BREAKFAST: Confused Deputy Attack on Infinity Fabric to Break AMD SEV-SNP

Philipp Giersfeld* Benedict Schlüter* Shweta Shinde
ETH Zurich

***Abstract*—SEV-SNP is AMD's offering of confidential computing in the cloud. It enables the creation of so-called confidential virtual machines. To achieve its security, SEV-SNP relies on a trusted co-processor called the Platform Security Processor (PSP). In this paper, we present the following novel observations: The hypervisor can change the MMIO address mapping of internal devices (e.g., SMU that manages power). Worse, these changes affect how PSP accesses to system DRAM are routed. With this ability, we showcase that we can mount a confused deputy attack on the PSP. Specifically, we can trick the PSP into writing attacker-controlled values to the MMIO range of internal devices. As there is little documentation on internal devices, it is challenging to identify: which device to write to, at what location, and with what value. To this end, we reverse engineer an undocumented device called FASTREG. Its role is to map the internal control network 4 GB address space as an addressable range (e.g., to the internal device, hypervisor, PSP) to facilitate read/write. Putting these two observations together, we use the PSP to update the internal control network to disable the IOMMU's SEV-SNP protections, to subsequently fake attestation and enable debug on production CVMs.**

1. Introduction

AMD Secure Encrypted Virtualization Secure Nested Paging (SEV-SNP) supports confidential computing on servers, where an untrusted hypervisor can launch Confidential Virtual Machines (CVMs) [1]. To enable SEV-SNP, AMD uses a trusted processor called PSP [2]. While the x86 cores and the PSP are well-known, AMD also places other devices (e.g., USB controller, integrated GPU) on the System-on-Chip (SoC). These elements communicate with each other over AMD's proprietary fast on-chip interconnect called Infinity Fabric [3].

By analyzing the open-source code for the platform initialization [4] and experimental validation, we identify that AMD exposes a set of On-SoC elements (e.g., SMU) via MMIO on the Infinity Fabric. Since the x86 cores that are under the untrusted hypervisor's control are also on this interconnect, we start with a preliminary investigation to answer: *can the hypervisor directly access the MMIO regions of On-SoC elements?* Our first set of experiments concludes that the untrusted hypervisor can read the MMIO regions of On-SoC elements by mapping them into the address space accessible by the x86 cores, even when SEV-SNP is enabled. However, certain write requests from the hypervisor are refused, potentially due to insufficient privilege on the interconnect.

Next, our goal is to attain higher privilege to circumvent this check. Our insight is to try to trick the PSP into writing to the MMIO regions of the On-SoC elements, with the hope that it will succeed because it has the highest privileges on the platform. To achieve this, we identify that the hypervisor can invoke PSP APIs (e.g., `SNP_PAGE_MOVE`, `SNP_DBG_ENCRYPT`). The hypervisor can control the source and destination addresses used by these APIs, such that they map to DRAM addresses of a CVM under its control. Thus, the hypervisor first overlaps the MMIO range of an On-SoC element with the DRAM address of a CVM it controls, and then instructs the PSP to perform API actions on these DRAM addresses. When we perform this experiment, the hypervisor succeeds in writing values of its choice to the MMIO regions of On-SoC elements. This experiment tells us two important points: the PSP has sufficient privileges to do the writes; and even though the PSP thinks it is operating on DRAM, the routing takes the MMIO mapping into account, directing the writes to the On-SoC element instead. This way, the hypervisor uses the PSP as a confused deputy to do its bidding.

The last challenge is to use the hypervisor's ability to write to the On-SoC elements in a way that undermines SEV-SNP enforcement. As there is almost no documentation of these On-SoC elements, we investigate: *which On-SoC elements to target, what value to write, and to which MMIO address?* In the process of answering these questions, we identify an On-SoC element called `FASTREG`. After reverse engineering its role, we uncover that it can change routing rules of the Infinity Fabric, potentially for performance as the name suggests. In particular, we observe that the writes to `FASTREG`'s MMIO range are directly reflected as changes to the control plane of the Infinity Fabric. This allows us to hijack control of the interconnect.

To demonstrate the utility of this new adversarial capability, we use malicious PSP writes to misconfigure `FASTREG` such that it points to IOMMU configuration registers. We then trigger another malicious PSP write, this time to the IOMMU's configuration space, to disable the IOMMU's SEV-SNP protection. This, in turn, allows hypervisor-controlled devices to directly read and write to CVMs in the absence of IOMMU-side SEV-SNP enforcement. Concretely, we demonstrate two end-to-end case studies on Zen 4 and Zen 5 that forge attestation and enable debug mode on victim CVMs. Our exploits do not require

* denotes equal contribution

any prior knowledge about the victim CVM or single-stepping, achieve 100% accuracy, take 1–25 ms, and are not easily detectable.

Based on our experimental findings and observations, we identify three underlying root causes that together enable BREAKFAST: (i) when handling hypervisor requests pertaining to CVMs, the PSP issues requests with high privileges even if the operation does not warrant it; (ii) the hypervisor can change the base addresses of On-SoC elements and map their MMIO address space such that it overlaps with DRAM; and (iii) when the PSP makes requests to addresses that it thinks are to DRAM, but in reality are maliciously mapped to the MMIO address space of an On-SoC element, the interconnect routes them to the MMIO address space. In summary, when enforcing the SEV-SNP threat model, multiple systemic weaknesses in the AMD interconnect lead to BREAKFAST. Fabricked is the first attack on the SoC interconnect that breaks AMD SEV-SNP [5]. This points to a systemic attack class that we call XCA, in which the untrusted hypervisor can modify the interconnect (AMD Infinity Fabric in this case) to break confidential computing guarantees. BREAKFAST is a second instance of XCA where we corrupt PSP interactions with DRAM.

To remediate the three underlying root causes of BREAKFAST, we propose pointwise mitigations that can act as a stop-gap for BREAKFAST. More importantly, we outline principled defenses not just to eliminate BREAKFAST but also to strengthen the AMD SEV-SNP architecture against other attacks that may stem from these systemic weaknesses.

Contributions. We present an attack on AMD SEV-SNP called BREAKFAST, that uses the PSP as a confused deputy to issue MMIO requests to On-SoC elements thereby obtaining complete access to the control plane of the interconnect. We show that both AMD Zen 4 and Zen 5 processors are susceptible to BREAKFAST. In the process of developing BREAKFAST, we uncover several new findings about the inner workings of AMD's Infinity Fabric and the PSP. BREAKFAST is public at https://xca-attacks.github.io/breakfast.

Responsible Disclosure. We responsibly reported our findings to AMD on 26 September 2025. AMD has acknowledged our findings and issued CVE-2025-61971 and CVE-2025-61972 for the vulnerability.

2. SEV-SNP

In this section, we provide background on the design of AMD SEV-SNP with a focus on the role of the Platform Security Processor (PSP).

2.1. Confidential Virtual Machines (CVMs)

Secure Encrypted Virtualization (SEV) is AMD's initial design for confidential computing, enabling users to run CVMs [6]. CVMs aim to protect against a malicious or compromised hypervisor while building on a familiar abstraction of virtual machines. SEV uses encryption to protect CVM memory. The memory controller transparently encrypts and decrypts memory requests based on the system physical address, so the same plaintext at the same physical address produces the same ciphertext. The PSP, AMD's hardware root-of-trust, programs the memory controller with per-VM encryption keys. The next iteration, SEV-Encrypted State (SEV-ES), builds on SEV and adds protection for the register state during context switching out of the CVM [7]. Traditionally, the hypervisor controls the register state. Since CVMs distrust the hypervisor, SEV-ES requires the register state to reside in encrypted guest memory.

Despite encryption, a CVM may still be vulnerable to ciphertext attacks such as replay attacks. To prevent this, SEV-Secure Nested Paging (SEV-SNP), AMD's latest design, protects the integrity of CVM memory. It enforces access control based on a Reverse Map Table (RMP), which also covers metadata such as the register state during context switching [1]. The RMP governs access by maintaining a reverse mapping entry for each system physical page. Each entry includes the Guest Physical Address (GPA) and other attributes, such as page ownership. By enforcing a unique owner and GPA for each Host Physical Address (HPA), the RMP prevents hypervisor attacks that encryption alone cannot mitigate, such as replay attacks. Following the page-table walk that converts a virtual address to a physical address, the hardware consults the RMP. It verifies that the currently executing entity is the owner and that the guest physical address used in the translation matches the address stored in the RMP. If not, the hardware raises an RMP fault and denies access to that memory. The PSP also programs the Input–Output Memory Management Unit (IOMMU) with the location of the RMP. The IOMMU uses the RMP to block all device accesses that do not target hypervisor-owned memory.

SEV-SNP also supports remote attestation of CVMs. When the hypervisor creates a CVM, the PSP computes a digest over the initial CVM memory layout and content. Later, once the guest requests attestation, the PSP uses a chip-unique key signed by AMD to sign an attestation report that includes the attestation measurement. The CVM owner can then use the attestation to remotely verify the trustworthiness of the CVM and the underlying platform. They can choose various configuration options, such as whether debug mode is enabled and whether the CVM runs with multi-threading. The PSP encodes this configuration in a bitmap, called the guest policy, which is fixed at launch and reflected in attestation. It saves both the attestation measurement and the guest policy, along with other guest metadata, in the guest context page. During runtime, the RMP protects this page from modification, which is also encrypted under a key exclusive to the PSP and unique to each boot.

2.2. Platform Security Processor

The PSP is a co-processor on the AMD System-on-Chip (SoC). During platform initialization, it configures essential

aspects such as the RMP and IOMMU enforcement of SEV-SNP checks. As the highest-privileged entity, it is also the only component that can perform certain security-critical operations, e.g., only the PSP can move CVM memory pages between different physical locations or perform certain updates to the RMP. As a concrete example, only the PSP can transition an RMP entry between specific page states, e.g., to allow the hypervisor to reclaim memory after the PSP has validated that it is no longer security-critical. Because SEV-SNP prevents the hypervisor from directly executing these necessary functional operations, the PSP exposes various APIs to safely act on behalf of the hypervisor.

API for the Hypervisor. The PSP exposes the following API calls among others:

`SNP_DBG_ENCRYPT`/`SNP_DBG_DECRYPT`. SEV-SNP provides a debug API that allows the hypervisor to encrypt and decrypt CVM memory. This API is only available if the hypervisor starts the guest with debugging enabled in its guest policy. The hypervisor supplies the PSP with a buffer in hypervisor memory and a page address of CVM memory. Using the page address, the PSP either writes decrypted CVM memory into this buffer or encrypts the buffer contents and writes them into CVM memory.

`SNP_GUEST_STATUS`. The hypervisor can use the `SNP_GUEST_STATUS` command to obtain information about a specific guest, e.g., the guest policy or the CVM identifier.

3. AMD Platform

Next, we give background on the AMD platform, including the SoC and on-chip interconnect. Figure 1 provides a simplified overview of Zen-based x86 CPUs [3], [8]. AMD groups cores into Core Complexes (CCXs), which, depending on the platform, contain either four or eight x86 cores.

3.1. Infinity Fabric

All components on the AMD SoC are connected with AMD's proprietary interconnect, Infinity Fabric. Similar concepts for fast interconnects also exist on other platforms, such as NVIDIA NVLink or Intel Ultra Path Interconnect. AMD Infinity Fabric consists of two planes: a data plane called the Scalable Data Fabric (SDF); and a control plane called the Scalable Control Fabric (SCF), which is also known as the SMN.

Data Plane: SDF. The SDF, which serves as the data interconnect on the AMD Zen architecture, connects components over a high-bandwidth link. This includes all *major* components such as x86 cores, memory controllers, and I/O controllers. The SDF is coherent and enables fast data movement across the SoC.

Control Plane: SCF/SMN. The SMN is AMD's control interconnect on Zen platforms and connects all components on the SoC [3], similar to Intel sideband fabric [9]. Compared

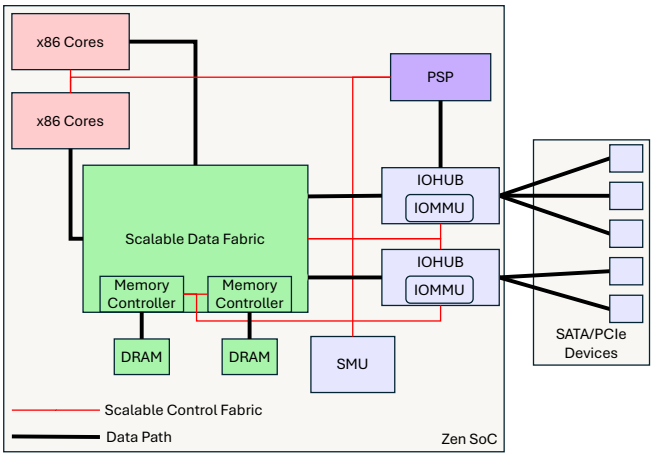

Figure 1. Overview of essential components of the AMD SoC. The data path provides a high-speed interconnect to move data quickly around the SoC. The SMN is the SoC internal control network through which each component can be configured using fixed 32-bit addresses. The IOHUB connects devices.

to the SDF, the SMN serves primarily as a register address space that comprises all hardware units on the SoC, i.e., it contains control registers for all components. The SMN is a 32-bit address space, where everything addressable in it has a fixed address. Platform components use it for most configuration operations, such as the IOMMU, memory controllers, or the SDF. Software running on the x86 cores primarily accesses the SMN for telemetry and configuration of devices that cannot be accessed via standard MMIO. For example, it uses the SMN to obtain temperature measurements, interact with the power management controller, or obtain information on hardware errors. Kernel code accesses registers in the SMN indirectly, using dedicated index and data registers in the PCI configuration space. Listing 1 shows a simplified version of this.

```
static int __amd_smn_rw(u16 node, u32 address,
      u32 *value, bool write)
{
 struct pci_dev *root;
 root = node_to_amd_nb(node)->root;
 pci_write_config_dword(root, 0x60, address);
 if(write) {
  pci_write_config_dword(root, 0x64, *value);
 } else {
  pci_read_config_dword(root, 0x64, value);
 }
}
```

Listing 1. Indirect Register Access to SMN.

3.2. I/O Subsystem and Non-x86 Devices

The SoC houses other devices aside from the x86 cores. Naturally, it also provides the option to attach devices to the platform, particularly to the SDF. These devices attach to the SDF via bridges, called the IOHUBs. Devices can be external devices (e.g., discrete GPUs or FPGAs) or integrated devices (e.g., USB or SATA controllers). An IOHUB

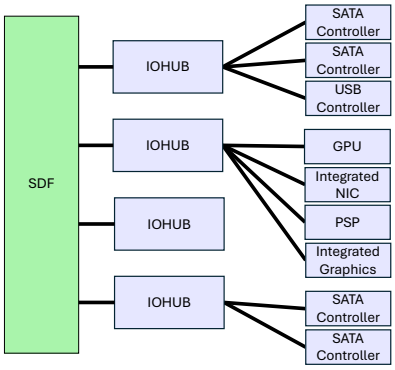

Figure 2. Example IOHUB configurations. A single IOHUB can connect both integrated and external devices, e.g., PCIe devices.

groups multiple devices as shown in Figure 2. Then, each IOHUB connects to the SDF, which in turn connects to other IOHUBs, CCXs, and other components such as the memory controller. Thus, the IOHUB serves two purposes: (i) as the router between different devices on the same IOHUB; (ii) as the bidirectional bridge between devices and the SDF. Figure 2 shows an example system with the devices connected to each of the four IOHUBs.

3.3. Routing MMIO Accesses from x86 Cores

So far, we have described the main components of the AMD SoC and how they are connected. We now explain how the SoC routes load and store requests through this interconnect. In particular, we elaborate on how the SDF interprets physical addresses as either DRAM or MMIO, where MMIO addresses map to devices behind a specific IOHUB.

Load/store requests from the x86 cores are routed either to DRAM or treated as MMIO accesses to a device. The physical address range for each device MMIO is configured during device attachment. Specifically, when a device attaches to the platform, it shares the size of physical address space it requires. Then, the device receives a continuous carve-out of the physical address space, known as the MMIO range. Subsequently, when an x86 core accesses a physical address in that range, the bus routes the access to the corresponding device.

Recall that on AMD, all device accesses pass through the IOHUBs. Therefore, the MMIO range for a device connected to a specific IOHUB is a subrange of that IOHUB's physical address range. In other words, IOHUB's physical address range is at least the union of the ranges of all the devices it groups. Bootcode programs the physical address range of the IOHUBs by writing to SMN registers that configure the SDF. For example, the first IOHUB receives a physical address space carve-out at `0xF0000000` – `0xF7FFFFFF`. A USB controller located behind this first IOHUB uses a subrange of that region, exposing its MMIO range at `0xF0100000` – `0xF01FFFFF`.

3.4. PSP on the SoC

The PSP is an ARMv7 Cortex-A5 core that also attaches to the SDF via an IOHUB. As mentioned in Section 2, the PSP is essential for the security of the whole platform. It has its own SRAM, but can also map DRAM and the SMN into its address space. The PSP executes the SEV firmware, which is part of the trusted computing base responsible for enforcing SEV security guarantees. AMD has open-sourced the firmware for Genoa Zen 4 processors [2].

PSP Privileges. AMD SoC implements multiple privilege levels ranging 0–7, with 0 being the highest privilege. For example, SEV firmware and AMD documentation describe security levels and `UnitIDs` on the AMD SoC [2], [10]. In Listing 2, a header file from the PSP source code shows that the PSP uses security level 0, microcode uses level 2, and all other accesses default to level 7. The assumption is further supported by public documents indicating that normal SMN accesses from x86 cores have security level 7 [10]. Thus, the PSP has the highest privilege on the platform.

Platform Initialization. During platform boot, the PSP executes as the first component and subsequently starts the x86 cores. The x86 cores run boot code. This boot code combines components supplied by AMD and by the motherboard vendor. AMD distributes a closed-source binary, AGESA, which implements various low-level initialization routines, but is in the process of replacing it with openSIL which is open-source [4]. Motherboard vendors develop their own firmware on top of the AGESA interface. However, the open-source firmware project coreboot can be used as an alternative. Both openSIL and coreboot provide valuable insight into the platform initialization process and the AMD platform as a whole.

Routing Security. PSP enforces checks for every access to DRAM. Otherwise, the hypervisor could mount attacks on the PSP, such as (i) tricking the PSP into corrupting one of the security-relevant data structures (e.g., RMP) or (ii) writing CVM data to MMIO, which the hypervisor can then read or modify by using the underlying device. To prevent such attacks, the PSP aims to be as comprehensive as possible, covering all DRAM accesses and verifying that target addresses do not overlap with critical memory regions.

4. BREAKFAST Overview

We provide a high-level overview of how BREAKFAST subverts SEV-SNP protections.

Base Register for On-SoC elements. An On-SoC element is a hardware unit on the SoC that can expose an MMIO interface to other parts of the system. Unlike both external and integrated devices in Section 3, each On-SoC element exposes a base address register in the SMN, which sets the start address of its MMIO range. Platform components can write a physical address *addr* into the base address register of the On-SoC element. This write places the On-SoC element MMIO in the system physical address space.

```
#define DF_TRUST_LVL_IMPLICIT_BIT       (0)     /* Implicitly trusted PSP (Secure MP0). */
#define DF_TRUST_LVL_HIGH_BIT           (2)     /* CPU Microcode, SMU (MP1), SCFCTP/Aspen. */
#define DF_TRUST_LVL_UNTRUSTED_BIT      (7)     /* Default. All non-microcode accesses from CPU,
                                                 * all accesses from I/O, and any unknown sources.*/
```

Listing 2. SDF Trust Levels. The PSP has the highest privilege. Microcode has a trust level of 2, e.g., it can perform certain RMP state transitions that the hypervisor is prohibited from performing. Most other components have trust level 7, such as the hypervisor or PCIe devices.

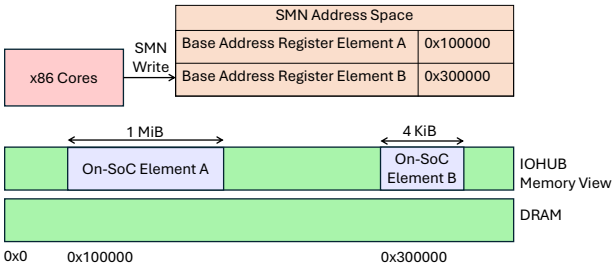

Figure 3. MMIO of two On-SoC elements. Element A has a size of 1 MiB and is mapped at `0x100000`. Element B has a size of 4 KiB and is mapped at `0x300000`.

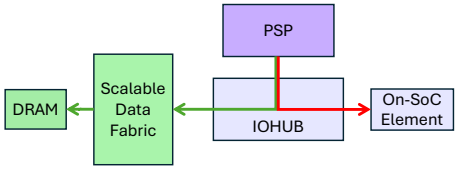

Figure 4. The IOHUB routes PSP memory requests to MMIO (red) with higher priority than to DRAM (green).

This MMIO range is then mapped at *addr* and spans exactly the fixed size of the On-SoC element.

Insight 1: x86 cores can access On-SoC element MMIO. We analyzed openSIL code and found that the code lists On-SoC elements[1] that expose an MMIO interface. An untrusted hypervisor can set the physical address range to which the On-SoC element should respond by configuring the base address register. Changing the base address of the On-SoC elements does not necessarily affect the system. For example, a base address register may correspond to an element that is excluded or deactivated in production hardware. To confirm whether this is the case, we access *addr* before and after the mapping from the x86 cores as outlined in Figure 3. When an x86 core accesses *addr* while no element maps to *addr*, the SoC returns `0xFF's` as a response for reads and silently discards the writes. Our experiments confirm this. As a second step, we map an On-SoC element by setting its base address to *addr*. We find that these On-SoC elements are connected by an IOHUB [4]. Thus, we choose *addr* such that it resolves to the correct IOHUB and observe that the values we read via the x86 cores change. This confirms that changing the base address to *addr* makes the On-SoC element reachable from the x86 cores.

Insight 2: x86 cores cannot write On-SoC element MMIO. After mapping an On-SoC element, reads to the element MMIO from x86 cores are of limited use, as the MMIO interface of these elements is undocumented and appears to expose only configuration registers. For writes, when we map the On-SoC element MMIO and attempt to modify its contents from the x86 cores, we observe that certain MMIO ranges are read-only. Overwriting the remaining, writable MMIO ranges has no visible side effects on the platform. Concretely, we do not observe any crashes or reported malfunctions of any On-SoC elements. This leaves us with two possible conclusions: either the On-SoC element MMIO ranges are in fact read-only, or the x86 cores lack sufficient privileges to modify them. Since the first conclusion would preclude further experimentation, we test whether the second conclusion holds.

Insight 3: PSP can write to On-SoC element MMIO. Assuming the x86 cores have insufficient privileges to modify these MMIO regions, we must instruct a higher-privileged entity to perform the writes for us. If we instruct the PSP to write to On-SoC element MMIO on our behalf, it may do so with higher privileges than the x86 cores. To perform our experiments, we need two primitives: (i) we need to trick the PSP into reading and writing to a physical address for us, and (ii) when the PSP writes to an address that overlaps with the On-SoC element MMIO, the routing should prefer MMIO over DRAM. If the second condition does not hold, the PSP will access DRAM instead of On-SoC element MMIO. We observe that both requirements are fulfilled. SEV-SNP has an API that allows the hypervisor to encrypt/decrypt CVM memory, which is effectively just a memory read and a memory write [2]. Then, if a PSP write targets a DRAM address, but *addr* overlaps with the destination of the access, the IOHUB routes the PSP request to On-SoC element MMIO, preventing it from reaching the SDF and thus DRAM (see also Figure 4). We find that, unlike experiments on x86 cores, we do not need to take special care to ensure *addr* is within the MMIO range of the corresponding IOHUB, since all PSP accesses targeting DRAM are routed through the IOHUB anyway. We confirm that we can use the PSP as a confused deputy to overwrite On-SoC element MMIO.

Insight 4: Writes from the PSP to `FASTREG` corrupt the control fabric. As we can indirectly write to the On-SoC element MMIO from the PSP, it remains to be explored how this can adversely impact SEV-SNP enforcement. There are six listed On-SoC element base address registers in openSIL. We defer to Section 5 for the detailed analysis of these elements. We observe that there are two elements,

1. The openSIL code uses the term *SMN non PCI devices*, but for readability we refer to these as On-SoC elements.

FASTREGCNTL and FASTREG, that allow us to create a mapping from the SDF to the SMN. In summary, FASTREG maps 1 MiB of the 4 GiB SMN address space into SDF. In other words, requests for the physical address range of FASTREG's 1 MiB range are actually performed on a 1 MiB range of the SMN. Furthermore, changing FASTREGCNTL lets us configure the 1 MiB range of SMN that FASTREG exposes. Thus, by writing to FASTREG and FASTREGCNTL, we can update any SMN register value, giving us complete access to the control plane.

Insight 5: Corrupting control fabric compromises SEV-SNP. Most of the SMN configuration fields are undocumented and potentially control dangerous devices such as voltage regulators. To prevent damage to the platform, we restrict our access to the publicly documented SMN registers used in the PSP SEV firmware [2]. The IOMMU exposes SMN registers that configure whether it enforces SEV-SNP protections. We can use Insight 3 and 4 to write attacker-controlled values to the SMN using the FASTREG mapping. Thus, we can disable the IOMMU SEV-SNP protections, such that there are no RMP checks on Direct Memory Access (DMA) accesses from devices to DRAM. We can therefore arbitrarily read and write to RMP-protected memory, including CVM memory.

5. Reverse Engineering On-SoC elements

This section details our experiments and reverse-engineering efforts to understand how On-SoC elements influence SoC routing and how we leverage this understanding to coerce the PSP into acting as a confused deputy. We perform all our experiments on an AMD Zen 5 EPYC 9135 processor with microcode version 0x0B002116 and SEV firmware version 1.55:44.

5.1. On-SoC elements

Our analysis of AMD manuals and openSIL source code reveals the existence and locations of several "SMN non PCI" devices–referred to here as On-SoC elements. The references list the MMIO ranges of these elements but do not document their functionality or the semantics of their MMIO registers [4], [11].

We found six On-SoC elements: DBG, FASTREG, FASTREGCNTL, IOAPIC, PSP, and SMU, summarized in Table 1. The openSIL code lists these elements alongside a base address register that is configurable through the SMN, and also lists the size of the corresponding MMIO range. Each base address register consists of two 32-bit registers that together form the low and high portions of a 64-bit physical address. Because the address is page-aligned, some of the lower bits are unused and are therefore repurposed to hold additional control information. For each On-SoC element, there are four such register pairs, with variable names indicating that there is one such register for each IOHUB.

TABLE 1. ON-SOC ELEMENTS, THEIR SMN ADDRESS ON THE FIRST IOHUB, AND SIZE [4], [11]

On-SoC element	SMN Address	Size
PSP	0x13B102E0	1 MiB
System Management Unit (SMU)	0x13B102E8	1 MiB
IOAPIC	0x13B102F0	256 B
DBG	0x13B102F8	512 KiB
FASTREG	0x13B10300	1 MiB
FASTREGCNTL	0x13B10308	4 KiB

5.2. Access to On-SoC elements from x86 Cores

Although openSIL and AMD documentation list the base address registers and MMIO sizes of these elements, they do not state whether, or under which conditions, these elements are mapped into the address space accessible by the x86 cores. We therefore experimentally attempt to establish an access path from the x86 cores to On-SoC elements, since the hypervisor can control the x86 cores.

Variable names in these references suggest that the IOHUB connects On-SoC elements to the system. As explained in Section 3, on Zen 5, the SoC contains four IOHUBs. The SDF contains 16 MMIO regions, four of which belong to each IOHUB. If a memory request falls within one of these regions, instead of being routed to DRAM, it is routed to the corresponding IOHUB. Because bootcode assigns a larger physical address range to each IOHUB than is required for the currently attached devices, parts of the assigned ranges remain unused. We can repurpose these unused ranges to map On-SoC element MMIO into the physical address space by satisfying two conditions. First, we must ensure that memory requests are routed to the IOHUB where the On-SoC element is connected. We guarantee this by programming the On-SoC element base address register to fall within an MMIO range that the SDF routes to that specific IOHUB. Failure to do so prevents the memory request from reaching the IOHUB, let alone the targeted On-SoC element. Second, to prevent a collision between the MMIO ranges of two devices on the same IOHUB, the chosen address range must not overlap with any already configured device on that IOHUB.

To find an unused range, we probe addresses within the MMIO region of one IOHUB until we identify a contiguous range that consistently reads as 0xFF and is large enough for the On-SoC element that we want to map, e.g., 1 MiB for FASTREG. We then configure the base address register of an On-SoC element to point to this range. To confirm that the IOHUB routes transactions from the x86 cores to the On-SoC element, we compare two configurations: one in which the element base address register points to this range, and one in which no device is mapped to this range. In the second configuration, reads from the window consistently return 0xFF. In contrast, when we map the On-SoC element, reads from the same addresses return element-specific data. This confirms Insight 1 from Section 4, the base address register of the On-SoC element maps the

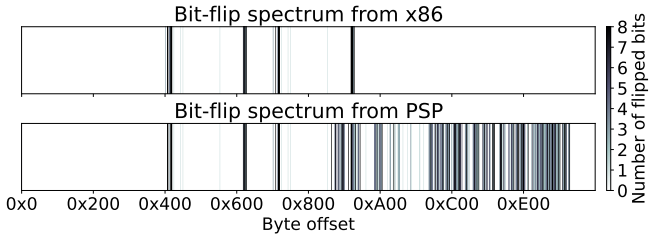

Figure 5. Access Control on FASTREGCNTL. The figure above shows how many bits the hypervisor can flip per byte in the FASTREGCNTL MMIO. The bottom figure shows the same using the confused deputy PSP.

element into MMIO accessible by the x86 cores. As a next step, we try to modify the contents of the On-SoC element MMIO.

5.3. Write Limitations for Access from x86 Cores

Having established an access path from the x86 cores to On-SoC elements, we next attempt to modify their state. When we write to On-SoC element MMIO, we observe that, for some address ranges, the values do not change, thus providing us with Insight 2 from Section 4. For example, Figure 5 shows which values can be modified from the x86 cores when attempting to flip all bits in the FASTREGCNTL 4 KiB MMIO.

We considered three explanations: the registers might not be implemented, they might be read-only, or our writes might lack sufficient privileges. As explained in Section 4, if one of the former two cases is correct, our experiments cannot continue. We therefore focus on the third possibility: that writes from the x86 cores lack sufficient privileges.

5.4. Attaining PSP Privileges

In Sections 5.2–5.3, we observed that x86 cores can read but cannot modify large parts of the On-SoC element MMIO space and hypothesize that these writes fail due to insufficient privileges rather than because the registers are unimplemented or intrinsically read-only. Next, we experimentally verify whether this is actually the case. To do so, we use the PSP as a confused deputy to access On-SoC element MMIO on behalf of the hypervisor. If we can perform MMIO accesses with the PSP privilege level and observe effects that the x86 cores cannot trigger, it strongly indicates that the registers are implemented and privilege-controlled rather than read-only. To do so, we need to instruct the PSP to write to On-SoC element MMIO.

As the hypervisor, we cannot gain code execution on the PSP to perform our experiment. However, we make two observations: (i) the PSP exposes hypervisor APIs that read and write data to and from DRAM, and (ii) we can influence IOHUB routing by configuring the base addresses of On-SoC elements. The SEV-SNP debug API allows the hypervisor to request that the PSP read a page from one DRAM location and write it to another (see Section 7). We leverage this to read and write data to the On-SoC element MMIO via the PSP at 4 KiB page granularity. In its intended use, this API is designed to be secure. It does not compromise SEV-SNP, as it must be explicitly enabled during CVM creation. The PSP also performs security checks on the source and destination addresses, ensuring that it reads from and writes to DRAM only.

We abuse this API by reconfiguring IOHUB routing. Specifically, we program the base address register of an On-SoC element so that its MMIO region overlaps with the physical address range that the PSP believes to be DRAM. Our experiments show that, in the presence of such an overlap, the IOHUB prioritizes the On-SoC element MMIO over DRAM. Consequently, when the hypervisor invokes the debug API on these addresses, the resulting reads and writes are issued by the PSP and the IOHUB routes them to the On-SoC element MMIO instead of DRAM. Note that we must use On-SoC elements on the same IOHUB as the PSP; otherwise, PSP accesses do not reach those elements.

This forms a classic confused-deputy scenario: the untrusted hypervisor tricks the highly privileged PSP (deputy) to move data between DRAM buffers and On-SoC element MMIO. The PSP thus unintentionally exercises its higher privileges on behalf of the hypervisor.

Using this to overwrite On-SoC element MMIO with PSP privileges, we observe two types of failures. Either the PSP becomes unresponsive, most likely due to a timeout or crash, or the entire platform stalls. We were unable to reproduce these crashes by writing the same data pattern from the x86 cores. Moreover, we sometimes observe failures even when the PSP only reads from these registers. These crashes and stalls indicate that the targeted MMIO registers are implemented and reachable with PSP privileges, whereas the platform drops identical accesses from the x86 cores. This in turn supports our hypothesis that the x86 cores lack sufficient privilege, rather than the registers being unimplemented or intrinsically read-only. Figure 5 demonstrates this behavior for the FASTREGCNTL MMIO range. Even though a large part of the range appears to be inherently read-only, we can overwrite specific values in FASTREGCNTL MMIO using the PSP that we cannot modify from the x86 cores; thus, we obtain Insight 3 from Section 4.

6. FASTREG & FASTREGCNTL

We have established in Section 5, both an x86-visible access path to On-SoC elements and a means to perform privileged MMIO writes via the confused deputy PSP. Next, our goal is to identify a suitable On-SoC element such that if we write an attacker-controlled value to an appropriate MMIO register, it will compromise SEV-SNP security guarantees. As shown in Table 1, there are six potential On-SoC elements to target. Our initial experiments with IOAPIC, DBG, SMU, and PSP were inconclusive due to a lack of documentation and of useful characteristics relevant to our research objectives. Only FASTREG and FASTREGCNTL showed promising behavior. Next, we outline our reverse engineering efforts to systematically study the semantics of FASTREG and FASTREGCNTL.

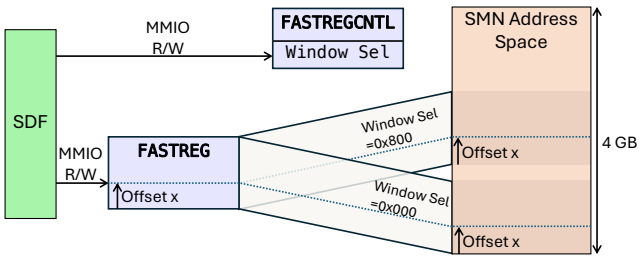

Figure 6. Reverse-engineered behavior of FASTREG and FASTREGCNTL. FASTREG exposes a 1 MiB (20-bit) sliding window into the SMN address space. The first 12 bits of FASTREGCNTL control the remaining upper 12 SMN address bits.

6.1. Reverse Engineering FASTREG

Based on their names, we hypothesize that FASTREGCNTL stores control information for FASTREG, which holds the associated data. Recall that the SoC exposes a separate base address register for each On-SoC element on each of the four IOHUBs, i.e., there are four MMIO instances per On-SoC element. As an initial step, we use the confused deputy to dump the contents of FASTREG and FASTREGCNTL for all four IOHUBs. We observe that the contents of FASTREGCNTL are identical on IOHUB 1, 2, and 3, but differ on IOHUB 0. In particular, the first four bytes of FASTREGCNTL on IOHUB 0 are 0x00000095, whereas on IOHUB 1–3 they are 0x0. Notably, the 1 MiB contents of FASTREG exhibit the same pattern: on IOHUB 1–3, it exposes identical values that differ from those of FASTREG on IOHUB 0.

Based on these observations, we hypothesize that the first four-byte word in the FASTREGCNTL MMIO region controls the values observed in FASTREG. To confirm our hypothesis, we use the PSP to write to FASTREGCNTL on IOHUB 0. We set the first four bytes to 0x00000000, matching the value on the other IOHUBs. However, because the PSP API only supports writing an entire 4 KiB page at once, we are forced to overwrite the entire FASTREGCNTL region. We use fine-grained writes, detailed in Section 7.2, which result in a successful overwrite without any observable adverse effects on the platform. Reading the FASTREG MMIO region on this IOHUB now returns the same values as on the other IOHUBs. We can also overwrite the first four bytes of FASTREGCNTL with 0x00000095 on the other IOHUBs, and observe that FASTREG on these IOHUBs now contains the same values as when FASTREGCNTL on the first IOHUB was configured with this value.

Thus, we infer that this first 32-bit word controls the contents of FASTREG. Note, however, that if we modify other values in FASTREGCNTL, the PSP times out or the platform stalls or crashes. Consequently, we focus on using the first four bytes to reverse-engineer the internal workings of FASTREG/FASTREGCNTL.

6.2. FASTREG Maps SMN into SDF Address Space

Section 6.1 showed that the first four-byte word in the FASTREGCNTL MMIO region directly influences the values observed through FASTREG. However, FASTREG still appears as an opaque 1 MiB MMIO range. We do not know whether FASTREG exposes scratch data from some internal device, mirrors an existing address space, or implements an entirely separate storage mechanism. Hence, our next goal in this section is to understand the semantics of the values exposed by FASTREG.

FASTREG maps unknown data into the normal address space accessible by the x86 cores. As discussed in the previous section, the value in the first four bytes of the FASTREGCNTL MMIO controls the contents of FASTREG. We hypothesize that these bytes encode some form of address or selector that determines which 1 MiB region is currently visible through FASTREG.

We experimentally characterize this behavior by modifying the first four bytes of FASTREGCNTL and observing the resulting contents of FASTREG. As a first step, we attempt to set all bits in these four bytes. When attempting this, we can only read back the value 0x800F0FFF, and we observe that the PSP either crashes or stalls. To systematically explore which bits are actually usable, we iteratively vary bitmasks over these four bytes and observe the resulting behavior. The bitmask with the most set bits that does not crash or stall the PSP is 0x7FF0FFFF. However, when we write this bitmask to FASTREGCNTL, the value we read back is 0x00000FFF. Comparing the effects of writing 0x7FF0FFFF and 0x00000FFF, we observe no difference in platform behavior and no change in the contents we can subsequently read from FASTREG. From this, we hypothesize that the bits in 0x7FF0F000 either (i) have side effects that are not visible through our current experiments, or (ii) correspond to read-only fields whose writes are masked. For the purposes of reverse-engineering the address mapping, this observation allows us to focus exclusively on the lower 12 bits.

According to openSIL, the FASTREG MMIO region is 1 MiB in size, which corresponds to 20 address bits. Put together, the lower 12 bits of FASTREGCNTL select a 1 MiB window, and the 1 MiB FASTREG MMIO region provides a 20-bit index within that window, for a total of 32 address bits. In other words, the lower 12 bits in FASTREGCNTL select the contents of the 1 MiB FASTREG MMIO range, yielding an effective 32-bit address space. We next investigate what corresponds to this 32-bit address space.

As a first step, we scan DRAM for any similarities with the contents from FASTREG, but we find no matches. We conclude that FASTREG most likely does not map to any DRAM. Next, we hypothesize that FASTREG instead maps into the SMN, since the SMN exposes a 32-bit address space that is not reachable from DRAM. To confirm the hypothesis, we read the SMN at offset 0x44000000 using the data/index register method explained in Section 3. Previous work identified that this address contains the initial

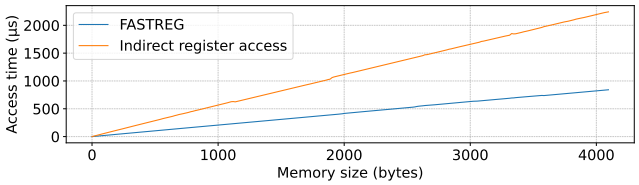

Figure 7. Access Latency to SMN using indirect register access and FASTREG.

encrypted bootcode of the PSP [12]. To determine whether we can access the same contents with FASTREG, we set the lower 12 bits of FASTREGCNTL to 0x440 and subsequently dump the values exposed by FASTREG. The two accesses return identical data, confirming that FASTREG indeed maps into the SMN address space.

To summarize, any address in the SMN can be mapped with FASTREG. To do so, we conceptually split a 32-bit SMN address into a 1 MiB-aligned window selector (the upper 12 bits) and an offset within that window (the lower 20 bits). We then write the window selector into the lower 12 bits of FASTREGCNTL and use the offset to index into FASTREG (see also Figure 6). Note that we must use the confused deputy PSP to write the window selector. We do not know the intended purpose of FASTREG. We measure the time required to access batches of data with different sample sizes in the SMN. Figure 7 compares these access times when using the standard indirect register method and when using FASTREG. We observe that, as the sample size grows, FASTREG is significantly faster. Recall that the early boot code of the PSP is mapped into the SMN, and if it needs to be read as a whole, it would benefit from using the FASTREG access method.

The semantics of FASTREG do not have direct security implications themselves. Although accesses to FASTREG itself are not privilege-controlled and x86 cores have full read/write access to its MMIO range, the underlying mapping into the SMN still enforces the same access checks as the standard index/data register pair method. However, by using the confused-deputy PSP, we can bypass the access control on the SMN and corrupt its state, thereby achieving Insight 4 from Section 4.

7. Putting It Together

In this section, we combine the confused deputy PSP with our reverse-engineered understanding of the FASTREG and FASTREGCNTL On-SoC elements to perform PSP-issued reads and writes to the SMN.

7.1. Selecting the Appropriate PSP API

We need to identify a suitable PSP API in which the hypervisor controls the content and address of a PSP write. After analyzing the available APIs, we specifically use SNP_DBG_ENCRYPT and SNP_DBG_DECRYPT to leverage the PSP as a confused deputy. Although these commands technically interact with encrypted CVM memory,

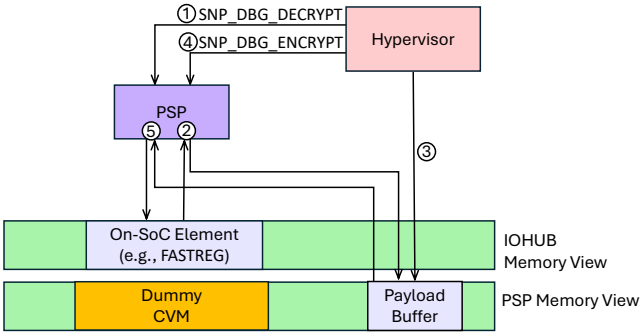

Figure 8. Fine-grained writes with the debug API. The malicious hypervisor uses a Read-Modify-Write pattern in conjunction with the confused deputy attack to perform fine-grained changes to the SMN register state.

because we overlap the destination address with the On-SoC element MMIO, the accesses are rerouted to MMIO instead of DRAM; i.e., they never reach the memory controller and are therefore never encrypted or decrypted. Most other PSP commands that write to DRAM either (i) instruct the PSP to perform an action (e.g., start a VM) or (ii) write only limited metadata (e.g., platform information) and therefore do not provide the desired control over the payload [13]. With these requirements in mind, only a few PSP commands remain suitable.

For example, SNP_GUEST_STATUS obtains the address of the guest-context page from the hypervisor and parses the entire page. However, the PSP returns only part of the page contents as metadata to the hypervisor. Thus, we do not control the data that the PSP writes to DRAM. Moreover, compared to the debug API, using SNP_GUEST_STATUS is more involved because the PSP first verifies that the page belongs to a guest context. Therefore, testing a different physical address would require creating a new CVM. Similar to the debug API commands, the SNP_PAGE_MOVE command also satisfies our requirements in principle. It also reads or writes a given physical address at page granularity. We were able to use the SNP_PAGE_MOVE command for the confused deputy attack. However, both the source and destination pages of SNP_PAGE_MOVE reside in encrypted memory. Thus, to obtain plaintext in the hypervisor when using SNP_PAGE_MOVE, we still require either the debug API or an additional interface between the dummy CVM and the hypervisor. In summary, we decided only to use the debug API.

Using the debug API, we can instruct the PSP to read and write data at any physical address that meets certain constraints. Primarily, the PSP checks that the corresponding CVM allows the usage of the debug API on its memory, the source and destination addresses are page-aligned, the CVM page belongs to the guest indicated in the command, and the page state is as expected. The hypervisor can easily satisfy all of those constraints. The hypervisor can start a dummy CVM and configure it to allow the usage of the debug API. As explained in Section 2, the hypervisor cannot perform all state transitions itself. However, the subset of

state transitions it can perform is sufficient for our purposes.

7.2. From Page to Byte Granularity

We now demonstrate how to use the debug API to instruct the PSP to issue reads and writes on behalf of the hypervisor, and then refine this mechanism to perform precise writes while minimizing side effects.

Reading & Writing Pages. To read from the On-SoC element MMIO, we invoke `SNP_DBG_DECRYPT` and choose the source page address so that it overlaps with the On-SoC element MMIO range. The PSP then copies the target On-SoC element MMIO values into hypervisor memory. The write path reverses this data flow. First, the hypervisor prepares a page in its own memory containing the desired payload. Then, the hypervisor invokes the `SNP_DBG_ENCRYPT` command and selects the destination page address such that it overlaps with the MMIO range of the On-SoC element. Subsequently, the PSP copies the payload to the MMIO region rather than to DRAM.

Granular Access with Read-Modify-Write. So far, the hypervisor can read from and write to On-SoC element MMIO by moving entire 4 KiB pages. As discussed in Section 5, reading from or writing to some registers can cause undesirable side effects, such as platform or PSP crashes. When only a specific register needs to be changed, we need fine-grained reads and writes instead to avoid side effects as much as possible.

As shown in Figure 8, we use the familiar Read-Modify-Write pattern. Low-level programming commonly uses this pattern to read a value, modify it (e.g., increment it), and then write the modified value back. In our case, we use it to read an entire page of register values, modify one or more registers, and then write the entire page back.

We again use our dummy CVM with debug mode enabled. Then, we invoke `SNP_DBG_DECRYPT` (①) to trick the PSP into reading an entire configuration page from the On-SoC element into hypervisor memory (②). After this step, the hypervisor only modifies the fields corresponding to the target registers (③). Next, the hypervisor uses `SNP_DBG_ENCRYPT` (④) to instruct the PSP to write the updated page back to the On-SoC element MMIO range (⑤). However, because this approach still writes an entire page back, it may cause side effects depending on register semantics, i.e., cause crashes as observed in Section 6.2.

7.3. Targeted SMN Access with PSP

Using the confused-deputy technique from Section 5.4 and our fine-grained read-write from Section 7.2, we now compose an end-to-end attack that drives the PSP to issue accesses directly into the SMN address space. This also provides us with Insight 5 from Section 4. We refer to this attack as BREAKFAST.

To determine the target SMN address and target value, we consult open-source references such as the PSP source code [2].

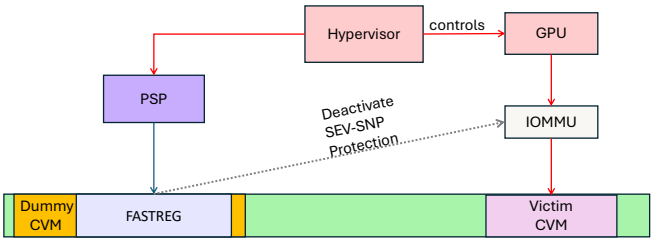

Figure 9. Compromising SEV-SNP with BREAKFAST. Malicious commands and subsequent GPU actions are in red, PSP confused-deputy writes are in blue, and `FASTREG` forwarding MMIO accesses to SMN is in grey. The hypervisor first uses the confused deputy PSP to disable IOMMU SEV-SNP protections. It then instructs a device, here a GPU, to tamper with protected memory.

Configuring `FASTREG` Window Selection. Recall from Section 6 that the first 12 bits in the `FASTREGCNTL` MMIO range select the 1 MiB window into the SMN that `FASTREG` will expose. The attacker overwrites it with the desired value, i.e., the upper 12 bits of the target SMN address.

Writing Arbitrary SMN Registers. Once the attacker sets the mapping, `FASTREG` forwards memory accesses to its MMIO range to the SMN window selected with `FASTREGCNTL`. We can then target specific registers. We confirm the end-to-end attack on Zen 5 and on Zen 4. On Zen 5, we used an AMD EPYC 9135 with microcode `0x0B002116`, SEV firmware version `1.55:44`, and BIOS version 1.5 on an ASRock BERGAMOD8-2L2T motherboard. On Zen 4, we used an AMD EPYC 9124 with microcode `0x0A101154`, SEV firmware version `1.55:43`, and BIOS version 10.023 on a Supermicro H13SSL. In both cases, we used a modified 6.15-rc5 host kernel and a 6.10 guest kernel.

8. Breaking SEV-SNP using BREAKFAST

We use BREAKFAST to disable RMP protection within the IOMMU (Section 8.1), thereby bypassing SEV-SNP access protection for DMA. After disabling SEV-SNP enforcement in the IOMMU, any DMA-capable device can write to RMP-protected memory (Section 8.2). We then exploit this capability to perform two attacks on SEV-SNP. In the first attack, we enable debug mode for a protected CVM (Section 8.3). In the second attack, we forge attestation (Section 8.4). Figure 9 illustrates the overall attack.

8.1. Disabling IOMMU SEV-SNP Protection

During SEV-SNP initialization, the PSP programs the IOMMU with the RMP base address, and configures SEV-SNP-related control registers to enforce access protections. During shutdown, the PSP clears these settings again. Listing 3 shows the corresponding PSP firmware snippet for clearing one such control register. Using BREAKFAST, we mimic this behavior and program the same registers to disable SEV-SNP enforcement in the IOMMU.

```
addr = iommul2bpsp_base_address(nbio,
    IOMMUL2B_VMGUARDIO_CNTRL_0_OFFSET);
status = sev_hal_map_smn_on_die(addr, (void
    **)&axi_addr, die);
if (status != SEV_STATUS_SUCCESS)
    goto end;
data = *axi_addr;
data &= ~(IOMMUL2B_VMGUARDIO_SNP_EN_MASK);
*axi_addr = data;
sev_hal_unmap_smn((void *)axi_addr);
```

Listing 3. PSP Firmware Code to Disable IOMMU SEV-SNP Protection.

For clarity, we describe the procedure here for a single representative register, `IOMMUL2B_VMGUARDIO_CNTRL_0`.

Because the IOMMU is part of the IOHUB, each IOHUB instance has its own copy of `IOMMUL2B_VMGUARDIO_CNTRL_0`. In the PSP source code, the base address for the registers containing IOMMU registers on the first IOHUB is denoted as `IOMMUL2BPSP_BASE_ADDRESS` with an address of `0x13FFE000`. The register `IOMMUL2B_VMGUARDIO_CNTRL_0` resides at offset `0x1E0` from this base. Thus, in the SMN address space, the register is located at `0x13FFE1E0`. To modify this register via BREAKFAST, we first map the page containing `IOMMUL2B_VMGUARDIO_CNTRL_0` into `FASTREG`. We achieve this by instructing the PSP to set the first 12 bits of the `FASTREGCNTL` MMIO register to `0x13F`, which are the upper 12 bits of the register address. Then, using our fine-grained write, we read an entire page at `0xFE000` from the 1 MiB `FASTREG` MMIO range. Inside this page, we find `IOMMUL2B_VMGUARDIO_CNTRL_0` at offset `0x1E0`. Following the PSP firmware, we clear bit seven (`IOMMUL2B_VMGUARDIO_SNP_EN_MASK`) of the register value. We then proceed by writing the entire modified page of register values back. We apply this procedure to all remaining SEV-SNP-related IOMMU registers listed in the PSP source code.

Implementation. We implement the code to disable the IOMMU inside a kernel module. The kernel module comprises 1625 LoC.

Validation. Across 1024 samples, disabling the IOMMU takes 133 ms on average and 137.8 ms worst-case, but since we perform this step only once per platform boot, the impact is negligible. After this step, the IOMMU no longer enforces RMP checks, which we confirm empirically by using a DMA-capable device to write to RMP-protected memory. Specifically, we test whether we can write to CVM memory to tamper with a counter.

8.2. Malicious DMA Accesses

We use a hypervisor-controlled GPU (GeForce RTX 3080) to initiate malicious DMA transactions targeting RMP-protected memory, to exploit the lack of RMP checks in IOMMU. We implement a CUDA kernel in the malicious hypervisor that copies data between two host-mapped buffers using DMA, providing the base for the attacks in Sections 8.3 and 8.4. Since the attacks write to the guest context page and CUDA runs as a userspace application, we create a small kernel module in the hypervisor that maps the guest context page into the userspace application. The userspace application registers this page with CUDA using `cudaHostRegister(<addr>, cudaHostRegisterPortable)` so that the hypervisor-controlled GPU can access it. The base CUDA program consists of 136 LoC, and the kernel module comprises 729 LoC.

8.3. Enabling Debug Mode

After disabling IOMMU SEV-SNP protections, we next demonstrate how to enable debug mode on a victim CVM. Recall that SEV-SNP includes a debug mechanism that allows a hypervisor to encrypt or decrypt guest pages. Only the guest owner can activate this functionality by supplying a policy bit during the creation of a CVM, which then resides in the guest context page. The platform encrypts this page with a boot-unique key, and RMP access control protects this page from unauthorized modifications, such as those made by the hypervisor.

We demonstrate that BREAKFAST can induce a runtime change to the guest debug policy. At a high level, our attack proceeds as follows. We begin by using BREAKFAST to disable SEV-SNP enforcement in the IOMMU. To prevent in-guest detection, the untrusted hypervisor halts the victim CVM. Next, the hypervisor instructs a malicious device to read the guest context page via DMA, saving it for later restoration. The device then flips bits in the encrypted guest context page via DMA. Because the memory controller encrypts this page, each flipped ciphertext bit randomizes the corresponding plaintext. We know from prior work that the debug policy is a single-bit flag; therefore, each attempt succeeds with a 50% probability. After each attempt, we invoke the `SNP_GUEST_STATUS` command to check whether we successfully flipped the debug bit. If the attempt is unsuccessful, we restore the original guest context and try again with a different combination of flipped bits. Once we have successfully set the debug bit, we can use the debug commands to read and write CVM memory. Finally, after we have extracted confidential information or tampered with the CVM using `SNP_DBG_ENCRYPT`, we restore the original guest context page to avoid detection and resume the CVM. Since the CVM owner can only examine the CVM policy during attestation, this attack is practically undetectable, aside from potential timing anomalies during VM halts, which can also arise during standard operation.

Implementation. SEV-SNP encrypts memory in 16-byte blocks, so we only need to modify the block that contains the policy. To locate the specific block within the guest context page that contains the policy field, we again use the `SNP_GUEST_STATUS` command: for each candidate block, we iteratively flip individual bits via the GPU and observe whether the reported policy changes. If a change occurs, we have identified the correct block; otherwise, we restore the original guest context page and proceed to the next candidate. Through this process, we determine that the

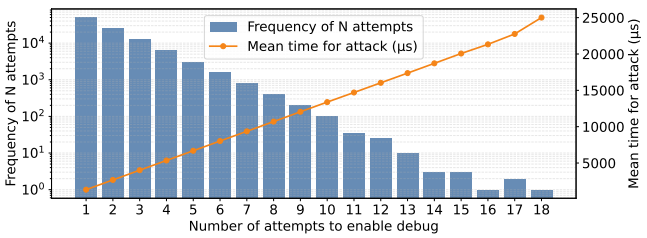

Figure 10. Histogram of attempts required to enable debug mode and corresponding mean time.

policy resides in block 31 of the guest context page. This identification step can be done once during an offline phase on a dummy CVM. During the actual attack, we instruct the GPU to flip bits within this block until `SNP_GUEST_STATUS` indicates that we have set the debug bit, after which the hypervisor can exploit the debug privileges. Afterward, we instruct the GPU to restore the original guest context page from the saved copy.

We add 26 LoC to the kernel module from Section 8.2 to invoke the `SNP_GUEST_STATUS` command. We also add 110 LoC to the base CUDA userspace application.

Evaluation. We performed this attack 100 k times, and the statistics for flipping the debug bit are shown in Figure 10. Our attack is probabilistic, but it succeeded every time. Across all trials, the maximum number of attempts to enable debug mode that we observed is 18. The time to enable the debug bit varies with the number of iterations. If only one iteration is necessary, which is the case around 50% of the time, the average time is 1.3 ms; for the rare case of 18 attempts, the time required is 25 ms. Across 100 k trials, enabling the debug bit requires, on average, 2.04 attempts and 3.2 ms, which matches the expected 50% success probability per attempt.

8.4. Bypassing Attestation

We demonstrate a second method of using the lack of IOMMU protections to bypass SEV-SNP by forging attestation. The PSP measures the guest only once, during CVM launch [2]. This measurement is a 48-byte digest over the initial CVM memory. Like the guest policy, the PSP stores the measurement in the (encrypted) guest context page. However, the encryption is deterministic with respect to the physical address. Therefore, if we place the guest context page at the same physical address, the same plaintext yields the same ciphertext [14]. We exploit this by copying the ciphertext of a benign guest measurement and later splicing it into the guest context page of a malicious CVM.

Implementation. We start by launching a benign image. Since the guest context page is encrypted with a tweak based on the physical address, we must place the guest context page for the victim and corrupted CVM at the same physical address. Using BREAKFAST, we disable the IOMMU. We then use the hypervisor-controlled GPU to read the portion of the guest context page that contains the measurement and store it. Since we know that the measurement is a 48-byte value, we attempt to replace the measurement at all possible offsets within the page until we succeed. For our firmware, the measurement resides in blocks 71–73, i.e., at byte offset `0x470`. We read these blocks and save them to forge the attestation at any later point in time. When the hypervisor creates the victim CVM, it replaces the CVM image with a malicious one and ensures that it places the guest context page to the same physical address as before, such that decryption semantics match. When the hypervisor starts the CVM, we halt it immediately after launch, before the CVM performs attestation (e.g., to obtain a disk decryption key). The attack does not depend on a race condition to halt the CVM before it requests attestation, since the guest sends the attestation request to the hypervisor, which then forwards it to the PSP. Then, we overwrite blocks 71–73 with the previously saved ciphertext from the benign CVM. The tampered CVM now reports the same measurement as the benign image, enabling forged attestation.

Evaluation. We reuse the kernel module from Section 8.2 and add 135 LoC to the base CUDA userspace application. Our attack succeeds 100% of the time. Across 100 k overwrites of the measurement value, the mean latency per overwrite is 1.1 ms, with a worst-case latency of 2.1 ms.

9. Discussion

We discuss the root cause of BREAKFAST, possible mitigations, and applicability to other architectures.

9.1. Root Cause Analysis

Three underlying issues enable BREAKFAST.

Violation of Least Privilege. We observe that the PSP violates the principle of least privilege by using its high privileges to access DRAM. An attacker can indirectly cause the PSP to write to memory, and those writes carry the PSP's highest privilege level. Although the details of the AMD interconnect are undocumented, open-source code indicates that memory transactions propagate the requester `UnitID` along with a memory request [2], [3]. However, the PSP APIs we study only access unprotected DRAM. For such calls, propagating the PSP's elevated security attribute with the memory request is unnecessary. Note that just having PSP privileges is not sufficient to carry out BREAKFAST, as we cannot use these privileges, since all default PSP operations only target CVM memory.

Device MMIO Relocation bypasses DRAM checks. The PSP validates each access that it believes to be a DRAM access. However, AMD allows the untrusted hypervisor to remap the MMIO range of On-SoC elements. Specifically, the hypervisor can overlap the address of benign PSP memory accesses with On-SoC element MMIO. Thus, the PSP thinks it is accessing DRAM and performs the checks that pass, ending in a request on the interconnect. Even though the PSP builds its memory map once during initialization, it neither checks whether On-SoC elements are mapped

within its known MMIO regions nor detects changes to those mappings over the platform lifecycle. In other words, the malicious remapping bypasses PSP checks. Thus, by virtue of these two root causes, we can perform a confused deputy attack on the PSP.

IOHUB Memory Routing. When we trick the PSP into issuing a DRAM request that is headed for the MMIO range, we find that the PSP issues the access request with software flags `Ax_DRAM` set to map DRAM [2]. When the interconnect receives such a request, it should not route it to MMIO, but to DRAM, as indicated by the flag. However, as per our experiments (Section 5), the interconnect forwards the access to the MMIO range.

9.2. Mitigation

BREAKFAST can be prevented by addressing at least one of the root causes above. These mitigations require changes to components outside our control (e.g., interconnect, PSP). Thus, we cannot implement any of the mitigation measures and validate their effectiveness.

Restricted Security Attributes. We propose that the PSP should use lower privileges when possible. Currently, the PSP does not distinguish between its memory requests requiring high privileges (e.g., PSP writes to the Trusted Memory Region [2]) or user-initiated memory accesses (e.g., `SNP_PAGE_MOVE`) that can be done with lower privileges. However, many user-initiated operations do not require PSP-level privileges, because they target unprotected DRAM. Thus, it is possible for the PSP to drop its privileges for the APIs we leverage for the confused deputy attack. This mitigation may require hardware changes, but it hardens the PSP so that it adheres to the principle of least privilege.

Locking Base Addresses. The PSP can prevent MMIO relocation by locking the base address registers of all On-SoC elements. This alone is insufficient to mitigate BREAKFAST; the PSP must also refuse to use locked On-SoC element MMIO for its own DRAM accesses. Otherwise, this may enable the confused deputy attack again. To stop such an attack, the PSP should additionally ensure that the range starting from the base address is within the MMIO region recorded in the PSP memory map that it constructs during its initialization. Additionally, the base address register must be locked early in the platform boot process, before any untrusted code executes. Otherwise, early boot code on the x86 cores can modify the base address before the locking occurs. Lastly, these checks must cover both documented and undocumented On-SoC elements, as public lists (e.g., openSIL) may be incomplete [4], [11].

Explicit Memory Routing. To address malicious routing (i.e., prevent PSP DRAM requests from being routed to On-SoC element MMIO), the IOHUB must not route PSP requests that indicate DRAM as the destination to On-SoC element MMIO. We cannot determine which hardware or firmware components need to be modified to enforce this. We find that the PSP already maps DRAM addresses with flags indicating they are DRAM. It is unclear whether the IOHUB misses a corresponding check, if the flag is not propagated at all, or if such a check is impossible.

9.3. Impact on Other CVM Designs

We discuss the feasibility of BREAKFAST to Intel SGX/TDX and Arm CCA.

Intel has a dedicated co-processor called the Management Engine with access to many internal buses [15]. Due to the closed-source nature of the Intel ecosystem, we were unable to conclude whether untrusted components could alter the routing configuration for the Management Engine such that it acts as a confused deputy. Intel platforms have multiple sideband interconnects that are similar to those in SMN [9]. The exact mechanisms of how these interconnects employ access controls are undocumented. However, previous work found an undocumented CPU instruction in multiple Intel processors that enables high-privilege access to various sideband interconnects [16].

Arm bases its trust on firmware running in the highest privilege level on the application processors. In the Arm ecosystem, it is up to the SoC vendors to customize the platform components and implement a trusted co-processor to assist the application processors with diverse tasks, e.g., bootstrapping or other security features [17], [18]. Whether these co-processors can compromise Arm CCA is unclear, as there is currently no commercially available Arm CCA CPU.

10. Related Work

We first outline other instances of confused deputy attacks and then discuss prior attacks on AMD SEV-SNP.

Confused Deputy Attacks. Confused Deputy attacks are well studied in numerous contexts. Hardy coined the term confused deputy [19]. Felt et al. showcase a special case of confused deputy attacks, permission re-delegation [20]. Less-privileged applications instruct applications with higher, user-controlled permissions to perform a privileged API call. ARF proposes a framework for automatically identifying permission re-delegation in system service entrypoints [21]. Fred proposes a tool to identify Android service entrypoints that re-delegate sensitive file access to third-party applications [22]. Similarly, iService proposes a system for detecting confused deputy attacks in higher-privileged AppleOS system services [23]. Duta et al. corrupt the stack to trick exception handling into acting as a confused deputy to achieve control-flow hijacking [24]. Furthermore, confused deputy attacks have been demonstrated within a blockchain environment [25], [26]. Just like BREAKFAST, these attacks abuse a higher-privileged entity to circumvent some form of access restriction.

SEV Attacks. There are various attacks on SEV and SEV-ES that abuse insufficient page table protection and other architectural flaws [27], [28], [29], [30], [31], [32], [33]. SEV-SNP mitigates many of the architectural weaknesses of its predecessors. Recent works built affordable hardware

attack kits to launch physical attacks. BadRam uses DRAM aliasing to bypass the RMP and compromise SEV-SNP [34]. WireTap and TEE.fail build affordable memory interposers capable of subverting SEV-SNP security measures [35], [36]. Google performed a preliminary security analysis of SEV-SNP prior to the public release [37]. Buhren et al. perform voltage fault injection to achieve code execution on the PSP [12]. CacheWarp induces cache incoherency to drop writes, enabling rollback of data and instructions [38]. By tampering with the stack engine, StackWarp extracts secrets and escalates privileges inside the CVM [39]. Heckler and WeSee exploit the hypervisor capability to inject malicious interrupts into a SEV-SNP CVM [40], [41]. RMPocalypse exploits insufficient protection of the RMP during initialization [42]. CounterSEVeillance uses performance counters as a side-channel to infer information from within the CVM [43]. BadAML exploits unattested guest firmware to compromise SEV-SNP CVMs [44]. Previous works demonstrated ciphertext and cache side-channel attacks against SEV-SNP [14], [45], [46], [47], [48], [49], [50], [51]. Unlike these prior works, BREAKFAST introduces a confused deputy attack against the PSP that compromises the control fabric.

Interconnect Corruption Attacks. Fabricked is the first work that shows how a malicious hypervisor can misconfigure the Infinity Fabric to break AMD SEV-SNP [5]. In particular, it uses the untrusted UEFI to tamper with the data fabric (SDF) to reroute DRAM accesses originating from the PSP. These accesses are eventually discarded. BREAKFAST is a second instance of Interconnect Corruption Attacks, as it also misconfigures the Infinity Fabric. Specifically, we target the IOHUB routing, which occurs before PSP DRAM accesses reach the SDF (Figure 1), to reroute these accesses to On-SoC elements. We then leverage this to obtain PSP privileges on the SMN. Furthermore, we do not modify the UEFI.

11. Conclusion

We present BREAKFAST, a novel attack that breaks AMD SEV-SNP. BREAKFAST tricks the PSP into writing attacker-controlled values to an On-SoC element called `FASTREG`, allowing the untrusted hypervisor to escalate its privileges on the SMN. We showcase that BREAKFAST can disable IOMMU SEV-SNP protections, to subsequently fake attestation and enable debug on production CVMs. We hope that this work motivates further investigation into interconnect-level security for confidential computing.

12. Acknowledgement

We thank the anonymous reviewers, Andrin Bertschi, and Mark Kuhne for their constructive feedback. Benedict Schlüter is supported by the Google PhD Fellowship for Privacy, Safety, and Security.

Ethics considerations

We reported the vulnerability to AMD and refrain from sharing any code until patches are publicly available. AMD acknowledged the vulnerability and asked for an embargo such that they can develop and roll out the patches before the public announcement.

LLM usage considerations

LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality. ChatGPT was used in the preparation of this manuscript. Additionally, Copilot was used for code development.

References

[1] AMD, "AMD SEV-SNP: Strengthening VM Isolation with Integrity protection and more," https://docs.amd.com/v/u/en-US/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more, 2020.

[2] ——, "AMD-ASPFW," https://github.com/amd/AMD-ASPFW, 2023.

[3] T. Burd, N. Beck, S. White, M. Paraschou, N. Kalyanasundharam, G. Donley, A. Smith, L. Hewitt, and S. Naffziger, ""Zeppelin": An SoC for Multichip Architectures," in *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, 2019, pp. 133–143.

[4] AMD, "openSIL: Open-Source Silicon Initialization Library," https://github.com/openSIL/openSIL, 2025, accessed: 2025-11-13.

[5] B. Schlüter, C. Wech, and S. Shinde, "Fabricked: Misconfiguring Infinity Fabric to Break AMD SEV-SNP," in *USENIX Security*, 2026.

[6] AMD, "AMD SEV," https://docs.amd.com/v/u/en-US/memory-encryption-white-paper, accessed 2023-05-04.

[7] D. Kaplan, "PROTECTING VM REGISTER STATE WITH SEV-ES," 2017.

[8] A. Kegel, P. Blinzer, A. Basu, and M. Chan, "Virtualizing IO through IO Memory Management Unit (IOMMU)," *ASPLOS Tutorials*, 2016.

[9] I. E. Papazian, "New 3rd Gen Intel® Xeon® Scalable Processor (Codename: Ice Lake-SP)," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, 2020, pp. 1–22.

[10] AMD, "Processor Programming Reference (PPR) (57228)," 2025.

[11] ——, "Dynamic Root of Trust Measurement (DRTM) Service Integration Guide (58453)," https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/user-guides/58453.pdf, 2025, accessed: 2025-11-13.

[12] R. Buhren, H.-N. Jacob, T. Krachenfels, and J.-P. Seifert, "One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2875–2889. [Online]. Available: https://doi.org/10.1145/3460120.3484779

[13] AMD, "SEV Secure Nested Paging Firmware ABI Specification, Rev 1.58," https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf, 2025.

[14] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, "CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 717–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan

[15] Peter Bosch, "Intel Management Engine deep dive," https://media.ccc.de/v/36c3-10694-intel_management_engine_deep_dive, 2019, accessed: 2025-11-13.

[16] M. Ermolov, D. Sklyarov, and M. Goryachy, "Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors," *Journal of Computer Virology and Hacking Techniques*, vol. 19, no. 3, pp. 351–365, Sep. 2023. [Online]. Available: https://doi.org/10.1007/s11416-022-00438-x

[17] Google, "Titan hardware chip," 2025.

[18] Apple, "Secure Enclave," 2024.

[19] N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," *SIGOPS Oper. Syst. Rev.*, vol. 22, no. 4, p. 36–38, Oct. 1988. [Online]. Available: https://doi.org/10.1145/54289.871709

[20] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission Re-Delegation: Attacks and Defenses," in *20th USENIX Security Symposium (USENIX Security 11)*. San Francisco, CA: USENIX Association, Aug. 2011. [Online]. Available: https://www.usenix.org/conference/usenixsecurity11/permission-re-delegation-attacks-and-defenses

[21] S. A. Gorski and W. Enck, "ARF: identifying re-delegation vulnerabilities in Android system services," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 151–161. [Online]. Available: https://doi.org/10.1145/3317549.3319725

[22] S. A. G. III, S. Thorn, W. Enck, and H. Chen, "FReD: Identifying file Re-Delegation in android system services," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[23] Y. Wang, Y. Hu, X. Xiao, and D. Gu, "iService: Detecting and Evaluating the Impact of Confused Deputy Problem in AppleOS," in *Proceedings of the 38th Annual Computer Security Applications Conference*, ser. ACSAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 964–977. [Online]. Available: https://doi.org/10.1145/3564625.3568001

[24] V. Duta, F. Freyer, F. Pagani, M. Muench, and C. Giuffrida, "Let Me Unwind That For You: Exceptions to Backward-Edge Protection," in *NDSS*, Feb. 2023, Intel Bounty Reward. [Online]. Available: https://download.vusec.net/papers/chop_ndss23.pdf

[25] F. Gritti, N. Ruaro, R. McLaughlin, P. Bose, D. Das, I. Grishchenko, C. Kruegel, and G. Vigna, "Confusum Contractum: Confused Deputy Vulnerabilities in Ethereum Smart Contracts," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 1793–1810. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/gritti

[26] Z. Zhong, Z. Zheng, H.-N. Dai, Q. Xue, J. Chen, and Y. Nan, "PrettySmart: Detecting Permission Re-delegation Vulnerability for Token Behaviors in Smart Contracts," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639140

[27] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "SEVered: Subverting AMD's Virtual Machine Encryption," in *Proceedings of the 11th European Workshop on Systems Security*, ser. EuroSec'18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3193111.3193112

[28] F. Hetzelt and R. Buhren, "Security Analysis of Encrypted Virtual Machines," vol. 52, no. 7. New York, NY, USA: Association for Computing Machinery, Apr. 2017, p. 129–142. [Online]. Available: https://doi.org/10.1145/3140607.3050763

[29] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "SEVurity: No Security Without Integrity : Breaking Integrity-Free Memory Encryption with Minimal Assumptions," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1483–1496.

[30] M. Li, Y. Zhang, and Z. Lin, "CrossLine: Breaking "Security-by-Crash" based Memory Isolation in AMD SEV," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2937–2950. [Online]. Available: https://doi.org/10.1145/3460120.3485253

[31] M. Radev and M. Morbitzer, "Exploiting Interfaces of Secure Encrypted Virtual Machines," in *Reversing and Offensive-Oriented Trends Symposium*, ser. ROOTS'20. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–12. [Online]. Available: https://doi.org/10.1145/3433667.3433668

[32] F. Hetzelt, M. Radev, R. Buhren, M. Morbitzer, and J.-P. Seifert, "VIA: Analyzing Device Interfaces of Protected Virtual Machines," in *Proceedings of the 37th Annual Computer Security Applications Conference*, ser. ACSAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 273–284. [Online]. Available: https://doi.org/10.1145/3485832.3488011

[33] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, "TLB Poisoning Attacks on AMD Secure Encrypted Virtualization," in *Proceedings of the 37th Annual Computer Security Applications Conference*, ser. ACSAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 609–619. [Online]. Available: https://doi.org/10.1145/3485832.3485876

[34] J. De Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede, and J. Van Bulck, "Badram: Practical memory aliasing attacks on trusted execution environments," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 4117–4135. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00104

[35] J. De Meulemeester, D. Oswald, I. Verbauwhede, and J. 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)*, 2026.

[36] J. Chuang, A. Seto, N. Berrios, S. van Schaik, C. Garman, and D. Genkin, "TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition," in *47th IEEE Symposium on Security and Privacy (IEEE S&P '26)*, 2026.

[37] Google, "Oh SNP! VMs get even more confidential," 2023.

[38] R. Zhang, L. Gerlach, D. Weber, L. Hetterich, Y. Lü, A. Kogler, and M. Schwarz, "CacheWarp: Software-based fault injection using selective state reset," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1135–1151. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-ruiyi

[39] R. Zhang, T. Hornetz, D. Weber, F. Thomas, and M. Schwarz, "StackWarp: Breaking AMD SEV-SNP Integrity via Deterministic Stack-Pointer Manipulation through the CPU's Stack Engine," in *USENIX Security*, 2026.

[40] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi, and S. Shinde, "HECKLER: Breaking Confidential VMs with Malicious Interrupts," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3459–3476. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/schl%C3%BCter

[41] B. Schlüter, S. Sridhara, A. Bertschi, and S. Shinde, "WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4220–4238.

[42] B. Schlüter and S. Shinde, "RMPocalypse: How a Catch-22 Breaks AMD SEV-SNP," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 3840–3854. [Online]. Available: https://doi.org/10.1145/3719027.3765233

[43] S. Gast, H. Weissteiner, R. Schröder, and D. Gruss, "CounterSEVeillance: Performance-Counter Attacks on AMD SEV-SNP," in *NDSS*, Feb. 2025. [Online]. Available: https://www.ndss-symposium.org/ndss2025/

[44] S. Takekoshi, M. Mori, T. Fukai, and T. Shinagawa, "BadAML: Exploiting Legacy Firmware Interfaces to Compromise Confidential Virtual Machines," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 4469–4483. [Online]. Available: https://doi.org/10.1145/3719027.3765123

[45] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, "CipherSteal: Stealing Input Data from TEE-Shielded Neural Networks with Ciphertext Side Channels," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025, pp. 4136–4154.

[46] M. Li, L. Wilke, J. Wichelmann, T. Eisenbarth, R. Teodorescu, and Y. Zhang, "A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 337–351.

[47] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, "HyperTheft: Thieving Model Weights from TEE-Shielded Neural Networks via Ciphertext Side Channels," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 4346–4360. [Online]. Available: https://doi.org/10.1145/3658644.3690317

[48] B. Schlüter, C. Wech, and S. Shinde, "Heracles: Chosen Plaintext Attack on AMD SEV-SNP," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 3810–3824. [Online]. Available: https://doi.org/10.1145/3719027.3765209

[49] Y. Yan, W. Huang, I. Grishchenko, G. Saileshwar, A. Mehta, and D. Lie, "Relocate-vote: using sparsity information to exploit ciphertext side-channels," in *Proceedings of the 34th USENIX Conference on Security Symposium*, ser. SEC '25. USA: USENIX Association, 2025.

[50] L.-C. Chiang and S.-W. Li, "Reload+Reload: Exploiting Cache and Memory Contention Side Channel on AMD SEV," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. New York, NY, USA: Association for Computing Machinery, 2025, p. 1014–1027. [Online]. Available: https://doi.org/10.1145/3676641.3716017

[51] L. Giner, S. R. Neela, and D. Gruss, "Cohere+Reload: Re-enabling High-Resolution Cache Attacks on AMD SEV-SNP," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 22nd International Conference, DIMVA 2025, Graz, Austria, July 9–11, 2025, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2025, p. 191–212. [Online]. Available: https://doi.org/10.1007/978-3-031-97620-9_11

Appendix A.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper proposes a novel attack called BREAKFAST, which breaks AMD SEV-SNP. It tricks the PSP into writing attacker controlled values to an On-SoC element called FASTREG, which allows the untrusted hypervisor to hijack control of the System Management Network. The experiment shows that the proposed attack can disable IOMMU's SEV-SNP protections.

A.2. Scientific Contributions

5) Identifies an Impactful Vulnerability
6) Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) Identifies an Impactful Vulnerability. This paper identifies a previously unknown vulnerability in AMD SEV-SNP.
2) The paper provides a valuable step forward in an established field. The proposed attack improves the understanding of the SEV-SNP threat surface, and it represents a meaningful step forward in confidential computing security research.