Intel SGX

Mengjia Yan
Fall 2020

Based on slides of Intel SGX Tutorial
Recap: Address Translation

Virtual Address Space (Programmer's View)

Physical Address Space (limited by DRAM size)

6.888 L3 - Intel SGX
Recap: Address Translation

Virtual Address Space (Programmer's View)

Page Table per process

Physical Address Space (limited by DRAM size)
Recap: Address Translation

Virtual Address Space (Programmer's View)

- 4KB
- 4KB

Page Table per process

Physical Address Space (limited by DRAM size)

- 4KB

System software handles “page fault”
Recap: Address Translation

Virtual Address Space (Programmer's View)

Physical Address Space
(limited by DRAM size)

Page Table per process

System software handles "page fault"
Recap: Address Translation

Virtual Address Space (Programmer's View)

Page Table per process

Physical Address Space (limited by DRAM size)

System software handles “page fault”
Recap: Process Isolation

Virtual Address Space (Programmer's View)

- Process 1
- Process 2

Page Table per process

Physical Address Space (limited by DRAM size)

4KB

6.888 L3 - Intel SGX
If two pages have the same content, use page de-duplication to reduce memory footprint.
Page De-duplication and Copy-on-write

If two pages have the same content, use page de-duplication to reduce memory footprint.

Physical Address Space (limited by DRAM size)

Virtual Address Space (Programmer's View)
Page De-duplication and Copy-on-write

Upon write access, copy the page and change the mapping.

Virtual Address Space (Programmer's View)

Physical Address Space (limited by DRAM size)

4KB (CoW)
Page De-duplication and Copy-on-write

Upon write access, copy the page and change the mapping.

Virtual Address Space (Programmer's View)
Page De-duplication and Copy-on-write

Upon write access, copy the page and change the mapping.

Virtual Address Space (Programmer's View)

4KB

4KB

4KB (copied)

Physical Address Space (limited by DRAM size)

4KB

Page Table per process

VA

PA

6.888 L3 - Intel SGX
Recap: Secure Boot

- Static root of trust for measurement (SRTM)

TPM + firmware

1. Measure (hash)

Boot Loader

2. Report (extend)

OS kernel

3. Load

Compared to expected values locally or submitted to a remote attester.

PCR: platform configuration register
Software Attestation

• Defend against replay attack: Freshness
Software Attestation

• Defend against replay attack: Freshness

Processor Chip (w/ BIOS, OS, Apps)

- TPM
  \( K_{pri} \)

\( \text{sign} \)

Measurement List + nonce

Verifier

\( K_{pub} \)

How to know this key belongs to a specific TPM?

nonce
Software Attestation

• Need public key infrastructure
Software Attestation

• Need public key infrastructure

Processor Chip (w/ BIOS, OS, Apps)

TPM
AIK_{pri}

sign
Measurement List + nonce

Verifier

Attestation Identity Key

AIK_{pub}

Works as Certificate Agent

Chip Manufactory

Root Key

RK_{pri}  RK_{pub}
Software Attestation

- Need public key infrastructure

Processor Chip (w/ BIOS, OS, Apps)

Send AIK_{pri} to TPM using secure communication protocol

Chip Manufactory

Root Key

RK_{pri}  RK_{pub}

Attestation Identity Key

AIK_{pub}

Verifier

Measurement List + nonce

 TPM

AIK_{pri}
Software Attestation

• Need public key infrastructure

Processor Chip (w/ BIOS, OS, Apps)

Send $AIK_{pri}$ to TPM using secure communication protocol

Chip Manufactory

Root Key

$RK_{pri}$  $RK_{pub}$

Work as Certificate Agent

Attestation Identity Key

$AIK_{pub}$

measurement List + nonce

Verifier

6.888 L2 - Secure Processors in Industry
Intel TXT, AMD PSP, Google Titan

Intel TXT Dynamic trust of measurement

Security Vulnerabilities of Using TPM

• Vulnerable to bus tapping attacks

• TPM Reset attacks
  • SW reports hash values

• Bugs in the trusted software

Han et al. A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping. Usenix Security’18
Wojtczuk et al. Attacking Intel TXT® via SINIT code execution hijacking. 2011
So Far ……

### Secure Processors in Industry

- **Guest OS**
- **App**
- **Hypervisor**
- **SMM (firmware)**

**Ring 3**
- App

**Ring 0**
- Host OS

**SMM**
- Hardware

**Ring -1**
- Hypervisor

**Ring -2**
- SMM (firmware)

**Ring 3**
- Guest OS

**Ring 0**
- Guest OS

**Hardware**
So Far ……

The trend: shrink TCB. Why?

Host OS

Guest OS

App

Ring 3

Ring 0

SMM

Hardware

App

Guest OS

Ring 0

Ring -1

Hypervisor

Ring -2

SMM (firmware)

Hardware

Trusted

6.888 L2 - Secure Processors in Industry
Why Shrink TCB?

• Software bugs
  • SMM-based rootkits
  • Xen 150K LOC, 40+ vulnerabilities per year
  • Monolithic kernel, e.g., Linux, 17M LOC, 100+ vulnerabilities per year

• Remote Computing
  • Remote computer and software stack owned by an untrusted party
  • Examples
Why Shrink TCB?

• Software bugs
  • SMM-based rootkits
  • Xen 150K LOC, 40+ vulnerabilities per year
  • Monolithic kernel, e.g., Linux, 17M LOC, 100+ vulnerabilities per year

• Remote Computing
  • Remote computer and software stack owned by an untrusted party
  • Examples

Shrink HW TCB?
Secure Remote Computing

• Example: Video processing
Shrink Trusted Computing Base (TCB)

Ring 3
- App

Ring 0
- Guest OS

Ring -1
- Hypervisor

Ring -2
- SMM

Hardware

Trusted
Shrink Trusted Computing Base (TCB)

Arm TrustZone

6.888 L2 - Secure Processors in Industry
Shrink Trusted Computing Base (TCB)

6.888 L2 - Secure Processors in Industry
Shrink Trusted Computing Base (TCB)

Arm TrustZone

Intel SGX

AMD SEV

Guest OS

App

Hypervisor

SMM

Hardware

Trusted

enclave

6.888 L2 - Secure Processors in Industry
Arm TrustZone

from Hua et al. vTZ: Virtualizing ARM TrustZone. Usenix’17
Arm TrustZone

Cache is dynamically partitioned by tagging each cache line with a security bit.

from Hua et al. vTZ: Virtualizing ARM TrustZone. Usenix’17
Recommend not to have secret data in DRAM.

But just recommendation.

Cache is dynamically partitioned by tagging each cache line with a security bit.
Privileged Software Attacks

• Manipulate everything
Privileged Software Attacks

• Manipulate everything

• Directly see and modify application code and data
Privileged Software Attacks

- Manipulate everything
- Directly see and modify application code and data
  - Need to encrypt secret data
  - Need to verify integrity (software attestation)
Privileged Software Attacks

• Manipulate everything

• Directly see and modify application code and data
  → Need to encrypt secret data
  → Need to verify integrity (software attestation)

• Mess up with
  • Address translation
  • Process initialization and context switch
  • Interrupts, I/Os
  • etc.

6.888 L3 - Intel SGX
Enclave High-level View

• Goal: A protected environment that contains the code and data of a security-sensitive computation.
Enclave High-level View

• Goal: A protected environment that contains the code and data of a security-sensitive computation.

6.888 L3 - Intel SGX
Enclave High-level View

• Goal: A protected environment that contains the code and data of a security-sensitive computation.
Enclave High-level View

- **Goal:** A protected environment that contains the code and data of a security-sensitive computation.

![Diagram of Enclave High-level View]

- **Ring 0:** Guest OS
- **Ring -1:** Hypervisor
- **Ring -2:** SMM
- **Ring 3:** App

**Isolation**

**Hardware**

**Processor Reserved Memory (PRM)**

6.888 L3 - Intel SGX
Enclave High-level View

• Goal: A protected environment that contains the code and data of a security-sensitive computation.

Problem: How to handle address translation, context switch, etc.?
SGX HW TCB
SGX HW TCB

Processor Chip (socket)

- core L1/L2
- core L1/L2
- LLC

Processor Chip (socket)

- core L1/L2
- core L1/L2
- LLC

System Bus (logically)

Memory (DRAM)

Non-volatile storage device

other I/O Device

6.888 L3 - Intel SGX
SGX HW TCB

- Processor Chip (socket)
  - core L1/L2
  - core L1/L2
  - LLC

- Memory Management Unit (MMU)

- Integrated Memory Controller

- System Bus (logically)

- Memory (DRAM)

- Non-volatile storage device

- other I/O Device

6.888 L3 - Intel SGX
SGX HW TCB

Processor Chip (socket)

Memory Management Unit (MMU)

Integrated Memory Controller

Memory (DRAM)

Non-volatile storage device

other I/O Device

Low impact on chip’s HW design

System Bus (logically)
Intel SGX Security Mechanisms

Isolation
- Ring 3
  - App
  - Enclave
- Ring 0
  - Guest OS
  - Guest OS
- Ring -1
  - Hypervisor
- Ring -2
  - SMM
  - Hardware
  - Processor Reserved Memory (PRM)

Attestation
- Client Application
- Enclave
- Remote Platform

DRAM Protection
- Processor Chip (socket)
  - core L1/L2
  - core L1/L2
  - ... (LLC)
- Integrated Memory Controller
- Memory (DRAM)
SGX Access Control

- Assume software attestation is done
- Can have multiple enclaves
SGX Access Control

• Assume software attestation is done
• Can have multiple enclaves

Performance issues.

PRM size is 128MB in SGX V1.0

All enclaves loaded at the same time cannot exceed said ~90MB
Enclave Address Translation

Virtual Address Space (Programmer’s View)

Page Table per process

Physical Address Space (limited by DRAM size)

4KB

VA

PA

4KB
Enclave Address Translation

Virtual Address Space (Programmer's View)

Enclave Linear Range (ELRANGE)

4KB

Page Table per process

VA

Physical Address Space (limited by DRAM size)

PA

4KB

Processor Reserved Memory (PRM)
Enclave Address Translation

Virtual Address Space (Programmer's View)

Physical Address Space (limited by DRAM size)

Page Table per process

VA
PA

4KB

4KB

Enclave Linear Range (ELRANGE)

Processor Reserved Memory (PRM)

4KB

4KB
Enclave Address Translation

Virtual Address Space (Programmer's View)

- Enclave Linear Range (ELRANGE)
- 4KB
- 4KB

Page Table per process

Physical Address Space (limited by DRAM size)

- 4KB
- Processor Reserved Memory (PRM)

This is what we want. But OS is not trustable.

But OS is not trustable.
Malicious Address Translation

Virtual Address Space (Programmer's View)

Physical Address Space (limited by DRAM size)

4KB

Enclave Linear Range (ELRANGE)

Page Table per process

Processor Reserved Memory (PRM)

4KB

4KB

4KB
Virtual Address Space (Programmer's View)

<table>
<thead>
<tr>
<th>4KB</th>
<th>4KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enclave Linear Range (ELRANGE)</td>
<td>Processor Reserved Memory (PRM)</td>
</tr>
</tbody>
</table>

Physical Address Space (limited by DRAM size)

<table>
<thead>
<tr>
<th>4KB</th>
</tr>
</thead>
</table>

Page Table per process

if (PA belongs to PRM) {
  check whether in enclave mode
  if (NOT enclave access) {
    return a value 0xffffffff
  }
}

Easy to implement in MMU
Malicious Address Translation

Virtual Address Space (Programmer's View)

- 4KB
- Enclave Linear Range (ELRANGE)
- 4KB

Page Table per process

- VA

Physical Address Space (limited by DRAM size)

- 4KB
- 4KB
- Processor Reserved Memory (PRM)
- 4KB
Malicious Address Translation

Virtual Address Space (Programmer's View)

Enclave Linear Range (ELRANGE)

4KB

4KB

Physical Address Space (limited by DRAM size)

Page Table per process

VA

PA

4KB

4KB

Processor Reserved Memory (PRM)

4KB

if (in enclave mode) {
    compare PA with RPM range
    if (NOT in RPM) {
        #Signal Fault
    }
}

Also Easy to implement in MMU
Malicious Address Translation

Virtual Address Space (Programmer's View)

Page Table per process

Physical Address Space (limited by DRAM size)

4KB

Enclave Linear Range (ELRANGE)

4KB

Processor Reserved

4KB (belong to a different enclave)

4KB

How to block such attacks?
SGX Memory Organization

- Keep page mapping metadata in PRM
- MMU performs extra checks
SGX Memory Organization

- Keep page mapping metadata in PRM
- MMU performs extra checks

- Enclave pages (code, data)
- Meta data per enclave
  - enclave page mapping information, enclave thread context information, etc.
Enclave Page Mapping Information

Virtual Address Space (Programmer's View)

- Enclave Linear Range (ELRANGE)

Physical Address Space (limited by DRAM size)

- Processor Reserved Memory (PRM)

Page Table per process

VA

PA

Enclave Page Cache Mapping (EPCM)

Stored in PRM

4KB

4KB

4KB

6.888 L3 - Intel SGX
Enclave Page Mapping Information

Virtual Address Space (Programmer's View)

- 4KB
- Enclave Linear Range (ELRANGE)
- 4KB

Page Table per process

- VA
- PA

Physical Address Space (limited by DRAM size)

- 4KB
- Processor Reserved Memory (PRM)
- 4KB

Enclave Page Cache Mapping (EPCM)

- Stored in PRM
- \{PA, VA, Enclave ID\}
Virtual Address Space (Programmer's View)

Enclave Linear Range (ELRANGE)

Page Table per process

VA

PA

Physical Address Space (limited by DRAM size)

4KB

Enclave Page Cache Mapping (EPCM)

Stored in PRM

{PA, VA, Enclave ID}

Processor Reserved Memory (PRM)

4KB

if (PA belongs to PRM) {
  compare VA in EPCM
  if (NOT match) {
    #Signal Fault
  }
}

6.888 L3 - Intel SGX
Enclave Page Mapping Information

Virtual Address Space (Programmer's View)

4KB

Enclave Linear Range (ELRANGE)

Page Table per process

4KB

VA

PA

Enclave Page Cache Mapping (EPCM)

Stored in PRM

{PA, VA, Enclave ID}

Physical Address Space (limited by DRAM size)

4KB

Processor Reserved Memory (PRM)

4KB

Problem: pages are allocated and selected by system software.
So far ......

• Once the enclave is initialized correctly, it can be isolated from system software using
  • Hardware access control (supported by MMU)
  • Hardware support for secure context switch

• How to ensure the initialization is correct?
So far ……

• Once the enclave is initialized correctly, it can be isolated from system software using
  • Hardware access control (supported by MMU)
  • Hardware support for secure context switch

• How to ensure the initialization is correct?
  • Software Attestation (similar to secure boot)
Enclave Initialization

• BIOS setup PRM region
Enclave Initialization

- Enclave creation (ECREATE)
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)

Virtual Address Space

Physical Address Space

Enclave 1 metadata
Update mapping information in EPCM
Enclave Measurement

- Hardware generates a cryptographic log of the build process
  - Code, data, stack, and heap contents
  - Location of each page within the enclave
  - Security attributes (e.g., page permissions) and enclave capabilities

- Enclave identity (MRENCLAVE) is a 256-bit digest of the log that represents the enclave
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
- Init (EINIT)
  - Finalize measurement
- Active (EENTER)
  - Switch to enclave mode
Enclave Initialization

- Add page (EADD)
- Measure (EEXTEND)
- Init (EINIT)
  - Finalize measurement
- Active (EENTER)
  - Switch to enclave mode

Problem:
No measurement after EINIT
Enclave Attestation and Sealing

- HW based attestation provides evidence that “this is the right application executing on an authentic platform” (approach similar to secure boot attestation)
Enclave Attestation and Sealing

- HW based attestation provides evidence that “this is the right application executing on an authentic platform” (approach similar to secure boot attestation)

HW-signed blob that includes enclave identity information

trusted communication channel
Protect Memory

- Processor Chip (socket)
  - core
  - L1/L2
  - core
  - L1/L2
  - LLC

- System Bus (logically)

- Memory (DRAM)
- Non-volatile storage device
- other I/O Device
Confidentiality Protection with Encryption

• Secret key is stored inside chip
• For freshness, encrypt with nonce (counter)
• \{nonce, ciphertext\} per cache block are stored externally in DRAM
Integrity Protection with Hash

• For each cache line: \{ciphertext + nonce + hash\}
Integrity Protection with Hash

• For each cache line: \{ciphertext + nonce + hash\}

• Problem:
  • Need to store hashes or nonces on-chip $\rightarrow$ high on-chip storage requirement
  • Too much storage requirement (~64bits / block) $\rightarrow$ high off-chip storage requirement
Integrity Protection with Hash

• For each cache line: \{ciphertext + nonce + hash\}

• Problem:
  • Need to store hashes or nonces on-chip \(\rightarrow\) high on-chip storage requirement
  • Too much storage requirement (~64bits / block) \(\rightarrow\) high off-chip storage requirement

• General solution:
  • Integrity Tree (Merkle tree)
Operations on Merkle Tree

• Only need to store the root node on chip

```
root = Hash(zőzi || zözi+1)
f_i = Hash(gzi || gzi+1)
g_i = Hash(hzi || hzi+1)
h_i = Hash(B_i)
```

Secure processor (trusted)
Operations on Merkle Tree

- Only need to store the root node on chip
- How to verify block B1?
- Write to block B3?

\[
\text{root} = \text{Hash}(f_{2i} \ || \ f_{2i+1})
\]

\[
f_i = \text{Hash}(g_{2i} \ || \ g_{2i+1})
\]

\[
g_i = \text{Hash}(h_{2i} \ || \ h_{2i+1})
\]

\[
h_i = \text{Hash}(B_i)
\]
Next Lecture:
Side Channel Introduction