

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

STALEUS: Breaking AMD SEV-SNP via Memory Incoherence

Benedict Schlüter and Shweta Shinde, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity26/presentation/schlueter-staleus

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

STALEUS: Breaking AMD SEV-SNP via Memory Incoherence

Benedict Schlüter Shweta Shinde
ETH Zurich

Abstract

AMD SEV-SNP provides confidential computing capabilities on AMD platforms. To secure confidential virtual machines, AMD relies on a trusted co-processor known as the Platform Security Processor (PSP). The PSP executes on an Arm core, which is connected to the x86 interconnect via dedicated hardware bridges. We show that a malicious hypervisor can configure one of these bridges to render the PSP cache-incoherent with respect to the x86 caches. We present a novel attack called STALEUS that demonstrates how to leverage this vulnerability to break SEV-SNP. We show the impact of STALEUS by achieving arbitrary read and write primitives within a fully attested CVM.

1 Introduction

Confidential computing has received industry adoption [2, 26, 41]. A cornerstone in this development is the notion of Confidential Virtual Machines, or CVMs. These environments provide a critical guarantee: shielding customer workloads from inspection or tampering, even by potentially malicious cloud providers. AMD Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP) is one such technology [1]. Unlike Intel TDX or Arm CCA [11, 30], which rely on different architectural foundations, AMD anchors its SEV-SNP Root of Trust (RoT) in a dedicated co-processor: the Platform Security Processor (PSP).

For scalability and performance, AMD employs a modular chiplet architecture, fabricating distinct platform components individually to optimize silicon yield. However, this distributed design introduces a critical challenge: efficiently interconnecting these discrete dies within a single package [47]. Since the introduction of the Zen microarchitecture [8], AMD has addressed this via its proprietary high-speed interconnect, the Infinity Fabric [14]. Designed for high throughput, the Infinity Fabric serves as the backbone of the SoC, facilitating both power efficiency and scalable performance.

Architecturally, the fabric is divided into two distinct planes: a data plane that manages memory request routing, and a control plane tasked with configuration and orchestration. This control plane is designated as the System Management Network (SMN), an area that has thus far largely escaped scrutiny from the security research community. Yet, the SMN encompasses control over almost all Intellectual Property (IP) blocks within the system. To mitigate this risk, AMD implements a basic access control mechanism on the SMN to restrict unauthorized modifications.

Beyond interconnectivity, AMD's heterogeneous architecture poses a second fundamental challenge: maintaining a coherent memory view across diverse platform units. AMD resolves this with a hardware mechanism where functional coherence units snoop caches prior to the memory controller, ensuring consistency across the system. However, strict coherency enforcement can induce latency variability, which is undesirable for latency-sensitive workloads. Consequently, the architecture permits specific devices to willingly opt out of the hardware-enforced coherency.

This architectural heterogeneity introduces yet another layer of complexity within the AMD SoC. Various IP blocks, some potentially integrated from third parties, such as the Arm-based PSP, often rely on different bus protocols. To bridge this gap, AMD employs specialized translation units, known as bridges, to convert external bus protocols into AMD-proprietary standards. Critically, these bridges require granular configuration to handle transaction semantics appropriately for different components. To support this configurability, AMD exposes bridge settings within the SMN, details of which appear in their open-source code.

In this paper, we present the STALEUS attack that exploits AMD bridge settings to compromise SEV-SNP. First, as the public code omits security-critical definitions, we reverse-engineer and map the bridge configuration landscape for a particular IP. Our methodology relies on the architectural convergence between AMD GPUs and CPUs, both of which use the Infinity Fabric. We discover that, despite layout differences, the functional register definitions for the fabric are similar across GPU and CPU implementations. This insight allows us to transpose the GPU Infinity Fabric documenta-

tion found in the Linux kernel to the CPU context. Guided by this documentation, we identify a security-critical bridge configuration that controls the coherency attributes assigned to externally connected IP blocks. We confirm that the PSP functions as one such external IP within this framework.

Second, our analysis reveals a critical gap: a malicious hypervisor can modify the bridge settings to alter the coherency attributes of the PSP. By actively desynchronizing the PSP's coherence state, we can artificially induce a split memory view between the PSP and the x86 cores. In this state, the PSP bypasses caches to access DRAM directly, whereas x86 cores operate on modified data retained within their caches. Consequently, a memory read initiated by the x86 cores returns a different value than a concurrent read performed by the PSP at the same physical address. This induced incoherency poses severe security risks, particularly because the PSP acts as the Root of Trust for SEV-SNP.

Finally, we exploit this divergent view to: (i) hide data updates from the PSP, effectively forcing it to use stale values resident in DRAM; (ii) selectively flush cached x86 data to DRAM, overwriting any legitimate state changes committed by the PSP. We demonstrate that STALEUS bypasses SEV-SNP guarantees. Specifically, we maliciously enable debug privileges in a fully attested CVM, inject arbitrary code, and bias the Linux kernel's random number generator.

STALEUS is a part of a larger attack family named Interconnect Corruption Attacks (XCA), wherein a malicious hypervisor reconfigures the Infinity Fabric. Contrary to its predecessors Fabricked and BreakFAST [24, 56], STALEUS does not redirect PSP memory transactions but instead alters their coherence setting by virtue of a different root cause.

Contributions We introduce the first attack vector that exploits the configuration of Intellectual Property (IP) blocks within AMD's System Management Network (SMN). By manipulating these configurations, we force the PSP into a non-coherent state with respect to x86 DRAM, thereby completely breaking the integrity and confidentiality guarantees of SEV-SNP. Furthermore, we provide the first analysis of the SMN to contextualize the vulnerability. All code and data are available at https://xca-attacks.github.io/staleus/

Responsible Disclosure. We responsibly disclosed our findings to AMD in September 2025. AMD acknowledged the vulnerability and assigned CVE-2025-54509.

2 Background

We introduce the AMD platform background, in particular the relevant aspects that are integral to STALEUS.

2.1 SEV-SNP

SEV-SNP is AMD's latest technology to enable confidential VMs (CVMs). It guarantees the confidentiality of guest data via memory encryption, while ensuring integrity through a dedicated data structure known as the Reverse Map Table (RMP). Residing in DRAM, the RMP maintains granular security attributes for every 4KiB page within the system. Microcode or hardware reads the RMP to verify whether a memory access is consistent with access restrictions. This validation occurs synchronously during every memory read or write operation initiated by the CPU or I/O devices. Any unauthorized access triggers a hardware-level page fault. x86 cores can change the RMP through specific assembly instructions (e.g., `RMPUPDATE`) that check the legitimacy of each request. The need for the RMP becomes apparent when examining the hypervisor's capabilities.

The hypervisor is in full control of the Second Level Address Translation (SLAT) page tables and can maliciously modify them. Earlier iterations (SEV and SEV-ES) proved susceptible to these SLAT manipulation attacks [45, 46, 60]. With SEV-SNP, the RMP detects such malicious changes from the hypervisor to the SLAT page tables to prevent the guest from using malicious data. To achieve this, the RMP cross-references SLAT translations against the reverse mapping stored within the RMP to ensure consistency.

2.2 Platform Security Processor

SEV-SNP establishes its root of trust in a dedicated secure co-processor known as the Platform Security Processor (PSP). The PSP exposes a set of management APIs to the hypervisor, facilitating guest creation and CVM lifecycle orchestration. Critical among these interfaces are the `SNP_PAGE_MOVE`, `SNP_DBG_ENCRYPT`, and `SNP_DBG_DECRYPT` APIs. The former instructs the PSP to migrate a page between physical locations in DRAM, strictly enforcing SEV-SNP invariants. The hypervisor is permitted to invoke the latter two APIs exclusively for CVMs where the owner has explicitly authorized debugging. Guests verify the debug-enable status by querying an attestation report generated by the PSP. AMD stores both the attestation report and the debugging status within a secure 4KiB page in DRAM called the Guest Context Page. Write access to the Guest Context Page is restricted solely to the PSP, which encrypts the data using its private key. Consequently, the write protection limits the hypervisor to reading only the encrypted ciphertext of the Guest Context Page.

Physically, AMD integrates the PSP directly onto the CPU SoC package. Designed as a lightweight co-processor, the PSP is built upon the Arm architecture. More specifically, AMD uses an Arm Cortex A5 processor. To facilitate interaction with x86 cores and the memory subsystem, the PSP interfaces via the AXI interconnect to the remaining parts of the system.

2.3 AMD Infinity Fabric

The AMD platform architecture comprises diverse heterogeneous components, including CPU cores, memory controllers,

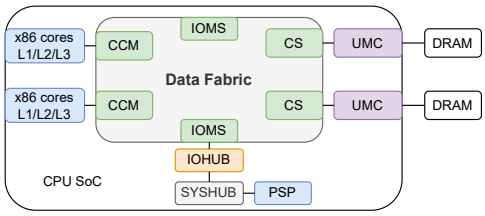


Figure 1: High-level overview of an AMD SoC.

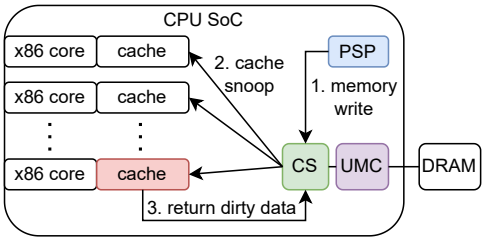


Figure 2: Memory coherence when the PSP writes to x86 DRAM. 1. PSP write arrives at CS block. 2. CS block snoops x86 caches to invalidate potential dirty data. 3. The x86 core responds with dirty cache data and marks the cache line as clean in its cache.

and I/O subsystems.This disaggregated design necessitates a fast and reliable interconnect for performance. Introduced with the Zen architecture, AMD leverages the Infinity Fabric, structurally divided into the Data Fabric (DF) and Control Fabric (CF) [14]. Figure 1 shows a high-level overview of Data Fabric components.

The Data Fabric serves as the primary transport layer, routing memory requests between master and slave endpoints across the fabric. The Control Fabric, also known as the System Management Network (SMN), provides a configuration plane to manage and modify Data Fabric transactions. Table 1 briefly introduces the most important IP units within an AMD SoC. Due to its architecture as an Arm-based processor, the PSP lacks native integration with the Data Fabric. To bridge this architectural gap, AMD utilizes specialized bridges to interface external IPs, such as the PSP, with the Data Fabric. The SYSHUB functions as one such bridge, translating standard external bus protocols (e.g., AXI) into the proprietary AMD SDP protocol.

2.4 Advanced eXtensible Interface (AXI)

The Advanced eXtensible Interface (AXI) is a high-speed on-chip interconnect specification developed by Arm. Within Arm-based SoCs, AXI serves as the primary fabric connecting the CPU to memory controllers and various peripheral devices. Arm architected the protocol to deliver high bandwidth and low-latency communication between on-chip components. Similar to contemporary bus protocols like TileLink, AXI offers significant flexibility, supporting the transmission of auxiliary signals alongside standard data and address lines. Crucially for this work, the AXI standard optionally includes a 4-bit field known as the AxCache signals [12].These four bits define the caching attributes of the memory request and instruct the memory controller not to snoop any caches when the request arrives. According to the specification, this bypass behavior occurs specifically when bits AxCache[2] and AxCache[3] are asserted [12].

Arm processors, which natively use AXI, are frequently embedded as co-processors within larger, heterogeneous SoCs (e.g., the AMD PSP). Consequently, protocol translation becomes essential to interface AXI with other, often proprietary, system interconnects. Hardware units known as bridges facilitate this translation by mapping AXI transactions to the target protocol. For instance, AMD integrates a dedicated bridge to translate AXI traffic to the Scalable Data Port (SDP) protocol [9]. Given the distinct signal definitions across protocols, these bridges transform signals based on their configuration, potentially pruning or synthesizing signals for compatibility.

2.5 System Management Network (SMN)

The System Management Network (SMN), also known as Control Fabric (CF), serves as the configuration backbone for all IP blocks on the AMD platform [14]. Although AMD documents the SMN sparsely, analysis of open-source repositories indicates it plays a pivotal role in platform initialization [5, 9]. Architecturally, the SMN spans a 4GiB address space that directly maps to the hardware registers of various on-chip IPs. Multiple IPs read and modify the SMN to execute their specific management functions [14]. For instance, the PSP leverages the SMN to configure security barriers within the memory controller, thereby enforcing SEV-SNP semantics [5]. The mechanism for accessing the SMN varies depending on the originating IP. Typically, the PSP maps the SMN directly into its own address space, whereas x86 cores interface with the SMN via a data/index register pair located in the PCIe configuration space [5, 6].

Given that the SMN houses security-critical configuration parameters (e.g., memory controller security settings), it enforces a strict access control model. Transactions initiated by x86 cores via PCIe configuration registers are assigned security level 7, effectively the least privileged state [6]. Contrary to this, accesses originating from the PSP are granted the highest privilege level [5]. Should x86 cores attempt to access a protected SMN register, the request is denied, returning a dummy value of `0x0` or `0xffffffff`.

2.6 Memory Coherence

Memory coherence constitutes a fundamental requirement for modern heterogeneous computing platforms. Distinct agents, including multiple CPU cores, external I/O peripherals, and

Table 1: Brief description of Infinity Fabric components.

State	Usage
CCM	Core/Cache Coherent Master units handle incoming requests from x86 CPU cores residing in the CCX units.
IOMS	I/O Master Slave units combine the master (source) and slave (destination) IP in one instance. They connect the I/O subsystem with the remaining parts of the \df and can act as the source or destination of transactions.
CS	Coherent Slave units are the destination for memory requests targeting DRAM. They ensure memory coherency and perform access controls.
UMC	Unified Memory Controller handles the physical configuration of the DRAM.
IOHUB	Routing crossbar that organizes and routes traffic coming from I/O devices.
SYSHUB	Routing crossbar that translates incoming transactions from external IPs into AMD's bus protocol.

internal co-processors such as the PSP, all contend for access to DRAM. Coherence protocols ensure that each agent maintains a consistent view of system memory, irrespective of local caching states. AMD enforces this coherence within its Coherent Slave (CS) units, as illustrated in Figure 2. Upon receiving a memory request targeting DRAM, a CS unit broadcasts probe requests to all system agents to verify if the target data is currently cached. If a cache holds the requested line, it responds to the probe by supplying the most recent data.

While effective, these broadcast probe requests generate substantial interconnect traffic, increasing latency for DRAM reads and writes. To mitigate this overhead, CS units incorporate a System Probe Filter (SPF) designed to minimize unnecessary snoop traffic [3]. The SPF maintains a directory of every cache line currently resident in system caches that maps to the DRAM managed by that CS unit. When a memory request arrives, the CS consults the directory and dispatches directed probes only to specific holders, or suppresses probing entirely if the line is uncached. This use of selective probing may yield performance gains [4]. However, for advanced optimization, specific memory transactions are permitted to bypass the coherence mechanism entirely. For instance, PCIe devices can assert the NoSnoop attribute within the TLP header to request a non-coherent transaction [49]. If such a request reaches the CS unit, it bypasses the SPF consultation and is forwarded directly to the memory controller, regardless of whether the line is dirty or resident in x86 caches. While this creates an intentionally non-coherent data view, it offers critical latency benefits for specific workloads.

3 Overview

We introduce the high-level workings of the STALEUS attack.

3.1 Threat Model

We operate in the AMD-defined SEV-SNP threat model [7]. We trust the hardware, AMD-supplied firmware, and the software within the CVM. All other components in the system, especially the hypervisor, are untrusted and act maliciously to violate the integrity or confidentiality of CVMs. The hypervisor has access to the initial CVM boot image such that it can bootstrap the CVM correctly.

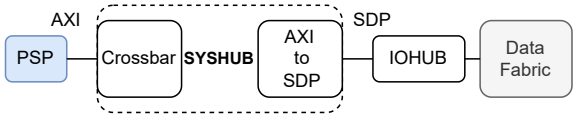


Figure 3: Data flow within the SYSHUB components connecting the PSP with the Data Fabric.

3.2 STALEUS Attack

The SYSHUB serves as AMD's architectural gateway for integrating third-party IP blocks into the Data Fabric. Among other functions, it converts standard AXI transactions into AMD's proprietary Scalable Data Port (SDP) bus protocol. Our reverse engineering efforts reveal that the SYSHUB services multiple clients, all of which interface via an internal crossbar (see Section 4.1). Crucially, the PSP functions as one of these clients whenever it initiates access to the Data Fabric (e.g., targeting x86 DRAM).

The SYSHUB is responsible for translating PSP AXI signaling into compatible SDP signals. Figure 3 visualizes this data flow from the PSP through the SYSHUB to the Data Fabric. Because the translation from AXI to SDP is not a one-to-one mapping, and given that data and control signals possess differing widths, the SYSHUB must aggregate specific AXI data lines or synthesize new SDP-specific signals. The SMN configuration for the SYSHUB exposes control bits that define how the cache coherence attributes propagate to the SDP interface. Modifying this configuration alters the memory coherence properties of the PSP with respect to x86 DRAM. PSP memory requests tagged with the NoSnoop attribute result in the transactions ignoring dirty data residing in x86 caches. Figure 4 visualizes the non-coherent PSP memory accesses. As depicted in a), the PSP accesses DRAM directly, disregarding whether an x86 core currently holds the corresponding cache line. This behavior establishes an incoherent memory view between x86 cores and the PSP.

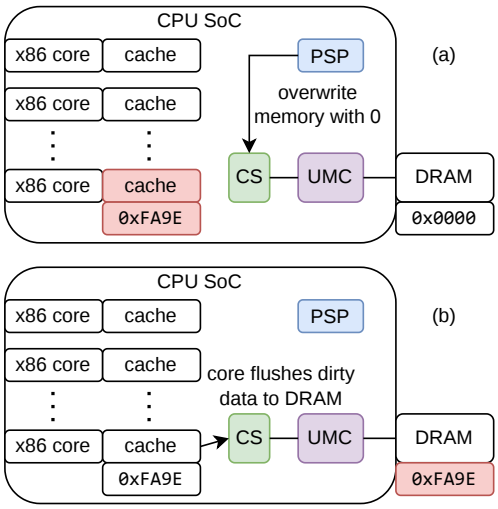


Figure 4: a) PSP memory requests targeting DRAM are non-coherent. b) x86 cores directly or indirectly flush cache lines to DRAM and overwrite PSP data.

Table 2: AMD processors vulnerable to STALEUS.

Generation	Test CPU	Launched	Platform Init
Zen 4	EPYC 9124	10/11/2022	1.0.0.F
Zen 5	EPYC 9135	10/10/2024	1.0.0.0

Figure 4 b) demonstrates the consequence, where x86 cores have residual dirty cache states not updated by the PSP write. When the core subsequently flushes its cache lines (directly or indirectly) to DRAM, the dirty x86 data overwrites the data previously written or read by the PSP.

In STALEUS, we demonstrate how an attacker can exploit non-coherent PSP memory accesses to compromise SEV-SNP. Crucially, this non-coherent access manifests only when x86 cache and DRAM data are divergent. We distinguish two scenarios: (i) If the PSP reads x86 DRAM disregarding x86 caches, it retrieves stale data. (ii) If the PSP writes to DRAM while x86 caches hold corresponding lines dirty, a subsequent cache flush will overwrite the PSP-committed data. We leverage this primitive to forge a critical CVM metadata structure, the Guest Context Page. By utilizing a forged Guest Context Page, we gain the ability to read and write arbitrary CVM memory, effectively violating SEV-SNP security guarantees. Our attack achieves a 100% success rate and does not require single-stepping. We evaluated STALEUS across multiple CPU generations, identifying the two latest AMD architectures available at the time of writing as vulnerable. Table 2 details the specific CPU models and Platform Initialization (PI) versions vulnerable to STALEUS.

4 Reverse Engineering

We discuss the discovery of our attack and how we abuse the lack of memory coherence to break PSP invariants.

4.1 Discovery

The PSP interfaces with the Data Fabric through a complex network of internal bridges and IP blocks. While the precise topology of the data flow within AMD's SoC remains proprietary, we can reconstruct parts of the architecture by gathering information from publicly released source code and technical documentation. A primary source for this reconstruction is openSIL, which handles early x86 silicon initialization [9]. The openSIL codebase exposes numerous SMN configuration registers that the UEFI programs during the boot sequence. A small subset of these registers belongs to the SYSHUB bridge, the likely interface point for the PSP. Specifically, openSIL identifies this bridge as `A2S_CNTL`, a component within the SYSHUB bridge. We hypothesize that this unit manages the AXI-to-SDP translation for various external components connected to the chipset, including the PSP [5, 9, 14]. Since openSIL defines only the `A2S_CNTL_SW0_SDP_WR_CHAIN_DIS` configuration, we extended our search to other public AMD documentation to uncover additional configuration options within the `A2S_CNTL` SMN region.

Both AMD CPU and GPU architectures leverage the Infinity Fabric as a common interconnect. Consequently, they likely integrate identical or analogous hardware blocks. By querying the Linux kernel source for the string `A2S_CNTL`, we discovered GPU headers defining internal `A2S_CNTL` parameters for the GPU SYSHUB [58]. However, because these headers correspond to the Vega GPU generation (released in 2017), the North Bridge Interface (NBIF), which houses the SYSHUB, has likely evolved. As a result, the exact offsets cannot be derived directly from the Linux headers. Nevertheless, given the persistence of `A2S_CNTL` SMN registers in both Vega and Zen 5, it is highly probable that several Vega-era registers persist in Zen 5, despite the lack of public documentation. We isolated the `A2S_CNTL_CL0__NSNOOP_MAP` control, defined in `drivers/gpu/drm/amd/include/asic_reg/nbif/nbif_6_1_sh_mask.h`, as a potentially security-critical register [58]. According to the header definition, the `NSNOOP_MAP` control occupies the two least significant bits of the `A2S_CNTL_CL` control register. Based on this nomenclature, we hypothesize that this register controls the SDP NoSnoop attribute. Although the IP configuration definitions in the Linux kernel are specific to the Vega series, if this register persists on Zen CPUs, is mutable by the hypervisor, and effectively controls the PSP connection to the SYSHUB, it could be exploited to violate PSP security assumptions. To summarize:

1. AMD connects the PSP through the SYSHUB with the Data Fabric

2. The SYSHUB contains an AXI to SDP bridge
3. The SMN exposes an A2S_CNTL configuration range in the SYSHUB IP AXI-to-SDP bridge.
4. The A2S_CNTL allows the definition of the cache coherence attributes for SDP memory requests

4.2 Finding the NoSnoop Bit

We hypothesize that the A2S_CNTL_CL0__NSNOOP_MAP configuration persists in the Zen CPU architecture and governs the memory coherence attributes of the PSP. In the next step, we validate this hypothesis. We analyze the SYSHUB SMN address space for Zen 5. The AMD EPYC platform for Zen 5 integrates four distinct SYSHUBs, each allocated a dedicated 1MiB region within the SMN [9]. Because AMD documentation does not specify which SYSHUB interfaces with the PSP, we systematically analyze all four regions. These SYSHUB blocks occupy contiguous 1 MiB address spaces starting at 0x01400000, 0x01500000, 0x01600000, and 0x01700000, respectively [9]. The Linux kernel defines a default initialization value, 0x2a80540, for the A2S_CNTL_CL0 register in Vega GPUs. Remember that the value includes the NoSnoop field configuration in the two least significant bits. Scanning the Zen 5 SMN address space for this specific value yields six partial matches. All six matches reside within three of the SYSHUB address ranges and share identical offsets of 0x3a90 or 0x3a94. Upon closer inspection, we identify 18 values in the range 0x3a80 to 0x3ac4 which likely correspond to A2S_CNTL_CL0 to A2S_CNTL_CL17. From this, we infer that the Zen 5 SYSHUB supports 18 internal clients, a significant expansion from the five clients (CL0–CL4) documented in the Vega GPU headers. If our hypothesis holds that the PSP is one of these 18 clients, we should be able to manipulate its memory coherence attributes by modifying the two least significant bits of the corresponding register.

We observe that for all identified clients (CL0–CL17), the two least significant bits are initially cleared. In our first experiment, we toggle the least significant bit for all 18 clients across all four SYSHUBs. This modification results in PSP error messages when attempting to invoke PSP API calls from the hypervisor. Detailed analysis of this behavior confirms that the faults stem from the PSP consuming stale data from DRAM, ignoring the most recent updates residing in x86 caches. In other words, we change PSP memory access to x86 DRAM to be non-memory coherent.

Identifying the PSP. In the final phase of our analysis, we isolate the specific client index assigned to the PSP and identify its corresponding SYSHUB. For the experiment, we systematically toggle bits within the A2S_CNTL_CL-X range across distinct SYSHUBs, followed by a PSP API invocation. We observed API failures exclusively when the least significant bit was flipped for SYSHUB #2 at offset 0x3AA8, a register likely corresponding to A2S_CNTL_CL10. Consequently, we infer that the PSP is architecturally located behind SYSHUB #2, operating as client 10 on the internal crossbar.

5 STALEUS Building Blocks

We introduce the building blocks for the case studies by showing how we abuse the non-coherent PSP in SEV-SNP.

5.1 Abusing Missing Memory Coherence

We demonstrate how non-coherent PSP memory requests induce observable architectural state changes. We induce this loss of coherence in PSP memory transactions by modifying the SYSHUB SMN configuration. Without memory coherence, the PSP may read or write stale data when accessing x86 DRAM. The hypervisor has the privilege to alter these PSP coherence settings at any arbitrary point in time.

Enabling the NoSnoop configuration forces the PSP to access DRAM directly, bypassing x86 cache snooping. If x86 caches do not contain lines corresponding to the data the PSP accesses, this non-coherence produces no side effects. This occurs because DRAM already holds the most recent data, and no divergent entries exist within the x86 cache hierarchy. Therefore, for the NoSnoop attribute to manifest a side-effect, x86 cores must cache the specific data region that the PSP targets. In summary, two preconditions must exist for the altered PSP memory coherence to result in a divergent memory view between the PSP and x86 cores:

1. The x86 cores previously wrote data to a target page, leaving dirty lines in its cache.
2. The PSP subsequently attempts to read from or write to that same page.

If both conditions are met, the lack of memory coherence violates PSP security invariants, which ultimately leads to a compromise of SEV-SNP, as we show later. Figure 5 shows the PSP memory access with and without memory coherence for both reads and writes.

PSP reads. Figure 5 a) shows the cache and DRAM states for coherent and non-coherent PSP reads. The PSP operates on x86 DRAM to fulfill its management tasks (e.g., CVM creation/initialization). During these operations, the RMP guarantees the PSP exclusive write access to the target DRAM [7]. This follows security best practices to mitigate ToCToU-type vulnerabilities. However, when the PSP reads from DRAM without regarding dirty x86 cache lines, it may retrieve stale data. Consequently, the x86 cores view a more recent version of the memory content than the PSP does.

PSP writes. Figure 5 b) shows the cache and DRAM state for coherent and non-coherent PSP writes. When PSP writes carry the NoSnoop attribute, they target DRAM directly, bypassing the x86 cache snoop mechanism. Thus, there may still be dirty

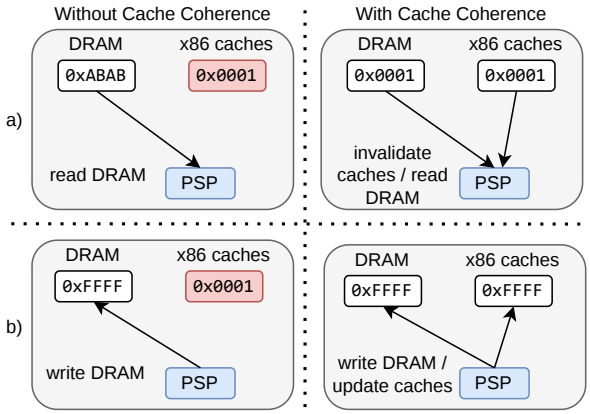


Figure 5: **(a)** The PSP reads DRAM. If the PSP memory requests are non-coherent, it will read stale DRAM data. **(b)** PSP writes DRAM. If the PSP memory requests are non-coherent, it will overwrite stale DRAM data. A future cache flush will overwrite the PSP written data.

data in the x86 caches. Once the PSP completes its write operation, the x86 cores will eventually evict this dirty data back to DRAM. This write-back process causes stale cache data to overwrite DRAM regions that should seemingly remain write-protected against x86 access. Thus, this flush overwrites the fresh data the PSP wrote during its non-coherent write sequence.

5.2 RMP Entry Cache Flush

STALEUS requires the PSP to access pages that have been previously written by x86. However, before the PSP accepts a page for use, the hypervisor must set the lock bit using RMPUPDATE. This ensures exclusive PSP access to the page enforced by the RMP. For STALEUS, an attacker needs to change the RMP state of the page holding the dirty cache lines. This poses a problem because prior research demonstrates that AMD caches RMP entries directly within the x86 cache hierarchy [13]. However, we need to ensure that the PSP reads the latest RMP state but not the latest copy of the page used for the attack.

The hypervisor cannot use `clflush` to synchronize the RMP entry cache with DRAM, since microcode accesses and caches the RMP based on the physical address [52]. Further, mapping the RMP in the x86 address space and using `clflush` on the virtual address does not work, as we experimentally confirm. To resolve the problem, we leverage insights from previous works [17,25]. Specifically, by accessing the RMP entry with a physical address where we set the C-bit (encrypted bit), we indirectly instruct the Coherent Slave (CS) unit to flush all corresponding non-C-bit (unencrypted) cache lines to DRAM. In essence, we trigger an encrypted memory read of the RMP entry to force the eviction of the non-encrypted RMP entry from the caches.

```
1  static void clflush_rmp_entry_of_hpa(u64 hpa){
2      u64 rmp_base, rmp_entry_hpa;
3      rdmsrl(MSR_AMD64_RMP_BASE, rmp_base);
4      rmp_entry_hpa = rmp_base + 0x4000 + ((hpa >> 8));
5
6      u64 rmp_entry_aligned = rmp_entry_hpa & ~0xFFFull;
7      // map with c-bit
8      u64 rmp_entry_virt = 0xFFFFFFFFFFF0F000;
9      map_physical_to_virtual(rmp_entry_aligned,
        rmp_entry_virt, (1ull << 51));
10     u64 *rmp_entry_virt = (u64*)(rmp_entry_virt + (
        rmp_entry_hpa & 0xFFF));
11     READ_ONCE(*rmp_entry_virt);
12 }
```

Listing 1: Flush an RMP entry from the x86 caches.

We investigated this behavior further to identify the architectural root cause overlooked by previous work [17,25]. Architecturally, the CS blocks integrate System Probe Filters (SPFs). These SPFs maintain a directory, organized by line and region, tracking all active system cache lines associated with data managed by that CS unit [4]. Upon receiving a request with the C-bit set, the CS block appears to invalidate all region directory entries for the corresponding non-C-bit values. Depending on the interleaving configuration, this mechanism impacts a region ranging from 256 bytes up to 2KiB [25].

Implementation We implement the selective flush as a Linux kernel function shown in Listing 1. Using calculations from previous work, we compute the base address of the RMP entry belonging to a host physical address (hpa) [52]. In the next step, we page-align the returned value and create a page table mapping. The page table mapping directly modifies the CR3 register in the kernel and sets bit 51, the c-bit, as part of the physical address. The c-bit acts as a notifier to the memory controller to encrypt the data associated with the physical address. We add the page offset to the virtual address in line 10 and perform the memory read in line 11. After the read, the modified RMP entry will be written back to DRAM, and the caches are synchronized for that cache line.

5.3 Forging Guest Context Pages

The PSP allocates a dedicated page, called the Guest Context Page, to store metadata for each CVM. The PSP keeps exclusive control over this page. Given that the Guest Context Page houses highly sensitive data, such as the attestation report, the PSP protects it using inline AES encryption [5]. This inline encryption mechanism utilizes the PSP's internal encryption engine and keys, rather than relying on standard memory controller-based encryption. Prior research, corroborated by our experiments, indicates that AMD employs neither AES-ECB nor AES-GCM, as the ciphertext reveals location dependence and lacks integrity protection [21]. SEV-SNP guest security relies fundamentally on the invariant that ex-

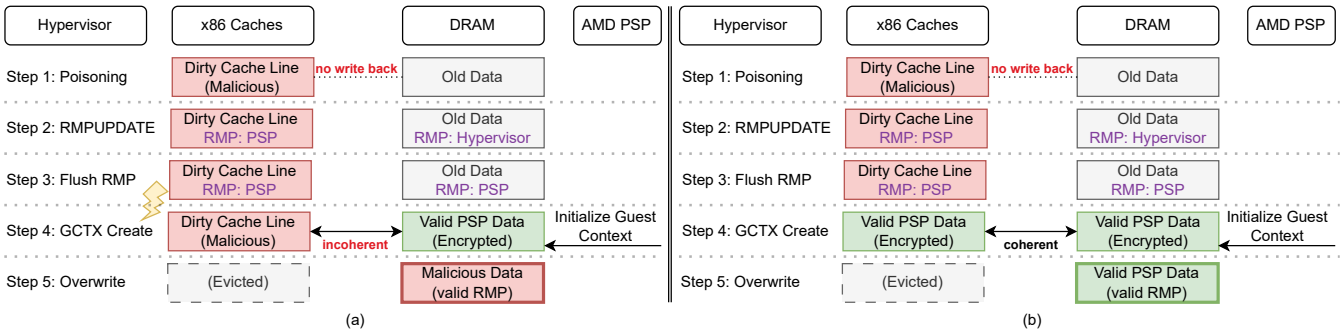


Figure 6: STALEUS attack to forge a Guest Context Page: (a) Attack flow with a non-coherent PSP after Step 3. (b) Benign flow with a memory coherent PSP.

actly one Guest Context Page exists for each Address Space Identifier (ASID), which serves as the unique CVM identifier. Thus, AMD enforces a strict bijective binding between the ASID and the Guest Context Page. The Guest Context Page contains the attestation report alongside the `GuestPolicy` fields. This 32-bit `GuestPolicy` value defines the security attributes of the guest. The debug-enable bit is part of the `GuestPolicy`. When set, this flag authorizes the hypervisor to read and write arbitrary CVM memory.

Figure 6 a) depicts the attack flow as well as the benign flow without STALEUS in b). We introduce the attack steps.

1. Poisoning: To instantiate a new CVM, the hypervisor first reserves a future Guest Context Page. Given the security relevance of the Guest Context Page, only the PSP holds the capability to initialize and control it. However, prior to transferring the page to the PSP, the hypervisor has unrestricted access. During this window, the hypervisor actively seeds dirty cache lines targeting that specific page.

2. RMPUPDATE: The PSP only operates on pages it has exclusive access to. Consequently, the hypervisor employs RMPUPDATE to transfer page ownership to the PSP. RMPUPDATE transitions the page state, effectively revoking write permissions for x86 cores. Crucially, the updated RMP entry remains within the x86 caches, leaving DRAM with a stale RMP state copy.

3. Flush the RMP: To synchronize the cached RMP entry with DRAM, the hypervisor can leverage the insights detailed in Section 5.2. By accessing the RMP entry with the C-bit set, the hypervisor forces the hardware to flush the RMP entry cache line to DRAM.

4. Page Move: The hypervisor now activates STALEUS to render the PSP non-memory coherent. Subsequently, it invokes the `SNP_GCTX_CREATE` PSP API to initiate the Guest Context Page creation. PSP source code analysis reveals that the PSP prepares the new Guest Context Page by zeroing the memory and flagging it as uninitialized [5]. As the PSP constructs the new Guest Context Page, it operates under the assumption that x86 cores possess no further access to the page.

5. Overwrite: Finally, the hypervisor triggers a flush of the x86 caches, either directly (e.g., via `wbinvd`) or indirectly (e.g., via natural eviction), overwriting the content of the Guest Context Page. Because the PSP successfully promoted the page to a Guest Context Page state, the RMP metadata remains valid. Nevertheless, the stale x86 data corrupts the actual page content. This effectively creates a second valid Guest Context Page for a CVM with hypervisor-controlled content.

5.4 Dropping Guest Writes

We show how an attacker can leverage STALEUS to selectively drop arbitrary CVM writes to DRAM. CVM operations rely on CPU-based caching to minimize DRAM access latency. Consequently, when a CVM yields control to the hypervisor for management tasks (e.g., handling I/O or timer interrupts), the caches retain dirty data written by the guest. Without STALEUS, the caching poses no security risk; the platform ensures coherency during eviction or concurrent access. However, we show how STALEUS allows an attacker to drop the writes in the caches.

Consider a scenario where the CVM writes data, rendering the corresponding cache lines dirty. We denote the page hosting this dirty data as the Page_v. The data resides within the L1/L2/L3 caches. Immediately following this write, the CVM exits to the hypervisor (e.g., triggered by a timer interrupt [61]). Thus, the CVM-written data remains dirty in the caches, leaving the physical DRAM with outdated content. The hypervisor then activates STALEUS, forcing PSP writes to be non-coherent. Next, it invokes the `SNP_PAGE_MOVE` API. The API call takes a page as source, which we denote as Page_s, and a page as destination, which we denote as Page_d. The hypervisor passes the Page_v as Page_s and a hypervisor-controlled page as Page_d. As the PSP executes the `SNP_PAGE_MOVE` call, it reads Page_s data directly from DRAM. Since the PSP bypasses the x86 caches, it retrieves stale data that lacks the recent CVM writes. Upon completion, the hypervisor reconfigures the SLAT page tables, redirecting

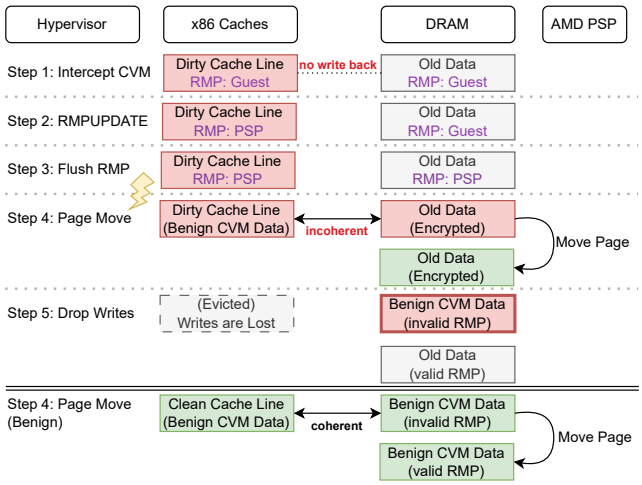


Figure 7: STALEUS attack to drop CVM writes. The lightning indicates we activate STALEUS and make the PSP non-coherent.

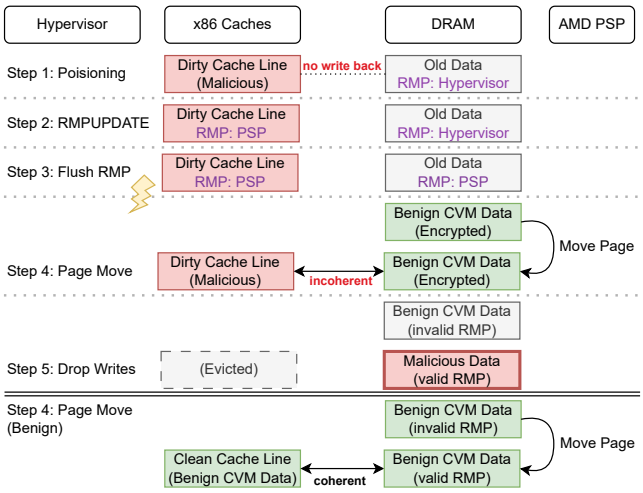


Figure 8: STALEUS attack to overwrite CVM memory. The lightning indicates we activate STALEUS and make the PSP non-coherent.

the CVM to Page_d. Page_d contains only the stale DRAM data, while the actual updates residing in Page_s cache lines eventually flush to the now-abandoned DRAM location From the perspective of the CVM, these writes effectively vanish. Figure 7 shows the data flow.

5.5 Overwriting Guest Memory

Using STALEUS, we demonstrate how an attacker can overwrite CVM memory. The hypervisor initiates the attack by writing data to a hypervisor-controlled Page_d. This action fills the x86 caches with dirty lines corresponding to this page. Next, it employs RMPUPDATE to transition the Page_d into a PSP-owned state. This step ensures the `SNP_PAGE_MOVE` API accepts the Page_d. Leveraging insights from Section 5.2, the hypervisor selectively flushes only the RMP entry to DRAM. As the PSP migrates guest data from the Page_s to the Page_d, the x86 caches retain the dirty data targeting the Page_d. The eventual eviction of these x86 cache lines overwrites the Page_d, corrupting the valid guest data. This directly violates SEV-SNP's integrity guarantees. However, because SEV-SNP encrypts these pages, the hypervisor lacks direct control over the plaintext produced when the guest decrypts this overwritten data. Figure 8 shows the attack steps. Section 6.3 describes how an attacker can circumvent encryption and use the primitive to inject arbitrary data.

6 Case Studies

Our setup consists of a Zen 5 EPYC 9135 16-core processor with 32 GiB DRAM. The platform runs AMD AGESA PI version 1.0.0.0 with microcode revision `0x0b002116` and SEV-SNP ABI version `1.55:44`. We use each of the primitives from Section 5 to break SEV-SNP.

6.1 Enabling Debug Mode

We use STALEUS to achieve arbitrary read/write within a CVM by enabling debug mode on production-level instances. Our attack consists of two phases.

1. Offline Phase. We first boot a CVM image with the debug-enable bit set. If a guest queries the attestation report, it will indicate that the PSP allows hypervisor debugging for this CVM instance. Thus, the guest would refuse to use the CVM and load any confidential data in it. However, we only use this CVM boot to snapshot the Guest Context Page of the CVM. We can read the ciphertext of the Guest Context Page. This is possible since the PSP only encrypts is but the RMP enforces the write-protection. We save the Guest Context Page content to a dedicated page in DRAM that we reserve during boot, such that we can later replay it. We shut down the CVM such that the PSP allows reusing the ASID.

2. Online Phase. We boot the benign image with the exact configuration that the guest requested. Importantly, we modify KVM such that it reuses the same ASID for this boot process that it used for the previous one. By default, KVM monotonically increments ASIDs because reusing ASIDs requires an API call to the PSP to flush Data Fabric caches [7]. Once the guest image is fully booted, we a correct Guest Context Page page associated with the guest. The Guest Context Page hosts the correct attestation value and has no debug bit set. Thus, when supplying the `SNP_DEBUG_DECRYPT` or `SNP_DEBUG_ENCRYPT` commands with the benign Guest Context Page, they refuse to execute.

We use STALEUS to create a new malicious Guest Context

```
/* create dirty cache entries*/
memcpy(new_gctx_virt_addr, memcpy_virt_addr, 0x1000);
rmpupdate_firmware(new_gctx_phys_addr);
clflush_rmp_entry_of_hpa(new_gctx_phys_addr);
```

Listing 2: Attack code to create dirty cache lines pointing to a future Guest Context Page and update the RMP.

```
smn_iohub_psp_a2s_ctrl_toggle_zen5();
swap_from_src_to_dst(kvm, hpa, 0x300005000, a0);
smn_iohub_psp_a2s_ctrl_toggle_zen5();
pr_info("swap from src to dst done\n");
swap_from_src_to_dst(kvm, 0x300005000, hpa, a0);
pr_info("swap from dst to src done\n");
```

Listing 3: Attack code to drop writes from a CVM.

Page using the primitive from Section 5.3. When creating the new Guest Context Page, we use the same physical address that we used when creating the snapshot. This is necessary since the PSP uses tweak values, similar to the memory controller, when encrypting data. Thus, if we chose a different address for the Guest Context Page, the content would be randomized, and the ASID would likely not match. However, as we chose the same physical address, the new Guest Context Page will have the same ASID as the current guest, but will have the debug-enable bit set. Subsequently, we can use the `SNP_DEBUG_DECRYPT` and `SNP_DEBUG_ENCRYPT` APIs, with the forged Guest Context Page to read and write arbitrary guest memory. This violates SEV-SNP guarantees. The attack is deterministic and works 100% of the time.

Implementation. We boot the Linux host kernel with a 1GiB reserved memory region at `0x300000000`. We modify the Linux kernel not to exclude the reserved memory for SEV-SNP. We adjust the ASID assignment of the CVM guests in the Linux kernel to always assign ASID 1 to a CVM. As we will reuse the ASID, we also add a PSP API call that flushes the Data Fabric caches. The PSP checks that this API call executes before allowing an ASID reuse. Lastly, we modify the `__sev_do_cmd_locked` function to execute `wbnoinvd` to make x86 writes visible to the PSP. This is necessary; the PSP would otherwise use stale command data and not execute the correct API call when the NoSnoop bit is set. After the `wbnoinvd` call but before making the final MMIO write to notify the PSP that all data is ready, we insert our attack code shown in Listing 2.

6.2 Poisoning Linux Randomness

We drop CVM Linux kernel writes in `get_random_bytes_user` using STALEUS. Many cryptographic userspace libraries use this function to obtain secure random numbers needed for seed generation. By reverting the write with secure random data from the Linux kernel, we effectively leave the userspace application with whatever randomness was previously in the page. To reset the write, we intercept the `get_random_bytes_user` from the hypervisor. This can be achieved using SLAT page faulting techniques from previous work [53, 54, 67, 68]. Once the function is intercepted, we execute `wbinvd` on all Linux cores. This is to ensure that the victim page does not hold other data that has not been written back to DRAM yet. Once we enter the guest again, we intercept it again after `get_random_bytes_user` copied the random bytes to userspace using `copy_to_iter`. When intercepting the guest the second time, we execute a page move of the userspace buffer where we disable PSP coherence. Thus, the PSP uses the stale data from DRAM as source when executing `SNP_PAGE_MOVE`. We enable PSP coherence again and revert the move so we do not have to adjust the SLAT page tables. After doing so, we have effectively skipped the effect of `copy_to_iter` in the CVM guest kernel. The userspace buffer has the same content it had before requesting random numbers from the Linux kernel. Thereby, we bias the random number usage of cryptographic libraries.

Implementation. We implement the attack in ∼235 LoC in the Linux host kernel. Listing 3 shows the host control code to perform the PSP non-coherence setting and swapping. We instruct the PSP to move the Page_v non-coherent to host physical address `0x300005000`. Subsequently, we revert the operation but with a coherent PSP.

6.3 Code Injection

We achieve arbitrary guest memory write, but using a different PSP API. Instead of relying on the debug-enabled APIs, we use `SNP_PAGE_MOVE`. Since this API may be optionally disabled by the guest due to previous attacks, we may use the mechanism from Section 5.3 to create another Guest Context Page page where the API is not disabled. Subsequently, we can supply the PSP with the malicious Guest Context Page when calling `SNP_PAGE_MOVE` to evade the restrictive guest policy in the original Guest Context Page.

An attacker can inject malicious code or data into the CVM address space. Prior work showed how CVM I/O interfaces can be used to bring attacker data into the CVM address space [46, 55]. To simplify exploit development, we omit single stepping and data injection in our implementation. We use the `SNP_PAGE_MOVE` API to move an already present payload within the guest CVM to the host physical address `0x300001000`. We snapshot the ciphertext at address `0x300001000`, containing the encrypted payload with the CVM key. Next, we use `SNP_PAGE_MOVE` to revert the initial move operation. After the PSP command finishes, we use `SNP_PAGE_RECLAIM` to transition address `0x300001000` back to the hypervisor state. This is necessary so we can write to page `0x300001000` and create dirty cache lines in the next step. As a final step, we move the CVM victim page containing the `icmp_rcv` function to physical address `0x300001000`.

```
/* create dirty cache entries*/
memcpy(0x300001000, ciphertext_snapshot, 0x1000);
rmpupdate_pre_guest(0x300001000, ASID, icmp_rcv_gpa);
clflush_rmp_entry_of_hpa(0x300001000);
```

Listing 4: Attack code to create dirty cache lines pointing to a future CVM page and update the RMP.

Before the final notification to the PSP to execute the API call, we disable the PSP memory coherency and create dirty cache lines pointing to address `0x300001000`. To do this, we use the Linux `memcpy` function and copy the snapshotted ciphertext to said address. When the PSP executes the `SNP_PAGE_MOVE` operation, it operates on DRAM, and thus, x86 cache lines remain untouched. Once the PSP completes, we make the PSP memory coherent and use `SNP_PAGE_MOVE` to revert the previous move. However, since the PSP is coherent again, it now snoops the x86 caches. After completion, the `icmp_rcv` page contains the plaintext we initially placed at a different place in the CVM. Under a real attack, the plaintext would have come from one of the various I/O interfaces of the CVM. Therefore, we have successfully demonstrated how an attacker could inject code into the CVM.

Implementation. We implement the attack in a kernel module with 653 LoC. The changes to the host Linux kernel are limited to the `__sev_do_cmd_locked` function and its helpers and amount to ∼135 LoC. We add the code in Listing 4 shortly before invoking the PSP page move API. Thus, we create the dirty cache lines as close as possible to the PSP API call to avoid potential eviction.

7 System Management Network

To contextualize the found vulnerability, we perform an initial SMN exploration. We analyze the security of the SMN and the number of registers accessible to the hypervisor.

7.1 Accessing the SMN

x86 cores access the SMN through a pair of index/data registers on the IOHUB. These IOHUB registers are embedded into the PCIe configuration space of an AMD internal device. The documented access is on every PCIe root port configuration space at offset `0x60` for the index and `0x64` for data. Kernel commits by AMD engineers unveil that there are other undocumented register offsets that can also be used to access the SMN [10].

For access, the x86 core writes the SMN address to read or write in the index register using a regular `mov` instruction. A subsequent read or write to the data register reads or writes to the respective SMN address. The access is subject to AMD's SMN security policy, and only a subset of the SMN registers is accessible to the hypervisor.

Table 3: Non `0x0` or `0xffffffff` SMN registers.

Generation	Test CPU	Platform Init	SMN register count
Zen 3	EPYC 7313	1.0.0.D	7417149
Zen 4	EPYC 9124	1.0.0.F	10209230
Zen 5	EPYC 9135	1.0.0.0	8609596

Security within the SMN. AMD defines 7 distinct security levels within its SMN, where 7 is the least and 0 the most privileged [5, 6]. The security level defines the access permissions for the SMN. Each SMN register (4-byte) likely has an assigned security level, and only when the security level of the requester is equal to or below the assigned level can it read or write the register. Access through the root port PCIe configuration space registers has a security level of 7. The PSP has the capability of directly mapping SMN address ranges into its address space. Accesses by the PSP have security level 0 [5]. AMD implicitly documents the security level of other on-chip components that have levels in between (e.g., System Management Unit (SMU) or x86 microcode with level 2). Important for STALEUS is which functional unit performs the security level controls. Our timing analysis shows that despite insufficient privileges to read certain SMN regions, we see differences in the cycle count when accessing those. This indicates that the access checks happen in a decentralized manner at an IP level. If that is the case, it raises the question of how and if the security level can be dynamically configured.

7.2 SMN compared to MSRs

Model Specific Registers control core-related configuration options for x86 cores. Reading or writing MSRs triggers a microcode routine that accesses the respective MSR data. Depending on the type of MSR accessed, the data may reside in a per-CPU exclusive SRAM or in DRAM [5, 31]. MSRs are scoped only to change x86-related controls and are independent of the bus architecture connecting the cores. Contrarily, the SMN exposes controls for almost every IP on the platform and even embeds certain MSR controls [14, 19]. Thus, the SMN acts as a superset of the MSR configuration. The MSR space consists of a few thousand active registers that can be actively used, while the SMN has millions of exposed registers [33]. Prior works fuzz the MSR space and search for undocumented control bits that may violate security guarantees [33, 68]. While such an analysis is feasible for the MSR space, it is infeasible for the SMN as we discuss in Section 8.2.

7.3 Organization within the SMN

The SMN is a 4GiB address space and can be extended by 4 bits for multi-socket configurations. The lower 20 bits define the offset configuration within a certain IP, whereas the upper

12 bits select the IP. The 4-bit extension selects the CPU if the motherboard has multiple sockets. Despite the 20-bit belonging to one IP, we observe access timing differences between different regions within a 20-bit sub-region. It follows that even within an IP block, different sub-IPs respond differently to configuration requests. Interestingly, we observe different timing for IP blocks even if the access is not permitted. Thus, the filtering based on the security attribute likely happens within an IP, rather than centralized [5, 9, 19, 50].

8 Discussion

We discuss STALEUS's root cause, possible mitigations, a more structured approach to similar vulnerabilities, and our results for the Zen 3 Infinity Fabric.

8.1 Root Cause Analysis

STALEUS is the first attack exploiting exposed SMN configuration registers. The root cause originates from an architectural flaw that allows kernel-mode x86 cores to modify the data flow configuration of the system's most privileged entity, the PSP. We demonstrate this vulnerability through a concrete instance where the hypervisor toggles the NoSnoop attribute of the PSP, thereby undermining its security invariants. It is highly probable that additional data flow configuration options exist capable of altering memory transaction behavior in exploitable ways. For instance, such configurations might permit transaction reordering or suppress the proper propagation of errors. To mitigate such attacks, the architecture must strictly prohibit unprivileged components from controlling data flow configurations for higher-privileged components.

8.2 SMN Exploration

The findings presented in STALEUS underscore the necessity for a more structured approach to discovering vulnerable SMN configurations. We scan the entire SMN range for all SEV-SNP capable CPUs, namely Zen 3 / 4 / 5, to examine the range of non-zero and non-`0xFF` values. We excluded these specific values as they typically indicate inactive or inaccessible registers. Table 3 shows the statistics for the respective CPU generations. Throughout the scanning process, we excluded crashing ranges, as they would significantly slow down the search. A single 20-bit sub-configuration range contains 2^{18} register fields. Given that a workstation reboot requires approximately five minutes, fully scanning just one IP configuration (1 out of 4096), assuming every access triggers a crash, would span 2.5 years. Testing individual bits scales this time by a factor of 32, resulting in an infeasible 80-year duration. Since we observed crashes on both read and write operations, acquiring SMN data for critical regions proves exceptionally challenging, rendering subsequent crash analysis even more demanding.

Table 4: SMN registers altering PSP memory transactions.

Register	SMN Address
NB_TOP_OF_DRAM3	0x13B10138
NB_TOP_OF_DRAM_SLOT1	0x13B00090
NB_UPPER_TOP_OF_DRAM2	0x13B10068
NB_LOWER_TOP_OF_DRAM2	0x13B10064

While MSRs are usually scoped per core, a MSR change-induced crash most likely only takes down one core and leaves the other ones functional, allowing for debugging. Conversely, crashes induced through SMN access usually occur at an interconnect level. This means the entire platform resets or freezes, and the exact error code is hidden in undocumented hardware registers that probably also reside within the SMN. While proprietary mechanisms like AMD HDT exist to extract this diagnostic data, they remain inaccessible to independent researchers [66]. We conclude that previous techniques to enumerate an undocumented configuration space are challenging to apply for the SMN [22, 33]. This limitation highlights the need to find new techniques to analyze platform and interconnect configuration through the SMN.

Other SMN Registers. Despite the challenges of SMN analysis, we encountered other SMN registers that alter the memory transaction behavior of the PSP. Table 4 lists the registers we found. We note that it may be possible to use these to achieve XCA-style attacks. These registers define the DRAM / MMIO split of all traffic passing through the IOHUBs. We discover that an untrusted hypervisor can reconfigure those registers with arbitrary values. When the PSP accesses DRAM, we use the registers to override the destination and cause them to be routed to MMIO devices. Doing so results in a PSP timeout. We believe that an attacker with better platform knowledge may be able to determine the reason for the timeouts and is capable of redirecting or dropping PSP memory requests to conduct a Fabricked-like attack [56].

8.3 Mitigation

To mitigate STALEUS, AMD must deploy firmware upgrades that guarantee PSP memory requests remain memory coherent. This may be achieved by preventing the hypervisor from changing the SMN configuration range for the `A2S_CNTL` registers. Although AMD platform internals remain undocumented, we suspect a dynamic mechanism exists to reconfigure SMN security controls. Specifically, this mechanism would allow for the dynamic assignment of security levels across different SMN ranges. We hypothesize that hypervisor SMN accesses via PCIe configuration space route to an internal IP, potentially the System Management Unit (SMU). The SMU performs the necessary access validation before forwarding the transaction onto the SMN. We ground this hypothesis on the presence of an SMU sub-entry within the

AMD firmware blob [50] and existing public documentation of the Zen platform [14]. As the exact mechanisms of SMN routing and access checks remain proprietary, we cannot conclude how such a mitigation will ultimately be implemented. Alternatively, AMD might leverage other undocumented control registers to convert non-coherent memory transactions into coherent ones. References to Vega GPU registers in the Linux kernel (e.g., `NoSnoopDis`) suggest that such functionality may already exist in the hardware [58].

To mitigate the guest write drop attack detailed in Section 5.4, the guest could modify its page tables to enforce non-cacheable memory. This configuration forces all guest memory accesses directly to DRAM, bypassing the cache hierarchy entirely. However, this approach only mitigates one specific exploitation vector of STALEUS without addressing the underlying root cause. Further, operating without caches introduces a severe performance overhead, significantly degrading CVM execution speed.

Alternatively, a microcode update could enforce that on RMPUPDATE, all cache lines get written back and invalidated. This would directly prevent all of our case studies, as the hypervisor must use RMPUPDATE to hand over the page to the PSP. While preventing our case studies, the microcode patch must also guarantee that no new clean cache lines can be allocated while the PSP operates on a page. Failure to enforce this restriction would allow the guest to consume stale x86 cache data rather than retrieving the most recent updates.

8.4 Applicability to Zen 3

Our SMN scanning indicates that Zen 3 uses a different Infinity Fabric than Zen 4 / 5. Attempting to modify the equivalent configuration register found in Zen 4 and Zen 5 produces no observable effect on Zen 3. This initially indicates that the register either resides at a different SMN location or remains inaccessible to the hypervisor. We search the entire SMN on Zen 3 for the configuration value `0x2a80540`, the same we found in Zen 4 / 5. This value represents the default `A2S_CNTL` configuration for Vega GPUs. We successfully located this value within the Zen 3 SMN address space. However, toggling the least significant bits failed to reproduce the effects observed on Zen 4 and Zen 5. Given the closed-source nature of the hardware, we cannot draw definitive conclusions; however, we hypothesize that a secondary IP block forcibly converts non-coherent transactions into coherent ones. We base the hypothesis on the `NoSnoopDis` string we found in the Linux kernel for a North Bridge I/O (NBIO) configuration of an AMD GPU [58]. Since openSIL misses documentation for the corresponding base register range, we cannot precisely isolate the target SMN region.

9 Related Work

We discuss closely related work to STALEUS.

SEV Attacks. Previous work studied SEV and SEV-ES security, which were susceptible to many design flaws that led to many different architectural attack vectors [28, 29, 36, 38, 44, 45, 45, 51, 59, 60].

SEV-SNP Attacks. Google performed a preliminary assessment of the PSP and SEV-SNP ecosystem [27]. Cipherleaks is the first work to use ciphertext side channels to violate SEV-SNP confidentiality guarantees [35, 37, 55, 62, 64, 65]. Heckler and WeSee abuse an open interrupt injection interface to break SEV-SNP [53, 54]. Cachewarp and Stackwarp exploit unlocked MSRs to revert caches and manipulate the stack engine, to break AMD security guarantees [67, 68]. With Cachewarp, at no point in time do CPUs or the PSP have a different memory view when they read the same address. With STALEUS, the CPU cores and the PSP have two different memory views. BadAML exploits the missing attestation of the CVM UEFI firmware to inject malicious code [57]. Lack of physical protection in SEV-SNP has been used for attacks [18, 20, 21]. CounterSEVeillance exploits the availability of guest performance counter data to exfiltrate confidential data [23]. RMPocalypse abuses a flaw during SEV-SNP initialization [52].

Interconnect Corruption Attacks. STALEUS also breaks SEV-SNP but is part of a larger family of attacks. We call the class Interconnect Corruption Attacks (XCA). Fabricked and BreakFAST are two prior instances of XCA, where we compromise SEV-SNP by manipulating the interconnect [24, 56]. STALEUS is the third instance of XCA and fully compromises SEV-SNP guarantees as well. All three attacks are instances of the XCA family with three different root causes. Fabricked targets the Data Fabric, BreakFAST the IOHUB, and STALEUS the SYSHUB i.e., three different components of the Infinity Fabric. In terms of attacker capabilities, Fabricked drops PSP writes by misrouting DRAM transactions. BreakFAST redirects PSP read/writes into the Control Fabric. STALEUS does neither; it alters the coherence attributes of PSP transactions, inducing a split memory view between the PSP and x86 cores without redirecting any transactions.

Platform Exploration. Domas measures access times of the `RDMSR` instruction to group MSRs into categories and identify potentially interesting ones [22]. Similarly, MSRevelio systematically analyzes the MSR space for configurations that violate platform security guarantees [33]. StackWarp employs akin techniques to scan for different instruction behavior when flipping MSR bits [68]. STALEUS performs an initial exploration of the SMN to find an unlocked configuration to alter PSP memory coherence.

Cache Coherence. The fundamental principles of modern memory coherence were established by Li and Hudak [34]. More recently, Nagarajan et al. [48] provide a comprehensive overview of the theory behind memory and cache consistency in contemporary systems. However, verifying these protocols in multiprocessor environments remains difficult,

as detailed by Cantin et al. [15, 16]. In terms of specific implementations, Lis et al. [40] propose a novel coherency model, while Molka et al. examine performance implications in the Intel Nehalem [42] and Haswell [43] architectures. These mechanisms are also targets for exploitation; Yao et al. [63] demonstrate how coherence states can be manipulated to form covert channels. Further, vulnerabilities such as Spectre [32] and Meltdown [39] leverage cache coherency timing differences to exfiltrate data. STALEUS exploits the absence of memory coherence to manipulate DRAM state.

10 Conclusion

We present STALEUS, an attack breaking SEV-SNP integrity and confidentiality. The root cause is that the untrusted hypervisor can alter the PSP's coherency via the SMN. STALEUS motivates further analysis of the SMN in the context of confidential computing.

11 Acknowledgements

We thank the reviewers, Christoph Wech, Philipp Giersfeld, and Andrin Bertschi for their valuable feedback. Benedict Schlüter is supported by the Google PhD Fellowship in Privacy, Safety, and Security.

Ethical Considerations

We conducted a stakeholder-based ethics analysis to evaluate the implications of our research on the AMD SEV-SNP architecture. Our analysis is guided by the ethical principles outlined in the Menlo Report, specifically *Beneficence*, *Respect for Persons*, and *Respect for Law and Public Interest*.

Stakeholders

We identified the following stakeholders impacted by our research procedures and subsequent publication:

- **Cloud Tenants:** The primary victims whose confidential data and privacy are at risk due to the vulnerability.
- **Cloud Service Providers (CSPs):** Entities managing the infrastructure, who rely on hardware guarantees to serve tenants.
- **Hardware Vendor (AMD):** The entity responsible for the integrity of the SEV-SNP technology.
- **The Security Community & Society:** The broader group benefiting from secure computing infrastructure.

Impacts and Ethical Principles

We analyzed the impacts through the lens of *Beneficence* (balancing benefits against risks) and *Respect for Law and Public Interest*.

Impact of Research Procedures: During the vulnerability discovery phase, there was a potential risk of violating the *Respect for Persons* principle if user data were accessed without consent. To prevent this violation, we conducted all experiments on equipment physically owned and controlled by the research team. No third-party cloud infrastructure or production workloads were involved. Consequently, no external stakeholders experienced tangible harms or rights violations during the experimental phase.

Impact of Publication: The publication of this work carries dual impacts:

1. **Negative Impact (Harm):** Public disclosure could enable malicious actors to exploit unpatched systems, causing financial loss and data exposure to Cloud Tenants and CSPs.
2. **Positive Impact (Benefit):** Publication forces the remediation of a critical hardware flaw, adhering to the principle of *Beneficence* by improving the long-term security posture of the ecosystem.

Mitigations

To minimize the harms identified above, we implemented the following mitigations:

- **Coordinated Vulnerability Disclosure (CVD):** We adhered to the *Respect for Law and Public Interest* by reporting our findings to AMD in September 2025. We operated under a strict embargo, ensuring the vendor had time to develop and deploy firmware patches before the vulnerability was made public.
- **Artifact Limitation:** To further mitigate the risk of misuse, we restricted the release of technical artifacts and will do so only after conditional acceptance. This is to mitigate the circle of exposure.

Decision

Our decision to proceed with the research and its publication is based on weighing the ethical harms against the ethical benefits.

The vulnerability in the SEV-SNP architecture existed regardless of our research; leaving it undiscovered would result in a false sense of security for Cloud Tenants while leaving them vulnerable to covert adversaries. By following a rigorous CVD process and limiting the lethality of our released artifacts, we determined that the benefits of public knowledge and the subsequent hardware hardening significantly outweigh the residual risks of disclosure. Therefore, we concluded that publishing this work is the ethical path to ensure the security and integrity of future confidential computing platforms.

Open Science

Our artifacts are available at https://zenodo.org/records/20266830. They contain the necessary code to reproduce the case studies on SEV-SNP. We further included the SMN scan for Zen 3, Zen 4, and Zen 5, which contains the data to build Table 3.

References

[1] Advanced Micro Devices, Inc. AMD SEV-SNP: Strengthening VM Isolation with Integrity protection and more. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020.

[2] Amazon. AWS Nitro Enclaves - Create additional isolation to further protect highly sensitive data within EC2 instances. https://aws.amazon.com/ec2/nitro/nitro-enclaves/.

[3] AMD. Probe Filter Directory Management (US12141066B2). https://patents.google.com/patent/US12141066B2/en, 2017.

[4] AMD. Region Probe Filter for Distributed Memory System (US20170177484A1). https://patents.google.com/patent/US20170177484A1/en, 2017.

[5] AMD. AMD-ASPFW. https://github.com/amd/AMD-ASPFW, 2023.

[6] AMD. Processor Programming Reference (PPR) for AMD Family 1Ah Model 02h, Revision C1 Processors (57238 Rev 0.24), 2024.

[7] AMD. SEV Secure Nested Paging Firmware ABI Specification (56860, Rev 1.58), 2025.

[8] AMD. AMD "Zen" Core Architecture. https://www.amd.com/en/technologies/zen-core.html, 2026.

[9] AMD. openSIL: Open-Source Silicon Initialization Library. https://github.com/openSIL/openSIL, 2026. Accessed: 2026-01-28.

[10] Yazen Ghannam (AMD). AMD NB and SMN rework. https://lore.kernel.org/all/20241206161210.163701-14-yazen.ghannam@amd.com/, 2024.

[11] ARM. Arm Confidential Compute Architecture (ARM-CCA). https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture.

[12] Arm. AMBA® AXI Protocol Specification (ARM IHI 0022), 2023.

[13] Alexis Bagia, Vincent Quentin Ulitzsch, Daniël Trujillo, Mengyuan Li, Mengjia Yan, and Jean-Pierre Seifert. A Close Look at RMP Entry Caching and Its Security Implications in SEV-SNP. In *ACM HASP*, 2025.

[14] Thomas Burd, Noah Beck, Sean White, Milam Paraschou, Nathan Kalyanasundharam, Gregg Donley, Alan Smith, Larry Hewitt, and Samuel Naffziger. "Zeppelin": An SoC for Multichip Architectures. *IEEE Journal of Solid-State Circuits*, 2019.

[15] Jason F Cantin, Mikko H Lipasti, and James E Smith. The complexity of verifying memory coherence. In *Proceedings of the fifteenth annual ACM symposium on Parallel Algorithms and Architectures*, pages 254–255, 2003.

[16] Jason F Cantin, Mikko H Lipasti, and James E Smith. The complexity of verifying memory coherence and consistency. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):663–671, 2005.

[17] Li-Chung Chiang and Shih-Wei Li. Reload+Reload: Exploiting Cache and Memory Contention Side Channel on AMD SEV. In *ACM ASPLOS*, 2025.

[18] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. Tee.fail: Breaking trusted execution environments via ddr5 memory bus interposition. In *47th IEEE Symposium on Security and Privacy (IEEE S&P '26)*. IEEE Computer Society, 2026.

[19] Oxide Computer Company. illumos. https://github.com/oxidecomputer/illumos-gate/blob/stlouis/usr/src/uts/intel/sys/amdzen/ccx.h, 2025.

[20] Jesse De Meulemeester, David Oswald, Ingrid Verbauwhede, and Jo Van Bulck. Battering RAM: Low-cost interposer attacks on confidential computing via dynamic memory aliasing. In *47th IEEE Symposium on Security and Privacy (S&P)*, May 2026.

[21] 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 *IEEE S&P*, 2025.

[22] Christopher Domas. GOD MODE UNLOCKED - Hardware Backdoors in x86 CPUs. https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs.pdf, 2018.

[23] Stefan Gast, Hannes Weissteiner, Robin Leander Schröder, and Daniel Gruss. Counterseveillance: Performance-counter attacks on amd sev-snp. In *Network and Distributed System Security Symposium 2025: NDSS 2025*, 2025.

[24] Philipp Giersfeld, Benedict Schlüter, and Shweta Shinde. BreakFAST: Confused Deputy Attack on Infinity Fabric to Break AMD SEV-SNP. In *47th IEEE Symposium on Security and Privacy (S&P 26)*, San Francisco, CA, May 2026. IEEE.

[25] Lukas Giner, Sudheendra Raghav Neela, and Daniel Gruss. Cohere+Reload: Re-enabling High-Resolution Cache Attacks on AMD SEV-SNP. In *DIMVA*, 2025.

[26] Google. Confidential Computing | Google Cloud. https://cloud.google.com/confidential-computing.

[27] Google. AMD Secure Processor for Confidential Computing. https://storage.googleapis.com/gweb-uniblog-publish-prod/documents/AMD_GPZ-Technical_Report_FINAL_05_2022.pdf, 2022.

[28] Felicitas Hetzelt and Robert Buhren. Security Analysis of Encrypted Virtual Machines. In *ACM SIGPLAN/SIGOPS*, 2017.

[29] Felicitas Hetzelt, Martin Radev, Robert Buhren, Mathias Morbitzer, and Jean-Pierre Seifert. VIA: Analyzing Device Interfaces of Protected Virtual Machines. In *ACM ACSAC*, 2021.

[30] Intel. Intel Trust Domain Extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.

[31] Josh Eads and Kristoffer Janke and Eduardo Vela Nava and Tavis Ormandy and Matteo Rizzo. EntrySign: Exploiting AMD Zen CPU Microcode Signature Verification. Blog post and technical disclosure, 2025.

[32] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[33] Andreas Kogler, Daniel Weber, Martin Haubenwallner, Moritz Lipp, Daniel Gruss, and Michael Schwarz. Finding and exploiting cpu features using msr templating. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[34] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[35] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *IEEE S&P*, 2022.

[36] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. CrossLine: Breaking "Security-by-Crash" Based Memory Isolation in AMD SEV. In *ACM CCS*, 2021.

[37] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *USENIX Security*, 2021.

[38] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. TLB Poisoning Attacks on AMD Secure Encrypted Virtualization. In *ACM ASAC*, 2021.

[39] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[40] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, and Srinivas Devadas. Memory coherence in the age of multicores. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 1–8. IEEE, 2011.

[41] Microsoft. Azure confidential cloud - protect data in use | microsoft azure. https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[42] Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S Muller. Memory performance and cache coherency effects on an intel nehalem multiprocessor system. In *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 261–270. IEEE, 2009.

[43] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Wolfgang E Nagel. Cache coherence protocol and memory performance of the intel haswell-ep architecture. In *2015 44th International Conference on Parallel Processing*, pages 739–748. IEEE, 2015.

[44] Mathias Morbitzer, Manuel Huber, and Julian Horsch. Extracting Secrets from Encrypted Virtual Machines. In *ACM CODASPY*, 2019.

[45] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. SEVered: Subverting AMD's Virtual Machine Encryption. In *EuroSec*, 2018.

[46] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. SEVerity: Code Injection Attacks against Encrypted Virtual Machines. In *IEEE S&PW*, 2021.

[47] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families: Industrial Product. In *ACM/IEEE ISCA*, 2021.

[48] Vijay Nagarajan, Daniel J Sorin, Mark D Hill, and David A Wood. *A primer on memory consistency and cache coherence*. Springer Nature, 2020.

[49] PCI-SIG. PCI Express® Base Specification Revision 5.0 Version 1.0), 2019.

[50] PSPReverse. PSPTool: Display, extract, and manipulate PSP firmware inside UEFI images. https://github.com/PSPReverse/PSPTool, 2026. Accessed: 2026-01-27.

[51] Martin Radev and Mathias Morbitzer. Exploiting Interfaces of Secure Encrypted Virtual Machines. ACM ROOTS, 2021.

[52] Benedict Schlüter and Shweta Shinde. RMPocalypse: How a Catch-22 Breaks AMD SEV-SNP. In *ACM CCS*, 2025.

[53] Benedict Schlüter, Supraja Sridhara, Andrin Bertschi, and Shweta Shinde. WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP. In *IEEE S&P*, 2024.

[54] Benedict Schlüter, Supraja Sridhara, Mark Kuhne, Andrin Bertschi, and Shweta Shinde. Heckler: Breaking Confidential VMs with Malicious Interrupts. In *USENIX Security*, 2024.

[55] Benedict Schlüter, Christoph Wech, and Shweta Shinde. Heracles: Chosen Plaintext Attack on AMD SEV-SNP. In *ACM CCS*, 2025.

[56] Benedict Schlüter, Christoph Wech, and Shweta Shinde. Fabricked: Misconfiguring Infinity Fabric to Break AMD SEV-SNP. In *35th USENIX Security Symposium (USENIX Security 26)*, Baltimore, MD, August 2026. USENIX Association.

[57] Satoru Takekoshi, Manami Mori, Takaaki Fukai, and Takahiro Shinagawa. BadAML: Exploiting Legacy Firmware Interfaces to Compromise Confidential Virtual Machines. In *ACM CCS*, 2025.

[58] Linus Torvalds et al. Linux kernel source tree. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git, 2026.

[59] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In *DIMVA*, 2023.

[60] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. SEVurity: No Security Without Integrity: Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In *IEEE S&P*, 2020.

[61] Luca Wilke, Jan Wichelmann, Anja Rabich, and Thomas Eisenbarth. SEV-Step A Single-Stepping Framework for AMD-SEV. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023.

[62] Yuqin Yan, Wei Huang, Ilya Grishchenko, Gururaj Saileshwar, Aastha Mehta, and David Lie. RelocateVote: Using Sparsity Information to Exploit Ciphertext Side-Channels. In *USENIX Security*, 2025.

[63] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 168–179. IEEE, 2018.

[64] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. HyperTheft: Thieving Model Weights from TEE-Shielded Neural Networks via Ciphertext Side Channels. In *ACM CCS*, 2024.

[65] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. CipherSteal: Stealing Input Data from TEE-Shielded Neural Networks with Ciphertext Side Channels . In *IEEE SP*, 2025.

[66] Bulat N. Zagartdinov. Undocumented Debug Interface HDT of Modern AMD CPUs. In *2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon)*, 2024.

[67] Ruiyi Zhang, Lukas Gerlach, Daniel Weber, Lorenz Hetterich, Youheng Lü, Andreas Kogler, and Michael Schwarz. CacheWarp: Software-based Fault Injection using Selective State Reset. In *USENIX Security*, 2024.

[68] Ruiyi Zhang, Tristan Hornetz, Daniel Weber, Fabian Thomas, and Michael Schwarz. StackWarp: Breaking AMD SEV-SNP Integrity via Deterministic Stack-Pointer Manipulation through the CPU's Stack Engine. In *USENIX Security*, 2026.