An Introduction to Intel Software Guard Extensions (SGX)

33C3

envy

December 28, 2016

Intel Software Guard Extensions (SGX)

- Extension of the x86 instruction set
- Introduced with the Skylake architecture
- Allows creation of *enclaves*
  - I.e. isolated compartments
- Enclaves are meant to process sensitive data securely

→ Small, tailored application parts can be enclavised

- All enclave memory is encrypted
- Enclave integrity is verified after creation by SGX
- Needed trust is reduced to Intel and the CPU package

Use Cases - Three Sides

I want to run software on my system

- Probably don't need SGX

Use Cases - Three Sides

I want to run software somewhere else securely

- I want to use a more powerful machine
- I don't trust the remote system/provider

- Computation without data disclosure
- Enclave contains my ssh server and host keys on my VPS
- Contains my luks master key
  - All crypto has to go through the enclave

→ *I* can do stuff without the provider seeing what I do

Use Cases - Three Sides

Someone else wants to run software on my system

- A remote party does not trust me
- Proprietary software can generate and hide secrets from me
- Can be used to implement rights management
  - e.g. only able to start software x times
  - or use feature y only x times

→ *Someone* can do stuff without me seeing what they do

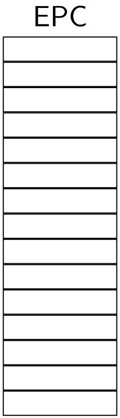
Enclave Overview

- Enclaves are part of normal applications
  - but even less privileged
- Only runnable in user space, no kernel enclaves
- No system calls allowed
- Some instructions not allowed
  - e.g. `cpuid`
- No direct calls into the enclave allowed
- No direct calls out of the enclave allowed

→ Secure computation in cooperation with untrusted software

Enclave Page Cache (EPC)

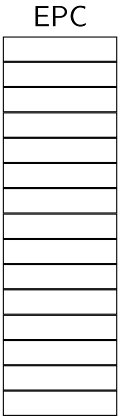
- Memory for all enclaves
- In current implementations part of system memory
  - Mapped as processor reserved memory
- Max 128 MB ($2^{15} = 32768 \times 4$K pages)



Enclave Page Cache (EPC)

- EPC is always encrypted
- Memory Encryption Engine inside CPU
- Encryption key is generated on boot
- Held inside the CPU
- Regenerated after sleep

→ Enclaves do not survive hibernation and standby

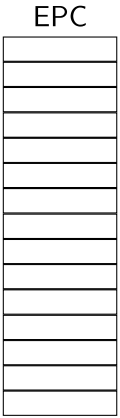


Enclave Page Cache (EPC)

- EPC is always encrypted
- Memory Encryption Engine inside CPU
- Encryption key is generated on boot
- Held inside the CPU
- Regenerated after sleep

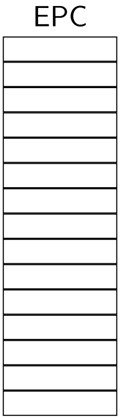
→ Enclaves do not survive hibernation and standby

- No idea if Intel ME can read it



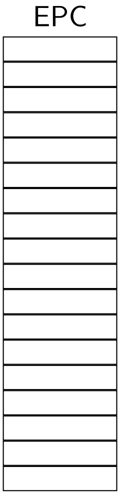
Enclave Page Cache (EPC)

- Swapping pages from EPC to memory is supported
- One EPC page is needed to hold version information of swapped out pages
- These can be swapped out, too
- High performance cost



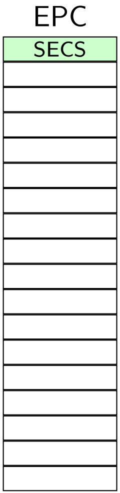
Enclave Page Cache Map (EPCM)

- Tracks state of all pages in EPC
- Is a "micro-architectural" data structure
  - So probably inside the processor package
- Contains information like
  - Valid mapping
  - Permissions
  - Page type
- Size of EPCM determines size of EPC



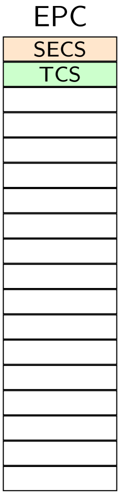
SGX Enclave Control Structure (SECS)

- One per enclave
- Contains most enclave metadata
  - Size, Hash, ...
- Inaccessible from untrusted and trusted side
  - Only by the processor itself
- Immutable after creation



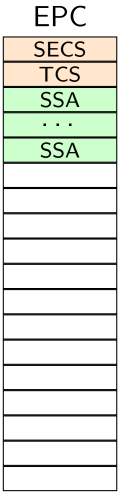
Thread Control Structure (TCS)

- Enclaves must have at least one TCS
- Describes an *entry point* into the enclave
- Multithreading is supported by SGX
- # of TCS = # of enclave threads
- Inaccessible from untrusted and trusted side
  - Only by the processor itself
- Immutable after creation
- References the SSA



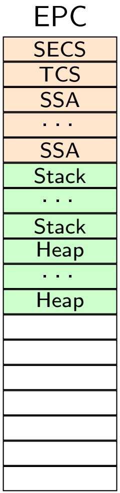
State Save Area (SSA)

- Saves the processor state during interrupts
- At least one SSA per TCS needed
- Written on interrupt
- Read on resume



Enclave Stack and Heap

- Enclave should use its own stack and heap
- Not enforced by SGX
  - Recommended for obvious reasons
- `rsp` and `rbp` are saved on enter
  - But not changed!
  - Restored on exit
- Nothing is done for the heap
  - You need your own allocator



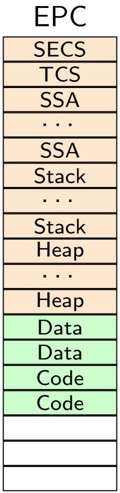
Code and data pages

- Your code and data goes here
- Not encrypted before creation
  - But integrity checked
- Enclave code and data is public until startup

→ Enclaves cannot have secrets during startup

- Have to inject them later

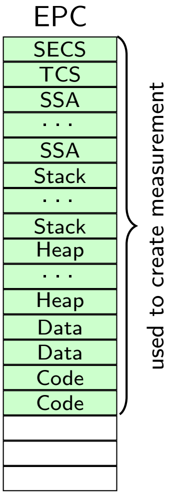
- Remaining pages can be used by other enclaves



Enclave Measurement

Each enclave has its own unique hash sum comprised of its page layout and contents

- *Measurement* of the enclave.
- Two enclaves with the same measurement are the same enclave
- Often called `MRENCLAVE`
- Used for integrity protection



Enclave Development

- No special compiler
- Enclave must be self contained
  - Statically linked
  - No system calls
  - `-nostdinc -nostdlib -nodefaultlibs -nostartfiles`
- Developer must create a `SIGSTRUCT` and `EINITTOKEN`
  - Contains the `MRENCLAVE` and is signed
- `EINITTOKEN` is signed by an Intel enclave
- Enter and exit through special instructions

Enclave Development

- No special compiler
- Enclave must be self contained
  - Statically linked
  - No system calls
  - `-nostdinc -nostdlib -nodefaultlibs -nostartfiles`
- Developer must create a `SIGSTRUCT` and `EINITTOKEN`
  - Contains the `MRENCLAVE` and is signed
- `EINITTOKEN` is signed by an Intel enclave
- Enter and exit through special instructions
- There exists a SDK to help you out

Instruction Overview

Enclave lifecycle instructions

ECREATE	Starts the enclave creation process
EADD	Adds pages to an enclave during creation
EEXTEND	Calculates the checksum of newly added pages
EINIT	Finalize the creation process
EENTER	Enter an enclave
EEXIT	Exit an enclave
ERESUME	Resume an enclave

kernel mode   user mode

Enclave Creation

Enclave creation is handled by the kernel

1. Create enclave with `ECREATE`
   - Creates SECS
2. Add pages with `EADD`
   - TCS, SSA and all other pages
3. Update hash sum with `EEXTEND`
4. Repeat 2 and 3 until all pages are added
5. Finalize creation with `EINIT`

`EINIT` will check the generated measurement with the signed measurement in `EINITTOKEN`.
The enclave will only be launched, if they match.

Enter and Exit

Entering an enclave can only be done from user space

- Call `EENTER`
- Specify which TCS to use

To exit an enclave synchronously

- Call `EEXIT`
- SGX does not clear registers for you!

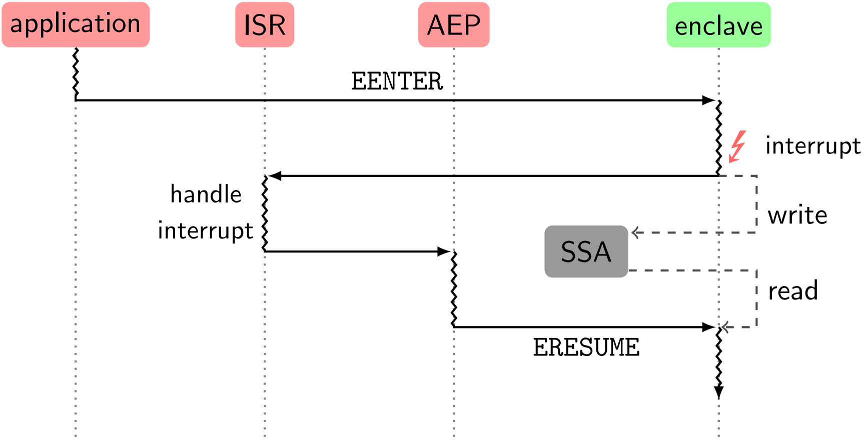
Enclave and Interrupts

Interrupts are transparent to enclaves

- They generate asynchronous enclave exits (AEX)
- On enter, an asynchronous exit handler (AEP) is registered
- AEP is called every time an AEX occurs
- AEP decides to resume enclave or not
  - By either calling `ERESUME` or ...not

The same mechanism is used if an exception or fault occurs inside the enclave

Handling an AEX



AEX and SSA

- On AEX, the state is saved in the SSA
- On `ERESUME` the last saved state is read
- TCS contains a counter that tracks the current SSA frame
  - `cssa`: current SSA frame
  - `nssa`: maximum number of SSA frames
- The AEP could reenter the enclave with `EENTER`
  - Subsequent AEX would create a new SSA frame

  → SSA works like a stack

Attestation

- Enclaves can prove to us, that they are enclaves
  - and that they are *the* enclave they claim to be
- This allows for secure communication with enclaves

- Local Attestation
  - Between two enclaves on the same system
- Remote Attestation
  - Between an enclave and a remote party

Local Attestation

- `EREPORT` instruction for generating reports
  - MAC'd by the processor
  - Made for a specific target enclave
- `EGETKEY` instruction to obtain report key
- Reports can have a payload
  - E.g. usable as a nonce for DH

→ Secure channel between two enclaves

Local Attestation

Enclaves Alice and Bob want to verify their identity to each other

1. Alice sends her `MRENCLAVE` to Bob
2. Bob calls `EREPORT` with Alice's `MRENCLAVE`
3. Bob sends the report to Alice
4. Alice verifies report
   - Using a report key obtained via `EGETKEY`
5. Repeat for other direction

Remote Attestation

- Special enclave called Quoting Enclave (QE)
  - Signed by Intel
- Creates a *quote* from a report

1. Enclave does local attestation with QE
2. Quote sent to remote party
3. Remote party asks Intel to verify quote

- Like reports, quotes can have a payload
  - E.g. usable as a nonce for DH

→ Secure channel between enclave and remote party

Sealing

- Enclave might want to store data persistently
- `EGETKEY` instruction to create a key based on either
  - `MRENCLAVE`
  - `MRSIGNER`
- Key also contains platform specific values

→ Same enclave on different platform gets different keys

1. Get key
2. Encrypt data
3. Give it to untrusted system to store it

You cannot trust the untrusted system, yet you need to give it the the sealed data and trust it to store them.

Sealing

Sealing with an `MRENCLAVE` key

- Only this enclave can read the data

Sealing with an `MRSIGNER` key

- All enclaves signed by the same signing key can read the data
- Allows for forward compatibility and enclave updates
  - Enclave v2 can read v1 sealed data

Monotonic counters

- SDK allows for monotonic counters
- Preserve state after enclave destruction

  Creating a monotonic counter (MC) involves writing to the non-volatile memory available in the platform. Repeated write operations could cause the memory to wear out during the normal lifecycle of the platform.

- I suspect the counters are managed by Intel ME
- They only work on Windows with installed ME drivers

SDK

- Windows and Linux SDK available
  - For C/C++ development
  - Ships with an in-enclave `libstdc`/`libstdcxx`
- Linux SDK is open source:
  - Driver `https://github.com/01org/linux-sgx-driver`
  - SDK `https://github.com/01org/linux-sgx/`
  - Binaries `https://01.org/intel-softwareguard-extensions`
- Windows SDK is not open source
  - They share the same in-enclave libc, so it's part open source
- SDK has an simulation mode, you don't need hardware
- Contains some samples

Enclave Interface

- SDK allows definition of ECalls and OCalls
  - ECalls are transitions from untrusted to trusted code
  - OCalls are the opposite

```
enclave {
  untrusted {
    void ocall_print_string([in, string] const char *str);
  };
  trusted {
    public void ecall_do_something(int foo,
        [in, size=len] uint8_t *bar, size_t len);
  };
};
```

See the Developer Reference for a detailed description

Graphene

- Library OS for Linux with SGX support
  - `https://github.com/oscarlab/graphene`

With the Intel SGX support, Graphene Library OS can secure a critical application in a hardware encrypted memory region. Graphene Library OS can protect applications against malicious system stack, with minimal porting effort.

→ Run legacy applications in enclaves

Attacks against SGX itself

- Rollback attacks against swapped out pages ✗
  - EPC page versions are tracked
- Rowhammer on the EPC ✗/?
  - EPC pages are protected by hashes stored in the EPCM
  - No idea when those are checked
- Critical bugs in the SGX *specification* ?
  - Someone would need to check, spec is open
  - Intel is fairly confident they don't have any
- Critical bugs in the SGX *implementation* ?
  - Someone would need to check
- Break into ME to break into SGX ?
  - Might break monotonic counters
  - ME might be able to read EPC unencrypted

Attacks against the enclave application

- Rollback attacks against sealed data ✓/?
  - Only between enclave versions
  - Not between sealed blobs
- Iago and TOCTTOU attacks ?
  - Depends on the implementation of the application

Attacks against the SDK

- Linux SDK does not use AES-NI ?
  - Implementation might be vulnerable to side channels
- TOCTTOU attacks ✗/?
  - SDK copies `ECall` arguments, if desired
  - Does not deep-copy structures
- SDK is open source, go find bugs!

Attacks

Secret extraction via side channels

- Page access tracking
  - Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems
  - `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ctrlchannels-oakland-2015.pdf`
- Branch history
  - Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing
  - `https://arxiv.org/pdf/1611.06952v2.pdf`

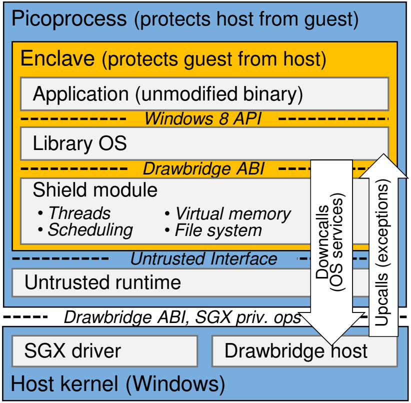
Attacks

Bug exploitation in enclaves

- Exploit synchronization bugs in enclaves
  - AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves
  - `https://www.ibr.cs.tu-bs.de/users/weichbr/papers/esorics2016.pdf`

Haven

Shielding Applications from an Untrusted Cloud with Haven[1]



[1] https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-baumann.pdf

Haven

Shielding Applications from an Untrusted Cloud with Haven[1]

- OSDI 2014, Microsoft Research
- SQL Server and Apache httpd
- They say low performance overhead
- Assume whole working set fits in enclave

→ With 128MB EPC limit unrealistic

- Not available

[1] https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-baumann.pdf

VC3

VC3: Trustworthy Data Analytics in the Cloud using SGX[2]

- IEEE Security & Privacy 2015, Microsoft Research
- Trusted MapReduce
- Encrypted map and reduce functions
- Small TCB
- Not available

[2] http://www.ieee-security.org/TC/SP2015/papers-archived/6949a038.pdf

SCONE

SCONE: Secure Linux Containers with Intel SGX[3]

- OSDI 2016, TU Dresden
- Docker meets SGX
- Smaller TCB than Haven, only modified libc
- Might be available next year

[3] http://www.ieee-security.org/TC/SP2015/papers-archived/6949a038.pdf

SecureKeeper

SecureKeeper: Confidential ZooKeeper using Intel SGX[4]

- Middleware 2016, TU Braunschweig
- Zookeeper coordination service on untrusted clouds
- Built with the SDK
- Low performance overhead
- Will be available soon
  - `https://github.com/sereca/SecureKeeper`

[4] `https://www.ibr.cs.tu-bs.de/users/brenner/papers/2016-middleware-brenner-securekeeper.pdf`

?

Your software here?

But please do not built DRM system

SGX v2

- Ability to dynamically add pages to a running enclave

→ Dynamically add threads to an enclave

→ Increase enclave heap during runtime

Availability unknown, maybe with Cannonlake/Coffeelake?

Sources

- Intel SGX Programming Reference (Oct. 2014) (https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf)
- Intel SGX Developer Reference (https://download.01.org/intel-sgx/linux-1.6/docs/Intel_SGX_SDK_Developer_Reference_Linux_1.6_Open_Source.pdf)
- Innovative Technology for CPU Based Attestation and Sealing (https://software.intel.com/sites/default/files/article/413939/hasp-2013-innovative-technology-for-attestation-and-sealing.pdf)

Sources

- SGX Secure Enclaves in Practice (https://www.blackhat.com/docs/us-16/materials/us-16-Aumasson-SGX-Secure-Enclaves-In-Practice-Security-And-Crypto pdf)
- Intel SGX Explained (118 p.) (https://eprint.iacr.org/2016/086.pdf)