MPTEE: Bringing Flexible and Efficient Memory Protection to Intel SGX

Wenjia Zhao\textsuperscript{1,2}, Kangjie Lu\textsuperscript{2}, Yong Qi\textsuperscript{1}, Sqiyu Qi\textsuperscript{3}

\textsuperscript{1}Xi’an Jiaotong University, China
\textsuperscript{2}University of Minnesota, USA
\textsuperscript{3}Xidian University, China

EuroSys’20, April 27–30, 2020
Intel Software eXecute Guard (SGX)

- Hardware-based trusted execution environment
- Provide secure region, namely enclave
- Enhance Application Security
Two types of SGX Research

Applications (to protect data/code)
- VC3 [OAKLAND’15]
- SCONE [OSDI’16]
- JITGuard [CCS’17]
- SGXCrypter [ASP-DAC’17]

Protection/attack to SGX itself
- Page Fault [OAKLAND’15]
- SGX-Shield [NDSS’17]
- SGXBOUNDS [EUROSYS’17]
- Side-channel [OAKLAND’18, SECURITY’17]
Examples and current disadvantages

SGXCrypter protects code by unpacking the packed code in enclave.
  • relies on the OS page table to remove the W perm of unpacked code
  • is incompatible with the SGX security model

SGX-Shield protects SGX code itself through randomization
  • uses software-based DEP to create an Non-RW boundary(R15)
  • wastes the R15 register
  • NRW boundary using a general register can be shifted[security’18]
Two types of SGX Research

Applications (to protect data/code)
• VC3 [OAKLAND’15]
• SCONE [OSDI’16]
• JITGuard [CCS’17]
• SGXCrypter [ASP-DAC’17]

Protection/attack to SGX itself
• Page Fault [OAKLAND’15]
• SGX-Shield [NDSS’17]
• SGXBOUNDS [EUROSYS’17]
• Side-channel [OAKLAND’18, SECURITY’17]
Unfortunately, the feature is missing
Unfortunately, the feature is missing. Why?
Unfortunately, the feature is missing
Why?

Security considerations (untrusted os)
Permissions are statically decided (sign-verify)
Challenges

- Limited hardware support
- Strong adversary
Challenges

- Limited hardware support
- Strong adversary
- A software-based solution, significant performance overhead
Challenges

- Limited hardware support

- Strong adversary

The privileged software (e.g., OS, hypervisor) is untrusted and SGX programs themselves might be vulnerable
Challenges

- Limited hardware support
- Strong adversary

A hardware-assisted solution
Challenges

- Limited hardware support
- Strong adversary

A hardware-assisted solution with low overhead
Challenges

Strong adversary

Limited hardware support

A hardware-assisted solution

low overhead
MPTEE: memory permission protection

Flexible, efficient, and isolated memory permission enforcement for SGX.

- Flexible and Efficient Memory-Permission Enforcement
- Enforcement Integrity
MPTEE: memory permission protection

Flexible, efficient, and isolated memory permission enforcement for SGX.

• Flexible and Efficient Memory-Permission Enforcement
• Enforcement Integrity
MPTEE: memory permission protection

Flexible, efficient, and isolated memory permission enforcement for SGX.

• Flexible and Efficient Memory-Permission Enforcement
• Enforcement Integrity
Memory-Permission Enforcement

Elastic Cross-Region Bound Check (CRBC)

Basic idea

*Use hardware-assisted technique (MPX) to bound-check access*
Elastic Cross-Region Bound Check (CRBC)

Memory Protection Extension (MPX)

- New instructions, bndcu, bndcl, bndmk...
- Four dedicated bound registers (BND0~BND3)
Elastic Cross-Region Bound Check (CRBC)

Memory Protection Extension (MPX)

- New instructions, bndcu, bndcl, bndmk...
- Four dedicated bound registers (BND0~BND3)
- More bounds will be stored in a bound table in memory

Significant performance overhead (over 60%)
Elastic Cross-Region Bound Check (CRBC)

region0

region1

region2

region3

region4

region5

OS kernel

env, argv, argc

Stack

...

dynamic libraries

program

Heap

.text

region0

region1

region2

region3

region4

region5
Elastic Cross-Region Bound Check (CRBC)

<table>
<thead>
<tr>
<th>Region</th>
<th>Permissions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>region0</td>
<td></td>
<td>OS kernel</td>
</tr>
<tr>
<td>region1</td>
<td>RW</td>
<td>.data, .bss,</td>
</tr>
<tr>
<td>region2</td>
<td>X</td>
<td>.text</td>
</tr>
<tr>
<td>region3</td>
<td></td>
<td>Heap</td>
</tr>
<tr>
<td>region4</td>
<td>RW</td>
<td>.data, .bss,</td>
</tr>
<tr>
<td>region5</td>
<td>X</td>
<td>.text</td>
</tr>
</tbody>
</table>

Bound tables impose high overhead
Elastic Cross-Region Bound Check (CRBC)

How can we use **limited number** of bound registers to protect **multiple** memory region access?
Elastic Cross-Region Bound Check (CRBC)

Key observation

The same permission memory range is continuous in an enclave
Elastic Cross-Region Bound Check (CRBC)

Key observation

The same permission memory range is continuous in an enclave

Because

All required libraries must be statically linked in the target enclave program
Elastic Cross-Region Bound Check (CRBC)

Permission change

Continuous \(\rightarrow\) Non-continuous

Exceeded the number of MPX registers
Elastic Cross-Region Bound Check (CRBC)

Enclave memory layout

- .text, .rodata, ...
- .got, .bss, .data, ...
- Thread context
- Unpack code/randomize code
- Remove W
Elastic Cross-Region Bound Check (CRBC)

Enclave memory layout

Remove W

3 regions $\rightarrow$ 5 regions
Elastic Cross-Region Bound Check (CRBC)

Enclave memory layout

3 regions → 5 regions

4 MPX registers are not enough
Elastic Cross-Region Bound Check (CRBC)

We design a new layout
Elastic Cross-Region Bound Check (CRBC)

Enclave memory layout

- .text, .rodata, ...
- .got, .bss, .data, ...
- Heap
- Thread context

New memory layout with CRBC

- non-permission
- .text, .rodata, ... (RX)
- .got, .bss, .data, heap (RW)

X
R
W
Elastic Cross-Region Bound Check (CRBC)

Enclave memory layout

- .text,.rodata,...
- .got,.bss,.data,...
- Heap
- Thread context

New memory layout with CRBC

- non-permission
- .text,.rodata,... (RX)
- .got,.bss,.data,heap (RW)

Update the memory layout with CRBC.
Elastic Cross-Region Bound Check (CRBC)

New memory layout with CRBC
Elastic Cross-Region Bound Check (CRBC)

Only three registers to offer six regions

Continuous after permission change

New memory layout with CRBC
Elastic Cross-Region Bound Check (CRBC)

Reserved area

JIT code generator

Generated code fragment 0
Elastic Cross-Region Bound Check (CRBC)

JIT code generator

non-permission

Remove W

Generated code fragment 0
Generated code fragment 1
Elastic Cross-Region Bound Check (CRBC)
Elastic Cross-Region Bound Check (CRBC)

- Initializing the bounds
- Updating the bounds
- Permission enforcement using CRBC
  - Four APIs, `mpt_mmap`, `mpt_mremap`, `mpt_uunmap`, `mpt_write`
- Improving EPC usage
- Optimizing CRBC: Adaptive Permission Enforcement

More details in the paper
Elastic Cross-Region Bound Check (CRBC)

- CRBC leverages MPX to efficiently bound-check multiple regions with different boundary registers.
  - Use **only** `regs`, `bnd0`, `bnd1`, and `bnd2`
  - Provide **six** different permission regions
  - Allow the **flexible changes** of the ranges of memory regions at runtime

<table>
<thead>
<tr>
<th>Permission</th>
<th>Bound range</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-perm.</td>
<td>(ImageBase, bnd0.lb)</td>
</tr>
<tr>
<td>X</td>
<td>(bnd0.lb, bnd2.lb)</td>
</tr>
<tr>
<td>RX</td>
<td>(bnd2.lb, bnd1.lb)</td>
</tr>
<tr>
<td>RWX</td>
<td>(bnd1.lb, bnd0.ub)</td>
</tr>
<tr>
<td>RW</td>
<td>(bnd0.ub, bnd1.ub)</td>
</tr>
<tr>
<td>R</td>
<td>(bnd1.ub, bnd2.ub)</td>
</tr>
</tbody>
</table>
Elastic Cross-Region Bound Check (CRBC)

- CRBC leverages MPX to efficiently bound-check multiple regions with different boundary registers
  - Use **only** reg, bnd0, bnd1, and bnd2
  - Provide **six** different permission regions
  - Allow the **flexible changes** of the ranges of memory regions at runtime

Without using MPX bound table to avoid the high performance overhead
CRBC may be attacked

Check-skipping attacks

- Control-flow attacks that bypass the bound checks and abuse the permission control

```
48 8b 14 d0  mov  (%rax,%rdx,8),%rdx
48 a1 ff d2  ff ff 00  movabs 0xffffffff, %rax
f2 0f 1a c7  bndcu  %rdi,%rdx
f3 0f 1a c7  bndcl  %rdi,%rdx
ff d2       callq  *%rdx
```

Unaligned call without check
CRBC may be attacked

Bound-manipulating attacks

- Data-flow attacks that manipulate bounds

Bndmk is called maliciously
Enforcement Integrity

control-data integrity + memory isolation
Enforcement Integrity

Control-data integrity

• Indirect calls/jumps

```
ptr = Index_A;
check ptr < table size;
Call *(base+ptr*8);
```

Trampolines table

Function A

Function B

Function A.Addr

Function B.Addr
Enforcement Integrity

Control-data integrity

• Indirect calls/jumps

```c
ptr = Index_A;
check ptr < table size;
Call *(base+ptr*8);
```

Function A
Function B

Trampolines table
Function A.Addr
Function B.Addr
Enforcement Integrity

Control-data integrity
• Indirect calls/jumps
• Return Address
  • SafeStack [OSDI’14]
Enforcement Integrity

Memory isolation

*ptrA = *ptrB + V

modify_bndregisters(*ptrA)

ptrA/ptrB

Trampoline table
Indirect calls

ret addr

Non-permission

No additional overhead

X(BND0)
Evaluation

Hardware platform
  • Intel Xeon E3-1225v5
  • 8GB memory

Software environments
  • Ubuntu 16.04 Server
  • SGX SDK v2.0
  • SGX driver v0.10
  • LLVM v6.0
Macro-benchmark

SQLite
  • Overhead from 2% to 8%
  • Average overhead 6.6%

Memcached
  • Average overhead 2.2%
Micro-benchmark and Case Studies

- Nbench
- Protecting SGXELIDE code
- Protecting SGX-Shield code

More details in the paper
Conclusions

MPTEE provides a flexible, isolated, and efficient memory permission protection mechanism

• Three bound registers offer six permission regions
• Efficient enforcement integrity
  • Control data integrity with efficient trampoline table
  • No additional overhead of isolation beyond the CRBC

Flexible and efficient memory permission protection for SGX