Software Guard eXtensions (SGX)

Overview

Software Guard eXtensions (SGX) hardware enables for user space applications to set aside private memory regions of code and data:

- Privileged (ring-0) ENCLS functions orchestrate the construction of the regions.
- Unprivileged (ring-3) ENCLU functions allow an application to enter and execute inside the regions.

These memory regions are called enclaves. An enclave can be only entered at a fixed set of entry points. Each entry point can hold a single hardware thread at a time. While the enclave is loaded from a regular binary file by using ENCLS functions, only the threads inside the enclave can access its memory. The region is denied from outside access by the CPU, and encrypted before it leaves from LLC.

The support can be determined by

```
grep sgx /proc/cpuinfo
```

SGX must both be supported in the processor and enabled by the BIOS. If SGX appears to be unsupported on a system which has hardware support, ensure support is enabled in the BIOS. If a BIOS presents a choice between "Enabled" and "Software Enabled" modes for SGX, choose "Enabled".

Enclave Page Cache

SGX utilizes an *Enclave Page Cache (EPC)* to store pages that are associated with an enclave. It is contained in a BIOS-reserved region of physical memory. Unlike pages used for regular memory, pages can only be accessed from outside of the enclave during enclave construction with special, limited SGX instructions.

Only a CPU executing inside an enclave can directly access enclave memory. However, a CPU executing inside an enclave may access normal memory outside the enclave.

The kernel manages enclave memory similar to how it treats device memory.

Enclave Page Types

SGX Enclave Control Structure (SECS)
: Enclave's address range, attributes and other global data are defined by this structure.

Regular (REG)
: Regular EPC pages contain the code and data of an enclave.

Thread Control Structure (TCS)
: Thread Control Structure pages define the entry points to an enclave and track the execution state of an enclave thread.

Version Array (VA)
: Version Array pages contain 512 slots, each of which can contain a version number for a page evicted from the EPC.

Enclave Page Cache Map

The processor tracks EPC pages in a hardware metadata structure called the *Enclave Page Cache Map (EPCM)*. The EPCM contains an entry for each EPC page which describes the owning enclave, access rights and page type among the other things.

EPCM permissions are separate from the normal page tables. This prevents the kernel from, for instance, allowing writes to data which an enclave wishes to remain read-only. EPCM permissions may only impose additional restrictions on top of normal x86 page permissions.

For all intents and purposes, the SGX architecture allows the processor to invalidate all EPCM entries at will. This requires that software be prepared to handle an EPCM fault at any time. In practice, this can happen on events like power transitions when the ephemeral key that encrypts enclave memory is lost.

Application interface

Enclave build functions

In addition to the traditional compiler and linker build process, SGX has a separate enclave “build” process. Enclaves must be built before they can be executed (entered). The first step in building an enclave is opening the **/dev/sgx_enclave** device. Since enclave memory is protected from direct access, special privileged instructions are then used to copy data into enclave pages and establish enclave page permissions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\x86\(linux-master)(Documentation)(x86)sgx.rst, line 97)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: arch/x86/kernel/cpu/sgx/ioctl.c
   :functions: sgx_ioc_enclave_create
               sgx_ioc_enclave_add_pages
               sgx_ioc_enclave_init
               sgx_ioc_enclave_provision
```

Enclave vDSO

Entering an enclave can only be done through SGX-specific EENTER and ERESUME functions, and is a non-trivial process. Because of the complexity of transitioning to and from an enclave, enclaves typically utilize a library to handle the actual transitions. This is roughly analogous to how glibc implementations are used by most applications to wrap system calls.

Another crucial characteristic of enclaves is that they can generate exceptions as part of their normal operation that need to be handled in the enclave or are unique to SGX.

Instead of the traditional signal mechanism to handle these exceptions, SGX can leverage special exception fixup provided by the vDSO. The kernel-provided vDSO function wraps low-level transitions to/from the enclave like EENTER and ERESUME. The vDSO function intercepts exceptions that would otherwise generate a signal and return the fault information directly to its caller. This avoids the need to juggle signal handlers.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\x86\(linux-master)(Documentation)(x86)sgx.rst, line 123)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: arch/x86/include/uapi/asm/sgx.h
   :functions: vdso_sgx_enter_enclave_t
```

ksgxd

SGX support includes a kernel thread called *ksgxd*.

EPC sanitization

ksgxd is started when SGX initializes. Enclave memory is typically ready for use when the processor powers on or resets. However, if SGX has been in use since the reset, enclave pages may be in an inconsistent state. This might occur after a crash and kexec() cycle, for instance. At boot, ksgxd reinitializes all enclave pages so that they can be allocated and re-used.

The sanitization is done by going through EPC address space and applying the EREMOVE function to each physical page. Some enclave pages like SECS pages have hardware dependencies on other pages which prevents EREMOVE from functioning. Executing two EREMOVE passes removes the dependencies.

Page reclaimer

Similar to the core kswapd, ksgxd, is responsible for managing the overcommitment of enclave memory. If the system runs out of enclave memory, *ksgxd* "swaps" enclave memory to normal memory.

Launch Control

SGX provides a launch control mechanism. After all enclave pages have been copied, kernel executes EINIT function, which initializes the enclave. Only after this the CPU can execute inside the enclave.

EINIT function takes an RSA-3072 signature of the enclave measurement. The function checks that the measurement is correct and signature is signed with the key hashed to the four **IA32_SGXLEPUBKEYHASH{0, 1, 2, 3}** MSRs representing the SHA256 of a public key.

Those MSRs can be configured by the BIOS to be either readable or writable. Linux supports only writable configuration in order to give full control to the kernel on launch control policy. Before calling EINIT function, the driver sets the MSRs to match the enclave's signing key.

Encryption engines

In order to conceal the enclave data while it is out of the CPU package, the memory controller has an encryption engine to transparently encrypt and decrypt enclave memory.

In CPUs prior to Ice Lake, the Memory Encryption Engine (MEE) is used to encrypt pages leaving the CPU caches. MEE uses a n-

ary Merkle tree with root in SRAM to maintain integrity of the encrypted data. This provides integrity and anti-replay protection but does not scale to large memory sizes because the time required to update the Merkle tree grows logarithmically in relation to the memory size.

CPUs starting from Icelake use Total Memory Encryption (TME) in the place of MEE. TME-based SGX implementations do not have an integrity Merkle tree, which means integrity and replay-attacks are not mitigated. B, it includes additional changes to prevent cipher text from being returned and SW memory aliases from being created.

DMA to enclave memory is blocked by range registers on both MEE and TME systems (SDM section 41.10).

Usage Models

Shared Library

Sensitive data and the code that acts on it is partitioned from the application into a separate library. The library is then linked as a DSO which can be loaded into an enclave. The application can then make individual function calls into the enclave through special SGX instructions. A run-time within the enclave is configured to marshal function parameters into and out of the enclave and to call the correct library function.

Application Container

An application may be loaded into a container enclave which is specially configured with a library OS and run-time which permits the application to run. The enclave run-time and library OS work together to execute the application when a thread enters the enclave.

Impact of Potential Kernel SGX Bugs

EPC leaks

When EPC page leaks happen, a WARNING like this is shown in dmesg:

"EREMOVE returned ... and an EPC page was leaked. SGX may become unusable..."

This is effectively a kernel use-after-free of an EPC page, and due to the way SGX works, the bug is detected at freeing. Rather than adding the page back to the pool of available EPC pages, the kernel intentionally leaks the page to avoid additional errors in the future.

When this happens, the kernel will likely soon leak more EPC pages, and SGX will likely become unusable because the memory available to SGX is limited. However, while this may be fatal to SGX, the rest of the kernel is unlikely to be impacted and should continue to work.

As a result, when this happpens, user should stop running any new SGX workloads, (or just any new workloads), and migrate all valuable workloads. Although a machine reboot can recover all EPC memory, the bug should be reported to Linux developers.

Virtual EPC

The implementation has also a virtual EPC driver to support SGX enclaves in guests. Unlike the SGX driver, an EPC page allocated by the virtual EPC driver doesn't have a specific enclave associated with it. This is because KVM doesn't track how a guest uses EPC pages.

As a result, the SGX core page reclaimer doesn't support reclaiming EPC pages allocated to KVM guests through the virtual EPC driver. If the user wants to deploy SGX applications both on the host and in guests on the same machine, the user should reserve enough EPC (by taking out total virtual EPC size of all SGX VMs from the physical EPC size) for host SGX applications so they can run with acceptable performance.

Architectural behavior is to restore all EPC pages to an uninitialized state also after a guest reboot. Because this state can be reached only through the privileged `ENCLS[EREMOVE]` instruction, `/dev/sgx_vepc` provides the `SGX_IOC_VEPC_REMOVE_ALL` ioctl to execute the instruction on all pages in the virtual EPC.

`EREMOVE` can fail for three reasons. Userspace must pay attention to expected failures and handle them as follows:

1. Page removal will always fail when any thread is running in the enclave to which the page belongs. In this case the ioctl will return `EBUSY` independent of whether it has successfully removed some pages; userspace can avoid these failures by preventing execution of any vcpu which maps the virtual EPC.
2. Page removal will cause a general protection fault if two calls to `EREMOVE` happen concurrently for pages that refer to the same "SECS" metadata pages. This can happen if there are concurrent invocations to `SGX_IOC_VEPC_REMOVE_ALL`, or if a `/dev/sgx_vepc` file descriptor in the guest is closed at the same time as `SGX_IOC_VEPC_REMOVE_ALL`; it will also be reported as `EBUSY`. This can be avoided in userspace by serializing calls to the ioctl() and to close(), but in general it should not be a problem.
3. Finally, page removal will fail for SECS metadata pages which still have child pages. Child pages can be removed by executing `SGX_IOC_VEPC_REMOVE_ALL` on all `/dev/sgx_vepc` file descriptors mapped into the guest. This means that the ioctl() must be called twice: an initial set of calls to remove child pages and a subsequent set of calls to remove SECS pages.

The second set of calls is only required for those mappings that returned a nonzero value from the first call. It indicates a bug in the kernel or the userspace client if any of the second round of `SGX_IOC_VEPC_REMOVE_ALL` calls has a return code other than 0.