Confining (Un)Trusted Execution Environments

Michael Schwarz
November 20, 2019 - SILM

Graz University of Technology
• Sandboxes assume trusted system and untrusted application
• Sandboxes assume trusted system and untrusted application
→ Protects the system from harm
Motivation

- Sandboxes assume trusted system and untrusted application
  → Protects the system from harm
- Protect the application from the system?
Motivation

- Sandboxes assume trusted system and untrusted application
  - Protects the system from harm
- Protect the application from the system?
- Assumption: untrusted system, trusted application
Motivation

- Sandboxes assume trusted system and untrusted application
  - Protects the system from harm
  - Protect the application from the system?
  - Assumption: untrusted system, trusted application
  - Isolation of application
Motivation

- Applications for isolation:
  - Working with sensitive data (e.g., passwords, money)
  - Distrusting the cloud provider
  - Intellectual property (e.g., algorithms)
  - Rights management (DRM)
  - Ensures security even against active attacks
Applications for isolation:
- Working with sensitive data (e.g., passwords, money)
Applications for isolation:
- Working with sensitive data (e.g., passwords, money)
- Distrusting the cloud provider
Applications for isolation:
- Working with sensitive data (e.g., passwords, money)
- Distrusting the cloud provider
- Intellectual property (e.g., algorithms)
Applications for isolation:

- Working with sensitive data (e.g., passwords, money)
- Distrusting the cloud provider
- Intellectual property (e.g., algorithms)
- Rights management (DRM)
Applications for isolation:
- Working with sensitive data (e.g., passwords, money)
- Distrusting the cloud provider
- Intellectual property (e.g., algorithms)
- Rights management (DRM)

Ensures security even against active attacks
• Requires some form of hardware support
- Requires some form of **hardware support**
- Well-known isolation: user space - kernel space
- Requires some form of hardware support
- Well-known isolation: user space - kernel space
  → Protects OS against malicious applications
• Requires some form of hardware support
• Well-known isolation: user space - kernel space
  → Protects OS against malicious applications
  → Applications also mutually isolated
- Requires some form of **hardware support**
- Well-known isolation: user space - kernel space
  - Protects OS against **malicious applications**
  - Applications also **mutually isolated**
- Enforced by the hardware (→ page table)
- Secure area of a CPU
- Secure area of a CPU
- Integrity and confidentiality guarantees for code and data
- Secure area of a CPU
- Integrity and confidentiality guarantees for code and data
- Hardware still shared with other applications
- Secure area of a CPU
- **Integrity and confidentiality** guarantees for code and data
- Hardware still shared with other applications
- (Nearly) no performance impacts
Assumptions in TEEs:
Assumptions in TEEs:
- Attacker controls the OS
Assumptions in TEEs:
- Attacker controls the OS
- Only the CPU is trusted
Assumptions in TEEs:
- Attacker controls the OS
- Only the CPU is trusted
- TEE memory is encrypted and inaccessible to OS
Assumptions in TEEs:

- Attacker controls the OS
- Only the CPU is trusted
- TEE memory is encrypted and inaccessible to OS
- TEE has access to OS
• Implementations for various CPUs
Implementations for various CPUs
- Intel: Software Guard Extension (SGX) and Management Engine (ME)
- ARM and AMD: TrustZone
- Implementations for various CPUs
  - Intel: Software Guard Extension (SGX) and Management Engine (ME)
  - ARM and AMD: TrustZone
- Widely used in mobile phones
Real-World TEE Example

- Netflix uses Widevine DRM
Real-World TEE Example

- Netflix uses Widevine DRM
- DRM in TrustZone
• Netflix uses Widevine DRM
• DRM in TrustZone
• Video is directly drawn on screen
• Netflix uses Widevine DRM
• DRM in TrustZone
• Video is directly drawn on screen
• No app (not even root) can access video data
Execution Flow (SGX)

Application

Untrusted part

Operating System

Michael Schwarz (@misc0110) | Graz University of Technology
Execution Flow (SGX)

Application

Untrusted part

Create Enclave

Operating System
Execution Flow (SGX)

Untrusted part
- Create Enclave
- Call Trusted Fnc.

Trusted part
- Call Gate
- Trusted Fnc.

Operating System
Execution Flow (SGX)

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System

Michael Schwarz (@misc0110) | Graz University of Technology
Execution Flow (SGX)

Application

Untrusted part
- Create Enclave
- Call Trusted Fnc.

Trusted part
- Call Gate
- Trusted Fnc.
- Return

Operating System
Execution Flow (SGX)

Application

Untrusted part

Create Enclave

Call Truste Fnc.

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Michael Schwarz (@misc0110) | Graz University of Technology
Execution Flow (SGX)

Application

Untrusted part
- Create Enclave
- Call Trusted Fnc.
- ...

Trusted part
- Call Gate
- Trusted Fnc.
- Return

Operating System

Michael Schwarz (@misc0110) | Graz University of Technology
Application

Untrusted part
- Create Enclave
- Call Trusted Fnc.
  - ...

Trusted part
- Trusted Fnc.
- Return

Operating System

Michael Schwarz (@misc0110) | Graz University of Technology
SGX

intel®

SGX
• Enclaves are black boxes
What if?

- Enclaves are black boxes
- Protected from all applications and OS
What if?

- Enclaves are black boxes
- Protected from all applications and OS
- What if they contain malicious code?
What if?

- Enclaves are black boxes
- Protected from all applications and OS
- What if they contain malicious code?
- Can we hide zero days?
Monday, September 23, 2013

Thoughts on Intel's upcoming Software Guard Extensions (Part 2)

In the first part of this article published a few weeks ago, I have discussed the basics of Intel SGX technology, and also discussed challenges with using SGX for securing desktop systems, specifically focusing on the problem of trusted input and output. In this part we will look at some other aspects of Intel SGX, and we will start with a discussion of how it could be used to create a truly irreversible software.

SGX Blackboxing - Apps and malware that cannot be reverse engineered?

http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html
[...] Intel is aware of this research which is based upon assumptions that are outside the threat model for Intel SGX. The value of Intel SGX is to execute code in a protected enclave; however, Intel SGX does not guarantee that the code executed in the enclave is from a trusted source [...]
Classical exploits cannot be mounted within SGX:
Classical exploits cannot be mounted within SGX:

- No syscalls
Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries
Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries
- No interprocess communication
Classical exploits cannot be mounted within SGX:

- No syscalls
- No shared memory/libraries
- No interprocess communication
- Blocked instructions
State-of-the-art Malicious Enclaves

- Side-channel attacks from SGX [Sch+17]
State-of-the-art Malicious Enclaves

- Side-channel attacks from SGX [Sch+17]
- Fault attacks from SGX [Gru+18; Jan+17]
- Side-channel attacks from SGX [Sch+17]
- Fault attacks from SGX [Gru+18; Jan+17]
- No real exploits from SGX so far
Side-channel attacks from SGX possible
- Side-channel attacks from SGX possible
- Allow attacker to spy on meta data
Side-channel attacks from SGX possible
- Allow attacker to spy on meta data
- Completely hide an attack
- Cache attacks preventable on source level
Countermeasures

• Cache attacks preventable on source level
• Side-channel resistant crypto
- Cache attacks preventable on source level
- Side-channel resistant crypto
- Default in most crypto libraries
SGX Encrypted Memory

Physical Memory

EPC (128 MB)

0 GB

16 GB
• What happens if a bit flips in the EPC?
What happens if a bit flips in the EPC?

- Integrity check will fail!
What happens if a bit flips in the EPC?
- Integrity check will fail!
→ Locks up the memory controller
What happens if a bit flips in the EPC?
- Integrity check will fail!
  - Locks up the memory controller
  - Not a single further memory access!
What happens if a bit flips in the EPC?
- Integrity check will fail!
  - Locks up the memory controller
  - Not a single further memory access!
  - System halts immediately
If a malicious enclave induces a bit flip, ...
If a malicious enclave induces a bit flip, ...

...the entire machine halts
• If a malicious enclave induces a bit flip, ...
• …the entire machine halts
• …including co-located tenants
If a malicious enclave induces a bit flip, ...

...the entire machine halts

...including co-located tenants

Denial-of-Service Attacks in the Cloud [Gru+18; Jan+17]
Cells leak faster upon proximate accesses!
Rowhammer

Cells leak faster upon proximate accesses!

Michael Schwarz (@misc0110) | Graz University of Technology
Cells leak faster upon proximate accesses → Rowhammer
Rowhammer

Cells leak faster upon proximate accesses → Rowhammer

Michael Schwarz (@misc0110) | Graz University of Technology
Cells leak faster upon proximate accesses → Rowhammer
Rowhammer

Cells leak faster upon proximate accesses → Rowhammer
How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)
How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)

First believed to be safe
- We showed bit flips in 2016
- 67% affected (estimation 2016)
Zero Days?

- Dangerous attacks but difficult in practice
Zero Days?

- Dangerous attacks but **difficult** in practice
- More relevant: **zero days** in enclaves
Dangerous attacks but difficult in practice

More relevant: zero days in enclaves

→ Super malware
Attack Overview

Enclave

Code

Data

Stack

Read Primitive (TAP)

Gadget
Attack Overview

Enclave

Code

Data

Stack

Read Primitive (TAP)

Write Primitive (CLAW)

Enclave

Michael Schwarz (@misc0110) | Graz University of Technology
Attack Overview

- **Enclave**
- **Code**
- **Data**
- **Stack**

**Read Primitive** *(TAP)*

**Write Primitive** *(CLAW)*

**ROP injection**

**Cave**

**Gadget**

**Enclave**
Enclave

Data

Code

Stack

Cave

Gadget

Write Primitive (CLAW)

Read Primitive (TAP)

ROP injection

Enclave

Michael Schwarz (@misc0110) | Graz University of Technology
Attack Overview

Write Primitive (CLAW)
Read Primitive (TAP)
ROP injection

Enclave

Michael Schwarz (@misc0110) | Graz University of Technology
Attack Overview

- **Enclave**
  - **Code**
  - **Data**
    - **Read Primitive** \((TAP)\)
    - **Write Primitive** \((CLAW)\)
  - **Stack**
    - **ROP injection**
      - **execute**
      - **chain**

Michael Schwarz (@misc0110) | Graz University of Technology
- Enclave can access host memory...
- Enclave can access host memory...
- ...but crashes on invalid access
- Enclave can access host memory...
- ...but crashes on invalid access
- No syscall or exception handler available
- Intel TSX: hardware transactional memory
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction
Transactional Memory

- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction
- Conflict → abort and roll back
- Intel TSX: hardware transactional memory
- Multiple reads and writes are atomic
- Operations in a transaction
- Conflict $\rightarrow$ abort and roll back
- Faults are suppressed
Transactional Memory

Thread 0

xbegin

(Thread 0)

else path of xbegin

Cache

(Thread 1)

xend

Michael Schwarz (@misc0110) | Graz University of Technology
Transactional Memory

Thread 0

```
xbegin
  mov
  xend
else path of xbegin
```

Cache

```
data
read set
```

Thread 1

```
read
  mov
```

Michael Schwarz (@misc0110) | Graz University of Technology
Transactional Memory

Thread 0

xbegin
mov
mov
xend
else path of xbegin

Thread 1

Cache

read
read
mov
mov
read
write
mov
mov

read set
Transactional Memory

Thread 0

xbegin
mov
mov
mov
xend
else path of xbegin

read
read
read

Cache

data
data
data

read set

Thread 1

read
write
write
mov
mov
mov

- Segmentation fault is a fault
- Segmentation fault is a **fault**
- Suppressed in TSX transaction
- Segmentation fault is a fault
- Suppressed in TSX transaction
- Abort code → “don’t try again”
- Segmentation fault is a fault
- Suppressed in TSX transaction
- Abort code $\rightarrow$ “don’t try again”
- Valid page $\rightarrow$ transaction succeeds
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

{[Thumb-up]}

Michael Schwarz (@misc0110) | Graz University of Technology
## Host Memory

<table>
<thead>
<tr>
<th>Valid</th>
<th>Valid</th>
<th>Valid</th>
<th>Invalid</th>
<th>Invalid</th>
<th>Valid</th>
<th>Invalid</th>
<th>Invalid</th>
</tr>
</thead>
</table>

Click to select valid memory.
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

Michael Schwarz (@misc0110) | Graz University of Technology
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

[Diagram of hand pressing a button with a red 'X']
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

{[Thumb up]}
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

![Diagram of a hand touching invalid memory blocks]
Host Memory

| Valid | Valid | Valid | Invalid | Invalid | Valid | Invalid | Invalid |

[Diagram showing a touch gesture and an 'X' symbol]
- **Entire memory**: 45 min
- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds
- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds
- Undetectable by OS
- Entire memory: 45 min
- Start from saved RIP/RSP: few seconds
- Undetectable by OS
- Used to find ROP gadgets
- Write to mapped page...
- Write to mapped page...
- ...abort immediately
- Write to mapped page...
- ...abort immediately
→ No architectural write
- Write to mapped page...
- ...abort immediately
  $\rightarrow$ No architectural write
- Abort code $\rightarrow$ explicit or implicit
### Host Memory

<table>
<thead>
<tr>
<th>Access</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/O</td>
<td>X</td>
</tr>
<tr>
<td>R/O</td>
<td>N/A</td>
</tr>
<tr>
<td>R/O</td>
<td>N/A</td>
</tr>
<tr>
<td>R/W</td>
<td>N/A</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The diagram illustrates unauthorized access with a red 'X' symbol.
Host Memory

<table>
<thead>
<tr>
<th>R/O</th>
<th>R/O</th>
<th>R/O</th>
<th>N/A</th>
<th>N/A</th>
<th>R/W</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
</table>

{\{ \text{hand} \}}
### Host Memory

<table>
<thead>
<tr>
<th></th>
<th>R/O</th>
<th>R/O</th>
<th>R/O</th>
<th>N/A</th>
<th>N/A</th>
<th>R/W</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟠</td>
<td>🟢</td>
<td>🟢</td>
</tr>
</tbody>
</table>

Michael Schwarz (@misc0110) | Graz University of Technology
### Host Memory

<table>
<thead>
<tr>
<th>R/O</th>
<th>R/O</th>
<th>R/O</th>
<th>N/A</th>
<th>N/A</th>
<th>R/W</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
</table>

![Warning Symbol]

Michael Schwarz (@misc0110) | Graz University of Technology
- TAP+CLAW → find writable memory
- TAP+CLAW → find writable memory
  → Robust write-anything-anywhere primitive
- TAP+CLAW → find writable memory
  → Robust write-anything-anywhere primitive
  → Store malicious payload
1. **TAP**: find ROP gadgets
1. **TAP**: find ROP gadgets
2. **CLAW**: find writable memory (data cave)
1. **TAP**: find ROP gadgets
2. **CLAW**: find writable memory (data cave)
3. **Inject** ROP gadgets into host stack
1. **TAP**: find ROP gadgets
2. **CLAW**: find writable memory (data cave)
3. **Inject** ROP gadgets into host stack
4. **Profit!
Stack

- Saved RIP
- Saved RBP
- ...
- Saved RIP
- Saved RBP
- ...
- Original saved RIP
- Original saved RBP

leave; ret
Stack

- Saved RIP
- Saved RBP
- ...
- Saved RIP
- Saved RBP
- ...
- Original saved RIP
- Original saved RBP

Fake stack frame

```leave; ret```
Stack

- Saved RIP
- Saved RBP
- ...
- Saved RIP
- Saved RBP
- ...
- Original saved RIP
- Original saved RBP

Fake stack frame

ROP Chain

leave; ret
SGX ROP

Stack

Saved RIP
Saved RBP
...
Saved RIP
Saved RBP
...
Original saved RIP
Original saved RBP

Fake stack frame

Original saved RIP
ROP Chain
Original saved RBP
leave; ret

Michael Schwarz (@misc0110) | Graz University of Technology
Injected RIP

Saved RIP

Saved RBP

...

Saved RIP

Saved RBP

...

Injected RIP

Injected RBP

Fake stack frame

Original saved RIP

ROP Chain

Original saved RBP

leave; ret

Stack

SGX ROP
64.8 MB writable data
mprotect ROP gadgets

Several pages writable data
mprotect ROP gadgets
Remote attestation + dynamic loading → no emulation, no binary
- Remote attestation + dynamic loading → no emulation, no binary
- Host continues normally → (nearly) no traces
- Remote attestation + dynamic loading → no emulation, no binary
- Host continues normally → (nearly) no traces
- Trigger-based → plausible deniability
• Remote attestation + dynamic loading → no emulation, no binary
• Host continues normally → (nearly) no traces
• Trigger-based → plausible deniability
→ Securely and stealthily deploying zero days
mschwarz@t480sms2 /tmp/sgxrop % ./app
Call trace:

+---- foo enter
  +---- bar enter
  +---- enclave enter

[ENCLAVE] <Start @ 0xffffffff0000>
[ENCLAVE] <Saved RSP: 7ff082d4320>
[ENCLAVE] <Saved RBP: 7ff082d47e0>
[ENCLAVE] <Searching for stack frame...>
[ENCLAVE] <Stack frame @ 296: 556108debe4c0 / 7ff082d4920 (3d8d4800000be / 7ff082d4940)>
[ENCLAVE] <Stack frame @ 328: 556108debe541 / 7ff082d4a28 (4800200c48058d48 / 7ff082d61ee)>
[ENCLAVE] <RIP @ 0x7ff082d4928>
[ENCLAVE] <RBP @ 0x7ff082d4920>
[ENCLAVE] <Searching for gadgets...>
[ENCLAVE] <Found gadget [SYSCALL] @ 0x7ff083f47ec>
```plaintext
--- foo enter
   --- bar enter
      --- enclave enter

[ENCLAVE] <Start @ 0x7ffffff000>
[ENCLAVE] <Saved RSP: 7fff082d4320>
[ENCLAVE] <Saved RBP: 7fff082d47e0>
[ENCLAVE] <Searching for stack frame...>
[ENCLAVE] <Stack frame @ 296: 556108dce4c0 / 7fff082d4920 (3d8d48000000008e / 7fff082d4940)>
[ENCLAVE] <Stack frame @ 328: 556108dce541 / 7fff082d4a28 (4800200c48058d48 / 7fff082d61ee)>
[ENCLAVE] <RIP @ 0x7fff082d4928>
[ENCLAVE] <RBP @ 0x7fff082d4920>
[ENCLAVE] <Searching for gadgets...>
[ENCLAVE] <Found gadget [SYSCALL] @ 0x7fff083f47ec>
[ENCLAVE] <Found gadget [POP_RDI] @ 0x7fe81a457287>
[ENCLAVE] <Found gadget [POP_RSI] @ 0x7fe81a456167>
[ENCLAVE] <Found gadget [LEAVE] @ 0x7fe81a4409ea>
[ENCLAVE] <Found gadget [POP_RDX] @ 0x7fe81a228f7a>
[ENCLAVE] <Found gadget [POP_RAX] @ 0x7fe819fe6af4>
[ENCLAVE] <Found gadget [XCHG_RAX_RDI] @ 0x7fe819f3e8e5>
[ENCLAVE] <Searching for data cave...>
```
<Searching for stack frame...>
<Stack frame @ 296: 556108dbe4c0 / 7fff082d4920 (3d8d4800000008be / 7fff082d4940)>
<Stack frame @ 328: 556108dbe541 / 7fff082d4a28 (4800200c48058d48 / 7fff082d61ee)>
<RIP @ 0x7fff082d4928>
<RBP @ 0x7fff082d4920>
<Searching for gadgets...>
<Found gadget [SYSCALL] @ 0x7fff083f47ec>
<Found gadget [POP_RDI] @ 0x7fe81a457287>
<Found gadget [POP_RSI] @ 0x7fe81a456167>
<Found gadget [LEAVE] @ 0x7fe81a4409ea>
<Found gadget [POP_RDX] @ 0x7fe81a228f7a>
<Found gadget [POP_RAX] @ 0x7fe819f6a6f4>
<Found gadget [XCHG_RAX_RDI] @ 0x7fe819f3e8e5>
<Searching for data cave...>
<Cave @ 0x7fe81a43a000>
<Building ROP chain...>
<Payload ready!>

| ++++ enclave exit |
| ++++ bar exit |
| ++++ foo exit |

QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-mschwarz'
Searching for stack frame...

Stack frame @ 296: 556108dbe4c0 / 7fff082d4920 (3d8d4800000008be / 7fff082d4940)
Stack frame @ 328: 556108dbe541 / 7fff082d4a28 (480000c48058d48 / 7fff082d61ee)

Break @ 0x7fff082d4928
Break @ 0x7fff082d4920

Searching for gadgets...

[ENCLAVE] <Found gadget [SYSCALL] @ 0x7fff083f47ec>
[ENCLAVE] <Found gadget [POP_RDI] @ 0x7fe81a457287>
[ENCLAVE] <Found gadget [POP_RSI] @ 0x7fe81a456167>
[ENCLAVE] <Found gadget [POP_RDI] @ 0x7fe81a457287>
[ENCLAVE] <Found gadget [LEAVE] @ 0x7fe81a4409ea>
[ENCLAVE] <Found gadget [POP_RDX] @ 0x7fe81a228f7a>
[ENCLAVE] <Found gadget [POP_RAX] @ 0x7fe819fe6af4>
[ENCLAVE] <Found gadget [KCHG_RAX_RDI] @ 0x7fe819f3e8e5>
[ENCLAVE] <Searching for data cave...>
[ENCLAVE] <Cave @ 0x7fe81a43a000>
[ENCLAVE] <Building ROP chain...>
[ENCLAVE] <Payload ready!>

    | +--- enclave exit
    |    | +--- bar exit
    |    | +--- foo exit
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-mschwarz'
https://github.com/IAIK/SGXROP
- Asymmetric threat model
Design Problems

- Asymmetric threat model
- Enclaves assumed always benign
Design Problems

- Asymmetric threat model
- Enclaves assumed always benign
- Not realistic in most scenarios
Design Problems

- Asymmetric threat model
- Enclaves assumed always benign
- Not realistic in most scenarios
- Full memory access avoidable → reduce attack surface
Takeaways

- Asymmetric threat model in SGX fosters malware
- SGX hides and protects malware
- Easy to port existing malware to SGX ROP
SGXJail [Wei+19]
Host application

- Stack
- Heap

Enclave

read/write
Enclave

- read/write
- corrupt RSP

Host application

- stack
- heap

RSP
Defense

Host application

- stack
- heap

RSP

Enclave

- read/write
- corrupt RSP

Michael Schwarz (@misc0110) | Graz University of Technology
Defense

Host application

- stack
- heap
- ROP
- code
- ret

Enclave

- read/write
- corrupt RSP
- EEXIT

Michael Schwarz (@misc0110) | Graz University of Technology
- Arbitrary read
- Arbitrary read
  - Bypass randomization-based defenses (ASLR)
- **Arbitrary read**
  - Bypass randomization-based defenses (ASLR)
  - Discover ROP gadgets
- Arbitrary read
  - Bypass randomization-based defenses (ASLR)
  - Discover ROP gadgets
- Arbitrary write
- **Arbitrary read**
  - Bypass randomization-based defenses (ASLR)
  - Discover ROP gadgets

- **Arbitrary write**
  - Memory corruption
- Arbitrary read
  - Bypass randomization-based defenses (ASLR)
  - Discover ROP gadgets
- Arbitrary write
  - Memory corruption
- Arbitrary EEXIT
SGX Primitives

- Arbitrary read
  - Bypass randomization-based defenses (ASLR)
  - Discover ROP gadgets
- Arbitrary write
  - Memory corruption
- Arbitrary EEXIT
  - Direct code-reuse attacks
Asymmetric Trust

- Root problem: asymmetric trust
- Root problem: asymmetric trust
- Assumption: Enclave is fully trusted
Root problem: asymmetric trust
Assumption: Enclave is fully trusted
Goal: mutual distrust
Host process

Application logic

- Process isolation breaks arbitrary read/write
- ECALLs and OCALLs via shared memory
- seccomp syscall filter breaks arbitrary EEXIT
Process isolation breaks arbitrary read/write
Process isolation breaks arbitrary read/write
SGXJail Sandboxing

- Process isolation breaks arbitrary read/write
- ECALLs and OCALLs via shared memory
Process isolation breaks arbitrary read/write

ECALLs and OCALLs via shared memory
- Process isolation breaks arbitrary read/write
- ECALLs and OCALLs via shared memory
- seccomp syscall filter breaks arbitrary EEXIT
Evaluation: OCALL

Vanilla vs. SGXJail

Ratio of OCALL vs. enclave workload

OCALLs / Esec
Evaluation: OCALL

![Graph showing the ratio of OCALL vs. enclave workload]

- **Vanilla**
- **SGXJail**

**OCALLs / Esec**
Evaluation: OCALL

Vanilla
SGXJail

Ratio of OCALL vs. enclave workload

OCALLs / Esec

SGXJail/Vanilla in %

0 10 20 30 40

0 1 10 100 1000 10000 100000 1000000

SGXJail/ Vanilla in %

Vanilla
SGXJail

Overhead

Michael Schwarz (@misc0110) | Graz University of Technology
Evaluation: Protected File System

Vanilla

Runtime in ms vs Payload size in bytes

Michael Schwarz (@misc0110) | Graz University of Technology
Evaluation: Protected File System

Payload size in bytes vs. Runtime in ms

- Vanilla
- SGXJail

Michael Schwarz (@misc0110) | Graz University of Technology
Evaluation: Protected File System

Payload size in bytes vs. Runtime in ms

- Vanilla
- SGXJail
- Overhead

Overhead in %

Michael Schwarz (@misc0110) | Graz University of Technology
- Compatible with unmodified enclaves
- Compatible with **unmodified** enclaves
- Fully integrated in SGX SDK
  
  https://github.com/IAIK/SGXJail
- Small overhead only due to ECALLs/OCALLs
Can we implement it in hardware?
Hardware SGXJail

Host application

- Intel Memory Protection Keys (MPK)
  - MPK disables host memory
  - Confined ENTER instruction
  - MPK enables host memory
  - Enclave accesses arguments
  - EEXIT only to call gate after ENTER
- Intel Memory Protection Keys (MPK)
- Intel Memory Protection Keys (MPK)
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
• Intel Memory Protection Keys (MPK)
• MPK disables host memory
• Confined EENTER instruction
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Confined EENTER instruction
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Confined EENTER instruction
Hardware SGXJail

- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Confined EENTER instruction

Enclave accesses arguments
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Confined EENTER instruction

- Enclave accesses arguments
- EEXIT only to call gate after EENTER
- Intel Memory Protection Keys (MPK)
  - MPK disables host memory
  - Confined EENTER instruction
- Enclave accesses arguments
  - EEXIT only to call gate after CEENTER
- Intel Memory Protection Keys (MPK)
  - MPK disables host memory
  - Confined EENTER instruction
  - MPK enables host memory

- Enclave accesses arguments
- EEXIT only to call gate after EENTER
- Intel Memory Protection Keys (MPK)
- MPK disables host memory
- Confined EENTER instruction
- MPK enables host memory

- Enclave accesses arguments
- EEXIT only to call gate after EENTER
- Almost zero runtime overhead
- Almost zero runtime overhead
- Highly compatible (opt-in)
- Almost zero runtime overhead
- Highly compatible (opt-in)
- New CEENTER instruction
- Almost zero runtime overhead
- Highly compatible (opt-in)
- New CEENTER instruction
  - Make MPK immutable inside enclave
Hardware SGXJail

- Almost zero runtime overhead
- Highly compatible (opt-in)
- New CEENTER instruction
  - Make MPK immutable inside enclave
  - Enforce exit call gate
Hardware SGXJail

- Almost zero runtime overhead
- Highly compatible (opt-in)
- New CEENTER instruction
  - Make MPK immutable inside enclave
  - Enforce exit call gate
  - Can be implemented in microcode
Key Insights

- Secure execution → enclave malware
- Better threat models
  - SGX: asymmetric trust
  - SGXJail: mutual distrust
- Protection almost for free
- Future: reason about security of enclave API
Thank you!
Confining (Un)Trusted Execution Environments

Michael Schwarz (@misc0110)
November 20, 2019 - SILM

Graz University of Technology


