AMD Memory Encryption: SME e SEV

João Gabriel Trombeta
João Paulo Taylor Ienczak Zanette
Ranieri Schroeder Althoff

9 de Abril de 2018

Conteúdo

Capítulo 1

Máquinas Virtuais

Máquinas virtuais (VM, na sigla em inglês para *Virtual Machines*) são emuladores de computadores lógicos, implementando arquiteturas e conjuntos de instruções para prover a funcionalidade de um computador físico. Chama-se de *Guest* ou convidado o computador executado pela máquina virtual, e de *Host* ou hospedeiro o computador cujo *hardware* oferece recursos para executar a máquina virtual. No *Host*, uma camada de software chamada hypervisor permite a execução máquinas virtuais em uma máquina física, independentes entre si e executando sistemas operacionais semelhantes ou não.

1.1 Hypervisor

Também chamado de monitor de máquina virtual (VMM, na sigla em inglês para *Virtual Machine Monitor*), um *hypervisor* é um componente em forma de *hardware*, *software* ou *firmware*, responsável por criar e executar uma máquina virtual. *Hypervisors* podem ser do tipo nativo (ou *bare metal*), quando executa suas máquinas virtuais diretamente no *hardware* do *Host*, ou *Hosted*, quando executa *software* sobre o sistema operacional do *Host*, sendo gerenciado pelo *kernel* do mesmo.

O *hypervisor* é responsável pela camada de abstração entre o *Host* e os *Guests*, realizando o gerenciamento de recursos, pois um *Guest* individualmente é isolado de forma que todos os recursos de *hardware* visíveis são considerados seus. Cada máquina virtual deve ser gerenciada de forma a evitar que comprometa o funcionamento das outras, por isso toda interação com o meio físico é intermediada pelo *hypervisor*, que é fortemente protegido das máquinas virtuais por ele gerenciadas.

1.2 Falhas em Hypervisors

1.2.1 Xen Hypervisor

Uma falha de segurança detectada com relação a *hypervisors* foi explorada no **Xen Hypervisor**, um dos *hypervisors* mais populares da atualidade, em que é possível executar uma função arbitrária alterando a tabela de *hypercalls*, algo semelhante a uma *vtable* no contexto de *hypervisors*. Uma *hypercall* é uma *software trap* (uma interrupção) do *hypervisor* para executar operações privilegiadas, como atualizar tabelas de página das máquinas virtuais.

Para explorar a falha, primeiramente é necessário descobrir a localização da tabela de *hypercalls* procurando pela assinatura da página, que é dada pelo *checksum* de seu conteúdo. Como a página não possui um formato tão previsível, é difícil de localizá-la, mas a tabela de argumentos dos *hypercalls* possui um formato previsível, já que seu conteúdo, que é o número de argumentos de cada *hypercall*, é fixo, e portanto seu *checksum* também é previsível. Além disso, a tabela de argumentos sempre se encontra na página seguinte à tabela de *hypercalls*, o que implica que basta encontrar uma das tabelas para que se tenha a localização da outra.

Aliado à possibilidade de leitura e escrita de código arbitrário, feito através de falhas nas regras de verificação de segurança de escrita em páginas do *hypervisor*, é possível então efetuar escape de máquina virtual, por exemplo, acessando recursos do *Host* que não estão alocados para máquina virtual.

Capítulo 2

AMD Memory Encryption

A AMD possui dois mecanismos de encriptação de memória com a finalidade de tornar mais seguro o uso de máquinas virtuais, são o Secure Memory Encryption (SME) e Secure Encrypted Virtualization (SEV). Para realizar tal função, ambos utilizam o AMD Secure Processor.

2.1 AMD Secure Processor

Grande parte das funções de criptografia executadas em um processador AMD utilizam um processador dedicado e independente, o **AMD Secure Processor** (AMD-SP, chamado anteriormente de **Platform Security Processor**), que garante que componentes sensíveis à segurança não recebam interferência do *software* sendo executado nos processadores principais.

O AMD-SP roda um *kernel* seguro de código fechado, que pode executar tarefas do sistema assim como tarefas de terceiros confiáveis, tendo o administrador controle sobre quais tarefas de terceiros são designadas ao processador. Além disso, possui uma memória dedicada e acesso direto ao coprocessador criptográfico (CCP), que é composto por um gerador de números aleatórios, várias engines para o processamento de algoritmos de criptografia, e um bloco para o armazenamento de chaves.

2.2 Security Memory Encryption

O **AMD Secure Memory Encryption** (SME) é um mecanismo que pode ser utilizado para criptografar os dados que são escritos na RAM, com a finalidade de evitar ataques físicos. Durante o boot, o AMD-SP gera uma chave simétrica que será usada para criptografar e descriptografar os dados que transitam pela DRAM. Como a *engine* de criptografia está dentro do chip, o impacto das operações é pequeno.

A figura 2.1 mostra como ocorre a criptografia. A engine é posicionada entre o OS/Hypervisor, onde dada a chave o dado é criptografado antes de ser

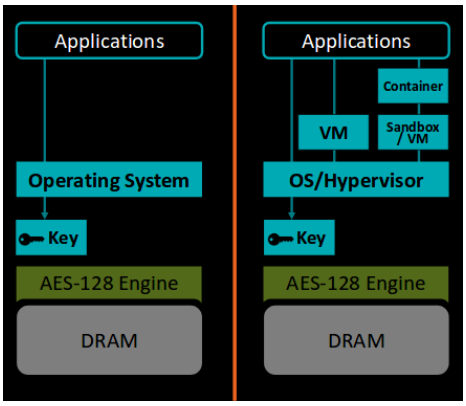

Figura 2.1: Método de criptografia do SME

salvo na DRAM. Ao passar da DRAM para o SO novamente, os dados são descriptografados.

Esse tipo de segurança não previne ataques advindos de um Hypervisor comprometido, uma vez que ele possui acesso aos dados de maneira direta, esse tipo de segurança tem com o objetivo evitar ataques como probe attack na DRAM, instalação de hardware que possa acessar a memória do *Guest*, ataques que possam capturar dados de DIMM e NVDIMM.

Para utilizar SME, é necessário verificar se o processador possui suporte para esse recurso, o que pode ser verificado através da chamada de CPUID Fn8000_001F, e que durante o boot o bit 23 de SYSCFG MSR esteja definido como 1 para sinalizar que esse recurso está habilitado. Após isso, ao fazer acesso à DRAM, é visto o último bit mais significativo do endereço, chamado de C-bit, que define se o dado deve ou não ser criptografado.

Como visto em 2.2, para a leitura, antes do dado passar para CPU, duas versões do dado são inseridas como entrada de um mux, um com o dado como estava na DRAM e outra com o dado após passar pelo circuito responsável pela criptografia. O controle do mux recebe o bit mais significativo do endereço, o C-bit, caso seja 1 significa que o dado está criptografado e a CPU precisa da informação descriptografada, caso seja 0 significa que o dado pode ser passado direto para a CPU.

Para a escrita a lógica é a mesma, caso o C-bit seja 1 o dado deve ser criptogrado antes de ser inserido na DRAM, caso seja 0 o dado pode ser salvo diretamente.

Ainda existe uma variação chamada Transparent SME, onde tudo é criptografado. Nesse caso não é necessário suporte do SO, tendo em vista que não é preciso fazer o controle de quais endereços serã criptografados e quais não. O processo de acesso à memória ocorre da mesma forma que em SME.

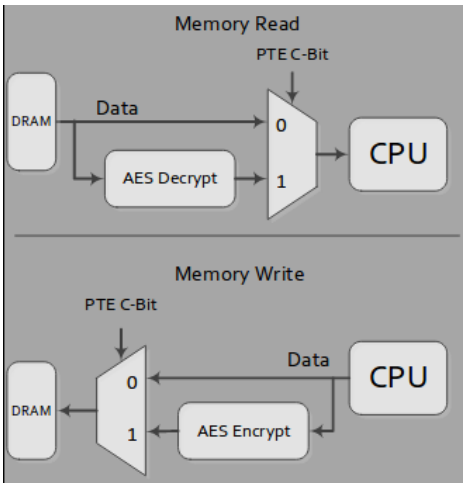

Figura 2.2: Leitura e escrita no SME

2.3 Secure Encrypted Virtualization

Secure Encrypted Virtualization (SEV) integra o SME com a capacidade de comportar várias VMs criptografadas. Durante o boot a máquina *Host* recebe uma chave, a qual é utilizada da mesma maneira que em SEV. O diferencial do SEV é que após a inicialização cada máquina virtual também ganha uma chave única, podendo assim criptografar seus dados e protegê-los tanto do hypervisor como de outros *Guest*s sendo executados no *Host*.

A figura 2.3, apesar de muito similar a 2.1, mostra como o acesso à memória se dá com a utilização sempre da chave correspondente à máquina que realizou a operação.

Na figura 2.4 é demonstrada a visão geral da arquitetura do SEV, nela é possível encontrar:

Guest Owner : Usuário que contratou o serviço e interage com sua VM.

Hypervisor : Interage com as VMs e com o Secure Processor através de drivers.

VM : Virtual Machine sendo executada no *Host*.

Secure Processor : Tem acesso exclusivo ao banco de chaves e interage com o SPI flash.

SPI Flash : Utilizado para o gerenciamento de chaves.

Memory Controller : Controla os acessos à memória aplicando conceitos visto acima, como por exemplo a criptografia.

DRAM : Memória que armazena os dados dos *Guest*s e do *Host*.

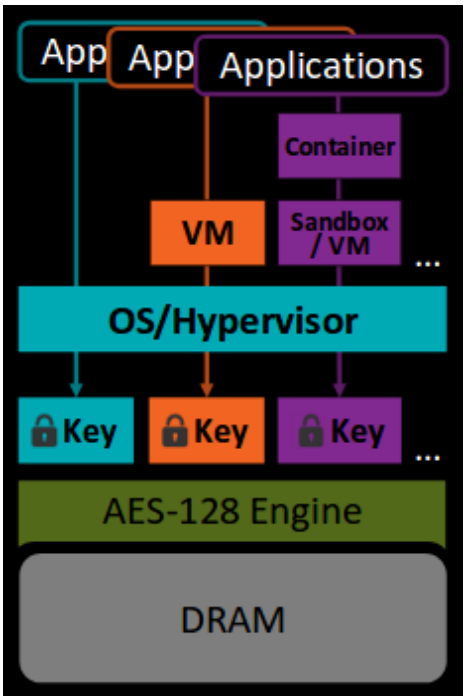

Figura 2.3: Demonstração do acesso à memória através de chaves.

Para implementar essa tecnologia também é usado o Address Space ID (ASID) para identificar a chave que pertence à VM. Para tirar proveito disso, o identificador também é utilizado para definir de quem é um dado na TLB, assim a ASID também é usado como uma *tag* na TLB. Após uma tentativa de acessar o dado, depois de ser encontrado pela TLB, é feita uma verificação para garantir que os identificadores são correspondentes antes de garantir um TLB hit. Isso faz com que não seja necessário esvaziar a TLB em caso de troca de contexto entre VMs, uma vez que as *tags* não irão colidir com uma mesma região da TLB. Esse comportamento é demonstrado em 2.5.

Um exemplo da tradução de endereço é visto em 2.6. Primeiramente o *Guest* traduz o endereço para o que ele acredita ser o endereço físico. Após isso o *Host* calcula o real endereço físico requisitado, deixando de lado o C-bit. Uma vez que o endereço físico é encontrado, o C-bit é novamente inserido e o dado pode ser consultado na memória.

Além disso, é possível existir conflito ao decidir se uma página é criptografada ou não. Uma página do *Guest* pode ter o C-bit marcado como 0, mas sua página física correspondente no *Host* ter C-bit 1. Nesses caso a página seria criptografada com a chave do *Host*, porém caso os dois C-bits sejam 1 o *Guest* teria prioridade, logo a chave usada seria a sua.

O SEV poderia garantir proteção contra o exploit visto em 1.3.1. Uma

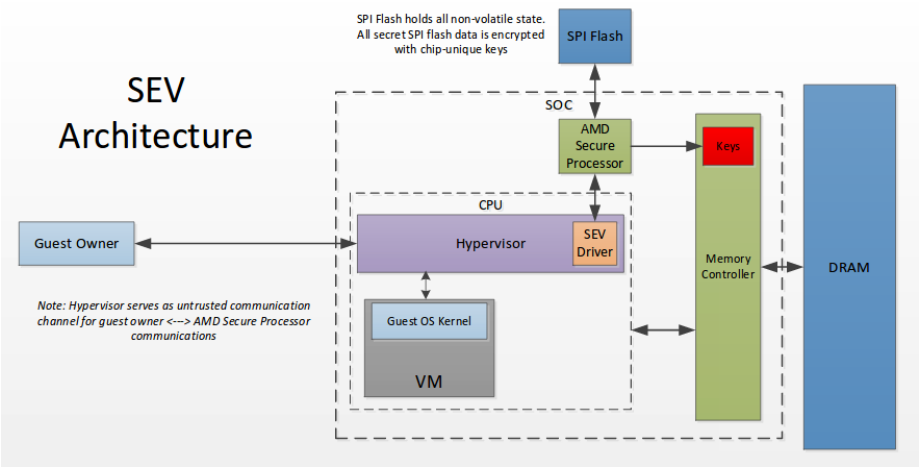

Figura 2.4: Visão geral da arquitetura do SEV.

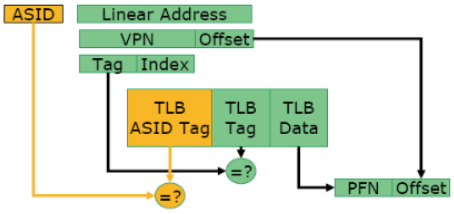

Figura 2.5: Comportamento da TLB mediante ASID.

tentativa de analisar o *checksum* com páginas criptografadas seria um desafio muito maior, já que o passo base do *exploit* consiste no *Guest* varrer dados do *Host*. Com a criptografia de memória, o *Guest* acabará descriptografando os dados do *Host* usando a própria chave, obtendo respostas diferentes do esperado e portanto não conseguindo identificar a tabela de argumentos de *Hypercall*. Caso a VM ainda assim conseguisse controle do *Host*, outras VMs estariam protegidas por utilizar sua própria chave. No geral o SEV consegue lidar tanto com ataques físicos quanto ataques de usuário, como o *Host* injetar código nos *Guest*s ou algum *Guest* tomar controle do *Host*.

2.4 Utilização do SME e SEV

2.4.1 *Set-up* e comunicação

A comunicação entre o AMD-SP e o x86 se dá por meio de registradores MMIO (*Memory Mapped IO*), chamados de *"mailbox registers"*. Um *mailbox* essencial é o *"Command buffer"*, que será lido/escrito pelo *Driver* responsável pela

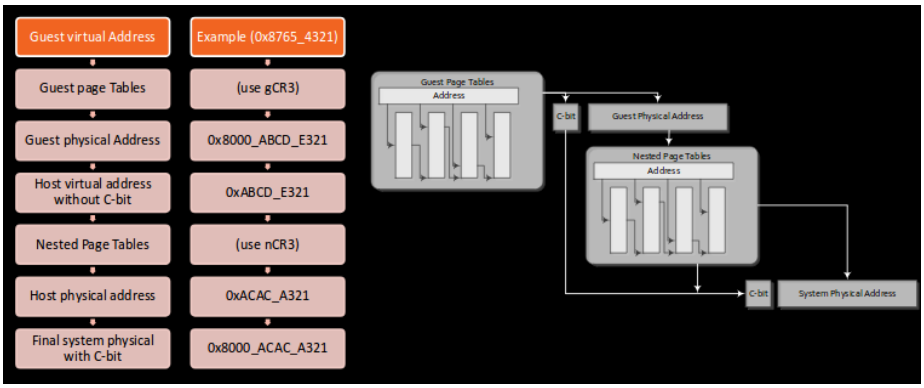

Figura 2.6: Tradução de endereços do SEV.

implementação do SEV para enviar comandos relacionados ao SEV para o *firmware*. Primeiramente é necessário, além de alterar o MSR, definir o endereço de memória em que ficará o *Command Buffer*, separado em dois registradores: `CmdBufAddr_Hi` e `CmdBufAddr_Lo` (32 bits mais e menos significativos, respectivamente). Internamente, o registrador do *Command Buffer* é chamado de `CmdResp`, e é utilizado da forma:

1. O driver (em x86) altera o `CmdResp` com:
   - Bit 31 em 0 (simbolizando a emissão de um comando);
   - Bits 30 a 26 mantidos em 0 (pois são reservados);
   - Bits 25 a 16 com o ID do comando a ser emitido;
   - Bits 15 a 1 também são mantidos em 0 (por serem reservados);
   - Bit 0 pode ser colocado em 1 para informar o *firmware* para habilitar uma interrupção no x86 ao completar o comando, do contrário pode ser mantido em 0.
2. O AMD-SP executa o comando;
3. Após completar o comando, o AMD-SP escreve no `CmdResp` com o resultado da execução do comando, da forma:
   - Bit 31 em 1 para indicar que se trata de uma resposta;
   - Bits 15 a 0 com o código de status descritos na tabela 2.1.

2.4.2 Comandos

INIT

O comando `INIT`, intuitivamente, inicializa a plataforma. Isso é feito carregando os dados de persistência relacionados ao SEV, vindos de algum armazenamento

Status	Código
SUCCESS	0000h
INVALID_PLATFORM_STATE	0001h
INVALID_GUEST_STATE	0002h
INVALID_CONFIG	0003h
INVALID_LENGTH	0004h
ALREADY_OWNED	0005h
INVALID_CERTIFICATE	0006h
POLICY_FAILURE	0007h
INACTIVE	0008h
INVALID_ADDRESS	0009h
BAD_SIGNATURE	000Ah
BAD_MEASUREMENT	000Bh
ASID_OWNED	000Ch
INVALID_ASID	000Dh
WBINVD_REQUIRED	000Eh
DFFLUSH_REQUIRED	0009h
INVALID_GUEST	0010h
INVALID_COMMAND	0011h
ACTIVE	0012h
HWERROR_PLATFORM	0013h
HWERROR_UNSAFE	0014h
UNSUPPORTED	0015h
INVALID_PARAM	0016h

Tabela 2.1: Relação dos códigos de status dados pelo AMD-SP. Mais detalhes sobre a tabela estão em [1]

não-volátil, e inicializando o contexto da plataforma. Geralmente, os casos em que este não é o primeiro comando executado são os que primeiramente utilizam comandos como `PLATFORM_STATUS` para determinar a versão da API. Requer que o estado da plataforma seja `PSTATE.UNINIT`;

SHUTDOWN

Utilizado pelo dono da plataforma afim de alterar o estado dela para não-inicializado. A plataforma pode estar em qualquer estado e, quando executado, todo o estado da plataforma e dos *Guests* é excluído do armazenamento volátil.

PLATFORM_RESET

Reinicia os dados não-voláteis de SEV, sendo útil quando o dono deseja transferir a plataforma para um novo dono ou apenas liberar os recursos do sistema seguramente.

PLATFORM_STATUS

Informa sobre o status atual da plataforma, incluindo versão da API (*major* e *minor*), dono (se é o próprio *Host* ou é externo), ID da *build* do *firmware*, número de *Guests* válidos mantidos pelo *firmware*, e código de status do *firmware*.

Outros comandos

Há outros comandos relativos às chaves, como `PEK_GEN`, `PEK_CSR`, `PEK_CERT_IMPORT`, `PDH_GEN` e `PDH_CERT_EXPORT`, que permitem gerar, trocar e validar chaves.

2.5 Possíveis falhas

Apesar das seguranças em criptografia os dois sistemas ainda podem ser vulneráveis a ataques. Nada impede um Hypervisor comprometido de modificar trechos de um *Guest*, mesmo que por textos sem sentido, ou até mesmo reescrever dados em um local da memória com trechos mais antigos do mesmo local.

Outra técnica de ataque que pode ser utilizada pelo Hypervisor é trocar os dados de dois endereços de uma virtual machine. Dessa forma a VM ainda conseguiria descriptografar a página porém resultaria em um comportamento inesperado.

Bibliografia

[1] I. Advanced Micro Devices, "Secure Encrypted Virtualization API version 0.14," tech. rep.

[2] S. Luan, "Exploit two Xen Hypervisor vulnerabilities."

[3] D. Kaplan, "AMD x86 Memory Encryption technologies."

[4] "AMD Memory Encryption tutorial," tech. rep.

[5] Z.-H. Du, Z. Ying, Z. Ma, Y. Mai, P. Wang, J. Liu, and J. Fang, "Secure Encrypted Virtualization is unsecure!," tech. rep.

[6] T. W. David Kaplan, Jeremy Powell, "AMD memory encryption," tech. rep.

[7] I. Advanced Micro Devices, "AMD-V nested paging," tech. rep.