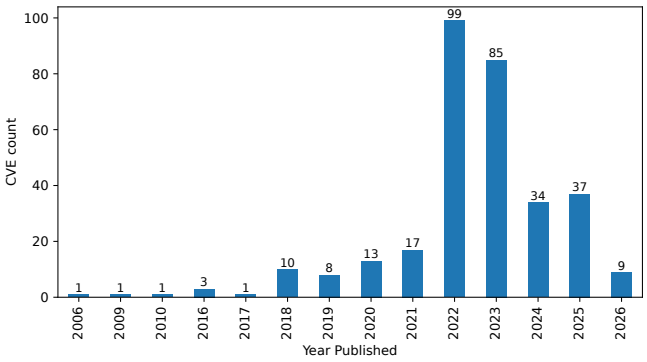

Figure 3: Historical analysis of SMM-related CVEs over time.

full CVE corpus and automatically filter entries with SMM related keywords in their description using regular expressions, yielding 319 CVEs. We then aggregate SMM-related CVEs by publication year and plot the result in Fig. 3, revealing a marked increase in reported SMM-related vulnerabilities over the past decade. Note, however, that because CVE publication dates are shaped by disclosure coordination and batch reporting, these counts are best interpreted as a proxy for reporting and analysis capacity rather than as a reliable indicator for underlying vulnerability prevalence or impact [177].

Vendor Advisory Review. To complement the rest of our analysis we create a CPU vendor-facing perspective, by reviewing the available Intel advisory and AMD bulletin lists. We crawl Intel Security Advisories and AMD Security Bulletins and filter for entries mentioning SMM keywords in the advisory or linked software guidance text. This process yielded 57 relevant advisories/bulletins. We then manually classified each entry using the classification tree in Fig. 2 to normalize vendor-specific wording into comparable attack classes. We summarize the resulting advisory set in Table 4 in Appendix A. While advisories vary in technical depth and disclosure practice, they provide an independent lens that is broadly consistent with the trends identified in the rest of our literature review and era analysis detailed in the next section.

4 Trends Across SMM Eras

Using the taxonomy introduced in Section 3.2, we analyze longitudinal trends in offensive SMM research across three eras. Our era boundaries align with shifts in dominant vulnerability classes and tooling maturity. We use Table 3 to characterize each era in terms of dominant attack surfaces, reproducibility, and the extent of automation. We also provide information where each work was published, demonstrating academic, industry, or practitioner efforts.

4.1 Era 1: Exploiting Platform Configuration

The first era is dominated by *platform configuration* weaknesses, where compromise is enabled primarily by misconfigured or insufficiently locked platform state rather than bugs in SMI handler code. We observe that early works of this era demonstrate how improper SMRAM access control, remapping, and cache-policy configuration can expose SMM with system-wide impact, enable end-to-end attacks [48, 67, 72, 125, 136, 174, 204–206] and early SMM-resident rootkit concepts [51, 57]. In this era we notice a recurring pattern of exploitation that hinges on boot-time policy errors (e.g., leaving SMRAM accessible or failing to lock configuration registers), which an attacker can later leverage to gain execution, modify SMRAM contents, or weaken its protections.

Methodologically, this era is characterized by manual reverse engineering and platform-specific validation, with limited open-source releases but several end-to-end demonstrations on real hardware as seen in Table 3. Discoveries of this era illustrate the consequences of weak platform lockdown in the presence of a highly privileged firmware execution mode.

A key inflection point is the introduction of CHIPSEC [134], which automates platform security assessment by providing reusable primitives for inspecting chipset configuration and firmware-controlled security settings. We observe that although not SMM-specific, CHIPSEC becomes the dominant tool for auditing and testing the misconfiguration classes of this and later eras (particularly on Intel platforms) and is subsequently used extensively for evaluation and exploit validation in Era 2.

> **Take-Away #1:** Boot-time platform configurations play a critical role as misconfigurations persist into run-time and enable compromise, even without SMM code vulnerabilities.

4.2 Era 2: Breaking SMM Code Apart

In Era 2 (2014–2020), we record a transition from sporadic platform misconfiguration issues to predominantly SMM code vulnerabilities and offensive malicious functionality concepts, which become the dominant theme of this period.

Compared to Era 1, this period provides more reproducible artifacts and demonstrations. A larger fraction of works include end-to-end exploitation examples and partial or full proof-of-concept releases, enabling independent reproduction and follow-up research. At the same time, automation becomes more visible, as CHIPSEC is used extensively for configuration auditing and exploit validation.

Tooling development remains relatively sparse but begins to solidify around a small set of widely used building blocks. Alongside CHIPSEC, Intel's Excite [43] project represents an early attempt to apply symbolic execution analysis techniques to firmware and SMM codebases. By the end of Era 2,

Binarly's efiXplorer [138] emerges as a practical static analysis workflow integrated into mainstream reverse-engineering environments (e.g., IDA), lowering the barrier for systematic inspection of SMM-heavy firmware images and helping drive the subsequent surge of SMM code vulnerability discovery in Era 3.

> **Take-Away #2:** Automated tooling, open-source writeups and artifacts demonstrate a shift from boot-time configuration to run-time SMM code issues.

4.3 Era 3: Towards Automated Validation

In Era 3 (2020–2026), we observe that SMM code vulnerabilities dominate the public offensive record, driving increased emphasis on program analysis. Works in this period repeatedly report memory-safety and isolation/shielding flaws in SMI handlers, and we see many reoccuring vulnerability classes over multiple years with limited variation. This persistence suggests that portions of SMM code remain insufficiently tested and that well-understood bug patterns continue to reappear in production firmware.

We also observe a growing role of automation in vulnerability discovery. Beyond static analysis, multiple efforts mostly academic [75, 180, 199, 201, 208, 218, 219] apply fuzzing and symbolic execution to firmware and SMM components, reflecting broader shifts toward scalable analysis. In parallel, CVE evidence (cf. Fig. 3) indicates a marked increase in reported SMM-related vulnerabilities from 2018–2026 with recurring categories. The same trend in recurring vulnerabilities is also present in our advisory analysis (cf. Table 4). We interpret this surge conservatively as a natural result due to improved analytical capability and tool availability.

Finally, we highlight cross-vendor asymmetry in assessment maturity. We find that AMD platform misconfiguration vulnerabilities become prominent roughly 15 years after analogous issues were widely explored on Intel platforms, where they remain visible to this day (cf. Table 4), suggesting delayed discovery driven in part by limited tooling and ecosystem attention. The emergence of cross-platform assessment tooling [119] enables more systematic evaluation and surfaces misconfiguration classes reminiscent of earlier Intel-era findings. Together, these trends motivate continued investment in open, scalable analysis tooling and evaluation methodologies for SMM across vendors.

> **Take-Away #3:** Recurring SMM code bug classes, and platform misconfiguration issues seen in Table 3 and Table 4 underscore the need for scalable testing and evaluation.

5 Gaps and Research Directions

Our era analysis indicates that offensive SMM progress is strongly shaped by analysis tooling and the visibility it enables. At the same time, deployed hardening remains difficult to scrutinize externally, and some threat models receive limited attention. We therefore distill gaps and research directions along six axes: (1) SMM hardening and isolation, (2) open evaluation tooling and reproducibility, (3) community coordination, (4) SMM as a defensive substrate, (5) microarchitectural security, and (6) cross-architecture transferability.

5.1 SMM Hardening and Isolation

Across our era analysis and advisory-driven evidence, we find that offensive SMM compromises are dominated by two broad vulnerability categories. First, *platform-configuration* failures that are prominent in earlier eras but become less common in the latest era as platforms adopt stricter initialization disciplines. Second, *SMM code vulnerabilities*, especially *memory-safety* and *shielding* vulnerabilities which remain prevalent and recur across years and vendors. This persistence suggests that SMM code vulnerabilities pose a long-term risk for the SMM ecosystem, and that there is a tooling adoption gap [134, 138] by firmware vendors.

SMM Hardening. From a hardening perspective, these observations point to two complementary directions. The first is reducing the prevalence and exploitability of memory corruption through memory-safe firmware implementations (e.g., Rust-based UEFI efforts [163]), addressing common vulnerabilities such as buffer overflows. Memory-safe languages alone, however, do not address confused-deputy and shielding vulnerabilities [35, 192]; the second direction is, therefore, the *containment* of SMM through deprivileged SMI handlers.

SMM Containment. In practice, both Intel and AMD pursue containment-style designs that enforce policy over SMI handler behavior, shifting from static primitives (e.g., register locking) toward runtime control over what SMM code may access and execute. Intel's "below-the-OS" security initiatives [61] frame this as a combination of paging-based protections and register locks, coupled with trust and measurement mechanisms that provide verifiable platform state.

On Intel platforms, practitioner analysis describes Intel System Resource Defense as an SMM isolation design that deprivileges SMI handlers by running most SMM code in a user-mode-like context, while a small trusted entry component mediates privileged operations [182]. This closely matches a "supervisor" model in which a minimal privileged core enforces policy and brokers MSR, I/O, and memory accesses for deprivileged handlers. Intel's public materials [62, 73] further emphasize reporting and attestation channels (e.g., Intel System Security Report) that provide OS- or loader-visible evidence of below-the-OS configuration. Because non-SMM software cannot read SMRAM or directly inspect active SMM configuration policies, SMM must expose its security-relevant state through an interface that is both trusted and

tamper-resistant. Intel addresses this gap with the Platform Properties Assessment Module (PPAM), intended to provide OS-visible evidence of SMM configuration; however, prior work shows PPAM itself can be subverted (e.g., disabled via a single-byte write [172]).

On AMD platforms, the SMM Supervisor [36] similarly introduces privilege separation by executing third-party handlers in a least-privileged context and mediating privileged actions through a narrow system-call interface enforced by a small trusted component, similar to Intel's initial STM approach [65].

Other modern firmware implementations also follow these architectural shifts and provide further defenses for SMM. Samsung Knox for x86 devices [175] hardens the SMM environment via SMI Guard, which dynamically intercepts and sanitizes inputs passed to SMI handlers. Knox further leverages Intel's Runtime BIOS Resilience to enforce read-only code sections and mark critical SMM data structures as non-executable.

Open-Source Concerns. Notably, the deployed implementations of the aforementioned containment mechanisms are primarily delivered as vendor firmware and are not fully open to independent inspection; while some reference components exist in open ecosystems [188, 193, 212], external evaluation on commercial platforms often requires substantial reverse engineering effort. Consistent with this concern, public work has reported vulnerabilities affecting SMM containment logic, illustrating that isolation layers require the same level of scrutiny as the SMI handlers they aim to protect [9, 148, 172, 193].

Take-Away #4: Persistent memory-safety and shielding code vulnerabilities motivate hardening and containment approaches for SMM.

5.2 Open Evaluation Tooling

Our era analysis yields the insight that vulnerability discovery scales with tooling availability and reuse, yet much of the firmware ecosystem remains closed and difficult to evaluate systematically. A practical obstacle is that representative firmware/SMM corpora are hard to obtain and normalize, since images are vendor-specific and updates are commonly distributed as capsules that must be installed to extract the resulting firmware state. These images often require non-trivial extraction and reverse engineering before analysis is possible. As a result, the community frequently stands on the shoulders of giants, reusing practitioner-developed parsing and inspection workflows to make analysis feasible [44, 119, 133, 134].

Open Firmware Ecosystems. Open firmware ecosystems lower the research barrier by making parts of the firmware stack inspectable and testable. EDK II [188], coreboot [59], and Microsoft Project Mu [141, 193] provide open UEFI and platform firmware components, including SMI/SMM-related services and supervisor-style hardening features. These ecosystems can serve as practical testbeds for reproducing SMM experiments and integrating configuration checks and SMI-handler analysis into continuous workflows.

Firmware Parsing and Modification. A first class of tooling supports firmware parsing and modification, which is a prerequisite for most evaluation pipelines. Examples include UEFITool [133] for parsing and extraction, FMMT [189] for modifying EDK II-based images, Fiano [1] for cross-platform manipulation, and scriptable parsers [186]. While these tools do not directly harden SMM, they reduce friction in dataset construction and improve reproducibility. In practice, many higher-level analyses depend on these utilities to locate and extract SMI-handler code for inspection and testing.

Platform Configuration Assessment. This second class comprises tools that automate security-relevant platform inspection and configuration checks. These tools validate lockdown assumptions and are repeatedly used to support exploit development and reproduction across eras [119, 134]. In Era 3, cross-platform assessment efforts reduce vendor asymmetry and make configuration testing more feasible beyond Intel-centric workflows [119].

SMM Code Analysis and Vulnerability Discovery. Era 3 introduces a broader set of automated approaches for SMI-handler analysis, including fuzzing [75, 200, 201, 208, 218] and symbolic execution [43, 97, 180] tools, which reduce reliance on manual reverse engineering and demonstrate the ability to automatically uncover SMM code vulnerabilities (e.g., memory-safety, confused deputy, SMM callouts etc.).

Complementing these approaches, Intel's INT31 research team introduced the BIOS Binary Instrumentation Framework (Intel.BIN) [166], a dynamic instrumentation tool analogous to Intel PIN, but operating withing the BIOS runtime. Though not yet publicly available, it currently focuses on the platform initialization stage of BIOS and plans to extend support to the SMM runtime prior to public release.

Taken together, the primary gap is not the absence of point solutions, but the lack of a *reproducible corpora* for SMM. We therefore encourage continued development of open analysis tooling for firmware and SMM, alongside public and representative datasets of SMM modules and configurations. We also see value in environments and frameworks for reproducing SMM experiments and in integrating configuration checks, static and dynamic code analysis into CI/CD-style workflows for both vendors and researchers. This infrastructure aims to lower the entry barrier and improve the comparability of platform security evaluation.

5.3 Aligning Industry, Academia, and Practice

Two decades of offensive SMM work, demonstrate that the hacker community has driven much of the vulnerability discovery, exploit validation, and disclosure effort, while CPU vendors have contributed a smaller set of foundational initiatives for platform hardening and analysis. Despite these efforts, recurring classes of SMM vulnerabilities persist across years, underscoring the importance of systematic testing across the UEFI firmware supply chain.

In parallel, we find that academia has often approached SMM from a different angle by leveraging its privileged position to build defensive capabilities and security services for other platform components. Only more recently, particularly in the latest era, has academic work engaged more directly with the need for a secure SMM mode by contributing program-analysis approaches for SMI-handler evaluation. In addition, some work explores hardening mechanisms such as SmmPack [140], which investigates encrypting SMM modules with keys protected by the Trusted Platform Module. However, during our analysis we notice that tools and datasets are not always released in reusable open-source form, limiting cumulative progress and reproducibility. We therefore encourage continued academic investment in open infrastructure that complements hacker community discoveries and vendor deployment. We expect that the experience gained from SMM-based defensive designs can help strengthen SMM practical evaluation and hardening.

> **Take-Away #5:** Closing the gap requires open-source frameworks for the SMM landscape, providing reusable datasets that lower the entry barrier and support systematic evaluation.

5.4 SMM as a Defensive Substrate

Representative academic work treats SMM as a defensive substrate that can provide security services beneath the OS and hypervisor. Prior systems leverage SMM for transparent monitoring and introspection of higher layers, especially protecting the integrity of the running system (OS and hypervisor) [128, 130, 171, 198, 222, 223, 226]. Others use SMM to implement security services analogous to those found in UEFI deployments, including secure kernel patching, enclave I/O, and protection of trusted system services in cloud settings [131, 132, 221, 224, 227]. Additional work explores isolated-computing designs that reduce OS or hypervisor trust assumptions by placing critical logic in SMM [39, 40, 197, 225]. Collectively, these efforts highlight SMM as a privileged capability for enforcing security properties when higher layers may be compromised.

Looking forward, we encourage further exploration of SMM as a defensive component. Since many prior systems combine SMM with Intel SGX, a natural direction is to revisit these ideas in the context of confidential VMs (e.g., Intel TDX or AMD SEV-SNP), where guests explicitly distrust the host hypervisor and rely on hardware-enforced isolation. An additional emerging direction is accelerator-centric confidential computing: as confidential GPU deployments mature and integrate with confidential VMs, it is important to understand whether SMM can play a narrowly scoped and auditable role in device configuration, IOMMU-enforced DMA isolation, and platform attestation evidence, without weakening the confidential-VM threat model.

> **Take-Away #6:** Prior work uses SMM as a defensive substrate for monitoring and security services, motivating integration with confidential VMs and GPU confidential computing.

5.5 Microarchitectural Security of SMM

Over the past decade, microarchitectural side-channel [74] and transient-execution [52] attacks have evolved into a major research domain, partially fueled by the emergence of isolation mechanisms such as Intel SGX [178]. Given this broader trend, it is striking that the microarchitectural security of SMM remains comparatively underexplored despite its high privilege level and unique execution semantics.

Among the 69 studies summarized in Table 3, we identified only a single work that explicitly considers microarchitectural attack surface, presenting an elementary proof-of-concept for Spectre v1 in SMM [69]. In contrast, the advisory record from Intel and AMD indicates that SMM is a relevant and active concern for microarchitectural hardening: of the 57 SMM-related advisories we examined, 8 explicitly provide guidance for protecting critical SMI handlers against microarchitectural attacks, primarily transient-execution vulnerabilities [95, 109, 111–116] as well as memory-aliasing attacks [26, 64].

While microcode updates commonly flush leaky microarchitectural buffers on SMM exit, SMI handlers must additionally coordinate all logical processors during SMM entry and exit [109, 112]. Whether such non-trivial "rendezvous" mechanisms are correctly and consistently implemented across heterogeneous, real-world firmware remains unclear, and the practical exploitability of microarchitectural leakage across SMM transitions is insufficiently understood. Given the maturity of microarchitectural attack and defense methodologies for other x86 isolation mechanisms, this gap presents a natural opportunity to transfer established analysis techniques into the privileged firmware domain. In particular, automated validation and fuzzing frameworks [142, 153] for assessing leakage across SMM transitions and the firmware ecosystem could substantially advance the state of the art.

> **Take-Away #7:** Offensive research assessing microarchitectural leakage across SMM transitions, as well as automated analysis of microarchitectural firmware mitigations, remains notably underexplored.

5.6 Lessons Beyond x86 Firmware

Finally, we discuss how insights learned from offensive SMM research may generalize to other architectures that rely on privileged firmware modes.

We have seen that incorrect or incomplete initialization of low-level x86 configuration registers can collapse the SMM isolation boundary and enable later compromise; the transferable lesson is that isolation in privileged firmware modes critically depends on correct platform configuration. Furthermore, firmware and secure monitor components are predominantly implemented in C/C++ and therefore remain susceptible to recurring memory-safety vulnerabilities. Communicating with privileged handler code from a less privileged context is a further universal risk: cross-world interfaces such as shared-buffer protocols can reproduce confused-deputy and callout-style patterns when inputs are insufficiently validated, untrusted pointers are dereferenced, or data is not copied into trusted memory before use. Similar failure modes have been observed in TEE ecosystems, motivating dedicated analysis tooling for enclave interfaces [35, 46, 54, 56, 192].

Arm TrustZone. On Arm Cortex-A, TrustZone [143, 165] isolation depends on vendor-configured partitioning of memory and device MMIO, together with the existence of a Secure Monitor (SM). The SM runs at the highest privilege and controls transitions between secure and non-secure states, establishing the access rules during early boot. On the other hand, TrustZone-M for Arm Cortex-M relies on vendor-configured memory attribution and protection rules, used to partition memory between secure and non-secure regions. Subtle misconfigurations on both Cortex-A and Cortex-M devices may expose secure code or data to software in the non-secure world. TrustZone firmware has, furthermore, been subject to a long history of confused-deputy vulnerabilities [54].

RISC-V. On RISC-V, firmware commonly executes in machine mode, with OpenSBI providing a runtime interface between the OS and M-mode services [118]. RISC-V provides Physical Memory Protection (PMP) [58] as an ISA-level primitive for constraining access to memory regions. TEE frameworks such as Keystone deploy an M-mode security monitor that relies on PMP for isolation [129]. Because PMP configuration defines the isolation boundary for both firmware and TEEs, misconfiguration or failure to lock PMP entries can nullify isolation guarantees. Compromise or insufficient sanitization of privileged M-mode firmware interfaces [192] can further allow permissions to be widened at runtime beyond the intended policy. Lastly, even a correctly configured PMP may be bypassed via microarchitectural attack surfaces [135].

Security Processors Beyond privileged CPU modes like x86 SMM, Arm EL3, and RISC-V M-mode, many commodity platforms rely on auxiliary security processors such as Intel's Management Engine and the AMD Platform Security Processor to provide security-critical services and participate in measured boot and attestation flows [36, 85]. These widely-deployed subsystems run on dedicated co-processors that are treated as part of the platform root of trust, placing them in an even more privileged and largely out-of-band position relative to the host OS. Their security guarantees are often difficult to validate externally due to closed-source proprietary implementations and limited visibility. Prior offensive security research has raised concerns about these ecosystems due to their privileged position (access to all platform resources) [126] and has already documented several vulnerabilities affecting these subsystems [127, 169, 172, 187]. This reinforces a broader lesson from the SMM landscape: platform trust increasingly depends on privileged firmware components whose behavior is hard to audit without dedicated evaluation tooling and, in many cases, significant reverse engineering effort to understand how they operate under the hood.

SMM Beyond Intel/AMD x86. Although almost all SMM security work focuses on Intel and AMD, SMM is a standard x86 feature that also appears in other vendor ecosystems. For example, VIA documentation describes SMI-driven SMM entry, SMRAM-backed execution, and state save and restore semantics [194, 195]. This behavior is not only described in VIA chipset manuals exposing the corresponding configuration surface [196], but also in other legacy x86 manuals as well [83]. However, publicly accessible documentation is less uniform across these vendors, complicating comparative analysis and leaving SMM hardening and vulnerability transferability beyond Intel/AMD underexplored.

6 Conclusion

This paper consolidates two decades of offensive SMM research into a structured corpus, taxonomy, and era-based analysis connecting vulnerability classes, tooling, and platform evolution. Our findings show how the attack surface has shifted from configuration-driven weaknesses toward recurring SMI handler vulnerabilities, and how tooling maturity shapes what becomes visible, reproducible, and tested. By systematically organizing the landscape and its trends, we provide researchers, practitioners, and vendors with a common reference point that aims to lowers the barrier to entry and to orient both offensive and defensive SMM research toward the knowledge gaps and priorities we have identified.

Acknowledgements

This research is partially funded by the Internal Funds KU Leuven and by the Cybersecurity Research Program Flanders.

References

[1] Google and Facebook. fiano: Go-based tools for modifying UEFI firmware. https://github.com/linuxboot/fiano. Accessed: 2026-01-27.

[2] Advanced Micro Devices, Inc. AMD server vulnerabilities – November 2021. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1021.html, 2021.

[3] Advanced Micro Devices, Inc. AMD client vulnerabilities – May 2022. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1027.html, 2022.

[4] Advanced Micro Devices, Inc. AMD client vulnerabilities – January 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1031.html, 2023.

[5] Advanced Micro Devices, Inc. AMD client vulnerabilities – November 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4002.html, 2023.

[6] Advanced Micro Devices, Inc. AMD server vulnerabilities – January 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1032.html, 2023.

[7] Advanced Micro Devices, Inc. AMD server vulnerabilities – May 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3001.html, 2023.

[8] Advanced Micro Devices, Inc. AMD server vulnerabilities – Nov 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3002.html, 2023.

[9] Advanced Micro Devices, Inc. AMD SMM supervisor vulnerability security notice. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7011.html, 2023. AMD Product Security.

[10] Advanced Micro Devices, Inc. Client vulnerabilities – May 2023. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4001.html, 2023.

[11] Advanced Micro Devices, Inc. SMM memory corruption vulnerability. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4003.html, 2023. AMD Product Security.

[12] Advanced Micro Devices, Inc. AMD embedded processors vulnerabilities – August 2024. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5002.html, 2024.

[13] Advanced Micro Devices, Inc. AMD embedded processors vulnerabilities – February 2024. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5001.html, 2024.

[14] Advanced Micro Devices, Inc. AMD processor vulnerabilities. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7009.html, 2024.

[15] Advanced Micro Devices, Inc. AMD server vulnerabilities – August 2024. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3003.html, 2024.

[16] Advanced Micro Devices, Inc. AMD64 architecture programmer's manual volumes 1—5. https://docs.amd.com/v/u/en-US/40332-PUB_4.08, 2024.

[17] Advanced Micro Devices, Inc. Client vulnerabilities – August 2024. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4004.html, 2024.

[18] Advanced Micro Devices, Inc. SMM lock bypass. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7014.html, 2024. AMD Product Security.

[19] Advanced Micro Devices, Inc. SPI lock bypass. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1041.html, 2024.

[20] Advanced Micro Devices, Inc. AMD client processor vulnerabilities – February 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4008.html, 2025.

[21] Advanced Micro Devices, Inc. AMD client vulnerabilities – August 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4012.html, 2025.

[22] Advanced Micro Devices, Inc. AMD CPU microcode signature verification vulnerability. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7033.html, 2025.

[23] Advanced Micro Devices, Inc. AMD embedded processors vulnerabilities – February 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5004.html, 2025.

[24] Advanced Micro Devices, Inc. AMD embedded vulnerabilities – August 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-5007.html, 2025.

[25] Advanced Micro Devices, Inc. AMD server processor vulnerabilities – February 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3009.html, 2025.

[26] Advanced Micro Devices, Inc. AMD server vulnerabilities – August 2025. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3014.html, 2025.

[27] Advanced Micro Devices, Inc. AMD SMM callout vulnerability. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7028.html, 2025. AMD Product Security.

[28] Advanced Micro Devices, Inc. AMD SMM vulnerabilities. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7027.html, 2025. AMD Product Security.

[29] Advanced Micro Devices, Inc. EDK2 SMM MCE enablement issue. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7043.html, 2025.

[30] Advanced Micro Devices, Inc. AMD Athlon™ and AMD Ryzen™ Processor Vulnerabilities – February 2026. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4013.html, 2026.

[31] Advanced Micro Devices, Inc. AMD embedded processor vulnerabilities – May 2026. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3030.html, 2026.

[32] Advanced Micro Devices, Inc. AMD EPYC™ and AMD EPYC™ Embedded Series Processor Vulnerabilities – February 2026. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3023.html, 2026.

[33] Advanced Micro Devices, Inc. AMD server vulnerabilities – May 2026. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-4017.html, 2026.

[34] Advanced Micro Devices, Inc. Incorrect use of locateprotocol service of the EFI_BOOT_services table in SMI handler. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7054.html, 2026.

[35] Fritz Alder, Lesly-Ann Daniel, David Oswald, Frank Piessens, and Jo Van Bulck. Pandora: Principled symbolic validation of Intel SGX enclave runtimes. In *45th IEEE Symposium on Security and Privacy (S&P)*, May 2024.

[36] AMD. AMD PRO technologies: A look at AMD PRO security and the AMD framework for secure, manageable, and reliable business pcs. https://www.amd.com/content/dam/amd/en/documents/products/processors/technologies/amd-pro-technologies-security-white-paper.pdf.

[37] Jacob Appelbaum, Laura Poitras, Marcel Rosenbach, Christian Stöcker, Jörg Schindler, and Holger Stark. Documents Reveal Top NSA Hacking Unit. https://www.spiegel.de/international/world/the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-a-940969.html, 2013.

[38] Paul Asadoorian. Firmware Security Realizations - Part 3 - SPI Write Protections. https://eclypsium.com/research/firmware-security-realizations-part-3-spi-write-protections/, 2022.

[39] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan 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, page 38–49, New York, NY, USA, 2010. Association for Computing Machinery.

[40] Ahmed M. Azab, Peng Ning, and Xiaolan 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, page 375–388, New York, NY, USA, 2011. Association for Computing Machinery.

[41] Oleksandr Bazhaniuk, Yuriy Bulygin, Andrew Furtak, Mikhail Gorobets, John Loucaides, Alexander Matrosov, and Mickey Shkatov. A new class of vulnerabilities in SMI handlers. https://www.c7zero.info/stuff/ANewClassOfVulnInSMIHandlers_csw2015.pdf, 2015. Accessed: 2025-11-25.

[42] Oleksandr Bazhaniuk, Andrew Furtak, Mikhail Gorobets, and Yuriy Bulygin. Exploring Your System Deeper with CHIPSEC is Not Naughty. https://www.c7zero.info/stuff/csw2017_ExploringYourSystemDeeper_updated.pdf, 2017.

[43] Oleksandr Bazhaniuk, John Loucaides, Lee Rosenbaum, Mark R. Tuttle, and Vincent Zimmer. Symbolic execution for BIOS security. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., August 2015. USENIX Association.

[44] Binarly. efixplorer hunting for UEFI firmware vulnerabilities at scale with automated static analysis. https://github.com/REhints/efiXplorer?tab=readme-ov-file, 2020.

[45] Binarly. UEFI firmware vulnerabilities: Past, present and future. https://github.com/binarly-io/Research_Publications/tree/main/OffensiveCon_2022, 2022.

[46] Marton Bognar and Jo Van Bulck. openIPE: An extensible memory isolation framework for microcontrollers. In *10th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2025.

[47] Kirk Brannock, Prashant Dewan, Frank McKeen, and Uday Savagaonkar. Providing a Safe Execution Environment. *Intel Technology Journal*, 13(2), 2009.

[48] BSDaemon, coideloko, and D0nand0n. Phrack Issue 65 file 7: System Management Mode Hack Using SMM for 'Other Purposes'. https://phrack.org/issues/65/7, 2008. Accessed: 2025-11-21.

[49] Yuriy Bulygin and Oleksandr Bazhaniuk. Digging into the core of boot. https://recon.cx/2017/montreal/resources/slides/RECON-MTL-2017-DiggingIntoTheCoreOfBoot.pdf, 2017.

[50] Yuriy Bulygin, Oleksandr Bazhaniuk, Andrew Furtak, John Loucaides, and Mikhail Gorobets. Baring the system: New vulnerabilities in SMM of Coreboot and UEFI based systems. https://www.c7zero.info/stuff/REConBrussels2017_BARing_the_system.pdf, 2017. Accessed: 2025-11-25.

[51] John Butterworth, Corey Kallenberg, Xeno Kovah, and Amy Herzog. BIOS chronomancy: fixing the core root of trust for measurement. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 25—36, New York, NY, USA, 2013. Association for Computing Machinery.

[52] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 249–266, 2019.

[53] Assaf Carlsbad. Zen and the Art of SMM Bug Hunting | Finding, Mitigating and Detecting UEFI Vulnerabilities. https://www.sentinelone.com/labs/zen-and-the-art-of-smm-bug-hunting-finding-mitigating-and-detecting-uefi-vulnerabilities/, 2022.

[54] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1416–1432. IEEE, 2020.

[55] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *ACM Computing Surveys*, 56(9):1–33, 2024.

[56] Tobias Cloosters, Michael Rodler, and Lucas Davi. TeeRex: Discovery and exploitation of memory corruption vulnerabilities in SGX enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 841–858. USENIX Association, August 2020.

[57] Core Collapse. Phrack Issue 66 file 11:A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers. https://phrack.org/issues/66/11, 2009. Accessed: 2025-11-21.

[58] RISC-V Consortium. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20250508. Accessed: 2025-10-30. https://docs.riscv.org/reference/isa/_attachments/riscv-privileged.pdf.

[59] coreboot. Fast, secure and flexible open source firmware platform. https://www.coreboot.org/.

[60] Xeno Kovah LEGBACORE Corey Kallenberg. How many million BIOSes would you like to infect? https://archive.conference.hitb.org/hitbsecconf2015ams/wp-content/uploads/2015/02/WHITEPAPER-How-Many-Million-BIOSes-Would-You-Like-To-Infect.pdf, 2015. Accessed: 2025-11-25.

[61] Intel Corp. Intel® Hardware Shield — Below-the-OS Security. https://contentmx.com/b/page/page.php?u=Abrahams73&i=2669077, 2019.

[62] Intel Corp. Intel® hardware shield: Trustworthy SMM on the intel vpro® platform. https://contentmx.com/b/page/page.php?u=Abrahams73&i=2669077, 2020.

[63] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[64] Jesse De Meulemeester, Luca Wilke, David Oswald, Thomas Eisenbarth, Ingrid Verbauwhede, and Jo Van Bulck. BadRAM: Practical memory aliasing attacks on trusted execution environments. In *46th IEEE Symposium on Security and Privacy (S&P)*, May 2025.

[65] Brian Delgado, Tejaswini Vibhute, and Karen L. Karavanic. Applying the principle of least privilege to system management interrupt handlers with the intel SMI transfer monitor. In *Proceedings of the 9th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, HASP '20, New York, NY, USA, 2021. Association for Computing Machinery.

[66] Christopher Domas. The memory sinkhole. https://blackhat.com/docs/us-15/materials/us-15-Domas-The-Memory-Sinkhole-Unleashing-An-x86-Design-Flaw-Allowing-Universal-Privilege-Escalation-wp.pdf, 2015. Accessed: 2025-11-25.

[67] Loic Duflot, Daniel Etiemble, and Olivier Grumelard. Using CPU system management mode to circumvent operating system security functions. In *CanSecWest 2006*, 2006.

[68] Loic Duflot, Olivier Levillain, Benjamin Morin, and Olivier Grumelard. System management mode design and security issues. In *White Paper*, 2010.

[69] Eclypsium. System management mode speculative execution attacks. https://eclypsium.com/research/system-management-mode-speculative-execution-attacks/, 2018.

[70] Embedi. UEFI BIOS holes. so much magic. don't come inside. https://web.archive.org/web/20190531081353/https://embedi.org/blog/uefi-bios-holes-so-much-magic-dont-come-inside/, 2017.

[71] Embedi. UEFI BIOS holes. so much magic. don't come inside. https://github.com/embedi/smm_usbrt_poc, 2017.

[72] Shawn Embleton, Sherri Sparks, and Cliff Zou. SMM rootkits: a new breed of OS independent malware. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, New York, NY, USA, 2008. Association for Computing Machinery.

[73] Microsoft Enterprise and OS Security. System management mode deep dive: How SMM isolation hardens the platform. https://www.microsoft.com/en-us/security/blog/2020/11/12/system-management-mode-deep-dive-how-smm-isolation-hardens-the-platform/, 2020.

[74] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, 8(1):1–27, 2018.

[75] Connor Glosner and Aravind Machiry. Fuzzuer: Enabling fuzzing of UEFI interfaces on edk-2. In *NDSS Symposium*, 2025.

[76] Mikhail Gorobets, Oleksandr Bazhaniuk, Alex Matrosov, Andrew Furtak, and Yuriy Bulygin. Attacking Hypervisors via Firmware and Hardware. https://c7zero.info/stuff/AttackingHypervisorsViaFirmware_bhusa15_dc23.pdf, 2015.

[77] NCC Group. Stepping insyde system management mode. https://www.nccgroup.com/research-blog/stepping-insyde-system-management-mode/, 2013. Accessed: 2025-11-25.

[78] NCC Group. A race to report a TOCTOU: Analysis of a bug collision in intel SMM. https://www.nccgroup.com/research-blog/a-race-to-report-a-toctou-analysis-of-a-bug-collision-in-intel-smm/, 2023. Accessed: 2025-11-25.

[79] Casual Hacking. A compendium to UEFI hacking. https://casualhacking.io/blog/2014/3/26/a-compendium-to-uefi-hacking, 2014. Accessed: 2025-11-21.

[80] Takahiro Haruyama. Detect me if you can - anti-firmware forensics. https://speakerdeck.com/takahiro_haruyama/detect-me-if-you-can-anti-firmware-forensics?slide=27, 2022.

[81] Jussi Hietanen and Diego Caminada. SMM rootkit. https://jussihi.kapsi.fi/2022-09-08-smmrootkit/, 2022. Accessed: 2025-11-25.

[82] Jussi Hietanen and Diego Caminada. HERMES: User process privilege escalation higher than the kernel. https://github.com/pRain1337/Hermes, 2023. Accessed: 2025-11-25.

[83] IBM Microelectronics / Cyrix. *6x86 Microprocessor Programmer's Reference Manual (Chapter 2, pp. 2–63)*, 1996.

[84] Advanced Micro Devices Inc. *AMD64 Architecture Programmer's Manual Volumes 1-5*. Accessed: 2026-01-22.

[85] Intel. Intel converged security and management engine (CSME) security white paper. https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/intel-csme-security-white-paper.pdf. Accessed: 2026-01-27.

[86] Intel. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Accessed: 2026-01-22.

[87] Intel. Intel® desktop and intel® mobile boards privilege escalation. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00017.html, 2008.

[88] Intel. Intel® desktop and intel® server boards privilege escalation. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00018.html, 2009.

[89] Intel. Intel® desktop boards privilege escalation. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00022.html, 2010.

[90] Intel. SINIT buffer overflow vulnerability. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00030.html, 2011.

[91] Intel. SINIT authenticated code module privilege escalation. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00035.html, 2013.

[92] Intel. Local APIC Elevation of Privilege. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00045.html, 2015.

[93] Intel. Abusing CPU hot-add weaknesses to escalate privileges in server datacenters. https://www.slideshare.net/slideshow/privilege-escalation-on-highend-servers-due-to-implementation-gaps-in-cpu-hotadd-flow/73217903, 2017. Accessed: 2025-12-01.

[94] Intel. Intel® NUC BIOS Security Updates. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00084.html, 2017.

[95] Intel. Q3 2018 speculative execution side channel update. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html, 2018.

[96] Intel. 2019.2 ipu – intel® processor security advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00240.html, 2019.

[97] Intel. Finding BIOS Vulnerabilities with Symbolic Execution and Virtual Platforms. https://www.intel.com/content/www/us/en/developer/articles/technical/finding-bios-vulnerabilities-with-symbolic-execution-and-virtual-platforms.html, 2019.

[98] Intel. Architecture specification: Intel trust domain extensions (Intel TDX) module. Specification 344425-005US, Intel, February 2023.

[99] Intel. Intel trusted execution technology (Intel TXT). Manual 315168-017, Rev. 017.4, Intel, April 2023.

[100] Intel. Intel® slim bootloader advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01395.html, 2025.

[101] Intel. 2025.3 IPU – UEFI reference firmware advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01234.html, 2026.

[102] Intel. 2026.1 ipu, intel® processor firmware advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01396.html, 2026.

[103] Intel Corporation. SMI transfer monitor (STM) user guide. https://www.intel.com/content/dam/develop/external/us/en/documents/stm-user-guide-001-819978.pdf, 2015.

[104] Intel Corporation. Intel® branded NUC's vulnerable to SMM exploit. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00057.html, 2016. Intel Product Security Center.

[105] Intel Corporation. SmmRuntime Escalation of Privilege. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00056.html, 2016. Intel Product Security Center.

[106] Intel Corporation. Intel® Branded NUC's Vulnerable to SMM exploit. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00068.html, 2017. Intel Product Security Center.

[107] Intel Corporation. EDK II untested memory not covered by SMM page protection. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00159.html, 2018. Intel Product Security Center.

[108] Intel Corporation. Intel® NUC BIOS SW SMI call-out. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00110.html, 2018. Intel Product Security Center.

[109] Intel Corporation. L1 terminal fault. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-l1-terminal-fault.html, 2018. Intel Software Security Guidance, Technical Documentation (ID 758376).

[110] Intel Corporation. 2019.2 IPU — intel® processor graphics SMM advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00254.html, 2019. Intel Product Security Center.

[111] Intel Corporation. Intel transactional synchronization extensions (intel tsx) asynchronous abort. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-tsx-asynchronous-abort.html, 2019. Intel Software Security Guidance, Technical Documentation (ID 758370; updated November 12, 2019).

[112] Intel Corporation. Microarchitectural data sampling. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-microarchitectural-data-sampling.html, 2019. Intel Software Security Guidance, Technical Documentation (ID 758366, Version 3.0; updated March 11, 2021).

[113] Intel Corporation. Speculative execution side channel mitigations. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-execution-side-channel-mitigations.html, 2021. Intel Software Security Guidance, Technical Documentation (ID 758363, Version 2.0; updated May 26, 2021).

[114] Intel Corporation. Branch history injection and intra-mode branch target injection / CVE-2022-0001, CVE-2022-0002 / INTEL-SA-00598. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/branch-history-injection.html, 2022. Intel Software Security Guidance, Technical Documentation (ID 824191, Version 2.5; updated May 12, 2025).

[115] Intel Corporation. Processor MMIO stale data vulnerabilities. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/processor-mmio-stale-data-vulnerabilities.html, 2022. Intel Software Security Guidance, Technical Documentation (ID 758360; updated June 14, 2022).

[116] Intel Corporation. Snoop-assisted l1 data sampling. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/snoop-assisted-l1-data-sampling.html, 2022. Intel Software Security Guidance, Technical Documentation (ID 758367; updated March 10, 2020).

[117] Intel Corporation. 2024.3 ipu - SMI transfer monitor advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01083.html, 2024. Intel Product Security Center.

[118] RISC-V International. Anup patel western digital. https://riscv.org/wp-content/uploads/2024/12/13.30-RISCV_OpenSBI_Deep_Dive_v5.pdf, 2024. Accessed: 2025-10-14.

[119] IOActive. UEFI and SMM assessment tool. https://github.com/IOActive/Platbox/tree/main, 2022. Accessed: 2025-11-25.

[120] Corey Kallenberg, Sam Cornwell, Xeno Kovah, and John Butterworth. Defeating signed BIOS enforcement. https://www.mitre.org/sites/default/files/publications/defeating-signed-bios-enforcement.pdf, 2014. Accessed: 2025-11-25.

[121] Corey Kallenberg, Sam Cornwell, Xeno Kovah, and John Butterworth. Setup for failure: Defeating secure boot. https://infocon.org/cons/SyScan/SyScan%202014%20Singapore/SyScan%202014%20presentations/SyScan2014_CoreyKallenberg_SetupforFailureDefeatingSecureBoot_WP.pdf, 2014. Accessed: 2025-11-25.

[122] Corey Kallenberg, Xeno Kovah, John Butterworth, and Sam Cornwell. Extreme Privilege Escalation on Windows 8/UEFI System. https://www.mitre.org/sites/default/files/publications/14-2221-extreme-escalation-presentation.pdf, 2014. Accessed: 2025-11-25.

[123] Xeno Kovah. Low level desktop and server attack and defense timeline. https://darkmentor.com/timeline.html, 2021. Accessed: 2025-11-21.

[124] Xeno Kovah, John Butterworth, Corey Kallenberg, and Sam Cornwelll. Copernicus 2: Senter the dragon! https://www.mitre.org/sites/default/files/publications/Copernicus2-SENTER-the-Dragon-CSW-.pdf, 2014. Accessed: 2025-11-25.

[125] Xeno Kovah, John Butterworth, Corey Kallenberg, and Sam Cornwelll. SENTER Sandman: Using Intel TXT to Attack BIOSes. https://papers.put.as/papers/firmware/2014/D1T1-SENTER_Sandman_-_Using_Intel_TXT_to_Attack_BIOSes.pdf, 2014. Accessed: 2025-11-25.

[126] Joanna Rutkowska Invisible Things Lab. Intel x86 considered harmful. https://blog.invisiblethings.org/papers/2015/x86_harmful.pdf, 2015. Accessed: 2025-11-25.

[127] CTS Labs. Ryzenfallen. https://github.com/depletionmode/ryzenfallen, 2019.

[128] Kevin Leach, Fengwei Zhang, and Westley Weimer. Scotch: Combining software guard extensions and system management mode to monitor cloud resource usage. In *Research in Attacks, Intrusions, and Defenses*, pages 403–424. Springer International Publishing, 2017.

[129] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.

[130] Tamas Lengyel, Thomas Kittel, George Webster, and Jacob Torrey. Pitfalls of virtual machine introspection on modern hardware. In *1st Workshop on Malware Memory Forensics (MMF)*, December 2014.

[131] Hongliang Liang, Mingyu Li, Yixiu Chen, Lin Jiang, Zhuosi Xie, and Tianqi Yang. Establishing trusted I/O Paths for SGX client systems with aurora. *IEEE Transactions on Information Forensics and Security*, 15:1589–1600, 2020.

[132] Hongliang Liang, Mingyu Li, Yixiu Chen, Tianqi Yang, Zhuosi Xie, and Lin Jiang. Architectural protection of trusted system services for SGX enclaves in cloud computing. *IEEE Transactions on Cloud Computing*, 9(3):910–922, 2021.

[133] LongSoft. UEFITool is a viewer and editor of firmware images conforming to UEFI Platform Interface (PI) Specifications. https://github.com/LongSoft/UEFITool, 2015. Accessed: 2025-11-25.

[134] John Loucaides and Yuriy Bulygin. Platform Security Assessment with CHIPSEC. https://www.c7zero.info/stuff/Platform%20Firmware%20Security%20Assessment%20wCHIPSEC-csw14-final.pdf, 2014. Accessed: 2025-11-25.

[135] Antonis Louka, Jesse De Meulemeester, Steven Keuchel, Ingrid Verbauwhede, and Jo Van Bulck. Physical Memory Please: Practical Memory-Aliasing Attacks on RISC-V PMP. In *2nd Microarchitecture Security Conference (uASC)*, February 2026.

[136] Benjamin Morin Loïc Duflot, Olivier Levillain and Olivier Grumelard. Getting into the SMRAM: SMM reloaded, 2009.

[137] Alex Matrosov and Eugene Rodionov. The UEFI firmware rootkits: Myths and reality. https://blackhat.com/docs/asia-17/materials/asia-17-Matrosov-The-UEFI-Firmware-Rootkits-Myths-And-Reality.pdf, 2017. Accessed: 2025-11-25.

[138] Alex Matrosov, Andrey Labunets Yndegor Vasilenko, and Philip Lebedev. efixplorer hunting for UEFI firmware vulnerabilities at scale with automated static analysis. https://i.blackhat.com/eu-20/Wednesday/eu-20-Labunets-efiXplorer-Hunting-For-UEFI-Firmware-Vulnerabilities-At-Scale-With-Automated-Static-Analysis.pdf, 2020.

[139] Alex Matrosov, Yegor Vasilenko, Alex Ermolov, and Sam Thomas. Breaking firmware trust from Pre-EFI: Exploiting early boot phases. https://i.blackhat.com/USA-22/Wednesday/US-22-Matrosov-Breaking-Firmware-Trust-From-Pre-EFI.pdf, 2022. Accessed: 2025-11-25.

[140] Kazuki Matsuo, Satoshi Tanda, Kuniyasu Suzaki, Yuhei Kawakoya, and Tatsuya Mori. SmmPack: Obfuscation for SMM Modules with TPM Sealed Key. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 439–459, Cham, 2024. Springer Nature Switzerland.

[141] Microsoft. Project Mu Basecore Repository. https://github.com/microsoft/mu_basecore, 2022.

[142] Daniel Moghimi, Moritz Lipp, Berk Sunar, and Michael Schwarz. Medusa: Microarchitectural data leakage via automated attack synthesis. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1427–1444, 2020.

[143] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. Trustzone explained: Architectural features and use cases. *IEEE 2nd International Conference on Collaboration and Internet Computing*, 2016.

[144] Enrique Nissim and Joseph Tartaro Krzysztof Okupski. Exploring the security configuration of AMD platforms. https://www.ioactive.com/exploring-the-security-configuration-of-amd-platforms/#anchor_ref, 2022. Accessed: 2025-11-25.

[145] Enrique Nissim and Joseph Tartaro Krzysztof Okupski. Adventures in the platform security coordinated disclosure circus. https://www.ioactive.com/adventures-in-the-platform-security-coordinated-disclosure-circus/, 2023. Accessed: 2025-11-25.

[146] Enrique Nissim and Joseph Tartaro Krzysztof Okupski. Back to the future with platform security. https://www.ioactive.com/back-to-the-future-with-platform-security/, 2023. Accessed: 2025-11-25.

[147] Enrique Nissim and Krzysztof Okupski. AMD sinkclose universal SMM privilege escalation. https://media.defcon.org/DEF%20CON%2032/DEF%20CON%2032%20presentations/DEF%20CON%2032%20-%20Enrique%20Nissim%20Krzysztof%20Okupski%20-%20AMD%20Sinkclose%20Universal%20Ring-2%20Privilege%20Escalation.pdf, 2024. Accessed: 2025-11-25.

[148] Enrique Nissim and Joseph Tartaro. Tales from the call-gate: An SMM supervisor vulnerability. https://www.ioactive.com/tales-from-the-call-gate-an-smm-supervisor-vulnerability/, 2024. Accessed: 2025-11-25.

[149] NSA. 20131230-Appelbaum-NSA ANT Catalog. https://www.eff.org/document/20131230-appelbaum-nsa-ant-catalog, 2007.

[150] NSA. NSA S3285 Intern Projects. https://www.eff.org/files/2015/01/27/20150117-spiegel-projects_of_the_tao-ato_department_such_as_the_remote_destruction_of_network_cards.pdf, 2015.

[151] Danny Odler. Attacking the golden ring on AMD mini-pc. https://dannyodler.medium.com/attacking-the-golden-ring-on-amd-mini-pc-b7bfb217b437, 2020.

[152] Krzysztof Okupski. Exploring AMD Platform Secure Boot. https://labs.ioactive.com/2024/02/exploring-amd-platform-secure-boot.html, 2024.

[153] Oleksii Oleksenko, Christof Fetzer, Boris Köpf, and Mark Silberstein. Revizor: Testing black-box cpus against speculation contracts. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 226–239, 2022.

[154] Oleksii Oleksenko, Dmitrii Kuvaiskii, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. Intel mpx explained: A cross-layer analysis of the intel mpx system stack. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–30, 2018.

[155] Dmytro Oleksiuk. Breaking UEFI security with software DMA attacks. https://web.archive.org/web/20181212181811/http://blog.cr4.sh/2015/09/breaking-uefi-security-with-software.html, 2015.

[156] Dmytro Oleksiuk. Exploiting UEFI bootscript table. https://github.com/Cr4sh/UEFI_boot_script_expl?tab=readme-ov-file, 2015.

[157] Dmytro Oleksiuk. SMM backdoor for UEFI based platforms. https://github.com/Cr4sh/SmmBackdoorNg, 2015.

[158] Dmytro Oleksiuk. Breaking UEFI security with software DMA attacks. https://web.archive.org/web/20161104071430/https://blog.cr4.sh/2016/10/exploiting-ami-aptio-firmware.html, 2016.

[159] Dmytro Oleksiuk. Breaking UEFI security with software DMA attacks. https://github.com/Cr4sh/Aptiocalypsis, 2016.

[160] Dmytro Oleksiuk. Exploiting SMM callout vulnerabilities in lenovo firmware. https://web.archive.org/web/20161105090533/http://blog.cr4.sh/2016/02/exploiting-smm-callout-vulnerabilities.html, 2016.

[161] Dmytro Oleksiuk. Exploring and exploiting lenovo firmware secrets. https://web.archive.org/web/20160916121946/https://blog.cr4.sh/2016/06/exploring-and-exploiting-lenovo.html, 2016.

[162] Dmytro Oleksiuk. Exploring and exploiting lenovo firmware secrets. https://github.com/Cr4sh/ThinkPwn?tab=readme-ov-file, 2016.

[163] Rust OSDev. Rusty wrapper for the unified extensible firmware interface. https://github.com/rust-osdev/uefi-rs, 2017.

[164] Vojtech Pavlik. Usb legacy support crashes. https://www.kernel.org/doc/Documentation/x86/usb-legacy-support.txt, 2004.

[165] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)*, 51(6):1–36, 2019.

[166] INT31 Przemyslaw Duda. Intel.bin: The bios binary instrumentation framework. https://www.intel.com/content/www/us/en/security/security-practices/blogs/bios-binary-instrumentation-framework.html, 2025.

[167] Bruno Pujos. Code check(mate) in SMM. https://www.synacktiv.com/en/publications/code-checkmate-in-smm, 2018. Accessed: 2025-12-1.

[168] Bruno Pujos. Through the SMM-class and a vulnerability found there. https://www.synacktiv.com/en/publications/through-the-smm-class-and-a-vulnerability-found-there, 2020. Accessed: 2025-12-1.

[169] Qualis. Vulnerabilities in AMD Processors RYZEN and EPYC. https://threatprotect.qualys.com/2018/03/21/vulnerabilities-in-amd-processors-ryzen-and-epyc/, 2018.

[170] Julian Rauchberger, Robert Luh, and Sebastian Schrittwieser. Longkit - a universal framework for BIOS/UEFI rootkits in system management mode. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*, pages 346–353. INSTICC, SciTePress, 2017.

[171] Alessandro Reina, Aristide Fattori, Fabio Pagani, Lorenzo Cavallaro, and Danilo Bruschi. When hardware meets software: a bulletproof solution to forensic memory acquisition. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 79–88, New York, NY, USA, 2012. Association for Computing Machinery.

[172] Binarly Research. The Intel PPAM attack story. Black Hat, https://www.binarly.io/blog/black-hat-2022-the-intel-ppam-attack-story, 2022. Accessed: 2025-11-25.

[173] ESET researchers. When "secure" isn't secure at all: High-impact UEFI vulnerabilities discovered in lenovo consumer laptops. https://www.welivesecurity.com/2022/04/19/when-secure-isnt-secure-uefi-vulnerabilities-lenovo-consumer-laptops/, 2022.

[174] Joanna Rutkowska and Rafał Wojtczuk Invisible Things Lab. Preventing and detecting xen hypervisor subversions. https://invisiblethingslab.com/resources/bh08/part2-full.pdf, 2008. Accessed: 2025-11-25.

[175] Samsung. Protecting the smm. https://docs.samsungknox.com/admin/fundamentals/whitepaper/samsung-knox-for-pc/protecting-the-smm/, 2024.

[176] Joshua Schiffman and David Kaplan. The SMM rootkit revisited: Fun with usb. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 279–286, 2014.

[177] Moritz Schloegel, Daniel Klischies, Simon Koch, David Klein, Lukas Gerlach, Malte Wessels, Leon Trampert, Martin Johns, Mathy Vanhoef, Michael Schwarz, Thorsten Holz, and Jo Van Bulck. Confusing value with enumeration: Studying the use of CVEs in academia. In *34th USENIX Security Symposium (USENIX Security 25)*, August 2025.

[178] Michael Schwarz and Daniel Gruss. How trusted execution environments fuel research on microarchitectural attacks. *IEEE Security & Privacy*, 18(5):18–27, 2020.

[179] Scumjr. Implementation of the SMM rootkit "the watcher". https://github.com/scumjr/the-sea-watcher/tree/master?tab=readme-ov-file, 2016.

[180] Md Shafiuzzaman, Achintya Desai, Laboni Sarker, and Tevfik Bultan. STASE: Static Analysis Guided Symbolic Execution for UEFI Vulnerability Signature Generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 1783–1794, New York, NY, USA, 2024. Association for Computing Machinery.

[181] Michal Szaknis and Krzysztof Szczypiorski. The design of the simple SMM rootkit. In *Proceedings of the 2022 9th International Conference on Wireless Communication and Sensor Networks*, icWCSN '22, pages 47–56, New York, NY, USA, 2022. Association for Computing Machinery.

[182] Satoshi Tanda. SMM isolation - security policy reporting (ISSR). https://tandasat.github.io/blog/2024/03/18/ISSR.html, 2024.

[183] Binarly REsearch Team. An in-depth look at the 23 high-impact vulnerabilities. https://www.binarly.io/blog/an-in-depth-look-at-the-23-high-impact-vulnerabilities, 2022.

[184] Binarly REsearch Team. Repeatable Failures: AMI UsbRt - Six Years Later, Firmware Attack Vector Still Affect Millions of Enterprise Devices. https://www.binarly.io/blog/ami-usbrt-repeatable-failures-a-6-year-old-attack-vector-still-affecting-millions-of-enterprise-devices, 2022.

[185] Binarly REsearch Team. Repeatable firmware security failures: 16 high impact vulnerabilities discovered in hp devices. https://www.binarly.io/blog/repeatable-firmware-security-failures-16-high-impact-vulnerabilities-discovered-in-hp-devices, 2022.

[186] Teddy Reed. UEFI firmware parser. https://github.com/theopolis/uefi-firmware-parser. Accessed: 2026-01-27.

[187] Alexander Tereshkin and Rafał Wojtczuk Invisible Things Lab. Introducing ring -3 rootkits. https://blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf, 2009. Accessed: 2025-11-25.

[188] TianoCore Community. Edk ii project repository. https://github.com/tianocore/edk2. Accessed: 2026-01-27.

[189] TianoCore Community. Fmmt tool. https://github.com/tianocore/edk2/tree/master/BaseTools/Source/Python/FMMT. Accessed: 2026-01-27.

[190] TianoCore Community. Tianocore EDK-II: Open source implementation of UEFI standard. https://www.tianocore.org/. Accessed: 2026-01-27.

[191] UEFI Forum. UEFI specifications and tools. https://uefi.org/specifications. Accessed: 2026-01-27.

[192] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *26th ACM Conference on Computer and Communications Security (CCS)*, pages 1741–1758, November 2019.

[193] Ilja van Sprundel. Assessing the security of an SMM supervisor. https://docs.google.com/presentation/d/11uex-xPyBjVThovkh1kVTXJ4aarVryhK/edit?slide=id.p21#slide=id.p21, 2022.

[194] Inc. VIA Technologies. Via c7-m datasheet. Technical report, 2006.

[195] Inc. VIA Technologies. Via c7 processor nanobga2 datasheet. Technical report, 2006.

[196] VIA Technologies, Inc. *VIA VX900 Series System Programming Manual*, rev. 1.41 edition, October 2011.

[197] Jiang Wang, Angelos Stavrou, and Anup Ghosh. Hypercheck: A hardware-assisted integrity monitor. In *Recent Advances in Intrusion Detection*, pages 158–177, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[198] Jiang Wang, Kun Sun, and Angelos Stavrou. Hardware-assisted application integrity monitor. In *2012 45th Hawaii International Conference on System Sciences*, pages 5375–5383, 2012.

[199] Jianqiang Wang. Breaking ring -2: Unveiling system management mode memory corruption vulnerability via fuzzing. https://i.blackhat.com/BH-EU-25/eu-25-Wang-Breaking-Ring-2-Unveiling.pdf, 2025. Accessed: 2026-01-12.

[200] Jianqiang Wang. Breaking ring -2: Unveiling system management mode memory corruption vulnerability via fuzzing. https://i.blackhat.com/BH-EU-25/eu-25-Wang-Breaking-Ring-2-Unveiling.pdf, 2025. Accessed: 2025-12-13.

[201] Jianqiang Wang, Yi Xiang, Meng Wang, Qinying Wang, Ali Abbasi, and Thorsten Holz. SmuFuzz: Enable Deep System Management Mode Fuzzing in Fully Featured UEFI Runtime Environment . In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 2964–2981, Los Alamitos, CA, USA, May 2026. IEEE Computer Society.

[202] Rafal Wojtczuk and Alexander Tereshkin Invisible Things Lab. Attacking intel BIOS. https://blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf, 2009. Accessed: 2025-11-25.

[203] Rafał Wojtczuk and Corey Kallenberg. Attacks on UEFI security, inspired by darth venamis's misery and speed racer. https://fahrplan.events.ccc.de/congress/2014/Fahrplan/system/attachments/2557/original/AttacksOnUEFI_Slides.pdf, 2014. Accessed: 2025-12-01.

[204] Rafał Wojtczuk and Joanna Rutkowska Invisible Things Lab. Attacking intel® trusted execution technology. https://blackhat.com/presentations/bh-dc-09/Wojtczuk_Rutkowska/BlackHat-DC-09-Rutkowska-Attacking-Intel-TXT-slides.pdf, 2009. Accessed: 2025-11-25.

[205] Rafał Wojtczuk and Joanna Rutkowska Invisible Things Lab. Attacking SMM memory via intel® CPU cache poisoning. https://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf, 2009. Accessed: 2025-11-25.

[206] Rafał Wojtczuk and Joanna Rutkowska Invisible Things Lab. Attacking intel TXT® via SINIT code execution hijacking. https://invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf, 2011.

[207] Rafał Wojtczuk and Corey Kallenberg Bromium Labs. Attacking UEFI boot script. https://bromiumlabs.wordpress.com/wp-content/uploads/2015/01/venamis_whitepaper.pdf, 2014.

[208] Zhenkun Yang, Yuriy Viktorov, Jin Yang, Jiewen Yao, and Vincent Zimmer. UEFI Firmware Fuzzing with Simics Virtual Platform. In *57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[209] Jiewen Yao and Vincent J. Zimmer Intel Corporation. Understanding the UEFI secure boot chain — intel® boot guard. https://tianocore-docs.github.io/Understanding_UEFI_Secure_Boot_Chain/draft/secure_boot_chain_in_uefi/intel_boot_guard.html, 2025.

[210] Jiewen Yao and Vincent J. Zimmer Intel Corporation. Understanding the UEFI secure boot chain — signed capsule update. https://tianocore-docs.github.io/Understanding_UEFI_Secure_Boot_Chain/draft/secure_boot_chain_in_uefi/boot_chain__putting_it_all_together/signed-capsule-update.html, 2025.

[211] Jiewen Yao and Vincent J. Zimmer Intel Corporation. Understanding the UEFI Secure Boot Chain — UEFI Secure Boot. https://tianocore-docs.github.io/Understanding_UEFI_Secure_Boot_Chain/draft/secure_boot_chain_in_uefi/uefi_secure_boot.html, 2025.

[212] Jiewen Yao, Vincent J. Zimmer, and Intel Corporation. A tour beyond BIOS launching a STM to monitor SMM in EFI developer kit ii. https://www.intel.com/content/dam/develop/external/us/en/documents/a-tour-beyond-bios-launching-stm-to-monitor-smm-in-efi-developer-kit-ii-819978.pdf, 2015. Accessed: 2026-01-22.

[213] Jiewen Yao, Vincent J. Zimmer, Star Zhang, and Intel Corporation. A tour beyond BIOS implementing s3 resume with EDKII. https://github.com/tianocore-docs/Docs/blob/main/White_Papers/A_Tour_Beyond_BIOS_Implementing_S3_resume_with_EDKII_V2.pdf, 2015. Accessed: 2026-01-22.

[214] Jiewen Yao, Vincent J. Zimmer, Star Zhang, and Intel Corporation. A tour beyond BIOS implementing UEFI authenticated variables in SMM with EDKII. https://github.com/tianocore-docs/Docs/blob/main/White_Papers/A_Tour_Beyond_BIOS_Implementing_UEFI_Authenticated_Variables_in_SMM_with_EDKII_V2.pdf, 2015. Accessed: 2026-01-22.

[215] Jiewen Yao, Vincent J. Zimmer, Star Zhang, and Intel Corporation. A tour beyond BIOS secure SMM communication in the EFI developer kit. https://github.com/tianocore-docs/Docs/blob/main/White_Papers/A_Tour_Beyond_BIOS_Secure_SMM_Communication.pdf, 2016. Accessed: 2026-01-22.

[216] Jiewen Yao, Vincent J. Zimmer, Star Zhang, Fan Jeff, and Intel Corporation. A tour beyond BIOS - security enhancement to mitigate buffer overflow in UEFI. https://tianocore-docs.github.io/ATBB-Mitigate_Buffer_Overflow_in_UEFI/draft/, 2016. Accessed: 2026-01-22.

[217] Jiewen Yao, Vincent J. Zimmer, Star Zhang, Fan Jeff, and Intel Corporation. A tour beyond BIOS implementing profiling in EDKII. https://github.com/tianocore-docs/Docs/blob/main/White_Papers/A_Tour_Beyond_BIOS_Implementing_Profiling_in_EDKII_V2.pdf, 2016. Accessed: 2026-01-22.

[218] Jiawei Yin, Menghao Li, Yuekang Li, Yong Yu, Boru Lin, Yanyan Zou, Yang Liu, Wei Huo, and Jingling Xue. RSFuzzer: Discovering Deep SMI Handler Vulnerabilities in UEFI Firmware with Hybrid Fuzzing. In *IEEE Symposium on Security and Privacy (SP)*, pages 2155–2169, 2023.

[219] Jiawei Yin, Menghao Li, Wei Wu, Dandan Sun, Jianhua Zhou, Wei Huo, and Jingling Xue. Finding SMM Privilege-Escalation Vulnerabilities in UEFI Firmware with Protocol-Centric Static Analysis. In *IEEE Symposium on Security and Privacy (SP)*, pages 1623–1637, 2022.

[220] Benny Zeltser and Jonathan Lusky. Ringhopper - how we almost zero day'd the world. https://www.youtube.com/watch?v=u8V4ofWpHZk, 2023. Accessed: 2025-12-01.

[221] Fengwei Zhang. IOCheck: A framework to enhance the security of I/O devices at runtime. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–4, 2013.

[222] Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. Using hardware features for increased debugging transparency. In *2015 IEEE Symposium on Security and Privacy*, pages 55–69, 2015.

[223] Fengwei Zhang, Kevin Leach, Kun Sun, and Angelos Stavrou. SPECTRE: A dependable introspection framework via system management mode. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2013.

[224] Fengwei Zhang, Kevin Leach, Haining Wang, and Angelos Stavrou. Trustlogin: Securing password-login on commodity operating systems. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, page 333–344, New York, NY, USA, 2015. Association for Computing Machinery.

[225] Ning Zhang, Ming Li, Wenjing Lou, and Y. Thomas Hou. Mushi: Toward multiple level security cloud with strong hardware level isolation. In *MILCOM 2012 - 2012 IEEE Military Communications Conference*, pages 1–6, 2012.

[226] Lei Zhou, Jidong Xiao, Kevin Leach, Westley Weimer, Fengwei Zhang, and Guojun Wang. Nighthawk: Transparent system introspection from ring -3. In *Computer Security – ESORICS 2019*, pages 217–238, Cham, 2019. Springer International Publishing.

[227] Lei Zhou, Fengwei Zhang, Jinghui Liao, Zhengyu Ning, Jidong Xiao, Kevin Leach, Westley Weimer, and Guojun Wang. Kshot: Live kernel patching with SMM and SGX. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13, 2020.

A Appendix: SMM Security Advisories

Table 4 shows all 57 SMM-related vulnerabilities automatically parsed from Intel and AMD advisories and manually classified according to the taxonomy in Fig. 2.

Table 4: SMM-related vulnerabilities found in Intel and AMD advisories, classified according to the taxonomy in Fig. 2.

Name	Year	Advisory	Vulnerability
Mobile boards privilege escalation [87]	2008	Intel	*Plat-config→FW→Memory-Controller*
Server boards privilege escalation [88]	2009	Intel	*Plat-config→FW→Memory-Controller*
Desktop boards privilege escalation [89]	2010	Intel	*Plat-config→FW→Memory-Controller*
SINIT Buffer Overflow [90]	2011	Intel	*SMM-code→MS→Spatial*
SINIT ACM priviledge escalation [91]	2013	Intel	*SMM-code→Shielding→CD*
Local APIC privilege elevation [92]	2015	Intel	*Plat-config→HW→APICRemap*
Memory corruption [104]	2016	Intel	*SMM-code→Shielding→CD*
SmmRuntime privilege escalation [105]	2016	Intel	*SMM-code→Shielding→SMMCallout*
NUC SMM callout [106]	2017	Intel	*SMM-code→Shielding→SMMCallout*
Intel NUC BIOS security updates [94]	2017	Intel	*SMM-code→Shielding→CD*
Foreshadow [95]	2018	Intel	*SMM-code→Uarch→MDS*
NUC SW SMI Callout [108]	2018	Intel	*SMM-code→Shielding→SMMCallout*
EDK2 SMM page protection [107]	2018	Intel	*Plat-config→FW→Memory-Controller*
MDS [112]	2019	Intel	*SMM-code→Uarch→MDS*
Intel graphics advisory [110]	2019	Intel	*Plat-config→FW→Memory-Controller*
Intel processor security [96]	2019	Intel	*Plat-config→FW→Memory-Controller*
Trust asynchronous abort [111]	2019	Intel	*SMM-code→Uarch→MDS*
AMD race-condition in ASP [2]	2021	AMD	*SMM-code→Shielding→Race-condition*
Spectre V1/V2 [113]	2021	Intel	*SMM-code→Uarch→Spectre*
MMIO stale data [115]	2022	Intel	*SMM-code→Uarch→MDS*
AMD client CPUs vulnerabilities May [3]	2022	AMD	*SMM-code→Shielding→CD*
Branch history injection [114]	2022	Intel	*SMM-code→Uarch→Spectre*
Snoop assisted L1DS [116]	2022	Intel	*SMM-code→Uarch→MDS*
AMD supervisor [9]	2023	AMD	*SMM-code→Shielding→SMMCallout*
AMD memory corruption [11]	2023	AMD	*SMM-code→Shielding→SMMCallout*
SMU input corrupts SMRAM [7]	2023	AMD	*SMM-code→Shielding→CD*
AMD client CPUs vulnerabilities Jan [4]	2023	AMD	*SMM-code→Shielding→CD*
AMD client CPUs vulnerabilities Nov [5]	2023	AMD	*SMM-code→Shielding→CD*
AMD server CPUs vulnerabilities Jan [6]	2023	AMD	*SMM-code→Shielding→CD*
AMD server CPUs vulnerabilities Nov [8]	2023	AMD	*SMM-code→Shielding→CD*
Insufficient input validation in ASP [10]	2023	AMD	*SMM-code→Shielding→CD*
SPI lock bypass [19]	2024	AMD	*Plat-config→FW→Memory-Controller*
AMD SMM lock bypass [18]	2024	AMD	*Plat-config→FW→Memory-Controller*
AMD Embedded processors [13]	2024	AMD	*[Plat-config→FW→Memory-Controller | SMM-code→Shielding→CD]*
STM priviledge escalation [117]	2024	Intel	*Plat-config→HW→ucode*
AMD server CPUs — SMM TOCTOU [15]	2024	AMD	*SMM-code→Shielding→CD*
AMD client CPUs — SMM TOCTOU [17]	2024	AMD	*SMM-code→Shielding→CD*
AMD embedded CPUs — SMM TOCTOU [12]	2024	AMD	*SMM-code→Shielding→CD*
SMM heap-overflow and access-control [14]	2024	AMD	*[Plat-config→FW→Memory-Controller | SMM-code→MS→Spatial]*
AMD SMM callout [27]	2025	AMD	*SMM-code→Shielding→SMMCallout*
SMM MCE on SMI entry [29]	2025	AMD	*SMM-code→Shielding→Race-condition*
SPD input validation [26]	2025	AMD	*SMM-code→Uarch→BadRAM*
AMD SMM vulnerabilities [28]	2025	AMD	*SMM-code→Shielding→SMMCallout*
Intel slim bootloader [100]	2025	Intel	*Plat-config→FW→Memory-Controller*
AMD client CPUs — input validation [20]	2025	AMD	*SMM-code→Shielding→CD*
AMD client CPUs vulnerabilites Aug [21]	2025	AMD	*SMM-code→Shielding→CD*
AMD embedded CPUs — input validation [23]	2025	AMD	*SMM-code→Shielding→CD*
AMD embedded CPUs — input validation [24]	2025	AMD	*SMM-code→Shielding→CD*
AMD microcode signature verification [22]	2025	AMD	*Plat-config→HW→ucode*
AMD server CPUs — input validation [25]	2025	AMD	*SMM-code→Shielding→CD*
AMD Ryzen & Athlon [30]	2026	AMD	*SMM-code→Shielding→CD*
Intel firmware advisory [102]	2026	Intel	*SMM-code→Shielding→CD*
AMD embedded CPUs — input validation [31]	2026	AMD	*SMM-code→[MS→[Spatial] | Shielding→[CD | SMMCallout]]*
AMD embedded CPUs vulnerabilities May [33]	2026	AMD	*SMM-code→[MS→[Spatial] | Shielding→[CD | SMMCallout]]*
AMD EPYC CPUs — input validation [32]	2026	AMD	*SMM-code→Shielding→CD*
Incorrect user of LocateProtocol [34]	2026	AMD	*SMM-code→Shielding→CD*
UEFI Reference Firmware Advisory [101]	2026	Intel	*SMM-code→Shielding→CD*