CROSSLINE: Breaking "Security-by-Crash" based Memory Isolation in AMD SEV

Mengyuan Li, Yinqian Zhang, Zhiqiang Lin
Department of Computer Science and Engineering
The Ohio State University
li.7533@osu.edu, {yinqian, zlin}@cse.ohio-state.edu

***Abstract*—AMD's Secure Encrypted Virtualization (SEV) is an emerging security feature on AMD processors that allows virtual machines to run on encrypted memory and perform confidential computing even with an untrusted hypervisor. This paper first demystifies SEV's improper use of address space identifier (ASID) for controlling accesses of a VM to encrypted memory pages, cache lines, and TLB entries. We then present the CROSSLINE attacks, a novel class of attacks against SEV that allow the adversary to launch an attacker VM and change its ASID to that of the victim VM to impersonate the victim. We present two variants of CROSSLINE attacks: CROSSLINE V1 decrypts victim's page tables or memory blocks following the format of a page table entry; CROSSLINE V2 constructs encryption and decryption oracles by executing instructions of the victim VM. We have successfully performed CROSSLINE attacks on SEV and SEV-ES processors.**

I. INTRODUCTION

AMD's Secure Encrypted Virtualization (SEV) is a security extension for the AMD Virtualization (AMD-V) architecture [4], which allows one physical server to efficiently run multiple guest virtual machines (VM) concurrently on encrypted memory. When SEV is enabled, the memory pages used by a guest VM are transparently encrypted by a secure co-processor using an ephemeral key that is unique to each VM, thus allowing the guest VMs to compute on encrypted memory. SEV is AMD's ambitious movement towards confidential cloud computing, which is gaining traction in the cloud industry. For instance, Google Cloud recently provides SEV-enabled VMs, called Confidential VMs, as its first product of Confidential Computing [9].

Unlike traditional security assumptions in which the trustworthiness of the system software is taken for granted, SEV is built atop a threat model where system software including hypervisor can be untrusted.

> *"SEV technology is built around a threat model where an attacker is assumed to have access to not only execute user level privileged code on the target machine, but can potentially execute malware at the higher privileged hypervisor level as well."* [15].

Consequently, such an audacious threat assumption has been examined under the microscope with numerous attacks (e.g., [10], [8], [6], [20], [19], [17], [27]) since its debut in 2017. With the assumption of a malicious hypervisor, these attacks successfully compromise the confidentiality and/or integrity provided by SEV's memory encryption by exploiting a number of design flaws, including unencrypted virtual machine control blocks (VMCB) [10], [27], unauthenticated memory encryption [10], [8], [6], [17], insecure ECB mode of memory encryption [8], [17], unprotected nested page tables [20], [19], and unprotected I/O operations [17].

In light of these security issues, AMD has enhanced SEV with a sequence of microcode and hardware updates, most notably SEV with Encrypted State (SEV-ES) and SEV with Secure Nested Paging (SEV-SNP). SEV-ES encrypts the VMCB of a VM to protect register values at VMEXITs; SEV-ES processors are already commercially available. To address the most commonly exploited flaw—the lack of memory integrity for SEV VMs (including unauthenticated memory encryption and unprotected nested page tables), AMD plans to release SEV-SNP, which introduces a Reverse Map Table (RMP) to dictate ownership of the memory pages, so that the majority of the previously known attacks will be mitigated.

However, in this paper, we move our attention to another, yet-to-be-reported design flaw of SEV—the improper ASID-based memory isolation and access control. Specifically, SEV adopts an ASID-based access control for guest VMs' accesses to SEV processor's internal caches and the encrypted physical memory. At launch time, each SEV VM is assigned a unique ASID, which is used as the tag of cache lines and translation lookaside buffer (TLB) entries. A secure processor (dubbed AMD-SP) that is in charge of generating and maintaining the ephemeral memory encryption keys also uses the current VM's ASID to index the keys for encrypting/decrypting memory pages upon memory access requests. As such, the ASID of an SEV VM plays a critical role in controlling its accesses to the private data in the cache-memory hierarchy. Nevertheless, the assignment of ASID to a VM is under complete control of the hypervisor. An implicit "security-by-crash" security principle is adopted in the SEV design:

> *"Although the hypervisor has control over the ASID used to run a VM and select the encryption key, this is not considered a security concern since a loaded encryption key is meaningless unless the guest was already encrypted with that key. If the incorrect key is ever loaded or the wrong ASID is used for a guest, the first instruction fetch of that guest will fail as memory will be decrypted with the wrong key, causing junk data to be executed (and very likely causing a fault)."*[15]

The aim of this paper, therefore, is to investigate the validity of this "security-by-crash" design principle. To do so, we first study how ASIDs are used in SEV processors to isolate encrypted memory pages as well as CPU caches and TLBs. We also explore how ASIDs are managed by the hypervisor, how an ASID of a VM can be altered by the hypervisor at runtime, and why the VM with altered ASID crashes afterwards. This exploration leads to the discovery of several potential opportunities for a VM with an altered ASID to *momentarily* breach the ASID-based memory isolation before it crashes.

Next, based on our exploration, we then present CROSSLINE attacks[1], which exploit such a *momentary execution* to breach the confidentiality and integrity of SEV VMs. Specifically, an adversary controlling the hypervisor can launch an attacker VM and, during its VMEXIT, assign it with the same ASID as the victim VM, and then resume it, leading to the violation of the ASID-based access control to the victim's encrypted memory.

We mainly present two variants of CROSSLINE. In CROSSLINE V1, even though no instructions are executed by the attacker VM after VMRUN, we show that it is possible to load memory pages encrypted with the victim VM's memory encryption key (VEK) during page table walks, thus revealing the encrypted content of the "page table entries" (PTE) through nested page faults. This attack variant enables the adversary to extract the entire encrypted page table of the SEV guest VM, as well as any memory blocks conforming to the PTE format. We have also successfully demonstrated CROSSLINE V1 on SEV-ES machines, in which we devise techniques to bypass the integrity checks of launching the attacker VM with the victim VM's encrypted VMCB, while keeping the victim VM completely unaffected. In CROSSLINE V2, by carefully crafting its nested page tables, the attacker VM could manage to momentarily execute arbitrary instructions of the victim VM. By wisely selecting the target instructions, the adversary is able to construct encryption oracles and decryption oracles, which enable herself to breach both integrity and confidentiality of the victim VM. CROSSLINE V2 is confined by SEV-ES, but its capability is stronger than V1.

As extensions of the two attack variants, we also discuss (1) another variant of CROSSLINE, which allows the attacker VM to reuse the TLB entries of the victim VM for address translation and execute some instructions, even without any successful page table walks; and (2) the potential applicability of CROSSLINE on SEV-SNP.

Differences from known attacks. CROSSLINE differs from all previously demonstrated SEV attacks in several aspects. *First*, CROSSLINE does not rely on SEV's memory integrity flaws, which is a common pre-requisite for all known attacks on SEV. Although CROSSLINE may not work on SEV-SNP, the protection does not come from memory integrity, but a side-effect of the RMP implementation. *Second*, CROSSLINE attacks do not directly interact with the victim VMs and thus enable *stealthy* attacks. As long as the ephemeral encryption key of the victim VM is kept in the AMD-SP and the victim's encrypted memory pages are not deallocated, CROSSLINE attacks can be performed even when the victim VM is shutdown. Therefore, CROSSLINE is undetectable by the victim VM. In contrast, prior attacks relying on I/O operations of the victim VM [17], [8], [20], [19] are detectable by the victim VM.

CROSSLINE questions a fundamental security assumption of "security-by-crash" underpinning the design of SEV's memory and cache isolation. The demonstration of these attack variants suggests that SEV should not rely on adversary-controlled ASIDs to mediate access to the encrypted memory. To eliminate the threats, a principled solution is to maintain the identity of VMs in the hardware, which unfortunately requires some fundamental changes in the architecture. As far as we know, SEV-SNP will not integrate such changes.

Responsible disclosure. We have disclosed CROSSLINE attacks to AMD via emails in December 2019 and discussed the paper with AMD engineers by phone in January 2020. The demonstrated attacks and their novelty have been acknowledged. As discussed in the paper, neither of the two attack variants directly affect SEV-SNP. Therefore, AMD would not replace ASID-based isolation in the short term, but may invest more principled isolation mechanisms in the future.

Contributions. This paper makes the following contributions to the security of AMD SEV and other trusted execution environments.

- It investigates SEV's ASID-based memory, cache, and TLB isolation, and demystifies its "security-by-crash" design principle (§III). It raises security concerns of the "security-by-crash" based memory and TLB isolation for the first time.
- It presents two variants of CROSSLINE attacks—the only attacks that breach the confidentiality and integrity of an SEV VM without exploiting SEV's memory integrity flaws (§IV).
- It presents successful attacks against SEV and SEV-ES processors (§V). It also discusses the applicability of CROSSLINE on the upcoming SEV-SNP processors (§VI).

II. BACKGROUND

Secure Memory Encryption (SME). SME is AMD's x86 extension for real-time main memory encryption, which is supported in AMD CPU with Zen micro architecture from 2017 [24]. Aiming to defeat cold boot attack and DRAM interface snooping, an embedded Advanced Encryption Standard (AES) engine encrypts data when the processor writes to the DRAM and decrypts it when processor reads it. The entire DRAM is encrypted with a single ephemeral key which is randomly generated each time the machine is booted. A 32-bit ARM Cortex-A5 Secure Processor (AMD-SP) [21] is integrated in the system-on-chip (SOC) alongside the main processor, providing a dedicated security subsystem, storing, and managing the ephemeral key. Although all memory pages are encrypted by default, the operating system can mark some pages as unencrypted by clearing the *C-bit* (the 48th bit) of the corresponding page table entries (PTE). However, regardless of the C-bit, all code pages and page table pages are encrypted by default. With Transparent SME (TSME), a special mode of

[1] CROSSLINE refers to interference between telecommunication signals in adjacent circuits that causes signals to cross over each other.

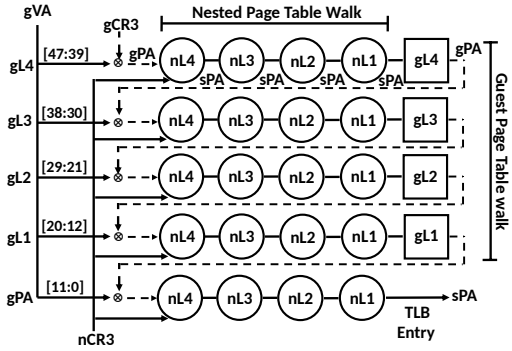

Figure 1: AMD-V nested page table walks [1].

operation of SME, the entire memory is encrypted, ignoring the C-bits of the PTEs.

AMD Virtualization (AMD-V). AMD-V is a set of extensions of AMD processors to support virtualization. Nested Page Tables (nPT) is introduced by AMD-V to facilitate address translation, which is officially marketed as Rapid Virtualization Indexing [1]. AMD-V's nPT provides two levels of address translation. When nPT is enabled, the guest VM and the hypervisor have their own CR3s: a guest CR3 (gCR3) and a nested CR3 (nCR3). The gCR3 contains the guest physical address of the gPT; the nCR3 contains the system physical address of the nPT. To translate a virtual address (gVA) used by the guest VM into the system physical address (sPA), the processor first references the guest page table (gPT) to obtain the guest physical address (gPA) of each page-table page. To translate the gPA of each page, an nPT walk is performed. During the nPT walk, the gPA is treated as host virtual address (hVA) and translated into the sPA using the nPT. These address translation steps are illustrated in Figure 1.

Translation lookaside buffers (TLB) and Page Walk Cache (PWC) are internal buffers in AMD processors for speeding up the address translation. AMD-V also relies on these internal buffers for performance improvements. AMD-V further introduces an nTLB for nPT. A successful nPT walk caches the translation from gPA to sPA in the nTLB for fast accesses [4], while the normal TLBs are used to store translations from virtual addresses of either the host or the guest to sPA.

To exchange data between the hypervisor and the guest VMs, a data structure dubbed the virtual machine control block (VMCB) is located on a shared memory page. VMCB stores the guest VM's register values and some control bits during VMEXIT. The VMCB is under the control of the hypervisor to configure the behaviors of the guest VM.

Secure Encrypted Virtualization (SEV). SEV combines AMD-V architecture with SME to allow individual VMs to have their own VM Encryption Key (VEK) [2]. Each VEK is generated by the processor and assigned to an SEV VM when launched by the hypervisor. All VEKs are stored in the AMD-SP and are never exposed to DRAM during their entire life cycle. SEV distinguishes different VEKs using ASIDs. When a memory request is made, the AMD-SP determines which key to be used with the current ASID. In combination with encryption modes specified in the guest page tables (gPT) and the nested page tables (nPT) [1], SEV achieves page-granular memory encryption with different keys.

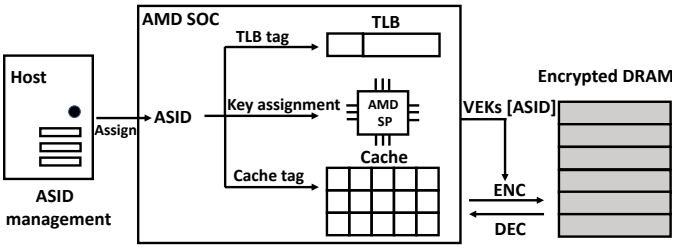

Figure 2: ASID-based memory isolation in SEV.

III. Demystifying ASID-based Isolation

ASID was initially designed by AMD to tag TLB entries so that unnecessary TLB flushes can be avoided when switching between guest VMs and the host. SEV reuses ASID as the indices of VEKs stored in AMD-SP. Cache tags are also extended accordingly to isolate cache lines with different ASIDs. As a result, ASID becomes the de-facto identifier used by SEV processors to control the software's accesses to virtual memory, caches, and TLBs (as shown in Figure 2).

However, following AMD-V, SEV allows the hypervisor to have (almost) complete authority over the management of ASIDs, which gives rise to security concerns as a malicious hypervisor may abuse this capability to breach ASID-based isolation. Interestingly, AMD adopts a "security-by-crash" and assumes if "*the wrong ASID is used for a guest*", the execution of the instruction will "*likely cause a fault*" [15]. In this section, we set off to understand and demystify how ASIDs are used to isolate memory, cache, and TLBs in SEV, and how ASIDs are managed by the hypervisor.

A. ASID-based Isolation

First, we explore in depth how ASID is used for access control in the virtual memory, CPU caches, and TLBs.

1) **ASID-based Memory Isolation:** ASIDs are used by the AMD-SP to index VEKs of SEV VMs. The SEV hardware ensures the data and code of an SEV VM is encrypted in the DRAM and only decrypted when loaded into the SOC. Specifically, each memory read from an SEV VM consists of memory fetches by the memory controller of a 128-bit aligned memory block, followed by an AES decryption by AMD-SP using the VEK corresponding to the current ASID. The current ASID is an integer stored in a hidden register of the current CPU core, which cannot be accessed by software in the guest VM.

SEV allows the guest OS to decide, by setting or clearing the C-bit of the PTE, whether each virtual memory page is (treated as) private (encrypted with the guest's VEK) or shared (either encrypted with the host's VEK or unencrypted). For instance, when the C-bit of a page is set, memory reads from this virtual-physical mapping is considered encrypted with the guest VM's VEK, regardless of its true encryption status, and thus a memory read in that page will be decrypted using the VEK of the current ASID.

However, the hypervisor is able to manipulate the nested C-bit (nC-bit) in nPT. When the gC-bit (the C-bit of the gPT)

Table I: Effects of gC-bits nC-bit. M is the plaintext; $E_k()$ is the encryption function under key k; k_g and k_h represent the guest VM and the hypervisor's VEK, respectively.

	nC-bit=0	**nC-bit=1**
gC-bit=0	M	$E_{k_h}(M)$
gC-bit=1	$E_{k_g}(M)$	$E_{k_g}(M)$

conflicts with the nC-bit, AMD-SP encrypts the memory pages according to rules specified in Table I: When gC-bit=0 and nC-bit=1, the page is encrypted with the hypervisor's VEK; when gC-bit=1, regardless of the nC-bit, the page is encrypted with the guest VM's VEK; when gC-bit=0 and nC-bit=0, the page is not encrypted. Following SME, the code pages are always considered private to the guest VM and thus is always encrypted regardless of the guest C-bits. Similarly, the gPT is also always encrypted with the guest's VEK, while the nPT is fully controlled by the hypervisor.

2) **ASID-based TLB Isolation:** ASID was originally introduced to avoid TLB flushes when the execution context switches between the guest VM and the hypervisor, which is achieved by extending each TLB tag with ASID. With the ASID capability, when observing activities like MOV-to-CR3, context switches, updates of `CR0.PG/CR4.PGE/CR4.PAE/CR4.PSE`, the hardware does not need to flush the entire TLB, but only the TLB entries tagged with the current ASID [4]. However, for the purpose of TLB isolation, the management of ASIDs for non-SEV VMs and SEV VMs is slightly different.

Non-SEV VMs. Each VCPU of a non-SEV VM may have different ASIDs, which can be assigned dynamically before each VMRUN. More specifically, before the hypervisor is about to resume a VCPU with VMRUN, it checks if the VCPU was the one running on this CPU core before the control was trapped into the hypervisor. If so, the hypervisor keeps the ASID of the VCPU unchanged and resumes the VCPU directly; if not, the hypervisor selects another ASID (from the ASID pool) and assign it to the VCPU. In the former case, TLB entries can be reused by the VCPU as its ASID is unchanged. However, in the latter case, the residual TLB entries (tagged with ASID of the hypervisor or the previous VCPU) should not be reused.

SEV VMs. SEV processors rely on a similar strategy to isolate entries in the TLBs with ASID. However, instead of dynamically assigning an ASID to a VCPU before VMRUN, all VCPUs of the same SEV VM are assigned the same ASID at launch time, which should in theory remain the same during the entire life cycle of the SEV VM.

3) **ASID-based Cache Isolation:** On platforms that support SEV, cache lines are tagged with the VM's ASID indicating to which VM this data belongs, thus preventing the data from being misused by entities other than its owner [15]. When data is loaded into cache lines, according to the current ASID, AMD-SP automatically decrypts the data with the corresponding VEK and stores the ASID value into the cache tag. When a cache line is flushed or evicted, AMD-SP uses the ASID in the cache tag to determine which VEK to use when encrypting this cache line before writing it back to DRAM. The cache tag is also extended to include the C-bit [15]. Because the cache is now tagged with ASID and C-bit, cache coherence of the same physical address is not maintained if the two virtual memory pages do not have the same ASID and C-bit.

B. ASID Management

1) **ASID Life Cycle:** The hypervisor reserves a pool (*i.e.*, a range of integers) of available ASIDs for all VMs (we call all-ASID pool for simplicity), and a separate pool of ASIDs for SEV VMs (SEV-ASID pool). The maximum ID number of the all-ASID pool is determined by CPUID 0x8000000a[EBX] (*e.g.*, 32768, thus the available ASIDs are whole numbers between 1 and 32768). The maximum ID number of the SEV-ASID pool is determined by CPUID 0x8000001f[ECX] (*e.g.*, 15, which suggests the legal ASIDs for SEV VMs are 1 to 15). Note that ASID 0 is reserved for the host OS (i.e., hypervisor), and is also not allowed to be assigned to a VCPU for processors with or without SEV extensions [4].

On SEV platforms, the hypervisor uses `ACTIVATE` command to inform AMD-SP that a given guest is bound with an ASID and uses `DEACTIVATE` command to de-allocate an ASID from the guest. The hypervisor may re-allocate an existing ASID to another VM, if there is no available ASID in the SEV-ASID pool [2].

At runtime, when the processor runs under the guest mode, the guest VM's ASID is stored in the ASID register that is hidden from software; when the processor runs under the host mode, the register is set to 0, which is the hypervisor's ASID. The guest VM's ASID is stored at the VMCB during VMEXIT. After VMRUN the processor restores the ASID in the VMCB. The VMCB State Cache allows the processor to cache some guest register values between VMEXIT and VMRUN for performance enhancement. The physical address of the VMCB is used to perform access control of the VMCB State Cache. However, the VMCB clean field controlled by the hypervisor can be used to force the processor to discard selected cached values. For example, bit-2 of the VMCB clean field indicates that an ASID reload is needed; bit-4 of the clean field indicates fields related to nest paging are dirty and needed to be reloaded from the VMCB. Some VMCB fields are strictly not cached and the corresponding register values will be reloaded from the VMCB every time. For example, offset 058h of the VMCB is a TLB control field to indicate whether the hardware needs to flush TLB after VMRUN; this field is always uncached.

2) **ASID Restrictions: Launch-time restrictions.** On processors supporting SEV, the hypervisor cannot bind a current active ASID in the SEV-ASID pool to an SEV VM during launch time [2]. However, an adversary is able to deactivate the victim SEV VM and then activate an attacker SEV VM with the same ASID. The hardware requires the hypervisor to execute a `WBINVD` instruction and a `DF_FLUSH` instruction after deactivating an ASID and before re-activating it. The `WBINVD` flushes all modified cache lines and invalidates all cache lines. The `DF_FLUSH` instruction flushes data fabric write buffers of all CPU cores. If these instructions are not executed before associating the ASID with a new VM, a `WBINVD_REQUIRED` or `DFFLUSH_REQUIRED` error will be returned by the AMD-SP and the VM launch process will be terminated.

This restriction is critical to the isolation of cache lines. Otherwise, victim VM's residual cache data can be read by subsequent attacker VM. In particular, the attacker VM can use the `WBINVD` instruction to flush the cache data to memory. Cache lines belonging to victim VM will thus be encrypted with the attacker VM's VEK and then flushed into the memory. Subsequent reads to those memory data will return plaintext and thus allow the adversary to extract the data.

Run-time restrictions. After a VM is launched, the hypervisor can change its ASID during VMEXITs, by changing the ASID field of its VMCB, which will take effect when the VM is resumed. There is no additional hardware restriction at runtime. As such, it is possible to have two SEV VMs concurrently with the same ASID on the same machine, though the one with a wrong ASID will crash very soon.

Moreover, the VMCB also contains a field (090h) to indicate if the VM is an SEV VM or a non-SEV VM. Therefore, it is possible to first launch an SEV VM and a non-SEV VM with the same ASID, and then, during VMEXITs of the non-SEV VM, change the non-SEV VM into an SEV VM by setting the corresponding bit in the VMCB. We have experimentally confirmed this possibility on our testbed (as shown in Section VI-A). It suggests that the hardware trusts the values of VMCB to determine (1) if the VM to be resumed is an SEV VM and (2) what ASID is associated with it. The hardware does not store this information to a secure memory region and use it for validation. The only additional validation performed by the AMD-SP is that the ASIDs of SEV VMs must fall into the valid ranges[2]. Therefore, while a VM was launched as a non-SEV VM, we can effectively (though momentarily) make it an SEV VM with the same ASID as another SEV VM.

3) **"Security-by-Crash":** As the hypervisor has the liberty of changing the ASIDs of both SEV VMs and non-SEV VMs, security concerns arise when the hypervisor is not considered a trusted party. However, it is believed that when an SEV VM is resumed with an ASID different from its own, its subsequent execution will lead to unpredictable results and eventually crash the VM.

Specifically, to change the ASID of a VM (either an SEV and non-SEV VM), the hypervisor can directly edit the ASID field of the VMCB, set the VMCB clean-field to inform the hardware to bypass the VMCB State Cache, and then resume the VM with VMRUN. After the VM is resumed, if the `RFLAGS.IF` bit in the VMCB is set, the virtual address specified by the interrupt descriptor-table register (IDTR) will be accessed, because the guest OS will try to handle interrupts immediately; if the `RFLAGS.IF` bit is cleared, the instruction pointed to by NRIP—the next sequential instruction pointer—is going to be fetched and executed. However, in either case, the virtual address translation will cause problems.

First, any TLB entries remaining due to its previous execution becomes invalid because its ASID has been changed; the ASID tag in the TLB entries would not match. Second, a page table walk is unlikely to succeed, as its own page tables are encrypted using the VEK indexed by its own ASID. As a result, the top-level page table will be decrypted into meaningless bit strings. References to a "page table entry" of this page will trigger an exception to be handled by the guest OS. Finally, a handler of the guest OS is to be invoked to handle the exception. However, any reference of this handler will be decrypted using a wrong VEK, leading to a *triple fault* that eventually crashes the VM.

[2]Specifically, the valid ASID range of SEV VMs are divided so that the lower values are for SEV-ES VMs. CPUID Fn8000_001F[ECX] specifies valid SEV ASIDs and CPUID Fn8000_001F[EDX] specifies the minimum ASID values used for SEV (but SEV-ES-disabled) VMs.

C. Summary

We highlight a few key points of SEV's "security-by-crash" based memory isolation mechanisms.

- **ASID is used for access control.** ASID is the only identifier used for controlling accesses to virtual memory, caches, and TLBs. Once a VM is successfully resumed from VMEXIT, the CPU and AMD-SP only rely on the ASID (loaded from its VMCB) to validate memory requests.
- **ASID is managed by the hypervisor.** The hypervisor may assign any ASID (including the ASID of another active SEV VM) to an SEV or non-SEV VM during VMEXIT. The only restriction enforced by the hardware is that the ASID must fall into the range in accordance to the VM's SEV type.
- **Security is achieved by VM crash.** The security of the mechanism relies solely on the faults triggered during the execution of the VM if its ASID has been changed. The faults can be caused by memory decryption with an improper VEK during instruction fetches or page table walks.
- **Cache/TLB entries are flushed by the hypervisor.** The hypervisor controls whether and when to flush TLB and cache entries associated with a specific ASID. Only limited constraints are enforced by the hardware during ASID activation. Misuse of these resources is possible.

IV. CrossLine Attacks

The goal of our CrossLine attacks is to extract the memory content of the victim VM that is encrypted with the victim VM's VEK. We make no assumption of the adversary's knowledge of the victim VM, including its kernel version, the applications running in it, *etc.* The common steps of the CrossLine attacks are the following: (1) the adversary who controls the hypervisor launches a carefully crafted attacker VM; (2) the hypervisor alters the ASID of the attacker VM to be the same as that of the victim VM during VMEXITs; (3) the hypervisor prepares a desired execution environment for the attacker VM by altering its VMCB and/or its nPT; (4) the attacker VM resumes after VMRUN, allowing a *momentary execution* before it crashes. During the momentary execution, memory accesses from the attacker VM will trigger memory decryption using the victim VM's VEK.

Although the attacker VM crashes shortly—due to the ASID-based isolation in TLB, caches, and memory—we show that this momentary execution, though very brief, already enables the attacker VM to impersonate the victim VM and breach its confidentiality and integrity. Note that the only requirement of the victim VM at the time of the attack is that

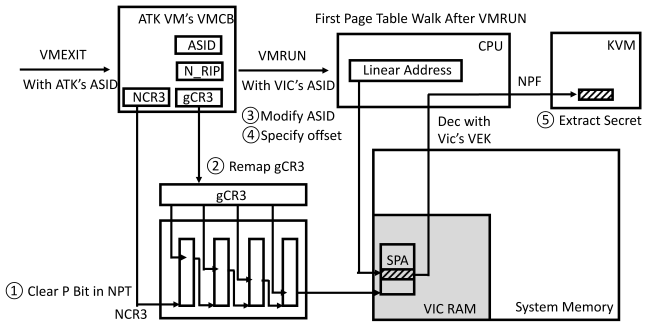

Figure 3: Workflow of CROSSLINE V1.

Algorithm 1: Determine NRIP when dumping one layer of page table (4096 bytes)

```
initialization;
while dumping the page do
    try to dump 8-byte memory block sPA_0 ;
    if sPA_0 % 0x1000 < 0x800 then
        NRIP = 0x8000000000* (sPA_0 % 0x1000 / 0x8);
    else
        NRIP = 0xffff000000000000 + 0x8000000000* (sPA_0 %
          0x1000 / 0x8);
    end
end
```

it has been launched and the targeted memory pages have been encrypted in the physical memory. Whether or not the victim VM is concurrently running during the attack is not important. Therefore, CROSSLINE is stealthy in that it does not interact with the victim VM at all. Detection of such attacks from the victim VM itself is unlikely.

A. Variant 1: Extracting Encrypted Memory through Page Table Walks

The CROSSLINE V1 explores the use of nested page table walks during the momentary execution to decrypt the victim VM's memory protected by SEV. To ease the description, let the victim VM's ASID be 1 and the attacker VM's ASID be 2. We use SPFN_0 to denote the system page frame number of the targeted memory page encrypted with the victim VM's VEK. We use sPA_0 to denote the system physical address of one 8-byte aligned memory block on SPFN_0, which is the target memory the adversary aims to read. The workflow of CROSSLINE V1 is shown in Figure 3. When the hypervisor handles a VMEXIT of the attacker VM, the following steps are executed:

① **Clear the Present bits.** The hypervisor alters the attacker VM's nPT to clear the Present bits of the PTEs of all memory pages. Thereafter, any memory access from the attacker VM after VMRUN will trigger a nested page fault, because the mapping from gPA to sPA in the NPT is missing.

② **Remap the current gCR3 of the attacker VM.** The hypervisor remaps the gCR3 of the current process in the attacker VM by altering the nPT. Now the gCR3 maps to SPFN_0. The hypervisor then sets the Present bit of this new mapping in the nPT.

③ **Modify the attacker VM's VMCB.** The hypervisor changes the attacker VM's ASID field in the VMCB to the victim VM's ASID (from 2 to 1 in this example).

④ **Specify the targeted page offset.** Before resuming the attacker VM with VMRUN, the hypervisor also modifies the value of NRIP in VMCB to specify which offset (*i.e.*, sPA_0) of the target page (*i.e.*, SPFN_0) to decrypt. Specifically, in a 64-bit Linux OS, bits 47 to 12 of a virtual address are used to index the page tables: bits 47-39 for the top-level page table; bits 38-30 for the second-level; bits 29-21 for the third; and bits 20-12 for the last-level page table. Each 4KB page in the page table has 512 entries (8 bytes each) and each entry contains the page frame number of the memory page of next-level page table or, in the case of the last-level page table, the page frame number of the target address. CROSSLINE V1 exploits the top-level page table walk to decrypt one 8-byte block each time. To control the offset of the 8-byte block within the page, the adversary modifies the value of NRIP stored in the VMCB so that its bit 47-39 can be used to index the top-level page table. The algorithm to choose NRIP properly is specified in Algorithm 1. Specifically, if the offset is less than 0x800, the NRIP is set to be in the range of 0x0000000000000000 - 0x00007fffffffffff (canonical virtual addresses of user space); if the offset is greater than or equal to 0x800, the NRIP is set to be in the range of 0xffff800000000000 - 0xffffffffffffffff (canonical virtual addresses of kernel space).

⑤ **Extract secrets from nested page faults.** After VMRUN, the resumed attacker VM immediately fetches the next instruction to be executed from the memory. This memory access is performed with ASID=1 (*i.e.*, the victim VM's ASID). The address translation is also performed with the same ASID. As the TLB does not hold valid entries for address translation, and thus an address translation starts with a page table walk from the gCR3, which maps to SPFN_0 in the nPT. Therefore, an 8-byte memory block on SPFN_0, whose offset is determined by bit 47-39 of the virtual address of the instruction, is loaded by the processor as if it is an entry of the page table directory. As long as the corresponding memory block follows the format of a PTE (to be described shortly), the data can be extracted and notified to the adversary as the faulting address (encoded in the EXITINFO2 field of VMCB).

1) **Dumping Victim Page Tables:** A direct security consequence of CROSSLINE V1 is to dump the victim VM's entire guest page table, which is deemed confidential as page-table pages are always encrypted in SEV VMs regardless of the C-bit in the PTEs.

To dump the page table, the adversary first locates the root of the victim VM's guest page table specified by its gCR3. She can do so by monitoring the victim VM's page access sequence using page-fault side channels. Specifically, during the victim VM's VMEXIT, the adversary clears the Present bit of all page entries of the nPT of the victim VM, evicts all the TLB entries, invalidates the nPT entries cached by nTLB and PWC. After VMRUN, the victim VM immediately performs a page table walk. The gPA of the first page to be accessed is stored in its gCR3. The adversary thus learns the gPA of the root of the guest page table. Once each of the entries of

the root page table is extracted by CROSSLINE V1, the rest of the page table can be decrypted one level after another.

Evaluation. We evaluated this page table dump attack using CROSSLINE on a blade server with an 8-Core AMD EPYC 7251 Processor. The host OS runs Ubuntu 64-bit 18.04 with Linux kernel v4.20 and the guest VMs run Ubuntu 64-bit 18.04 with Linux kernel v4.15 (SEV supported since v4.15). The QEMU version used was QEMU 2.12. The victim VMs were SEV-enabled VMs with 4 virtual CPUs, 4 GB DRAM and 30 GB disk storage. The attacker VMs were SEV-enabled VMs with only one virtual CPU, 2 GB DRAM and 30 GB disk storage. All the victim VMs were created by the ubuntu-18.04-desktop-amd64.iso image with no additional modification. The guest OS version does not matter in our attack.

To decrypt one 8-byte memory block, the adversary needs to launch the attacker VM, let it run until a VMEXIT, change its ASID, clear the Present bit of all PTEs. Roughly, it takes 2 seconds to decrypt one 8-byte memory block (which includes time to deactivate the ASID, reboot the VM, and clear the Present bit of all PTEs).

To speed up the memory decryption, the adversary could perform the following *VMCB rewinding attack*. Particularly, after extracting one 8-byte block through a VMEXIT caused by the nested page fault, the adversary could continue to decrypt the next 8-byte block without rebooting the attacker VM. To do so, the adversary directly repeats the attack steps by rewinding the VMCB of the attacker VM to the previous state and changing the NRIP to perform the next round of attack. With this approach, we found the average time to decrypt a 4KB memory page (with 1 attacker VM in 500 trials) was only 39.580ms (with one standard deviation of 4.26ms).

2) **Reading Arbitrary Memory Content:** Beyond page tables, the adversary could also extract regular memory pages of the victim VM. For example, if the data of an 8-bytes memory block is `0x00 0x00 0xf1 0x23 0x45 0x67 0x8e 0x7f`, the extracted data through page fault is `0x712345678`; if the data is `0x00 0x00 0x0a 0xbc 0xde 0xf1 0x20 0x01`, the extracted data is `0xabcdef12`. However, as CROSSLINE V1 only reveals the encrypted data as a page frame number embedded in the PTE, such memory decryption only works on 8-byte aligned memory blocks (*i.e.*, the begin address of the block is a multiple of 8 and the size of the block is also 8 bytes) that conforms to the format of a PTE.

Concretely, as shown in Figure 4, the 8-byte memory block to be extracted from CROSSLINE, must satisfy the following requirements: The Present bit (bit 0) must be 1; Bits 48-62 must be all 0s, and Bits 7-8 are both 0s (optional). This is because the Present bit must be 1 to trigger nested page fault. Otherwise, non-present faults in the guest VM will be handled without involving the hypervisor. Bits 48-62 are reserved and must be 0. The Page Size (PS) bit (bit-7) is used to determine the page size (*e.g.*, 4KB vs. 2MB); the Global Page (G) bit (bit-8) is used to indicate whether the corresponding page is a global page. These 2 bits can only be set 1 in the last level of the page table. Therefore, if CROSSLINE V1 generates page faults at the top-level page table, they must be set 0. However,

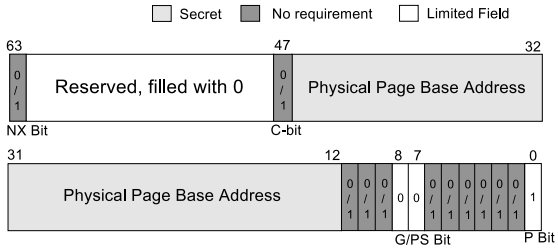

Figure 4: Valid PTE format.

we find it possible to configure the nPT so that the first three levels of the guest page table walk all pass successfully, and only trigger the nested page fault at the last-level page table. In this way, the target memory block can be regarded as a PTE of the last-level page table and hence these two bits are not restricted to be 0s.

Performance evaluation. The speed of memory decryption for arbitrary memory content is the same as dumping page tables, as long as the they are of PTE format. If the target block does not conform to the PTE format, a triple fault takes place instead of nested page fault, in which case the adversary could perform the VMCB rewinding attack and target another memory block in the next round of attacks.

Percentage of readable memory blocks. We studied the binary file of ten common applications, `python` 2.7, `OpenSSH` 7.6p1, `perl` 5.26.1, `VIM` 8.0.1453, `tcpdump` 4.9.3, `patch` 2.7.6, `grub-install` 2.02.2, `sensors` 3.4.0 , `Nginx` 1.14.0, and `diff` 3.6, which are installed from the default package archives in Ubuntu 18.04 (64-bit). The percentages of 8-byte aligned memory blocks that can be directly read using this method is 1.00%, 1.53%, 1.79%, 1.81%, 2.10%, 3.50%, 4.00%, 5.88%, 6.10%, and 6.50%. While they only account for a small portion of the whole memory space, they leak enough information for process fingerprinting purposes.

B. Variant 2: Executing Victim VM's Encrypted Instructions

In CROSSLINE V2, we show that, when certain conditions are met, it is possible for the attacker VM to momentarily execute a few instructions that are encrypted in the victim VM's memory. Apparently, CROSSLINE V2 is more powerful than the previous variant. Fortunately, the only prerequisite of CROSSLINE V2 is the consequence of CROSSLINE V1.

Similar to the settings in the previous attack variant, two SEV VMs were configured so that the ASID of the victim VM is 1 and the ASID of the attacker VM is 2. We assume that the attacker VM aims to execute one instruction—"`movl $2020, %r15d`"—in the victim VM's encrypted memory. Let the virtual address of this target instruction be gVA_0 and the corresponding gCR3 of the target process be $gCR3_0$. The adversary's strategy is to follow the common steps of CROSSLINE attacks and manipulate the nPT of the attacker VM so that it finishes a few nested page table walks to successfully execute this instruction. More specifically, CROSSLINE V2 can be performed in the following steps:

① **Prepare nPT.** The hypervisor clears the Present bit of all PTEs of the attacker VM's nPT. It also prepares valid mappings for the gVA_0 to the physical memory encrypted with

the victim's VEK. To do so, the hypervisor needs to prepare five gPA to sPA mappings (for the gPFNs of the four levels of the gPT and the instruction page), respectively.

② **Set nRIP.** The hypervisor sets nRIP as gVA_0. It also clears the Interrupt Flag of the RFLAGs register (RFLAGS.IF) in the VMCB, so that the attacker VM directly executes the next instruction specified by nRIP, instead of referring to Interrupt-Descriptor-Table Register.

③ **Change ASID.** The hypervisor changes the attacker VM's ASID to the victim's ASID, marks the VMCB as dirty, and resumes the attacker VM with VMRUN. During the next VMEXIT, the value of $\%r15$ has been changed to \$2020, which means the attacker VM has successfully executed an instruction that is encrypted with the victim's VEK.

These experiments suggest that CROSSLINE allows the attacker VM to execute some instruction of the victim VM. We exploit this capability to construct decryption oracles and encryption oracles.

1) **Constructing Decryption Oracles:** A decryption oracle allows the adversary to decrypt an arbitrary memory block encrypted with the victim's VEK. With CROSSLINE V2, the attacker VM executes one instruction of the victim VM to decrypt the target memory.

The first step of constructing a decryption oracle is to locate an instruction in the victim VM with the format of "`mov (%`reg_1`),%`reg_2", which loads an 8-byte memory block whose virtual address is specified in reg_1 to register reg_2. As most memory load instructions follow this format, the availability of such an instruction is not an issue. The adversary can leverage CROSSLINE V1 to scan the physical memory of the victim VM, in the hope that the readable memory blocks contain such a 3-byte instruction. Alternatively, if the kernel version of the victim VM is known, the adversary can scan the binary file of the kernel image to locate this instruction and then obtain its runtime location by reading the gPT, which can be completely extracted by CROSSLINE V1.

Let the virtual address of this instruction be gVA_0, its corresponding system physical address be sPA_0, and the gCR3 value of the process in the victim VM be $gCR3_0$. The virtual address and the system physical address of the target memory address to be decrypted are gVA_1 and sPA_1. Note since the adversary is able to extract the gPT of the victim, the corresponding translation for gVA_0 and gVA_1 can be obtained. Then following the three steps outlined above, during a VMEXIT of the attacker VM, the adversary prepares the nPT of the attacker VM (including one mapping for $gCR3_0$, four mappings for gVA_0, and four mappings for gVA_1), configures the VMCB (including nRIP, ASID, the value of $\%reg_1$), and then resumes the attacker VM.

In the next VMEXIT, the adversary is able to extract the secret stored in sPA_1 by checking the value of $\%reg_2$. The adversary can immediately perform the next round of memory decryption. The system physical page frame number can be manipulated in the last-level nPT and the page offset can be controlled in $\%reg_1$.

Performance evaluation. We measured the performance of the decryption oracle described above for decrypting a 4KB memory page. With only one attacker VM, the average decryption time (of 5 trials) for a 4KB page was 113.6ms with one standard deviation of 4.3ms. Note the decryption speed is slower than the optimized version of CROSSLINE V1, but the decryption oracle constructed with CROSSLINE V2 is more powerful as it is not limited by the format of the target memory block.

2) **Constructing Encryption Oracles:** An encryption oracle allows the adversary to alter the content of an arbitrary memory block encrypted with the victim's VEK to the value specified by the adversary. With CROSSLINE V2, an encryption oracle can be created in ways similar to the decryption oracle. The primary difference is that the target instruction is of the format "`mov %`reg_1`,(%`reg_2`)`", which moves an 8-byte value stored in reg_1 to the memory location specified by reg_2.

With an encryption oracle, the adversary could breach the integrity of the victim VM and force the victim VM to (1) execute arbitrary instruction, or (2) alter sensitive data, or (3) change control flows. Note that our encryption oracle differs from those in the prior works [8], [6], [17] as it does not rely on SEV's memory integrity flaws.

Performance evaluation. We measured the performance of the encryption oracle by the time it takes to updates the content of a 4KB memory page. The average time of 5 trials was 104.8ms with one standard deviation of 6.1ms. Note in a real-world attack, the attacker may only need to change a few bytes to compromise the victim VM, which means the attack can be done within 1ms.

3) **Locating Decryption/Encryption Instructions:** In the previous experiments, we have already shown that once the instructions to perform decryption and encryption can be located, the construction of decryption and encryption oracles is effective and efficient. Next, we show how to locate such decryption/encryption instructions to bridge the gap towards an end-to-end attack.

Specifically, on the victim VM, an OpenSSH server (SSH-2.0-OpenSSH-7.6p1 Ubuntu-4ubuntu0.1) is pre-installed. *First*, the adversary learns the version of the OpenSSH binary by monitoring the SSH handshake protocol. More specifically, the adversary who controls the hypervisor and host OS monitors the incoming network packets to the victim VM to identify the SSH `client_hello` message. The victim VM would immediately respond with an SSH `server_hello` message, which contains the version information of the OpenSSH server. As these messages are not encrypted, the adversary could leverage this information to search encryption/decryption instructions offline from a local copy of the binary.

Second, the adversary extracts the gCR3 of the *sshd* process. To do so, upon observing the `server_hello` message, the adversary immediately clears the Present bits of all PTEs of the victim VM. The next memory access from the *sshd* process will trigger an NPF VMEXIT, which reveals the value of gCR3. We empirically validated that this approach allows the adversary to correctly capture *sshd*'s gCR3, by repeating the above steps 50 times and observing correct gCR3 extraction every time.

Third, the adversary uses CROSSLINE V1 to dump a portion of the page tables of *sshd* process. More specifically, the

adversary first dumps the 4KB top-level page-table page pointed to by gCR3; she identifies the smallest offset of this page that represents a valid PTE, and then follow this PTE to dump the second-level page-table page. The adversary repeats this step to dump all four levels of page tables for the lowest range of the virtual address. In this way, the adversary could obtain the physical address corresponding to the base virtual address of the OpenSSH binary.

Fourth, with the knowledge of the memory layout of the code section of the OpenSSH binary, the adversary can calculate the physical address of the decryption/encryption instructions within the OpenSSH binary. In our demonstrated attack, the adversary targets two instructions inside the `error` function of OpenSSH, "`mov (%rbx),%rax`" for decryption and "`mov %rax,(%r12)`" for encryption. The offsets of the two instructions are `0xca9a` and `0xca18`, respectively.

Performance evaluation. We measured the time needed to locate these two instructions. Once the adversary has intercepted the SSH handshake messages, it takes on average 504.74ms (over 5 trials) to locate these two instructions.

C. Discussion on Stealthiness and Robustness

CROSSLINE attacks are stealthy. The attacker VM and the victim VM are two separate VMs. They have different NPTs and VMCBs. Therefore, any state changes made in the attacker VM are not observable by the victim. As such, it is impossible for victim VM to sense the presence of the attacker VM. In contrast to all known attacks to SEV, CROSSLINE cannot be detected by running a detector in the victim VM. More interestingly, the adversary can rewind the attacker VM's VMCB to eliminate the side effects caused by the attacker VM's attack behaviors (*e.g.*, triggering a NPF with non-PTE format or executing an illegal instruction). This method also increases the robustness of the attack: Even if the encryption/decryption instructions are not correctly located, CROSSLINE V2 will not affect the execution of the victim VM. Therefore, the adversary can perform the attack multiple times until succeeds.

V. APPLICABILITY TO SEV-ES

A. Overview of SEV-ES

To protect VMCB during VMEXIT, SEV-ES was later introduced by AMD [13]. With SEV-ES, a portion of the VMCB is encrypted with authentication. Therefore, the hypervisor can no longer read or modify arbitrary register values during VMEXITs. To exchange data between the guest VM and the hypervisor, a new structure called Guest Hypervisor Control Block (GHCB) is shared between the two. The guest VM is allowed to indicate what information to be shared through GHCB.

VMEXITs under SEV-ES modes are categorized into Automatic Exits (AE) and Non-Automatic Exits (NAE). AE VMEXITs (*e.g.*, those triggered by most nested page faults, by the `PAUSE` instruction, or by physical and virtual interrupts) are VMEXITs, which do not need to expose register values to the hypervisor. Therefore, AE VMEXITs directly trigger a VMEXIT to trap into the hypervisor. To enhance security, NAEs (*e.g.*, those triggered by `CPUID`, `RDTSC`, `MSR_PROT` instructions) are first emulated by the guest VM instead of the hypervisor. Specifically, NAEs first trigger #VC exceptions, which are handled by the guest OS to determine which register values need to be copied into the GHCB. This NAE VMEXIT will then be handled by hypervisor that extracts the register values from the GHCB. After the hypervisor resuming the guest in VMRUN, the #VC handler inside the guest OS reads the results from the GHCB and copies the relevant register states to corresponding registers.

SEV-ES VMs can run concurrently with SEV VMs and non-SEV VMs. After VMEXIT, the hardware recognizes an SEV-ES VM by the SEV control bits (bit-2 and bit-1 of `090`h) in the VMCB [4]. Therefore, the hypervisor may change the SEV type (from an SEV VM to an SEV-ES VM) during VMEXIT. The legal ASID ranges of SEV-ES and SEV VMs, however, are disjoint, and thus it is not possible to run an SEV-ES VM with an ASID in the range of SEV VMs.

1) **VMCB's Integrity Protection:** With SEV-ES, the VMCB is divided into two separate sections, namely the control area and the state save area (VMSA) [4]. The control area is unencrypted and controlled by the hypervisor, which contains the bits to be intercepted by the hypervisor, the guest ASID (058h), control bits of SEV and SEV-ES (090h), TLB control (058h), VMCB clean bits (`0C0`h), NRIP (`0C8`h), the gPA of GHCB (`0A0`h), the nCR3 (`0B0`h), VMCB save state pointer (`108`h), *etc.* The state save area is encrypted and integrity protected, which contains the saved register values of the guest VM. The VMCB save state pointer stores the system physical address of VMSA, the encrypted memory page storing the state save area.

The integrity-check value of the state save area is stored in the protected DRAM, which cannot be accessed by any software, including the hypervisor [4]. At VMRUN, the processor performs an integrity check of the VMSA. If the integrity check fails, VMRUN terminates with errors [4]. Because the integrity-check value (or the physical address storing the value) is not specified by the hypervisor at VMRUN, we conjecture the value is index by the system physical address of the VMSA. Therefore, a parked virtual CPU is uniquely identified by the VMSA physical address.

B. CROSSLINE V1 *on SEV-ES*

The primary challenge to apply CROSSLINE on SEV-ES machines is to bypass the VMSA check. Directly resuming the attacker VM using the victim's ASID would cause VMRUN to fail immediately, because the VMSA integrity check takes place before fetching any instructions in the attacker VM. Since the attacker VM's VMSA is encrypted using the VEK of the attacker VM, when resuming the attacker VM with the victim's ASID, the decryption of VMSA leads to garbage data, crashing the attacker VM immediately.

Therefore, to perform CROSSLINE V1, the adversary must change the save state pointer (0108h) of the attacker VM's VMCB so that the attacker VM will reuse the victim VM's VMSA. As such, the attacker VM cannot change the register values that are stored in the VMSA, which includes RIP,

gCR3, and all general-purpose registers (if not exposed in the GHCB). Therefore, with SEV-ES, the adversary is no longer able to arbitrarily control the execution of the attacker VM by simply manipulating its nRIP in its VMCB's control area [4].

However, by pausing victim's VCPU during VMEXIT and changing attacker's VMSA pointer (0108h) to victim's VMSA, the adversary is still able to perform CrossLine V1 on SEV-ES VMs to achieve the same goal—extracting the entire gPT or decrypting any 8-byte memory block conforming to a PTE format. To show this, we have performed the following experiments:

Two SEV-ES VMs were launched. The ASID of the victim VM is set to be 1 and that of the attacker VM is 2. The hypervisor pauses the victim VM at one of its VMEXIT, so that its VMSA is not used by itself. The attack is performed in the following steps:

① **Prepare nPT.** During the VMEXIT of the attacker VM, the hypervisor clears the Present bits of the all PTEs in the attacker VM's nPT.

② **Manipulate the attacker VM's VMCB.** The hypervisor first changes the attacker VM's ASID from 2 to 1. It also informs the hardware to flush all TLB entries of the current CPU, by setting the TLB clearing field (058h) in the VMCB control area. Finally, it changes the VMCB save area pointer to point to the victim's VMSA.

③ **Resume the attacker VM.** Because the attacker VM runs with the victim's ASID, the victim's VMSA is decrypted correctly. The integrity check also passes, as no change is made in the VMSA, including its system physical address. Once resumed, the attacker VM will try to fetch the first instruction determined by RIP (in VMSA) or the IDTR using the victim's VEK. Since there is no valid TLB entry, the processor has to perform a guest page table walk to translate the virtual address to the system physical address. A nested page fault can be observed with the faulting address being the victim VM's gCR3 value.

④ **Remap gCR3 in nPT.** When handling this NPF VMEXIT, the hypervisor remaps the gCR3 in the nPT to the victim VM's memory page to be decrypted. The Present bits of the corresponding nested PTEs are set to avoid another NPF of this translation. Moreover, the `EXITINTINFO` field in the unencrypted VMCB control area needs to be cleared to make sure the attacker VM complete the page table walk. After resuming the attacker VM, an NPF for the translation of another gPA (embedded in the target memory block) will occur, which reveals the content of the 8-byte aligned memory block if it follows the format of a PTE.

⑤ **Reuse the VMSA.** The hypervisor repeats step ④ so that its gCR3 is remapped to the next page to be decrypted in the victim VM. Then, the next NPF VMEXIT reveals the corresponding memory block. This could work because the attacker VM has not successfully fetched a single instruction yet; it is trapped in the first page table walk (more specifically, the first nested page table walk of the first gPA). Therefore, the VMSA is not updated and no valid TLB entry is created. During the remapping of gCR3, the hypervisor is able to invalidate the previously generated entry in the nTLB. Thus, from the perspective of the attacker VM, step ④ does not change its state. Therefore, the attacks can be carried out repeatedly.

⑥ **Handling triple faults.** In step ④ or ⑤, if the targeted 8-byte memory block does not conform to the PTE format, a triple fault VMEXIT (error code 0x7f) will be triggered instead of the NPF VMEXIT. The adversary can continue to decrypt the next page if this happens. However, after a triple fault, the RIP in the VMSA has been updated to the fault handler to deal with the fault. As such, resuming from a triple fault will lead to the decryption of a different offset of the target page. However, the attack can still continue.

Resuming the victim VM. After performing CrossLine V1, the VMSA of the victim VM is still usable by the victim. We empirically validated this by resuming the victim VM after the attacker VM has used this VMSA to decrypt several memory blocks and has encountered both nested page faults and triple faults. The victim VM was resumed successfully, without observing any faults or abnormal kernel logs.

To better understand the victim VM's state changes when its VMSA is used by the attacker VM, we instrumented the hypervisor to check which regions of the encrypted VMSA have been changed after the attacker VM has performed several rounds of CrossLine V1, which triggers both nested page faults and triple faults. The result shows that the entire VMSA remains the same, except the value of CR2, which stores the most recent faulting address. The change of the CR2 value does not affect the execution of the victim VM as this value is not used by the guest OS after NPFs.

Controlling page offsets. Because the integrity protection of VMSA prevents the adversary from controlling the RIP after VMRUN, the page offset of the memory blocks to be decrypted cannot be controlled in CrossLine V1. However, the adversary may resume the victim VM and allow it to run till a different RIP is encountered. In total, 512 different RIPs are needed to decrypt any memory blocks conforming to the PTE formats. To diversify the exploited RIPs, one strategy is to pause the victim when the VMEXIT is a NPF-triggered AE. When VMEXITs are NAEs or interrupt-triggered AEs, the next instruction to be executed after VMRUN is an instruction of the #VC handler, whose virtual address is fixed in the kernel address space. To differentiate NPF-triggered AEs and interrupt-triggered AEs, although the adversary cannot read the RFLAG.IF directly, which indicates pending interrupts, she can inspect Bit 8 (V_IRQ) of the Virtual Interrupt Control field (offset 60h) in the unencrypted VMCB control area. Moreover, as two consecutive NPF-triggered AEs may be caused by the same RIP, it is preferred to pause the victim VM after a few AEs. To trigger more NPF VMEXITs, one could periodically unset the Present bit of all PTEs of the victim VM.

With these strategies in place, we empirically evaluated the time needed for the adversary to find all 512 offsets. In our test, we let the victim VM run a build-in program of Ubuntu Linux, called "cryptsetup benchmark". The attack can be performed on any level of the page tables; bits 47-39, 38-30, 29-21, and 20-12 of the same RIP can all be used as the page offset by the attacker. Therefore, with any RIP, there

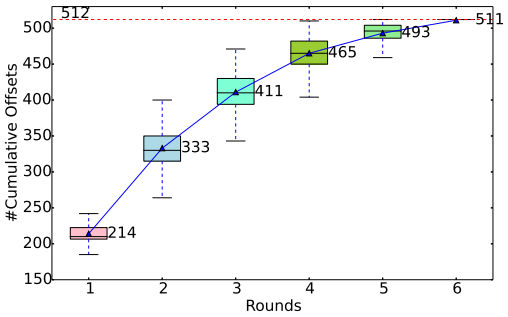

Figure 5: Covered offsets after N rounds.

are 1∼4 different offsets that the attacker may use to extract data on any encrypted page. The experiments were performed in the following manner: Each round of the experiments, the cryptsetup benchmark were run several times and each time with a different address space layout due to ASLR; every 30 seconds, the adversary unset all Present bits of the victim VM to trigger NPFs; the adversary pauses the victim VM every 13 AE VMEXITs to extract one RIP. The adversary concludes the round of monitoring after 60 seconds. In total, 15 rounds of experiments were conducted. Figure 5 shows the number of offsets that can be covered after N rounds of experiments, where $N = 1$ to 6. Each data point is calculated over all combinations of selecting N rounds from the 15 rounds, *i.e.*, $C(15, N)$, of data collected in the experiments above. Specifically, on average, after 5 rounds of experiments, the adversary could obtain 493 offsets; after 6 rounds, she could obtain 511 offsets (out of the 512 offsets). These experiments show that when the victims run an application that has diverse RIPs (*i.e.*, not running in idle loops), the adversary has a good chance of performing CROSSLINE V1 on almost all page offsets after some efforts (in these experiments, after 6 minutes of the victim's execution).

Performance evaluation. We have evaluated the attack mentioned above on a workstation with an 8-Core AMD EPYC 7251 Processor. The motherboard of our testbed machine was GIGABYTE MZ31-AR0, with which we successfully configured Fn8000_001F[EDX] to return 5, which means ASID 1 to 4 were reserved for SEV-ES VMs. Since the source code supporting SEV-ES for both host OS and guest OS has not been added into the mainstream Linux kernel yet, we used the source code provided in the SEV-ES branch of AMD's official repositories for SEV, which is available on Github [5]. The kernel version for the host and guest were branch sev-es-5.1-v9. The QEMU version used was QEMU sev-es-v4 and the OVMF version was sev-es-v11. Both victim VMs and attacker VMs were configured as SEV-ES-enabled VMs with 1 virtual CPU, 2 GB DRAM and 30 GB disk storage. All VMs were created by the kernel image generated from sev-es-5.1-v9 branch without any additional modification.

On average over 200 trials, it takes 2.0ms to decrypt one 8-byte memory block, which is slower than the attack against SEV VMs (0.077ms per block). This is because the AMD-SP must calculate the hash of the VMSA and store it to the secure memory region during VMEXITs, and validate its integrity after each VMRUN. This happens in between of decrypting two memory blocks.

C. CROSSLINE V2 *on SEV-ES*

Applying CROSSLINE V2 on SEV-ES would be challenging, because with the encrypted VMCB, RIP is no longer controlled by the adversary. As such, the attacker VM will resume from the RIP stored in the VMSA, which prevents the attacker VM from executing arbitrary instructions. Moreover, constructing useful encryption or decryption oracles requires the manipulation of specific register values, which is only possible without SEV-ES.

D. *Discussion on Stealthiness*

Unlike CROSSLINE on SEV, to attack SEV-ES machines, the attacker VM must reuse the victim VM's VMSA. However, CROSSLINE V1 is still stealthy and undetectable by the victim VM for two reasons. First, the attack only alters the CR2 field of the victim's VMSA. As this field is not examined by the guest OS after resumption from a NPF, the victim VM cannot detect the anomaly. Second, even if the guest OS is modified, the change of the CR2 cannot be detected, because the AE NPFs are directly trapped into the hypervisor, such that the guest OS does not have a chance to record the original value of CR2 to be compared with.

VI. DISCUSSION

A. *A New Variant: Reusing Victim's TLB Entries*

First, we discuss a proof-of-concept attack that extends the other two variants. In particular, in this attack, we show that the ASID-based TLB isolation can be breached. There were two VMs involved: the victim VM is an SEV VM whose ASID is 1; the attacker VM is a non-SEV VM whose ASID is 16. Both VMs only have one VCPU, which are configured by the hypervisor to run on the same logical CPU core. We assume the victim VM executes the following code snippet:

```
d83:41 bb e4 07 00 00    mov     $0x7e4,%r11d
d89:41 bc e4 07 00 00    mov     $0x7e4,%r12d
d8f:0f a2                cpuid
d91:eb f0                jmp     d83
```

Specifically, the code updates the values of `%r11d` and `%r12d`, and then executes a `CPUID` to trigger a VMEXIT. Following the common steps of CROSSLINE, the adversary launches an attacker VM, changes its ASID during VMEXIT, sets the nRIP of the attacker VM to the virtual address of the code snippet above, changes offset 090h of VMCB to make it an SEV VM, and resumes the attacker VM. Unlike CROSSLINE V1 and CROSSLINE V2, the nPT of the attacker VM is not changed in this step. Therefore, if the attacker VM performs a page table walk, a NPF will be triggered.

Interestingly, the execution of the attacker VM triggers `CPUID` VMEXITs before a triple fault VMEXIT crashes it. Since no NPF is observed, the attacker VM apparently does not perform any page table walk. However, during the attacker VM's `CPUID` VMEXITs, we observe that the values of `%r11d` and `%r12d` have been successfully changed to `$0x7e4`. It is clear that the two MOV instructions and the

subsequent CPUID instruction have been executed by the attacker VM. This is because the attacker VM was able to use the victim VM's TLB entries left in the TLB to translate the virtual address of the instructions. The triple fault might be caused by code executed outside the page, whose translation is not cached in the TLB.

While the consequences of this attack are close to V2, it highlights the following flaws in AMD's TLB isolation between guest VMs: (1) ASIDs serve as the only identifier for access controls to TLBs, which can be *forged* by the hypervisor, and (2) TLBs cleansing during VM context switch is performed at the discretion of the hypervisor, which may be *omitted* intentionally.

B. Applicability to SEV-SNP

To address the attacks against SEV that exploit memory integrity flaws, AMD recently announced SEV-SNP [14] and released a whitepaper describing its high-level functionality in January, 2020 [3]. The key idea of SEV-SNP is to provide memory integrity protection using a Reverse Map Table (RMP). An RMP is a table indexed by system page frame numbers. One RMP is maintained for the entire system. Each system page frame has one entry in the RMP, which stores information of the page state (*e.g.*, hypervisor, guest-invalid, guest-valid) and ownership (*i.e.*, the VM's ASID and the corresponding gPA) of the physical page. The ownership of a physical page is established through a new instruction, PVALIDATE, which can only be executed by the guest VM. Therefore, the guest VM can guarantee that each guest physical page is only mapped to one system physical page; by construction, RMP allows each system physical page to have only one validated owner.

After each nested page table walks that leads to a system physical page belonging to an SEV-SNP VM (and also some other cases), an RMP check is to be performed. The RMP check compares the owner of the page (*i.e.*, the ASID) with the current ASID and compares the recorded gPA in the RMP entry with the gPA of the current nPT walk. If a mismatch is detected, a nested page fault will be triggered.

- **CROSSLINE V1 on SEV-SNP.** When applying CROSSLINE V1 on SEV-SNP by following the same attack steps for SEV-ES, it seems step ① to ④ would work the same. As the VMSA is also protected by the RMP, loading VMSA would lead to an RMP check. However, as the attacker VM uses the victim's ASID, the check would pass. However, the NPF in step ⑤ that reveals the page content would not occur. Instead, an NPF due to RMP check would take place, because the gPA used in nPT walk is different from the one stored in the RMP entry. Therefore, from the description of the RMP, it seems CROSSLINE V1 can be prevented.
- **CROSSLINE V2 on SEV-SNP.** As CROSSLINE V2 does not work on SEV-ES, it cannot be applied on SEV-SNP.

Nevertheless, SEV-SNP is still in its planning phase. Some implementation details are still unclear[3]. For instance, in our discussion with AMD engineers, AMD is developing technologies to better isolate TLBs [4], which will thwart the attack variant we discuss in Section VI-A. But it is not yet clear when the technology can be officially announced and implemented on SEV-SNP processors.

C. Intel MKTME

Similar to AMD's SEV, Intel's Total Memory Encryption (TME) and Multi-Key Total Memory Encryption (MK-TME) [11] also provide memory encryption to software. The concept of TME is similar to AMD SME: a memory encryption engine is placed between the direct data path and external memory buses, which encrypts data entering or leaving the SOC using 128-bit AES encryption in the XTS mode of operation. MKTME is built atop TME and supports multiple encryption keys. When used in virtualization scenarios, MKTME is close to AMD's SEV. However, different from SEV, where each VM only possesses one encryption key, multiple keys can be used in each VM on an MKTME platform, allowing cross-VM memory sharing when the same keys are used. The selection of encryption keys is controlled by software, by specifying the key id in the upper bits of a page table entry (PTE). In a virtualization scenario, the hypervisor has to be trusted because it has the capability of mapping guest VM's memory and controlling the memory encryption keys in the PTE. CROSSLINE is not needed in the MKTME setting as a malicious hypervisor may directly read encrypted guest memory. Therefore, the hypervisor is included in the TCB of MKTME, which could greatly limit its real-world adoption.

D. Relation to Speculative Execution Attacks

CROSSLINE is *not* a speculative execution attack. Meltdown [18], Spectre [16], L1TF [25], and MDS [26], [22], [7] are prominent speculative execution attacks that exploit transiently executed instructions to extract secret memory data through side channels. In these attacks, instructions are speculatively executed while the processor awaits resolution of branch targets, detection of exceptions, disambiguation of load/store addresses, *etc.*. However, in the settings of CROSSLINE V1, no instructions are executed, as the exceptions take place as soon as the frontend starts to fetch instructions from the memory. The other two variants of CROSSLINE execute instructions with architecture-visible effects.

CROSSLINE does not rely on micro-architectural side channels, either. Speculative execution attacks leverage micro-architectural side channels (*e.g.*, cache side channels) to leak secret information to the program controlled by the attacker. In contrast, CROSSLINE reveals data from the victim VM as page frame numbers, which can be learned by the hypervisor directly during page fault handling.

[3] "This white paper is a technical explanation of what the discussed technology has been designed to accomplish. The actual technology or feature(s) in the resultant products may differ or may not meet these aspirations. Each description of the technology must be interpreted as a goal that AMD strived to achieve and not interpreted to mean that any such performance is guaranteed to be fully achieved."[3].

VII. Related Work

Past work mainly studied the insecurity of AMD SEV from the following aspects.

Unencrypted VMCB. Before SEV-ES, the VMCB is not encrypted at the time of VMEXIT. Hetzelt and Buhren [10] first reported that an adversary who controls the hypervisor could directly observe the machine states of the guest VM by reading the VMCB structure. Moreover, they show that the adversary could also manipulate the register values in the VMCB before resuming the guest VM to perform return-oriented programming (ROP) attacks [23] against the guest VM. As a result, the adversary is able to read or write arbitrary memory in the SEV VM. These security issues have been completely mitigated by SEV-ES [13].

Werner *et al.* also explored security vulnerabilities caused by unencrypted VMCB [27]. Their study suggests that an adversary is able to identify applications running inside the SEV VMs by recording register values in VMCB. The study also shows that it is practical to inject data by locating certain system calls and modify some registers to mislead the guest VM. However, SEV-ES restricts most of their attacks and the only working attack that remains is application fingerprinting.

Unauthenticated encryption. The lack of authentication in the memory encryption is one major drawback of the SME design, which has been demonstrated in fault injection attacks [6]. SEV inherits this security issue. Therefore, a malicious hypervisor may alter the ciphertext of the encrypted memory without triggering faults in the guest VM. Another problem with SME's memory encryption design is that SME uses Electronic Codebook (ECB) mode of operation in its AES-based memory encryption. In such a scheme, the plaintext of a 16-byte memory block is first XORed with the output of a *tweak function*, which takes as input the system physical address of the memory block and deterministically generates a 16-byte long bit string. The outcome of the XOR operation is then encrypted with the memory encryption key to produce the 16-byte ciphertext.

This design choice unfortunately has enabled a chosen plaintext attacks. Du *et al.* [8] reverse-engineered the tweak function and recovered the mapping between the system physical address and the output of the tweak functions. With this knowledge, an adversary can relate the plaintext of any two 16-byte memory blocks if they have the same ciphertext. A chosen plaintext attack can be conducted if the adversary is able to force the victim VM to encrypt some plaintext blocks chosen by the adversary (*e.g.*, via HTTP requests): The adversary first identifies the corresponding ciphertext of the chosen plaintext (*e.g.*, by recognizing repetitive patterns) and then replaces with it (after applying the corresponding tweak functions) some critical instructions of the victim VM (*e.g.*, `sshd`) [8].

Wilke *et al.* [28] studied the Xor-Encrypt-Xor (XEX) mode of memory encryption of AMD's Epyc 3xx1 series processors, where the tweak function XOR with the plaintext twice, both before and after the encryption. However, in the Epyc 3xx1 processor that was studied by the authors, the entropy of the tweak functions is only 32 bits, making brute-force attacks practical. It is demonstrated that the adversary who breaks the tweak function can insert some arbitrary 2-byte instruction into encrypted memory with the help of 8MB plaintext-ciphertext pairs. The vulnerability is also caused by the lack of authentication in the memory encryption. Fortunately, the XEX tweak function vulnerability exploited in the paper was fixed in Zen 2 architecture that was released in May, 2019. Therefore, later AMD processors are not affected by this attack.

Unprotected nPT. Hetzelt and Buhren [10] demonstrated address translation redirection attacks (an idea first explored by Jang *et al.* in the context of hardware-based external monitors [12]) in SEV and discussed remapping guest pages in the nPT to replay previously captured memory pages. This idea was later realized by SEVered [20], [19], which manipulates the nPT to breach the confidentiality of the memory encryption. More specifically, in the SEVered attack, the hypervisor triggers activities of the victim VM's network-facing application and concurrently monitor its accesses to the encrypted memory using a page-level side channel. In this way, the hypervisor can determine the system physical page used to store the response data. Then, by changing the memory mapping in the nPT, the hypervisor tricks the guest VM to respond to network requests from the target page, leaking secrets to the adversary.

Unprotected I/O. Li *et al.* [17] exploited unprotected I/O operations to construct encryption and decryption oracles that encrypts and decrypts arbitrary memory with the victim's VEK. As SEV's IOMMU hardware can only support DMA with hypervisor's VEK, a shared region within SEV VM called Software I/O Translation Lookaside Buffer (SWIOTLB) is always needed for SEV I/O operations. SEV VM itself needs to copy I/O streaming from SWIOTLB to its private memory when there are incoming I/O data; it needs to copy I/O data to the SWIOTLB when there are outgoing I/O. This design gives the hypervisor an opportunity to monitor and alternate I/O streaming to build encryption and decryption oracles. The paper also showed these unprotected I/O problems still exist in SEV-ES.

Summary. We summarize the attacks against SEV, their exploited vulnerabilities, the attack consequences, and the stealthiness of the attacks in Table II. SEV-SNP can defeat all known attacks against these design flaws, including unencrypted VMCB, unauthenticated encryption, unprotected nPT, and unprotected I/O. However, as SEV-SNP is not designed to mitigate ASID abuses and the CrossLine attacks, although it prevents CrossLine V1 as it disallows nPT remapping, its effectiveness against CrossLine in general deserves further exploration.

VIII. Conclusion

In conclusion, this paper demystifies AMD SEV's ASID-based isolation for encrypted memory pages, cache lines, and TLB entries. For the first time, it challenges the "security-by-crash" design philosophy taken by AMD. It also proposes the CrossLine attacks, a novel class of attacks against SEV that allow the adversary to launch an attacker VM and change its ASID to that of the victim VM to impersonate the victim. Two variants of CrossLine attacks have been presented and

Table II: Demonstrated attacks against SEV. I/O Interaction: the attack requires interaction with applications inside the victim VM through I/O operations (*e.g.*, Network, disk). Stealthiness: the attack cannot be detected by the victim VM.

Research Papers	Exploited Vulnerabilities	I/O Interaction	Breach Confidentiality	Breach Integrity	Stealthiness	Mitigated by
Du *et al.* [8]	Unauthenticated encryption	✓	✗	✓	✗	SEV-SNP
Buhren *et al.* [6]	Unauthenticated encryption	✓	✓	✗	✗	SEV-SNP
Wilke *et al.* [28]	Unauthenticated encryption	✓	✓	✓	✗	SEV-SNP
Werner *et al.* [27]	Unencrypted VMCB	✓	✓	✗	✗	SEV-ES
Hetzelt & Buhren [10]	Unencrypted VMCB Unprotected PT	✓	✓	✓	✗	SEV-SNP
Morbitzer *et al.* [20]	Unprotected PT	✓	✓	✗	✗	SEV-SNP
Morbitzer *et al.* [19]	Unprotected PT	✓	✓	✗	✗	SEV-SNP
Li *et al.* [17]	Unprotected I/O Unauthenticated encryption	✓	✓	✓	✗	SEV-SNP
CROSSLINE V1	Security-by-Crash	✗	✓	✗	✓	SEV-SNP
CROSSLINE V2	Security-by-Crash Unencrypted VMCB	✗	✓	✓	✓	SEV-ES

successfully demonstrated on SEV machines. They are the first SEV attacks that do not rely on SEV's memory integrity flaws.

ACKNOWLEDGEMENT

We thank David Kaplan and other engineers in the AMD SEV team for their valuable feedback and constructive suggestions, which have helped improve this paper.

REFERENCES

[1] AMD. AMD-V nested paging, 2008. http://developer.amd.com/.
[2] AMD. Secure encrypted virtualization api version 0.22, 2019.
[3] AMD. Amd sev-snp: Strengthening vm isolation with integrity protection and more. *White paper*, 2020.
[4] AMD. Amd64 architecture programmer's manual volume 2: System programming, 2020.
[5] AMD. Amdsev/sev-es branch, 2020. https://github.com/AMDESE/AMDSEV/tree/sev-es.
[6] Robert Buhren, Shay Gueron, Jan Nordholz, Jean-Pierre Seifert, and Julian Vetter. Fault attacks on encrypted general purpose compute platforms. In *7th ACM on Conference on Data and Application Security and Privacy*. ACM, 2017.
[7] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, et al. Fallout: Leaking data on meltdown-resistant cpus. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 769–784, 2019.
[8] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, Yufei Mai, Phoebe Wang, Jesse Liu, and Jesse Fang. Secure encrypted virtualization is unsecure. *arXiv preprint arXiv:1712.05090*, 2017.
[9] Google. Introducing google cloud confidential computing with confidential vms, 2020. https://cloud.google.com/blog/products/identity-security.
[10] Felicitas Hetzelt and Robert Buhren. Security analysis of encrypted virtual machines. In *ACM SIGPLAN Notices*. ACM, 2017.
[11] Intel. Intel architecture memory encryption technologies specification, April 2019. Ref:336907-002US, Rev: 1.2.
[12] Daehee Jang, Hojoon Lee, Minsu Kim, Daehyeok Kim, Daegyeong Kim, and Brent Byunghoon Kang. Atra: Address translation redirection attack against hardware-based external monitors. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 167–178, 2014.
[13] David Kaplan. Protecting VM register state with sev-es. *White paper*, 2017.
[14] David Kaplan. Upcoming x86 technologies for malicious hypervisor protection, 2020. https://developer.amd.com/sev/.
[15] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. *White paper*, 2016.
[16] 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*, pages 1–19. IEEE, 2019.
[17] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected i/o operations in amd's secure encrypted virtualization. In *28th USENIX Security Symposium*, pages 1257–1272, 2019.
[18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium*, pages 973–990, 2018.
[19] Mathias Morbitzer, Manuel Huber, and Julian Horsch. Extracting secrets from encrypted virtual machines. In *9th ACM Conference on Data and Application Security and Privacy*. ACM, 2019.
[20] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. SEVered: Subverting AMD's virtual machine encryption. In *11th European Workshop on Systems Security*. ACM, 2018.
[21] AMD Roger Lai. Amd security and server innovation. *UEFI PlugFest-March*, pages 18–22, 2013.
[22] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. *arXiv preprint arXiv:1905.05726*, 2019.
[23] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *14th ACM Conference on Computer and Communications Security*. ACM, 2007.
[24] Teja Singh, Alex Schaefer, Sundar Rangarajan, Deepesh John, Carson Henrion, Russell Schreiber, Miguel Rodriguez, Stephen Kosonocky, Samuel Naffziger, and Amy Novak. Zen: An energy-efficient high-performance x86 core. *IEEE Journal of Solid-State Circuits*, 53(1):102–114, 2017.
[25] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium*, pages 991–1008, 2018.
[26] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. *S&P (May 2019)*, 2019.
[27] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. The severest of them all: Inference attacks against secure virtual enclaves. In *ACM Asia Conference on Computer and Communications Security*, pages 73–85. ACM, 2019.
[28] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. Sevurity: No security without integrity–breaking integrity-free memory encryption with minimal assumptions. 2020.