Intel® Software Guard Extensions (Intel® SGX) Architecture for Oversubscription of Secure Memory in a Virtualized Environment

Somnath Chakrabarti, Rebekah Leslie-Hurd, Mona Vij, Frank McKeen,
Carlos Rozas, Dror Caspi, Ilya Alexandrovich, Ittai Anati

Intel Corporation

{somnath.chakrabarti, rebekah.leslie-hurd, mona.vij, frank.mckeen, carlos.v.rozas, dror.caspi, ilya.alexandrovich, ittai.anati}@intel.com

ABSTRACT

As workloads and data move to the cloud, it is essential that software writers are able to protect their applications from untrusted hardware, systems software, and co-tenants. Intel® Software Guard Extensions (SGX) enables a new mode of execution that is protected from attacks in such an environment with strong confidentiality, integrity, and replay protection guarantees. Though SGX supports memory oversubscription via paging, virtualizing the protected memory presents a significant challenge to Virtual Machine Monitor (VMM) writers and comes with a high performance overhead. This paper introduces SGX Oversubscription Extensions that add additional instructions and virtualization support to the SGX architecture so that cloud service providers can oversubscribe secure memory in a less complex and more performant manner.

Keywords

SGX; Software Guard Extensions; Virtualization; Memory Management; Oversubscription

1 INTRODUCTION

Computing is evolving into a virtualized resource that Cloud Service Providers (CSPs) supply on-demand to a broad spectrum of users from corporate customers to application providers. The variety of workloads requires dynamic allocation of compute resources to maximize performance.

HASP '17, June 25, 2017, Toronto, ON, Canada

ACM 978-1-4503-5266-6/17/06...$15.00
http://dx.doi.org/10.1145/3092627.3092634

Memory is a key element of the resource management strategy because the memory requirements of cloud application can vary dramatically over time. Many cloud workloads contain personal or confidential data that must not be exposed. CSPs must provide confidentiality and integrity, especially in an environment shared between multiple unknown users, so that their customers can rest assured that their data is protected from both accidental and malicious disclosure or tampering. In today's environment, privileged software provided and maintained by the CSP is central to protect customer workloads and data. Bugs in the CSP software could result in the compromise of a customer's data, and establishing the correctness of such large and complicated software codebases is extremely difficult. Intel® Software Guard Extensions (SGX) was developed to meet the confidentiality and integrity needs of applications even when the software environment may not be bug-free.

SGX provides the hardware support needed to defeat privileged software attacks and assures separation between critical data and untrusted application code without sacrificing the platform manageability features expected by CSPs such as memory oversubscription. A virtual machine monitor (VMM) may virtualize the memory of SGX programs using the standard ballooning [6] or paging [9] techniques; however, performing SGX paging in a VMM requires expensive VM exits and emulation of guest behavior. In addition, the VMM must keep extensive metadata to manage the SGX memory correctly, which wastes space and complicates the VMM algorithms.

To address the difficulties in virtualizing SGX memory using the existing paging instructions, Intel® developed the SGX Oversubscription Extensions to SGX described in this paper. The new instructions and virtualization support enable CSPs to support SGX in the cloud in a simpler and more performant manner without compromising the security guarantees or functionality of legacy SGX. The remainder of this paper is devoted to the description of these extensions and their usage, with a review of the relevant background knowledge in Section 2. Section 3 provides background on oversubscription techniques and details of the oversubscription extensions. Section 4 provides a reference software flow using the new extensions. Section 5 reviews

related work with Section 6 providing concluding remarks.

2 SGX MEMORY MANAGEMENT

SGX provides an execution environment called an enclave where code and data are protected from privileged and unprivileged code. Enclave code and data are stored in a protected area of memory called the Enclave Page Cache (EPC), as are security-critical control structures introduced by SGX. SGX provides the building blocks for system software to manage EPC memory, including instructions for oversubscribing the EPC, dynamically allocating EPC pages, and changing the permissions or type associated with an EPC page. In this section, the structure of enclave memory and the hardware instructions for enclave memory management are reviewed. For a detailed overview see [8].

2.1 Enclave Layout

Unlike regular memory, enclave memory has a structure that connects each page of the enclave to the enclave control structure for the enclave to which that page belongs. This connection is recorded by the CPU in a protected per-page data structure called the Enclave Page Cache Map (EPCM). The enclave control structure, the Secure Enclave Control Structure (SECS), resides in a separate page of EPC memory and stores configuration information such as the linear address range associated with the enclave. Each enclave page is linked to its SECS parent via an identifier stored in the EPCM, as illustrated in Figure 1.

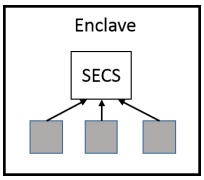

Figure 1: Unlike traditional applications, enclave memory layout is not flat. VMMs must understand the layout of each enclave to effectively support paging-based oversubscription.

The tree-like structure of enclave memory introduces constraints for enclave paging in that all of the child pages associated with an enclave must be removed before their SECS parent can be paged out of the enclave. Without special support, the VMM must keep track of the type of each enclave page as well as the location of its parent.

In addition to the SECS identifier, each enclave page has metadata associated with it that further constrains the ways in which the page may be used. This metadata is also stored in the EPCM, which contains the following fields:

- VALID: Indicates whether a page is in use.
- RWX: Permissions associated with page.
- PAGETYPE: Indicator of the kind of EPC page, for example PT_REG or PT_SECS.
- BLOCKED, PENDING, MODIFIED, PR: Status flags that indicate there is an update to the page in-progress.

VMM must maintain the EPCM state of a page across paging so as to be transparent to the guest. In some cases the EPCM state may lead the VMM to make different memory management choices. For the full definition of the EPCM and SGX details, see the SDM Chapter 38 [4] and previous SGX papers [7,8].

2.2 Enclave Memory Paging

SGX provides instructions for enclave memory paging including EWB, which securely evicts an EPC page to regular memory and ELDB/ELDU, which load a previously evicted page back into the EPC. Enclave pages evicted from the EPC to main memory have the same integrity, confidentiality and replay protection as when the contents resided within the EPC. To achieve this protection, the paging instructions enforce the following rules:

1. An enclave page can be evicted only after all cached address translations to that page have been evicted from all logical processors.
2. The contents of the evicted enclave page must be encrypted before being written out to main memory.
3. When an evicted enclave page is reloaded into EPC it must have identical page type, permissions, virtual address, content, and be associated to the same enclave as at the time of eviction.
4. Only the last evicted version of an enclave page can be allowed to be reloaded.

To prepare the enclave page for eviction, system software marks the page to be evicted as BLOCKED using the EBLOCK instruction. Once an EPC page has been marked as BLOCKED, the processor prevents any new Translation Lookaside Buffer (TLB) entries that map that EPC page from being created. However, TLB entries that reference this page may exist in one or more logical processors. These TLB entries must be removed before the page can be removed from the EPC. The ETRACK instruction configures microarchitectural trackers that detect when all cached references to a blocked page have been flushed. Hardware uses these trackers to determine when it is safe to evict an enclave page. To avoid races on the per-enclave trackers and simplify the architecture, ETRACK may not be called concurrently on the same enclave. To summarize, the software flow for evicting enclave memory in an OS or VMM is as follows:

1. Identify the target page to be evicted using usual algorithms (aging, etc.)
2. Unmap the page from the application space.
3. Execute EBLOCK on the target page.
4. Execute ETRACK on the SECS page to which the target page belongs.
5. Flush the TLB mappings to the target page, for example, by sending IPIs to the threads running in the enclave to which the page belongs. Threads may resume immediately after the mappings have been flushed.
6. Execute EWB on the target page.

To load an evicted page back into the enclave, software uses the ELDU or ELDB instruction. These instructions both load an encrypted page from main memory into the EPC, after first checking the integrity of the contents. The only difference is whether the page is loaded in the blocked state (ELDB) or not (ELDU). Typically software uses

ELDU, but ELDB is useful if the guest OS has blocked the page prior to the VMM evicting it. Both instructions require the address of the SECS parent with which the newly loaded page should be associated.

2.3 Challenges in Virtualizing SGX Paging

The enclave paging flow described in the previous section presents multiple challenges for a VMM:

- The VMM must know the location of the SECS parent of any page that it would like to evict and must remember this location while the page is written out.
- The VMM and guest may execute ETRACK concurrently on the same enclave, causing an unexpected error in the guest.
- A guest VM might evict or remove SECS pages without the VMM's knowledge, preventing the VMM from loading evicted pages associated with that parent.

The only strategy available to the VMM in legacy SGX for handling these challenges is to emulate paging and enclave configuration instructions for the guest and to track the enclave memory layout of the guest.

Our evaluation of the VMM software flows required to support SGX paging and oversubscription found that the performance and complexity overheads were significant. Table 1 shows the performance overheads for the enclave paging and enclave build flows. The numbers are captured on Intel® Xeon E3-1280 v5 based server using KVM as the VMM on Ubuntu 14.04 host. The guest OS is Ubuntu 14.04 tested with enclaves upto 4MB in size.

Table 1: Overhead of exit roundtrip and emulation during paging and during enclave build/teardown.

VMM Guest Enclave Paging	#Cycles (average)	VMM Enclave Build/Teardown	#Cycles (average)
ELD/EWB	10436	EREMOVE/EADD	3550
VM exit roundtrip	2872	VM exit roundtrip	2872
VMM emulation	4746	VMM emulation	4746
Overhead	73%	Overhead	214%
VMM paging operation incurs an average of 70% overhead		Guest enclave build/teardown incurs an average of 200% overhead	

As shown in the table, the cost of exiting on guest instructions and emulating them in the VMM significantly increases the cost of paging and doubles the cost of enclave build. (Note that the cycle numbers shown are based on a prototype and may vary between various machine configurations.) The next section describes SGX Oversubscription processor extensions that enable VMM oversubscription in a simpler and more performant manner.

3 SGX OVERSUBSCRIPTION SUPPORT

The SGX Oversubscription Extensions is an architecture extension designed to address the challenges in virtualizing EPC paging discussed in the previous section.

3.1 Background

Oversubscription is a term used to describe the VMM practice of assigning more resources to virtual machines (VMs) than is actually available on the physical platform. VMs typically do not use all the resources all the time, so oversubscription enables cloud service providers to increase the density of VMs per platform. Resource oversubscription can be done for CPU, memory, and I/O resources. For example, the VMM can assign more virtual CPUs (VCPU) to a VM than physical cores available, scheduling the VCPUs on actual CPUs as governed by the scheduler policy. The remainder of this section explores the practice of memory oversubscription and the various ways a VMM might allocate memory to a VM.

3.1.1 Static Partitioning

One approach to memory allocation is for the VMM to statically partition the available memory amongst the guest VMs, as shown in Figure 2(a). In this scheme, the memory available to a VM does not change over its lifetime. The advantage of static partitioning is its simplicity and the fact that guest applications observe deterministic performance because the VM is always in possession of the expected amount of memory. The disadvantage is that static partitioning can lead to memory waste when a VM is not making use of its full allotment.

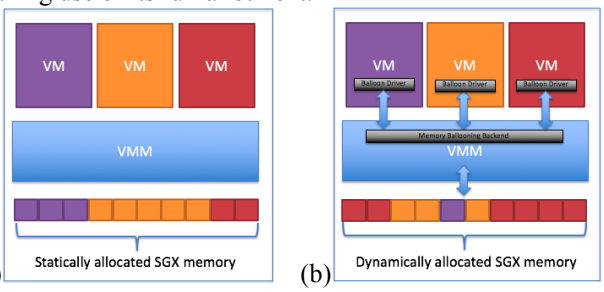

Figure 2: SGX memory partitioning options. VMM may configure memory (a) statically for each VM or (b) dynamically based on each VM's run-time needs.

3.1.2 Dynamic Partitioning Using Ballooning

To address the resource utilization issues with static partitioning, some VMMs allocate memory dynamically, growing the size of a VM's memory footprint on demand. The memory layout arising from this technique, which is known as ballooning, is illustrated in Figure 2(b). Dynamic partitioning using ballooning requires guest co-operation, which is not always feasible. Each VM informs the VMM when it needs more memory or when it wants to release free memory. Based on this protocol, the hypervisor can maintain a free pool of memory that it uses to satisfy dynamic memory allocation requests. Since this model requires guest awareness and cooperation with the VMM, it is best-suited for enterprise datacenter environments where both the VM and VMM are trusted. In public cloud environment, this model is not suitable because cloud service providers do not trust the guest VMs and often do not even allow direct VM-to-VMM communication.

3.1.3 Dynamic Partitioning Using Paging

In scenarios where guest cooperation with memory management is not feasible, but VMM writers would still like to take advantage of dynamic memory partitioning. paging-based oversubscription can be done by the VMM. Paging-based oversubscription presents the guest with

virtual memory that may or may not be fully resident at any given time. The VMM manages the resident memory based on the memory usage of the VM and the load of the system. A variety of heuristics are employed to determine which memory to page out to disk, such as identifying the least recently used page by scanning the guest's memory and the associated accessed and dirty flags. Though paging-based oversubscription does not require guest modifications, it does expose non-deterministic timing behavior to the guests and might not be suitable for certain real-time applications. In the context of SGX, extra work must be done by the VMM to page out EPC memory in a way that is functionally transparent to the guest.

3.2 Overview

The SGX Oversubscription Extensions architecture addresses the challenges that VMM writers encounter when deploying dynamic partitioning of EPC memory by introducing new paging support for VMMs. The architecture provides three key components to address the requirements presented in the previous section:

1. Ability to determine guest EPC layout, including the location of each page's SECS, without tracking guest allocation instructions (Section 3.4).
2. Ability to prevent, detect and recover from conflicts that occur when the VMM and guest are paging the same enclave in a way that is transparent to the guest VM (Section 3.5).
3. Ability to prevent a guest OS from incorrectly paging out an SECS page that has no children due to VMM paging activities (Section 3.6).

With this support, a VMM and its guests can simultaneously perform SGX paging without requiring the VMM to perform emulation of guest paging instructions. If used correctly, a guest VM will not observe functional differences in machine behavior when an underlying VMM is paging its EPC memory. Timing differences may occur, but are beyond the scope of this paper.

3.3 SGX Instruction Format

The SGX Oversubscription Extensions include instruction support for VMM EPC paging. Legacy SGX provides two instructions: ENCLU (for user-mode actions) and ENCLS (for supervisor-mode actions). To perform a particular action, software invokes the appropriate instruction with a *leaf function* parameter that tells the hardware which SGX action to perform. The formal name for an SGX action is *instruction[leaf_function_name]*, though we often refer to SGX actions by their leaf function name alone. For example, Section 2.1 referred to the EBLOCK instruction, which would formally be referred to as ENCLS[EBLOCK].

SGX Oversubscription Extensions adds an additional instruction, ENCLV, as well as new leaf functions. Table 2 summarizes these new leaf functions.

Table 2: SGX Oversubscription instructions and purpose.

Instruction	Purpose
ENCLS[ERDINFO]	Provides metadata of EPC page to VMM
ENCLV[ESETCONTEXT]	Store information in SECS for later access by VMM through child pages
ENCLV[EINCVIRTCHILD]	Blocks guest from paging out SECS page in use by the VMM
ENCLV[EDECVIRTCHILD]	Allows guest to page out SECS page
ENCLS[ETRACKC]	For conflict free tracking of SECS page
ENCLS[ELDC]	For conflict free loading of EPC pages

ENCLV is a special instruction that can only be executed when VMX operation is enabled, that is, between the execution of the instructions VMXON and VMXOFF. ENCLV may be executed by the VMM (root mode) or a guest (non-root mode). We introduce this instruction because some of the oversubscription features do not make sense in an OS bare metal context. Though we don't envision a typical guest needing to execute ENCLV instruction leaf functions, there are VMM designs where a guest VM performs privileged actions on behalf of the VMM.

The VMM enables non-root execution of ENCLV leaf functions by setting the "ENABLE_ENCLV" execution control in the Virtual Machine Control Structure (VMCS) on a per-guest basis. An execution control is per-guest configuration setting. If this control is disabled, guests will receive an invalid opcode exception when attempting to execute an ENCLV instruction leaf. In some situations, such as nested virtualization the VMM may enable ENCLV but also configure ENCLV exiting so that the VMM can emulate ENCLV leaf functions on behalf of the guest.

Table 3 shows the hardware behavior of ENCLV in different modes and the corresponding use-cases:

Table 3: Behavior of ENCLV in different processor modes

Mode	H/W Behavior	Use Case
VMXOFF	Causes fault	Bare-metal OS
VMXON (root)	ENCLV Available	Hosted VMMs
VMXON (non-root)	ENCLV available if enabled else fault	VMMs with VT-based control domain

3.4 Determining Guest Enclave Layout

In order to correctly page out guest EPC memory, a VMM must know the EPCM status of the page and where the parent SECS structure resides for that page. SGX Oversubscription Extensions enables a VMM to determine the parent SECS location without scanning the guest's memory or exiting on enclave configuration instructions. SGX Oversubscription extends SGX hardware to track the physical address of each enclave page's SECS and introduces a new leaf function that allows system software to extract metadata information regarding an EPC page. This section discusses these aspects of the oversubscription.

3.4.1 *SECS Location Tracking*

When the VMM is performing paging operations of guest enclave pages, it must be able to locate the parent SECS of an enclave page so that it can be loaded back into the enclave later. However, the VMM does not know when guests create, evict, or load an SECS page. One way to keep track of guest enclave layout is to simply trap all SGX ring-0 instructions and emulate those instructions on guest's behalf. In this case the VMM also needs to maintain a dynamic table that maps all the SECS addresses and their enclave's hardware-assigned identifier. When loading an EPC page, the VMM performs a lookup into the table to find the corresponding SECS location. This mechanism is extremely expensive in terms of execution time and VMM memory consumption, and adds significant overhead in the guest paging and enclave creation flows. Figure 3 shows an example enclave layout inside the guests and the corresponding VMM mapping of those enclaves.

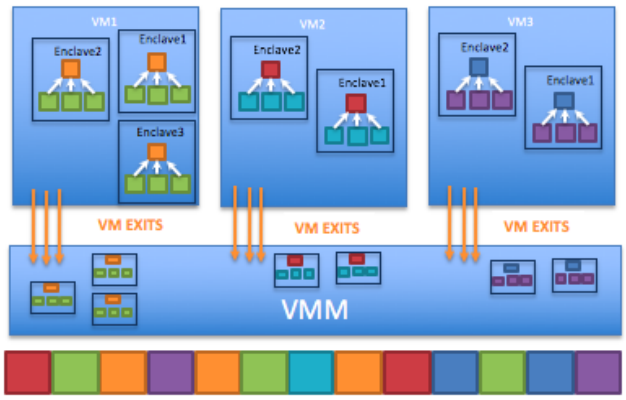

Figure 3: Example layout of a running system. Without additional support, the VMM must track the enclave structure present in the guest.

To reduce the overheads associated with SECS location tracking, SGX Oversubscription introduces a 64-bit field in the SECS called ENCLAVECONTEXT that VMMs may use to simplify the process of locating the SECS page associated with an EPC page. This field is initialized by the hardware during instructions that create SECS pages and may be accessed by the VMM later using ENCLS[ERDINFO] (see next section), avoiding the need for the VMM to maintain a mapping table. The hardware uses the physical address produced by page table translation. This corresponds to a guest-physical address when executed in non-root mode and a host-physical address when executed within the VMM. In some scenarios, such as when a VMM is executing the EPC allocation on behalf of a guest, the address produced by paging will not be the correct address for locating the page again later. In which case, the VMM may overwrite the default value using a new leaf function called ENCLV[ESETCONTEXT].

For nested virtualization cases, the lowest level VMM can hide SGX Oversubscription instructions from higher level guest VMMs. In that case the lower level VMM can inject a general protection fault into higher level VMMs if they incorrectly attempt to execute these instructions. However, if VMMs wish to expose SGX Oversubscription instructions to higher level VMMs, then the lowest level VMM may need to use ENCLV[ESETCONTEXT] to properly manage the ENCLAVECONTEXT field of SECS during paging operations. To determine the correct location context value to use, the lowest level VMM may need to execute enclave context related instructions on the nested VMM's behalf.

3.4.2 Reading EPC Metadata

ENCLS[ERDINFO] is a new supervisor instruction that provides software with access to the metadata associated with each valid EPC page. This avoids the need to track page properties such as the type and corresponding SECS page in the VMM. The information is returned via a new RDINFO data structure, shown in Table 4.

The STATUS field of the structure describes the status of the page. The FLAGS field contains the page's access permissions; the page type; and the BLOCKED, PENDING, and MODIFIED and PR status of the page. If the page is an enclave child page or an SECS page, then ERDINFO also returns the value of ENCLAVECONTEXT from the corresponding SECS page.

For other page types, the ENCLAVECONTEXT field is considered reserved. For invalid or non-EPC pages, the instruction returns an information code to indicate why ERDINFO did not succeed. ERDINFO returns an error code when other instructions that modify the EPCM are executed simultaneously.

Table 4: Information returned by ERDINFO

Field Name	Bit Name	Description
STATUS	CHILDPRESENT	Indicates SECS has one or more child pages present
	VIRTCHILDPRESENT	Indicates SECS has one or more virtual child pages present
FLAGS	RWX	EPCM.RWX permission
	PENDING	PENDING status of the page
	MODIFIED	MODIFIED status of the page
	PR	PR status of the page
	PAGE_TYPE	The type of EPC page
	BLOCKED	BLOCKED status of page
ENCLAVECONTEXT		Context pointer to SECS

3.5 Handling Conflicts

When both guest and VMM are paging enclave memory simultaneously, conflicts may arise between instructions accessing a shared resource. For example, ETRACK modifies a micro-architectural tracking structure in the SECS of a target enclave. Concurrent calls to ETRACK on the same enclave may cause a synchronization error in the guest or in the VMM. In legacy SGX this failure results in a general protection fault, which is difficult to handle in a VMM. The fault is particularly problematic when observed by the guest, because it may allow the guest to detect that it is being virtualized and may even result in a fatal error if the

guest is not prepared to handle the fault.

Section 3.5.1 describes new variants of the paging instructions ETRACKC and ELDB/U that return recoverable error codes when a synchronization error occurs. The capability to hide these errors from the guest is described in Section 3.5.2.

3.5.1 Preventing Unnecessary Paging Conflicts

To help the VMM manage conflicts with the guest, we introduce variants of the ETRACK and ELDB/ELDU instructions with better support for concurrency. These instructions produce error codes instead of exceptions when synchronization conflicts occur. This allows the VMM to catch and recover from races with the guest. When the VMM encounters an error code, it may retry the instruction or abort the operation to work on a different page. When the VMM encounters an error code, it may retry the instruction or abort the operation to work on a different page.

3.5.2 Recovering from Paging Conflicts

In legacy SGX, the VMM must execute all potentially conflicting instructions on the guest's behalf to avoid synchronization errors. Ideally, the VMM would allow the guest to execute SGX instructions, but be able to detect and recover from any conflicts transparently to the guest. To support this desired behavior, SGX Oversubscription introduces a new VM exit reason, called an SGX conflict exit, which transfers execution control of the machine to the VMM whenever the guest encounters a synchronization error that might have been caused by VMM activity.

Table 5: VM Exit Qualification

Field Name	Bit Name
CODE	TRACKING_LOCK_CONFLICT
	REFERENCE_COUNT_CONFLICT
	EPCM_LOCK_CONFLICT_EXCEPTION
	EPCM_LOCK_CONFLICT_ERROR
ERROR	ERROR CODE (for EPCM_LOCK_CONFLICT_ERROR only)

The SGX conflict VM exit provides the VMM with additional information that describes the nature of the conflict in the exit qualification field, as shown in Table 5. The new instruction leaf functions ETRACKC and ELDC may cause an SGX conflict exit, as well as the legacy leaf functions EADD, ECREATE, ELDB/ELDU, EPA, EREMOVE, ETRACK, and EWB. The VMM configures whether SGX conflict exits should occur on a per-guest basis by setting a control bit in the corresponding VMCS.

When the VMM receives an SGX conflict exit, it uses its own internal data structures to determine whether the fault could have been caused by the VMM, for example, if the VMM is currently paging the same enclave. The VMM will take different courses of action depending on whether or not the VMM may have caused the conflict.

If the fault may have been caused by the VMM, then the VMM has two possible options: (a) allow the guest to retry the conflicting instruction or (b) synchronize with the conflicting VMM thread and execute the guest instruction in the VMM. Approach (a) is usually safe because synchronization failures are transient and unlikely to occur again; however, in degenerate cases allowing the guest to retry could lead to a failure of the guest to make progress. The VMM may choose to employ heuristics, retrying the instruction some number of times and then falling back to a executing the problematic instruction within the VMM.

If the fault was not caused by the VMM, then the failure should be reported to the guest. The VMM resumes guest execution at the next instruction, injecting the appropriate error code by updating the guest's register values.

3.6 Preventing Guest Interference

For the most part, a VMM prevents interference from a guest by unmapping pages that it is operating on from the extended page table (EPT) entries of the guest. This prevents the guest from, for example, executing an instruction on the same page that the VMM is working on. In other cases, the VMM can detect and recover from interference using the error code information returned by a failing instruction. However, the area that still needs to be addressed is preventing the guest VM from paging out an SECS of an enclave that the VMM is operating on. If this isn't prevented, the VMM will not be able to bring back pages for that enclave that are currently paged out. The unmapping approach does not work because the SECS page must be present to execute enclave programs.

An important feature of the EPC page de-allocation instructions, EREMOVE and EWB, is that they do not allow software to remove or evict an SECS page when child pages associated with that SECS are resident in the EPC. This ensures that every enclave child page always points to a valid enclave parent. Each SECS page contains a "child count" that tracks the number of resident EPC pages associated with this enclave and is checked during EREMOVE and EWB. When a VMM writes out a guest's child page to disk, the child count associated with that SECS might go to zero, allowing the guest VM to EREMOVE or EWB an SECS page without informing the VMM. If the guest touches one of the evicted pages, the VMM has no way to reload the page back into the enclave and resume the guest because it can't associate the page with its SECS. The remainder of this section describes how SGX Oversubscription Extensions avoids this scenario by introducing a virtual child count.

SGX Oversubscription introduces an additional SECS field to store the *virtual child count,* a count of pages that the guest believes to be resident but have actually been evicted by the VMM. The VMM explicitly manages this counter during paging operations, calling the new leaf function ENCLV[EINCVIRTCHILD] to increment the virtual child count during the eviction flow and ENCLV[EDECVIRTCHILD] to decrement the count during the load flow. This field is initialized to zero by hardware during the creation of the SECS (ECREATE) and preserved when the SECS itself is paged out.

Whenever the virtual child count is non-zero, the guest VM is blocked from evicting the SECS page, just as if it had resident children present in the EPC. The guest VM isn't

expected to attempt such an eviction, after all, the guest thinks the virtual children are resident, but if the guest does attempt an EREMOVE or a EWB on the page, it will receive an SGX_CHILD_PRESENT error code, as if a physical child was there.

SGX Oversubscription introduces a new execution control that the VMM uses to enable or disable virtual child count checking on a per-guest basis. Typically, VMMs that perform oversubscription would enable virtual child count tracking on all guests. VMMs that make use of a privileged guest to perform paging and EPC page removal on behalf of other guests would disable this tracking for that VM. In VMX root or bare metal mode, the virtual child count is ignored, ensuring the VMM or OS can always manage EPC memory as it sees fit. Note that executing ERDINFO on an SECS page returns status bits indicating whether physical and virtual children are present. This helps the VMM to quickly determine the state of the enclave at any point in time as described in Table 6.

Table 6: Behavior of ERDINFO instruction under different processor configurations.

VMX Off || VMX root mode || ENABLE_EPC_VIRTUALIZATION_EXTENSIONS VT control is OFF	VMX non-root mode && ENABLE_EPC_VIRTUALIZATION_EXTENSIONS VT control is ON
CHILDPRESENT = (CHILDCOUNT != 0) VCHILDPRESENT = (VIRTCHILDCOUNT != 0)	CHILDPRESENT = (CHILDCOUNT != 0) OR (VIRTCHILDCOUNT !=0) VCHILDPRESENT = 0

4 EXAMPLE SOFTWARE FLOWS

There are various ways a VMM can implement EPC oversubscription using the new SGX extensions. However, at a high level, a VMM will typically implement a basic flow as discussed in the following sections.

4.1 Paging out EPC pages by the VMM

When the VMM wishes to evict memory pages from the EPC, it executes the following steps:

1. Start the aging cycle to find victim pages using EPT A/D bit to track recently used pages
2. Mark victim page(s) not present in EPT
3. Flush TLBs of threads running in the guest to remove stale translations. VMMs typically track which hardware threads are assigned to which VMs. So VMM needs to send TLB flush IPI only to those threads that belong to that specific VM.
4. For every aged page in the guest EPC address space, call ENCLS[ERDINFO] to determine the page's type and SECS location. Then take the following actions:
    a. Evict enclave child pages, saving the location of each page's parent for later use and incrementing the virtual child count with EINCVIRTCHILD. Use normal SGX paging flow (EBLOCK, ETRACK, EWB) on the victim page.
    b. Evict SECS pages if their child count is zero.

ERDINFO gives the VMM information about the EPC memory without the VMM maintaining this information itself. By using ETRACKC, the VMM avoids causing faults in guest. EINCVIRTCHILD blocks the guest from evicting an in-use SECS.

4.2 Loading enclave pages back into EPC

Typically, an EPT violation due to a guest EPC access to a non-present page triggers the VMM to reload the page. At that point, the VMM executes the following steps:

1. Identify the SECS page in VMM data structures for the page being restored
2. Create VMM mappings to target page and SECS page
3. Execute ELDC on page. ELDC helps VMM avoid causing faults in guest.
4. For every regular page successfully loaded, execute EDECVIRTCHILD to allow guest to be able to evict SECS
5. If loading SECS, execute ESETCONTEXT to restore context. This ensures that VMM can track this SECS
6. Add EPT mapping for page
7. During an EPT violation caused due to guest access to non-present page, as an optimization, VMMs may choose to load more contiguous pages instead of just one. The probability of the guest accessing next set of contiguous pages is typically high, hence this strategy may reduce number of EPT violations and improve guest performance.

5 RELATED WORK

Secure Encrypted Virtualization (SEV) [5] is an extension of AMD-V™ architecture that supports running multiple encrypted VMs under the control of the hypervisor. SEV is focused on VM isolation, rather than application protection like SGX, and does not provide generic oversubscription support for VMMs.

VMware® and Hyper-V both support memory oversubscription [1] for regular memory but use slightly different techniques. VMware® uses both ballooning and paging for overcommitting memory as opposed to Hyper-V which gives finer control over per-VM memory assignment.

VMware® ESX server supports memory oversubscription [2] as an important feature of the hypervisor and claims that this is an important feature even with continuing fall of memory cost. Memory oversubscription is important for disaster recovery, high availability and distributed power management to ensure good performance.

6 CONCLUSIONS

SGX Oversubscription Extensions address the difficulties of virtualizing SGX memory using the existing paging instructions. In particular, it allows VMMs to avoid the performance overhead and complexity of tracking, virtualizing and maintaining the parent-child relationship between an SECS and the pages belonging to that enclave. The extensions include new instructions and extensions for programmatically discovering the parent-child relationship (ERDINFO), virtualizing the parent-child relationship (EINCVIRTCHILD, EDECVIRTCHILD, and

ESETCONTEXT), and handling conflicts due to concurrent operations (ETRACKC).

Using these instructions, VMMs can both significantly reduce the overhead and complexity of oversubscribing the EPC. Thus, enabling EPC oversubscription without guest co-operation.

7 ACKNOWLEDGEMENTS

The authors of this paper wish to acknowledge the contributions of many hardware and software architects and designers who have worked in developing this innovative technology.

8 REFERENCES

[1] A. Abernathy, "Hyper-V Dynamic Memory vs. VMware Memory Overcommitment - Another Reason to Use Microsoft for VDI," http://blog.unidesk.com/hyper-v-dynamic-memory-vs-vmware-memory-overcommittment-vdi. [Accessed 7 April 2017]

[2] I. Banerjee, F. Guo, R. Venkatasubramanian, "Memory Overcommitment in ESX Server" VMware® Technical Journal, Summer 2013.

[3] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas, V. Phegade and J. Del Cuvillo, "Using Innovative Instructions to Create Trustworthy Software Solutions," in *HASP*, Israel, 2013.

[4] Intel® Corp., "Intel® 64 and IA-32 Architectures Software Developer's Manual," March 2017. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf. [Accessed 10 May 2017].

[5] David Kaplan, Jeremy Powell, Tom Woller, "AMD Memory Encryption," http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf. [Accessed 7 April 2017]

[6] Dan Magenheimer, "Memory Overcommit... without the commitment," in *Xen Summit*, 2008. https://oss.oracle.com/projects/tmem/dist/documentation/papers/overcommit.pdf. [Accessed 7 April 2017]

[7] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd and C. Rozas, "SGX Instructions to Support Dynamic Memory Allocation Inside an Enclave," in *HASP*, South Korea, 2016.

[8] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue and U. Savagaoankar, "Innovative Instructions and Software Model for Isolated Execution," in *HASP*, Israel, 2013.

[9] W.Zhao and Z. Wang, "Dynamic Memory Balancing for Virtual Machines," in Virtual Execution Environments, 2009.

9 Disclaimer