SEIMI: Efficient and Secure SMAP-Enabled Intra-process Memory Isolation

Zhe Wang¹, Chenggang Wu¹, Mengyao Xie¹, Yinqian Zhang², Kangjie Lu³, Xiaofeng Zhang¹, Yuanming Lai¹, Yan Kang¹, and Min Yang⁴

¹Institute of Computing Technology, Chinese Academy of Sciences, ²The Ohio State University, ³University of Minnesota at Twin-Cities, ⁴Fudan University
Intra-process Memory Isolation

• Memory corruption defenses need to keep their metadata safe.
  – The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.
Intra-process Memory Isolation

• Memory corruption defenses need to keep their metadata safe.
  – The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  – The software-based randomization method has been proven to be vulnerable.

• The strict memory isolations for the metadata in defenses are needed.
Intra-process Memory Isolation

• Memory corruption defenses need to keep their metadata safe.
  – The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  – The software-based randomization method has been proven to be vulnerable.

• The strict memory isolations for the metadata in defenses are needed.
  – Intel MPX uses bounds checks for isolation.
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.

- The strict memory isolations for the metadata in defenses are needed.
  - Intel MPX uses bounds checks for isolation.
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.

- The strict memory isolations for the metadata in defenses are needed.
  - Intel MPX uses bounds checks for isolation.
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.

- The strict memory isolations for the metadata in defenses are needed.
  - Intel MPX uses bounds checks for isolation.
Intra-process Memory Isolation

• Memory corruption defenses need to keep their metadata safe.
  – The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  – The software-based randomization method has been proven to be vulnerable.

• The strict memory isolations for the metadata in defenses are needed.
  – Intel MPX uses bounds checks for isolation.
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.

- The strict memory isolations for the metadata in defenses are needed.
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.
Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The *safe region* in CPI, the *shadow stack* in CFI, the *randomization secrets* in ...
  - The software-based randomization method has been proven to be vulnerable.

- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.
Intra-process Memory Isolation

- Memory corruption defenses need to keep their metadata safe.
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  - The software-based randomization method has been proven to be vulnerable.

- The strict memory isolations for the metadata in defenses are needed.
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.
Intra-process Memory Isolation

• Memory corruption defenses need to keep their metadata safe.
  – The safe region in CPI, the shadow stack in CFI, the randomization secrets in ...
  – The software-based randomization method has been proven to be vulnerable. ⚠

• The strict memory isolations for the metadata in defenses are needed.
  – Intel MPX uses bounds checks for isolation.
  – Intel MPK changes permissions of pages.

But they are not efficient enough as we expect.
Threat Model

• We consider a defense that protects a vulnerable application against memory corruption attacks.
  – Web servers, databases or browsers.

• The design of this defense is secure:
  – Breaking memory isolation is a prerequisite for compromising the defense (e.g., attackers cannot hijack the control flow before it).

• Attackers’ capabilities:
  – Arbitrary read and write by exploiting memory corruption vulnerabilities.
Outline

Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation
Motivation

• Problem:
  – Hardware-assisted memory isolations could achieve better performance.
  – But existing methods are not fast enough for isolating in the user-mode process.
Motivation

• Problem:
  – Hardware-assisted memory isolations could achieve better performance.
  – But existing methods are not fast enough for isolating in the user-mode process.
Motivation

• Problem:
  – Hardware-assisted memory isolations could achieve better performance.
  – But existing methods are not fast enough for isolating in the user-mode process.

The user-mode hardware features are not fast.
Motivation

• Problem:
  – **Hardware**-assisted memory isolations could achieve better performance.
  – But existing methods are **not fast** enough for isolating in the **user-mode** process.

The **user-mode hardware features are not fast**.

How about the privileged hardware feature?
Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are **not fast** enough for isolating in the **user-mode** process.

  The **user-mode hardware features are not fast**.

How about the privileged hardware feature?

Is there a privileged hardware feature which is more efficient than Intel MPX/MPK for the memory isolation???
Motivation —— SMAP in Processors 101

• To prevent the kernel from inadvertently accessing malicious data in user space,  
  – dereferencing a corrupted data pointer
Motivation —— SMAP in Processors 101

• To prevent the kernel from inadvertently accessing malicious data in user space,
  – dereferencing a corrupted data pointer

• Intel and AMD provide the **Supervisor-mode Access Prevention (SMAP)** hardware feature to disable the kernel access to the user space memory.
Motivation —— SMAP in Processors 101

• **Supervisor-mode Page (S-page) vs. User-mode Page (U-page)**
  – Divided by the U/S bit in the page table entry.
Motivation —— SMAP in Processors 101

• **Supervisor-mode Page (S-page) vs. User-mode Page (U-page)**
  – Divided by the U/S bit in the page table entry.

• **SMAP disallows the code access to the U-page in the supervisor-mode.**
  – S-mode is short for supervisor-mode (ring 0-2).
  – U-mode is short for user mode (ring 3).
Motivation —— SMAP in Processors 101

• Supervisor-mode Page (S-page) vs. User-mode Page (U-page)
  – Divided by the U/S bit in the page table entry.

• SMAP disallows the code access to the U-page in the supervisor-mode.
  – S-mode is short for supervisor-mode (ring 0-2).
  – U-mode is short for user mode (ring 3).

<table>
<thead>
<tr>
<th></th>
<th>Ring 0</th>
<th>Ring 1</th>
<th>Ring 2</th>
<th>Ring 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged Instruction Fetch</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>S-page Access Permission</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
</tr>
<tr>
<td>U-page Access Permission</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

SMAP is disabled
Motivation —— SMAP in Processors 101

• Supervisor-mode Page (S-page) vs. User-mode Page (U-page)
  – Divided by the U/S bit in the page table entry.

• SMAP disallows the code access to the U-page in the supervisor-mode.
  – S-mode is short for supervisor-mode (ring 0-2).
  – U-mode is short for user mode (ring 3).

<table>
<thead>
<tr>
<th></th>
<th>Ring 0</th>
<th>Ring 1</th>
<th>Ring 2</th>
<th>Ring 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged Instruction Fetch</td>
<td>✔️</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>S-page Access Permission</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗️</td>
</tr>
<tr>
<td>U-page Access Permission</td>
<td>✗️</td>
<td>✗️</td>
<td>✗️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

SMAP is enabled
Motivation —— SMAP in Processors 101

• X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.
  – When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.
Motivation —— SMAP in Processors 101

• X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.
  – When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

• POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.
  – popfq could be execute in S-mode (ring 0-2).
  – stac/clac are privileged instructions that can only be execute in ring 0.
Motivation —— SMAP in Processors 101

- X86 processors provide a `RFLAGS.AC` flag to disable/enable SMAP.
  - When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

- POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.
  - `popfq` could be execute in S-mode (ring 0-2).
  - `stac/clac` are privileged instructions that can only be execute in ring 0.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Cycles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrpkru</td>
<td>18.9</td>
<td>Update the access right of a pkey in Intel MPK</td>
</tr>
<tr>
<td>popfq</td>
<td>22.4</td>
<td>Pop stack into the RFLAGS register.</td>
</tr>
<tr>
<td>stac/clac</td>
<td>8.6</td>
<td>Set/Clear the AC flag in the RFLAGS register.</td>
</tr>
</tbody>
</table>
Motivation —— SMAP in Processors 101

• X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.
  – When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

• POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.
  – `popfq` could be execute in S-mode (ring 0-2).
  – `stac/clac` are privileged instructions that can only be execute in ring 0.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Cycles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrpkru</td>
<td>18.9</td>
<td>Update the access right of a pkey in Intel MPK</td>
</tr>
<tr>
<td>popfq</td>
<td>22.4</td>
<td>Pop stack into the RFLAGS register.</td>
</tr>
<tr>
<td><code>stac/clac</code></td>
<td>8.6</td>
<td>Set/Clear the AC flag in the RFLAGS register.</td>
</tr>
</tbody>
</table>

Intel SMAP is more efficient than Intel MPK for controlling memory access permission.
Outline

Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation
High-level Design —— SEIMI

• The Memory Layout Setting
  – The isolated memory region are set to be \textit{U-pages}.
  – Other memory regions are set to be \textit{S-pages}.

• The Running State Setting
  – The process runs in \textit{ring 0}, due to the \textit{stac/clac} are privileged instructions.
High-level Design —— SEIMI

• **The Memory Layout Setting**
  – The isolated memory region are set to be **U-pages**.
  – Other memory regions are set to be **S-pages**.

• **The Running State Setting**
  – The process runs in **ring 0**, due to the stac/clac are privileged instructions.
High-level Design —— SEIMI

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-pages**.
  - Other memory regions are set to be **S-pages**.

- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.

![Diagram of Memory Layout and Running State]

- **Heap (RW)**
- **Stack (RW)**
- **Code (RX)**
- **RW**

<table>
<thead>
<tr>
<th>Ring 0</th>
<th>Heap (RW)</th>
<th>Stack (RW)</th>
<th>Code (RX)</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

- **S-page**
- **U-page**
- **Access Denied**

**SMAP is disabled**
**High-level Design —— SEIMI**

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-pages**.
  - Other memory regions are set to be **S-pages**.

- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.

![Diagram showing memory layout and running state](image)
High-level Design —— SEIMI

• Problem:
  – Running untrusted code in ring 0 may corrupt the OS kernel.
High-level Design — SEIMI

• **Problem:**
  – Running untrusted code in ring 0 may corrupt the OS kernel.

• **Our Solution** —— Placing the OS kernel in “ring -1”
  – Using the Intel VT-x technique to separate the target application and the OS kernel

![Diagram showing VMX non-root (guest) and VMX root (host) with Intel VT-x for separation]
High-level Design —— SEIMI

• **Problem:**
  – Running untrusted code in ring 0 may corrupt the OS kernel.

• **Our Solution** —— Placing the OS kernel in “ring -1”.
  – Using the Intel VT-x technique to separate the target application and the OS kernel.
High-level Design — Challenges in SEIMI

• C-1: Distinguishing SMAP reads and writes.
  – Sensitive data may require only integrity protection.
  – Preventing reads from untrusted code can lead to unnecessary overhead.
High-level Design —— Challenges in SEIMI

• **C-1: Distinguishing SMAP reads and writes.**
  – Sensitive data may require only **integrity** protection.
  – Preventing reads from untrusted code can lead to **unnecessary** overhead.

• **C-2: Preventing the leaking/manipulating of the privileged data structures.**
  – In general, a guest VM needs to manage the memory, interrupts, exceptions, etc.
  – Some data structures are **privileged**, e.g., the page tables.
High-level Design —— Challenges in SEIMI

• **C-1: Distinguishing SMAP reads and writes.**
  – Sensitive data may require only **integrity** protection.
  – Preventing reads from untrusted code can lead to **unnecessary** overhead.

• **C-2: Preventing the leaking/manipulating of the privileged data structures.**
  – In general, a guest VM needs to manage the memory, interrupts, exceptions, etc.
  – Some data structures are **privileged**, e.g., the page tables.

• **C-3: Preventing the abusing of the privileged hardware features.**
  – Besides the stac/clac, **other** privileged instructions can also run in ring 0.
Outline

Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation
Approaches Overview —— Challenge-1

• C-1: Distinguishing SMAP reads and writes.

• Solution —— The shared-memory based read/write separation method.
Approaches Overview —— Challenge-1

• C-1: Distinguishing SMAP reads and writes.

• Solution —— The shared-memory based read/write separation method.

Ring 0

<table>
<thead>
<tr>
<th>Heap (RW)</th>
<th>Stack (RW)</th>
<th>Code (RX)</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>U</td>
</tr>
</tbody>
</table>

Regular Memory Region

Isolated Memory Region

S-page

U-page

Access Denied
Approaches Overview —— Challenge-1

• **C-1: Distinguishing SMAP reads and writes.**

• **Solution** —— The shared-memory based read/write separation method.

![Diagram showing SMAP regions and permissions]

- **Rule:** SMAP is enabled
- **Diagram:**
  - Regular Memory Region
  - Isolated Memory Region
  - S-page
  - U-page
  - Access Denied

- **Legend:**
  - Ring 0
  - Heap (RW)
  - Stack (RW)
  - Code (RX)
  - RW
  - RO

- **Access Permissions:**
  - S
  - U
  - Access Denied

- **Note:** The shared-memory based read/write separation method is used to distinguish between SMAP reads and writes.
Approaches Overview —— Challenge-1

• C-1: Distinguishing SMAP reads and writes.

• Solution —— The shared-memory based read/write separation method.

![Diagram showing memory regions and their access operations.](image-url)
Approaches Overview —— Challenge-2

• C-2: Preventing the leaking/manipulating of the privileged data structures.
Approaches Overview ——— Challenge-2

- C-2: Preventing the leaking/manipulating of the privileged data structures.

- Observation:
  - The operations to these structures are only performed when the process accesses the OS kernel through specific events, e.g., interrupts, exceptions, and system calls.
Approaches Overview —— Challenge-2

• C-2: Preventing the leaking/manipulating of the privileged data structures.

• Observation:
  – The operations to these structures are **only** performed when the process **accesses** the OS kernel through specific **events**, e.g., interrupts, exceptions, and system calls.

• Solution:
  – Placing the privileged data structures and their operations **into the VMX root mode**.
  – We leverage the Intel VT-x technique to force all these events to **trigger VM exits** and enter into the VMX root mode.
Approaches Overview —— Challenge-3

• C-3: Preventing the abusing of the privileged hardware features.
Approaches Overview —— Challenge-3

• C-3: Preventing the abusing of the privileged hardware features.

• Solution:
  – SEIMI sanitizes the execution of all privileged instructions in the VMX non-root mode.
Approaches Overview —— Challenge-3

• C-3: Preventing the abusing of the privileged hardware features.

• Solution:
  – SEIMI sanitizes the execution of all privileged instructions in the VMX non-root mode.

1 Triggering the VM exits and stopping the execution;
Approaches Overview —— Challenge-3

• C-3: Preventing the abusing of the privileged hardware features.

• Solution:
  – SEIMI sanitizes the execution of all privileged instructions in the VMX non-root mode.
    1. Triggering the VM exits and stopping the execution;
    2. Invalidating the execution effects;
Approaches Overview —— Challenge-3

• C-3: Preventing the abusing of the privileged hardware features.

• Solution:
  – SEIMI **sanitizes** the execution of all privileged instructions in the VMX non-root mode.

1. Triggering the VM exits and stopping the execution;
2. Invalidating the execution effects;
3. Raising processor exceptions and disabling the execution.
Outline

Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation
System Overview

- SEIMI is implemented on Linux/X86_64 platform.
System Overview

- **SEIMI** is implemented on Linux/X86_64 platform.
- Two Phases in **SEIMI** —— Compilation Phase and Runtime Phase
**System Overview**

- **SEIMI** is implemented on Linux/X86_64 platform.
- Two Phases in **SEIMI** —— Compilation Phase and Runtime Phase

**Compilation phase**

Users could use the SEIMI’s APIs to management the isolated memory region.
**System Overview**

- **SEIMI** is implemented on Linux/X86_64 platform.
- Two Phases in **SEIMI** —— Compilation Phase and Runtime Phase

**Compilation phase**

Users could use the SEIMI’s APIs to management the isolated memory region.
System Overview

- **SEIMI** is implemented on Linux/X86_64 platform.
- Two Phases in **SEIMI** —— Compilation Phase and Runtime Phase

**Compilation phase**

Users could use the **SEIMI’s APIs** to management the isolated memory region.

**Runtime Phase**

The core of **SEIMI** is a kernel module which monitors the startup of the target application and places it into ring 0 of the VMX non-root mode.
**System Overview**

- **SEIMI** is implemented on Linux/X86_64 platform.
- Two Phases in **SEIMI** —— Compilation Phase and Runtime Phase

**Compilation phase**

Users could use the SEIMI’s APIs to manage the isolated memory region.

**Library.a**

**Runtime Phase**

The core of SEIMI is a kernel module which monitors the startup of the target application and places it into ring 0 of the VMX non-root mode.

**Target Process**

- HW(VMX non-root, Ring 0)

**Other Processes**

- HW(VMX root, Ring 3)

**Kernel Module**

- OS Kernel

- HW(VMX root, Ring 0)
SEIMI —— Compilation Phase

- **SEIMI** provides APIs to *allocate/free* the isolated region, and *enable/disable* the SMAP.
SEIMI —— Compilation Phase

- SEIMI provides APIs to allocate/free the isolated region, and enable/disable the SMAP.

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *sa_alloc(size_t length, bool need_ro, long *offset);</td>
<td>Allocate an isolated U-page region at the page granularity. If specified, it also allocates a shared isolated S-page region.</td>
</tr>
<tr>
<td>bool sa_free(void *addr, size_t length);</td>
<td>Free an isolated U-page region with the specified length.</td>
</tr>
<tr>
<td>#define SWITCH_IN \ \ asm(&quot;stac\n&quot;);</td>
<td>Disable SMAP—access the isolated U-page region is allowed.</td>
</tr>
<tr>
<td>#define SWITCH_OUT \ \ asm(&quot;clac\n&quot;);</td>
<td>Enable SMAP—access to the isolated U-page region is denied.</td>
</tr>
</tbody>
</table>
SEIMI — Runtime Phase

- The core of SEIMI is a kernel module, includes three key components.
SEIMI —— Runtime Phase

• The core of SEIMI is a kernel module, includes **three key components**.

1. **Memory Management Component**
   - Configures the regular/isolated memory region.
SEIMI —— Runtime Phase

- The core of SEIMI is a kernel module, includes three key components.

1. Memory Management Component
   - Configures the regular/isolated memory region.

2. Privileged Instructions Prevention Component
   - Prevents these instructions from being abused.

![Diagram showing the relationship between components and processes during the runtime phase.]
SEIMI —— Runtime Phase

- The core of SEIMI is a kernel module, includes three key components.

1. **Memory Management Component**
   - Configures the regular/isolated memory region.

2. **Privileged Instructions Prevention Component**
   - Prevents these instructions from being abused.

3. **Events Redirection Component**
   - Handles system calls, interrupts, exceptions, and Linux signals.
SEIMI — Memory Management Component

- A shadow mechanism for (only) page-table root.
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.
SEIMI — Memory Management Component

- A shadow mechanism for (only) page-table root.
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.

![Diagram of memory management structure]
SEIMI — Memory Management Component

- A shadow mechanism for (only) page-table root.
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.
SEIMI — Memory Management Component

- A shadow mechanism for (only) page-table root.
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.

### Entries in PML4

<table>
<thead>
<tr>
<th>Entries in PML4</th>
<th>Size(TB)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0 ~ #254</td>
<td>127.5</td>
<td>Regular Memory</td>
<td>S-page</td>
</tr>
<tr>
<td>#255</td>
<td>0.5</td>
<td>Isolated Memory</td>
<td>U-page</td>
</tr>
<tr>
<td>#255 ~ #511</td>
<td>128.0</td>
<td>Kernel Space</td>
<td>NULL</td>
</tr>
</tbody>
</table>
SEIMI — Memory Management Component

- Support the read-only isolated S-page memory region.
  - Flipping the R/W bit to set the read-only permission neatly.
SEIMI — Memory Management Component

- Support the read-only isolated S-page memory region.
  
  - Flipping the R/W bit to set the read-only permission neatly.

<table>
<thead>
<tr>
<th>Entries in PML4</th>
<th>Size(TB)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0 ~ #253</td>
<td>127</td>
<td>Regular Memory</td>
<td>S-page</td>
</tr>
<tr>
<td>#254</td>
<td>0.5</td>
<td>Isolated Memory</td>
<td>S-page</td>
</tr>
<tr>
<td>#255</td>
<td>0.5</td>
<td>Isolated Memory</td>
<td>U-page</td>
</tr>
<tr>
<td>#255 ~ #511</td>
<td>128</td>
<td>Kernel Space</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Set the Read-only Region
- The R/W bit of the #254 entry is set to 0.
SEIMI — Memory Management Component

- Support the read-only isolated S-page memory region.
  - Flipping the R/W bit to set the read-only permission neatly.

<table>
<thead>
<tr>
<th>Entries in PML4</th>
<th>Size(TB)</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0 ~ #253</td>
<td>127</td>
<td>Regular Memory</td>
<td>S-page</td>
</tr>
<tr>
<td>#254</td>
<td>0.5</td>
<td>Isolated Memory</td>
<td>S-page</td>
</tr>
<tr>
<td>#255</td>
<td>0.5</td>
<td>Isolated Memory</td>
<td>U-page</td>
</tr>
<tr>
<td>#255 ~ #511</td>
<td>128</td>
<td>Kernel Space</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Set the Read-only Region
- The R/W bit of the #254 entry is set to 0.

Set the Shared Memory
- The #254 and the #255 reference the same PDPT.
SEIMI —— Privileged Instruction Prevention Component

• We identify all privileged instructions and the instructions that will change the behaviors in different rings in the 64-Bit mode of X86_64.
SEIMI —— Privileged Instruction Prevention Component

• We identify all privileged instructions and the instructions that will change the behaviors in different rings in the 64-Bit mode of X86_64.

• Our identification method:

1 Automated filtering

– We embed each instruction with random operands into a test program and run it in ring 3.
– By capturing the #GP and the #UD, we automatically and completely filter all privileged instructions.
SEIMI — Privileged Instruction Prevention Component

- We identify all privileged instructions and the instructions that will change the behaviors in different rings in the 64-Bit mode of X86_64.

- Our identification method:

1. **Automated filtering**
   - We embed each instruction with random operands into a test program and run it in ring 3.
   - By capturing the #GP and the #UD, we automatically and completely filter all privileged instructions.

2. **Manual Verification**
   - We manually review the description of all X86 instructions by reading the Intel Software Developers’ Manual.
   - Confirm the first step is complete, and also find the instructions that behave differently in ring 0 and ring 3.
SEIMI —— Privileged Instruction Prevention Component

- We group them into 20 categories based on their different functionality.
SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.

<table>
<thead>
<tr>
<th>Line</th>
<th>Detailed Instructions</th>
<th>Is Privileged Instruction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VM[RESUME</td>
<td>READ</td>
</tr>
<tr>
<td>2</td>
<td>INVD. XSETBV</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>ENCLS(e.g., ECREATE, EADD, EINIT, EDBG...)</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>RDMSR, WRMSR</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>IN, OUT, IN[S</td>
<td>SB</td>
</tr>
<tr>
<td>6</td>
<td>HLT, INVLPG, RDPNC, MONITOR, MWAIT, WBINVD</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>LGDT, LLDT, LTR, LIDT</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>MOV to/from DR0-DR7</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>MOV to/from CR3, MOV to/from CR8</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>MOV to/from CR0/CR4, CLTS, LMSW, SMSW</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>MOV to/from CR2</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>SWAPGS</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>CLI, STI</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>LAR, LSL. VERR, VERW</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>POPF, POPFQ</td>
<td>N</td>
</tr>
<tr>
<td>16</td>
<td>L[FS</td>
<td>DS</td>
</tr>
<tr>
<td>17</td>
<td>Far CALL, Far RET, Far JMP</td>
<td>N</td>
</tr>
<tr>
<td>18</td>
<td>IRET, IRETD, IRETQ</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>SYSEXIT, SYSRET</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>XSAVES, XRSTORS, INVPCID</td>
<td>Y</td>
</tr>
</tbody>
</table>
SEIMI — Privileged Instruction Prevention Component

- We group them into 20 categories based on their different functionality.

<table>
<thead>
<tr>
<th>Line</th>
<th>Detailed Instructions</th>
<th>Is Privileged Instruction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VM[RESUME</td>
<td>READ</td>
</tr>
<tr>
<td>2</td>
<td>INVD. XSETPV</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD...)</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>RDMSR, WRMSR</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>IN, OUT, IN[S</td>
<td>SB</td>
</tr>
<tr>
<td>6</td>
<td>HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>LGDT, LLDT, LTR, LIDT</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>MOV to/from DR0-DR7</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>MOV to/from CR3, MOV to/from CR8</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>MOV to/from CR0/CR4, CLTS, LMSW, SMSW</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>MOV to/from CR2</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>SWAPGS</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>CLI, STI</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>LAR, LSL. VERR, VERW</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>POPF, POPFQ</td>
<td>N</td>
</tr>
<tr>
<td>16</td>
<td>L[FS</td>
<td>DS</td>
</tr>
<tr>
<td>17</td>
<td>Far CALL, Far RET, Far JMP</td>
<td>N</td>
</tr>
<tr>
<td>18</td>
<td>IRET, IRETD, IRETQ</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>SYSEXIT, SYSRET</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>XSAVES, XRSTORS, INVPCID</td>
<td>Y</td>
</tr>
</tbody>
</table>

Triggering VM Exit and Stopping Execution.
- Using the Intel VT-x technique to configure the VM exits directly.
SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.

<table>
<thead>
<tr>
<th>Line</th>
<th>Detailed Instructions</th>
<th>Is Privileged Instruction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VM[RESUME</td>
<td>READ</td>
</tr>
<tr>
<td>2</td>
<td>INVD, XSETBV</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD...)</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>RDMSR, WRMSR</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>IN, OUT, IN[S</td>
<td>SB</td>
</tr>
<tr>
<td>6</td>
<td>HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>LGDT, LLDT, LTR, LIDT</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>MOV to/from DR0-DR7</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>MOV to/from CR3, MOV to/from CR8</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>MOV to/from CR0/CR4, CLTS, LMSW, SMSW</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>MOV to/from CR2</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>SWAPGS</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>CLI, STI</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>LAR, LSL, VERR, VERW</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>POPF, POPFQ</td>
<td>N</td>
</tr>
<tr>
<td>16</td>
<td>L[FS</td>
<td>DS</td>
</tr>
<tr>
<td>17</td>
<td>Far CALL, Far RET, Far JMP</td>
<td>N</td>
</tr>
<tr>
<td>18</td>
<td>IRET, IRETD, IRETQ</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>SYSEXIT, SYSRET</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>XSAVES, XRSTORS, INVPCID</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Triggering VM Exit and Stopping Execution.
- Using the Intel VT-x technique to configure the VM exits directly.

### Invalidating the Execution Effects.
- The execution does not change any state.
SEIMI — Privileged Instruction Prevention Component

- We group them into 20 categories based on their different functionality.

<table>
<thead>
<tr>
<th>Line</th>
<th>Detailed Instructions</th>
<th>Is Privileged Instruction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VM[RESUME</td>
<td>READ</td>
</tr>
<tr>
<td>2</td>
<td>INVD. XSETBV</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD...)</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>RDMSR, WRMSR</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>IN, OUT, IN[SB][SW][SD], OUT[SB][SW][SD]</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>LGDT, LLDT, LTR, LIDT</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>MOV to/from DR0-DR7</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>MOV to/from CR3, MOV to/from CR8</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>MOV to/from CR0/CR4, CLTS, LMSW, SMSW</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>MOV to/from CR2</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>SWAPGS</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>CLI, STI</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>LAR, LSL, VERR, VERW</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>POPF, POPFQ</td>
<td>N</td>
</tr>
<tr>
<td>16</td>
<td>[FS][DS][SS], MOV to [DS][ES][FS][GS][SS], POP [FS][GS]</td>
<td>N</td>
</tr>
<tr>
<td>17</td>
<td>Far CALL, Far RET, Far JMP</td>
<td>N</td>
</tr>
<tr>
<td>18</td>
<td>IRET, IRETD, IRETQ</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>SYSEXIT, SYSRET</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>XSAVES, XRSSTORS, INVPCID</td>
<td>Y</td>
</tr>
</tbody>
</table>

Triggering VM Exit and Stopping Execution.
- Using the Intel VT-x technique to configure the VM exits directly.

Invalidate the Execution Effects.
- The execution does not change any state.

Raising the Execution Exception and Stopping Execution.
- Configure the execution condition.
SEIMI —— Privileged Instruction Prevention Component

- We group them into 20 categories based on their different functionality.

<table>
<thead>
<tr>
<th>Line</th>
<th>Detailed Instructions</th>
<th>Is Privileged Instruction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VM[RESUME</td>
<td>READ</td>
</tr>
<tr>
<td>2</td>
<td>INVD. XSETBV</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>ENCLS (e.g., ECREATE, EADD, EINIT, EDBGRD...)</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>RDMSR, WRMSR</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>IN, OUT, IN[S</td>
<td>SB</td>
</tr>
<tr>
<td>6</td>
<td>HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>LGDT, LLDT, LTR, LIDT</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>MOV to/from DRo-DR7</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>MOV to/from CR3, MOV to/from CR8</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>MOV to/from CR0/CR4, CLTS, LMSW, SMSW</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>MOV to/from CR2</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>SWAPGS</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>CLI, STI</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>LAR, LSL. VERR, VERW</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>POPF, POPFQ</td>
<td>N</td>
</tr>
<tr>
<td>16</td>
<td>L[FS</td>
<td>DS</td>
</tr>
<tr>
<td>17</td>
<td>Far CALL, Far RET, Far JMP</td>
<td>N</td>
</tr>
<tr>
<td>18</td>
<td>IRET, IRETQ, IRETQ</td>
<td>Y</td>
</tr>
<tr>
<td>19</td>
<td>SYSEXIT, SYSRET</td>
<td>Y</td>
</tr>
<tr>
<td>20</td>
<td>XSAVES, XRSTORS, INVPCID</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Triggering VM Exit and Stopping Execution.**
- Using the Intel VT-x technique to configure the VM exits directly.

**Invalidate the Execution Effects.**
- The execution does not change any state.

**Raising the Execution Exception and Stopping Execution.**
- Configure the execution condition.
Raising the Execution Exception.

– We choose to raise an exception during their execution and to trigger the VM exits.
SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.
   – We choose to raise an exception during their execution and to trigger the VM exits.

1 Raising the #UD (invalid opcode exception)
   – xsaves, xrstors, invpcid ... via configuring the VMCS to disable the support in guest.
SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.
- We choose to raise an exception during their execution and to trigger the VM exits.

1. Raising the #UD (invalid opcode exception)
   - xsaves, xrstors, invpcid ... via configuring the VMCS to disable the support in guest.

2. Raising the #PF (page fault exception)
   - sysexit, sysret... due to the S-page setting in all code pages.
SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.

– We choose to raise an exception during their execution and to trigger the VM exits.

1. Raising the #UD (invalid opcode exception)
   – xsaves, xrstors, invpcid ... via configuring the VMCS to disable the support in guest.

2. Raising the #PF (page fault exception)
   – sysexit, sysret... due to the S-page setting in all code pages.

3. Raising the #GP (general protection exception)
   – Segment-switching related instructions: mov to %ds, lcall...
SEIMI —— Privileged Instruction Prevention Component

• Since the application runs in ring 0, attackers may use the segment-switching instructions to switch to any segment, we need to control them.
SEIMI —— Privileged Instruction Prevention Component

• Since the application runs in ring 0, attackers may use the segment-switching instructions to **switch to any segment**, we need to control them.

• Problem:
  – Intel VT-x cannot intercept these instructions that could change the segment.
SEIMI —— Privileged Instruction Prevention Component

• Since the application runs in ring 0, attackers may use the segment-switching instructions to **switch to any segment**, we need to control them.

• Problem:
  – Intel VT-x cannot intercept these instructions that could change the segment.

!! How to intercept this type of instructions ???
• Observation
  – When changing a segment register, the hardware will use the target selector to access the segment descriptor table to obtain the target segment information.
SEIMI —— Privileged Instruction Prevention Component

• Observation
  – When changing a segment register, the hardware will use the target selector to access the segment descriptor table to obtain the target segment information.
  – If the segment descriptor table is empty, the CPU will raise a general protection exception (#GP) in this process.
SEIMI —— Privileged Instruction Prevention Component

• Observation
  – When changing a segment register, the hardware will use the target selector to access the segment descriptor table to obtain the target segment information.
  – If the segment descriptor table is empty, the CPU will raise a general protection exception (#GP) in this process.

Solution —— Emptying out this table to intercept such instructions.
SEIMI —— Privileged Instruction Prevention Component

• Observation
  – When changing a segment register, the hardware will use the target selector to access the segment descriptor table to obtain the target segment information.
  – If the segment descriptor table is empty, the CPU will raise a general protection exception (#GP) in this process.

Solution —— Emptying out this table to intercept such instructions.

• Problem:
  – How to ensure the normal execution (segment addressing)?
  – How to ensure the correct functionality of the segment-switching instructions?
SEIMI —— Privileged Instruction Prevention Component

- Segment-switching exception using descriptor cache.

<table>
<thead>
<tr>
<th>%ds:</th>
<th>Visible Part</th>
<th>Hidden Part</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selector</td>
<td>Base, Limit, Access</td>
</tr>
</tbody>
</table>
SEIMI —– Privileged Instruction Prevention Component

- Segment-switching exception using descriptor cache.
SEIMI —— Privileged Instruction Prevention Component

- Segment-switching exception using descriptor cache.

```
mov %ds:(%rax), %rbx
```

Segment-switching instruction will load the information into the descriptor cache.
SEIMI —— Privileged Instruction Prevention Component

• Segment-switching exception using descriptor cache.
  – X86 allows the descriptor cache to be inconsistent with the descriptor table.
SEIMI —— Privileged Instruction Prevention Component

- Segment-switching exception using descriptor cache.
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.

Filling the correct information into the cache via configuring the VMCS.

Segment-switching instruction will load the information into the descriptor cache.
**SEIMI —— Privileged Instruction Prevention Component**

- Segment-switching exception using descriptor cache.
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.

---

**Diagram:**

- **VMCS**
  - Guest
  - **CS/SS/DS...GS:**
  - Selector
  - Base, Limit, Access

- **%ds:**
  - Visible Part
  - Hidden Part
  - **Selector**
  - Base, Limit, Access

- **GDTR**
  - GDT
  - LDT Descriptor

- **LDTR**
  - LDT
  - Base, Limit, Access

---

1. **mov %ds:(%rax), %rbx**
2. **Filling the correct information into the cache via configuring the VMCS.**
3. **Segment-switching instruction will load the information into the descriptor cache.**
Segment-switching exception using descriptor cache.
- X86 allows the descriptor cache to be inconsistent with the descriptor table.

At last, emptying out the segment descriptor table.

Segment-switching instruction will load the information into the descriptor cache.

Filling the correct information into the cache via configuring the VMCS.
**SEIMI —— Privileged Instruction Prevention Component**

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.

At last, emptying out the segment descriptor table.

Emulating the function of the normal segment-switching instructions.

Segment-switching instruction will load the information into the descriptor cache.
SEIMI —— Privileged Instruction Prevention Component

Invalidating the Execution Effects.

– We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.
Invalidate the Execution Effects.

- We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.

- **CR0/CR4-related instructions.**
  - Configure guest/host masks and read shadows in VMCS.
  - The value of the %CR0/%CR4 read is all 0.
  - Write to them does not really modify the values;
Invalidating the Execution Effects.

– We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.

• CR0/CR4-related instructions.
  – Configure guest/host masks and read shadows in VMCS.
  – The value of the %CR0/%CR4 read is all 0.
  – Write to them does not really modify the values;

• SWAPGS, L[AR/SL], VER[R/W], CLI/STI …
  – More details are in the paper.
SEIMI —— Events Redirection Component

• **System-call Handling**
  – Convert the system calls to the hypercalls via mapping a code page.
    • Containing two instructions: `VMCALL` and `JMP *%RCX`.
    • The `IA32_LSTAR` MSR register in guest points to this page.
• **System-call Handling**
  – Convert the system calls to the hypercalls via mapping a code page.
    • Containing two instructions: `VMCALL` and `JMP *%RCX`.
    • The `IA32_LSTAR` MSR register in guest points to this page.
  – The kernel module vectors the `system_call_table` and calls the handlers.
SEIMI —— Events Redirection Component

• System-call Handling
  – Convert the system calls to the hypercalls via mapping a code page.
    • Containing two instructions: \texttt{VMCALL} and \texttt{JMP *%RCX}.
    • The \texttt{IA32_LSTAR} MSR register in guest points to this page.
  – The kernel module vectors the \texttt{system\_call\_table} and calls the handlers.

• Interrupts and Exceptions Handling
  – All these events trigger the VM exit via configuring the VMCS.
  – The kernel module checks the call gates and vectors the IDT.
SEIMI —— Events Redirection Component

• **System-call Handling**
  – Convert the system calls to the hypercalls via mapping a code page.
    • Containing two instructions: `VMCALL` and `JMP *%RCX`.
    • The `IA32_LSTAR` MSR register in guest points to this page.
  – The kernel module vectors the `system_call_table` and calls the handlers.

• **Interrupts and Exceptions Handling**
  – All these events trigger the VM exit via configuring the VMCS.
  – The kernel module checks the call gates and vectors the IDT.

• **Linux Signal Handling**
  – Check the signal queue, and switch the context via configuring the VMCS.
SEIMI —— Some Implementations

• Memory Management Implementations
  – Avoiding overlaps in the 254th and 255th entries.
  – Handling the VSYSCALL page.
  – Tracking updates of the PML4 page.
  – Avoiding accessing the kernel by exploiting the TLB.

• Other Implementations
  – Hardening system calls against confused deputy.
  – Starting and exiting the target process.
  – Defeating the concurrent attacks.
Outline

Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation
Performance Evaluation

- **Defenses and Isolation Schemes:**
  - **Defenses**: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  - **Isolation**: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes
Performance Evaluation

• Defenses and Isolation Schemes:
  – **Defenses:** O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  – **Isolation:** IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes

• **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  – lmbench v3.0-a9
Performance Evaluation

• **Defenses and Isolation Schemes:**
  – **Defenses:** O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  – **Isolation:** IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes

• **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  – lmbench v3.0-a9

• **Macrobenchmark** —— the overheads on different isolation schemes.
  – SPEC CPU2006 C/C++ benchmark with the ref input.
Performance Evaluation

• Defenses and Isolation Schemes:
  – **Defenses**: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  – **Isolation**: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes

• **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  – Imbench v3.0-a9

• **Macrobenchmark** —— the overheads on different isolation schemes.
  – SPEC CPU2006 C/C++ benchmark with the ref input.

• **Real-world applications**:
  – 4 Web servers: Nginx, Apache, Lighttpd, and Openlitespeed.
  – 4 Databases: MySQL, SQLite, Redis, and Memcached.
  – 4 JavaScript engines: ChakraCore, Google V8, JavaScriptCore, SpiderMonkey.
Microbenchmark —— Imbench

• We run Imbench directly on SEIMI to only evaluate the overhead on kernel operations.
Microbenchmark —— Imbench

- We run *Imbench* directly on SEIMI to only evaluate the overhead on kernel operations.

<table>
<thead>
<tr>
<th>Config</th>
<th>null call</th>
<th>null I/O</th>
<th>open close</th>
<th>select TCP</th>
<th>signal install</th>
<th>signal handle</th>
<th>fork proc</th>
<th>exec proc</th>
<th>sh proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>0.21</td>
<td>0.26</td>
<td>0.57</td>
<td>1.23</td>
<td>5.35</td>
<td>0.27</td>
<td>0.99</td>
<td>355</td>
<td>870</td>
</tr>
<tr>
<td>SEIMI</td>
<td><strong>0.71</strong></td>
<td><strong>0.82</strong></td>
<td><strong>1.33</strong></td>
<td><strong>2.58</strong></td>
<td><strong>6.11</strong></td>
<td><strong>0.79</strong></td>
<td><strong>3.02</strong></td>
<td><strong>463</strong></td>
<td><strong>1029</strong></td>
</tr>
<tr>
<td>Slowdown</td>
<td><strong>2.4X</strong></td>
<td><strong>2.2X</strong></td>
<td><strong>1.3X</strong></td>
<td><strong>1.1X</strong></td>
<td><strong>14%</strong></td>
<td><strong>1.9X</strong></td>
<td><strong>2.1X</strong></td>
<td><strong>30.4%</strong></td>
<td><strong>18.3%</strong></td>
</tr>
</tbody>
</table>

Latency on process-related kernel operations (in μs): smaller is better.
Microbenchmark —— lmbench

- We run lmbench directly on SEIMI to only evaluate the overhead on kernel operations.

1. Latency on process-related kernel operations (in $\mu$s): smaller is better.

<table>
<thead>
<tr>
<th></th>
<th>null call</th>
<th>null I/O</th>
<th>open close</th>
<th>select close</th>
<th>TCP install</th>
<th>signal handle</th>
<th>fork proc</th>
<th>exec proc</th>
<th>sh proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>0.21</td>
<td>0.26</td>
<td>0.57</td>
<td>1.23</td>
<td>5.35</td>
<td>0.27</td>
<td>0.99</td>
<td>355</td>
<td>870</td>
</tr>
<tr>
<td>SEIMI</td>
<td>0.71</td>
<td>0.82</td>
<td>1.33</td>
<td>2.58</td>
<td>6.11</td>
<td>0.79</td>
<td>3.02</td>
<td>463</td>
<td>1029</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2p/0K</th>
<th>2p/16K</th>
<th>2p/64K</th>
<th>8p/16K</th>
<th>8p/64K</th>
<th>16p/16K</th>
<th>16p/64K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>2.05</td>
<td>2.06</td>
<td>3.1</td>
<td>8.13</td>
<td>12.2</td>
<td>8.43</td>
<td>12.6</td>
</tr>
<tr>
<td>SEIMI</td>
<td>2.46</td>
<td>2.45</td>
<td>3.6</td>
<td>10.1</td>
<td>14.8</td>
<td>11.52</td>
<td>15.9</td>
</tr>
</tbody>
</table>

2. Context-switching latency (in $\mu$s): smaller is better.
Microbenchmark —— lmbench

- We run lmbench directly on SEIMI to only evaluate the overhead on kernel operations.

Latency on process-related kernel operations (in μs): smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>null call</th>
<th>null I/O</th>
<th>open close</th>
<th>select</th>
<th>signal install</th>
<th>signal handle</th>
<th>fork proc</th>
<th>exec sh proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>0.21</td>
<td>0.26</td>
<td>0.57</td>
<td>1.23</td>
<td>5.35</td>
<td>0.27</td>
<td>0.99</td>
<td>355</td>
</tr>
<tr>
<td>SEIMI</td>
<td>0.71</td>
<td>0.82</td>
<td>1.33</td>
<td>2.58</td>
<td>6.11</td>
<td>0.79</td>
<td>3.02</td>
<td>463</td>
</tr>
<tr>
<td>Slowdown</td>
<td>2.4X</td>
<td>2.2X</td>
<td>1.3X</td>
<td>1.1X</td>
<td>14%</td>
<td>1.9X</td>
<td>2.1X</td>
<td>30.4%</td>
</tr>
</tbody>
</table>

Context-switching latency (in μs): smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>2p/0K</th>
<th>2p/16K</th>
<th>2p/64K</th>
<th>8p/16K</th>
<th>8p/64K</th>
<th>16p/16K</th>
<th>16p/64K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>2.05</td>
<td>2.06</td>
<td>3.1</td>
<td>8.13</td>
<td>12.2</td>
<td>8.43</td>
<td>12.6</td>
</tr>
<tr>
<td>SEIMI</td>
<td>2.46</td>
<td>2.45</td>
<td>3.6</td>
<td>10.1</td>
<td>14.8</td>
<td>11.52</td>
<td>15.9</td>
</tr>
<tr>
<td>Slowdown</td>
<td>20.0%</td>
<td>18.9%</td>
<td>16.1%</td>
<td>24.2%</td>
<td>21.3%</td>
<td>36.7%</td>
<td>26.2%</td>
</tr>
</tbody>
</table>

File & VM system latency (in μs): smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>0K File Create</th>
<th>0K File Delete</th>
<th>10K File Create</th>
<th>10K File Delete</th>
<th>Mmap Latency</th>
<th>Prot Fault</th>
<th>Page Fault</th>
<th>100fd select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>5.4717</td>
<td>4.7816</td>
<td>10.9</td>
<td>6.6214</td>
<td>6779</td>
<td>0.636</td>
<td>0.1593</td>
<td>1.016</td>
</tr>
<tr>
<td>SEIMI</td>
<td>6.9623</td>
<td>5.3421</td>
<td>14.5</td>
<td>7.4527</td>
<td>12500</td>
<td>1.038</td>
<td>0.2128</td>
<td>1.705</td>
</tr>
<tr>
<td>Slowdown</td>
<td>27.2%</td>
<td>11.7%</td>
<td>33.0%</td>
<td>12.6%</td>
<td>84.4%</td>
<td>63.2%</td>
<td>33.6%</td>
<td>67.8%</td>
</tr>
</tbody>
</table>
Microbenchmark —— Imbench

- We run **Imbench** directly on SEIMI to only evaluate the overhead on kernel operations.

1. **Latency on process-related kernel operations (in μs):** smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>null call</th>
<th>null I/O</th>
<th>open close</th>
<th>select</th>
<th>signal TCP</th>
<th>signal install</th>
<th>fork proc</th>
<th>exec proc</th>
<th>sh proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>0.21</td>
<td>0.26</td>
<td>0.57</td>
<td>1.23</td>
<td>5.35</td>
<td>0.27</td>
<td>0.99</td>
<td>355</td>
<td>870</td>
</tr>
<tr>
<td>SEIMI</td>
<td>0.71</td>
<td>0.82</td>
<td>1.33</td>
<td>2.58</td>
<td>6.11</td>
<td>0.79</td>
<td>3.02</td>
<td>463</td>
<td>1029</td>
</tr>
<tr>
<td>Slowdown</td>
<td>2.4X</td>
<td>2.2X</td>
<td>1.3X</td>
<td>1.1X</td>
<td>14%</td>
<td>1.9X</td>
<td>2.1X</td>
<td>30.4%</td>
<td>18.3%</td>
</tr>
</tbody>
</table>

2. **Context-switching latency (in μs):** smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>2p/0K File Create</th>
<th>2p/16K File Create</th>
<th>2p/64K Mmap Latency</th>
<th>8p/16K Prot Fault</th>
<th>8p/64K Page Fault</th>
<th>16p/16K select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>2.05</td>
<td>2.06</td>
<td>3.1</td>
<td>8.13</td>
<td>12.2</td>
<td>8.43</td>
</tr>
<tr>
<td>SEIMI</td>
<td>2.46</td>
<td>2.45</td>
<td>3.6</td>
<td>10.1</td>
<td>14.8</td>
<td>11.52</td>
</tr>
<tr>
<td>Slowdown</td>
<td>20.0%</td>
<td>18.9%</td>
<td>16.1%</td>
<td>24.2%</td>
<td>21.3%</td>
<td>36.7%</td>
</tr>
</tbody>
</table>

3. **File & VM system latency (in μs):** smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>0K File Create</th>
<th>0K File Delete</th>
<th>10K File Create</th>
<th>10K File Delete</th>
<th>Mmap Prot</th>
<th>Mmap Page</th>
<th>100fd select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>5.4717</td>
<td>4.7816</td>
<td>10.9</td>
<td>6.6214</td>
<td>6779</td>
<td>0.636</td>
<td>0.1593</td>
</tr>
<tr>
<td>SEIMI</td>
<td>6.9623</td>
<td>5.3421</td>
<td>14.5</td>
<td>7.4527</td>
<td>12500</td>
<td>1.038</td>
<td>0.2128</td>
</tr>
<tr>
<td>Slowdown</td>
<td>27.2%</td>
<td>11.7%</td>
<td>33.0%</td>
<td>12.6%</td>
<td>84.4%</td>
<td>63.2%</td>
<td>33.6%</td>
</tr>
</tbody>
</table>

4. **Local-communication latency (in μs):** smaller is better.

<table>
<thead>
<tr>
<th>Config</th>
<th>Pipe/AF_UNIX</th>
<th>UDP/RPC/ TCP</th>
<th>TCP/RPC/TCP</th>
<th>TCP_CONN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>5.582</td>
<td>9.2</td>
<td>9.883</td>
<td>14.9</td>
</tr>
<tr>
<td>SEIMI</td>
<td>7.428</td>
<td>11.7</td>
<td>11.7</td>
<td>20</td>
</tr>
<tr>
<td>Slowdown</td>
<td>33.1%</td>
<td>27.2%</td>
<td>18.4%</td>
<td>34.2%</td>
</tr>
</tbody>
</table>
Macrobenchmark —— SPEC CPU 2006 benchmark

• Compared with the **MPX-based scheme**, **SEIMI** achieves a lower performance overhead on average, with the average reduction of **33.97%**.
Macrobenchmark —— SPEC CPU 2006 benchmark

- Compared with the **MPX-based scheme**, **SEIMI** achieves a lower performance overhead on average, with the average reduction of **33.97%**.
- Compared to the **MPK-based scheme**, **SEIMI** is more efficient in almost all test cases, and with the average reduction of **42.3%** (maximum is **133.33%**).
Macrobenchmark —— SPEC CPU 2006 benchmark

• Performance Analysis: MPK vs. SEIMI

The impact of permission-switching frequency on performance of MPK and SEIMI.
Macrobenchmark —— SPEC CPU 2006 benchmark

- Performance Analysis: MPK vs. SEIMI

The impact of permission-switching frequency on performance of MPK and SEIMI.

Compared to MPK, as the access permission switching frequency increases, the performance gain of SEIMI becomes more apparent.
The impact of bound-checking frequency (\texttt{CFreq}) and permission-switching frequency (\texttt{SFreq}) on performance.
Macrobenchmark —— SPEC CPU 2006 benchmark

- Performance Analysis: **MPX vs. SEIMI**

When the bound-checking frequency is 52 times of the access permission switching frequency, **SEIMI** is more efficient than **MPX** in most cases.

The impact of bound-checking frequency (**CFreq**) and permission-switching frequency (**SFreq**) on performance.
Real-world Applications

- **SEIMI** is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.

<table>
<thead>
<tr>
<th>Applications</th>
<th>OCFI IH</th>
<th>MPX</th>
<th>MPK</th>
<th>SEIMI</th>
<th>SS IH</th>
<th>MPX</th>
<th>MPK</th>
<th>SEIMI</th>
<th>CPI IH</th>
<th>MPX</th>
<th>MPK</th>
<th>SEIMI</th>
<th>AG IH</th>
<th>MPX</th>
<th>MPK</th>
<th>SEIMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nginx</td>
<td>1.10%</td>
<td>3.86%</td>
<td>5.32%</td>
<td>1.77%</td>
<td>1.86%</td>
<td>7.33%</td>
<td>10.49%</td>
<td>2.43%</td>
<td>0.90%</td>
<td>6.38%</td>
<td>8.95%</td>
<td>3.08%</td>
<td>0.74%</td>
<td>7.60%</td>
<td>5.27%</td>
<td>2.01%</td>
</tr>
<tr>
<td>Apache</td>
<td>1.58%</td>
<td>4.71%</td>
<td>5.18%</td>
<td>1.82%</td>
<td>1.64%</td>
<td>6.36%</td>
<td>6.83%</td>
<td>2.15%</td>
<td>1.45%</td>
<td>5.01%</td>
<td>2.58%</td>
<td>1.80%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>2.94%</td>
<td>3.42%</td>
<td>5.74%</td>
<td>4.46%</td>
<td>2.77%</td>
<td>6.85%</td>
<td>6.33%</td>
<td>3.78%</td>
<td>1.70%</td>
<td>6.83%</td>
<td>3.42%</td>
<td>2.46%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Olpenitespeed</td>
<td>1.44%</td>
<td>5.39%</td>
<td>3.88%</td>
<td>1.61%</td>
<td>1.04%</td>
<td>1.92%</td>
<td>3.39%</td>
<td>1.42%</td>
<td>0.91%</td>
<td>2.89%</td>
<td>2.99%</td>
<td>1.38%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>MySQL</td>
<td>1.75%</td>
<td>12.09%</td>
<td>8.08%</td>
<td>3.79%</td>
<td>3.17%</td>
<td>9.60%</td>
<td>11.99%</td>
<td>3.94%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SQLite</td>
<td>1.61%</td>
<td>2.11%</td>
<td>2.70%</td>
<td>1.84%</td>
<td>1.42%</td>
<td>3.46%</td>
<td>2.19%</td>
<td>1.94%</td>
<td>1.36%</td>
<td>3.11%</td>
<td>2.66%</td>
<td>2.18%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Redis</td>
<td>4.51%</td>
<td>5.46%</td>
<td>13.12%</td>
<td>10.31%</td>
<td>1.18%</td>
<td>2.81%</td>
<td>5.36%</td>
<td>5.06%</td>
<td>1.24%</td>
<td>4.47%</td>
<td>4.81%</td>
<td>3.93%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Memcached</td>
<td>1.64%</td>
<td>6.64%</td>
<td>7.46%</td>
<td>2.74%</td>
<td>2.38%</td>
<td>5.57%</td>
<td>8.13%</td>
<td>3.44%</td>
<td>1.04%</td>
<td>6.02%</td>
<td>7.28%</td>
<td>1.60%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ChakraCore</td>
<td>3.03%</td>
<td>12.09%</td>
<td>9.90%</td>
<td>4.10%</td>
<td>4.37%</td>
<td>7.92%</td>
<td>10.09%</td>
<td>5.15%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>V8</td>
<td>2.57%</td>
<td>11.63%</td>
<td>5.04%</td>
<td>3.37%</td>
<td>2.05%</td>
<td>8.01%</td>
<td>4.05%</td>
<td>2.96%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>JavaScriptCore</td>
<td>2.22%</td>
<td>22.87%</td>
<td>39.65%</td>
<td>26.81%</td>
<td>20.69%</td>
<td>38.34%</td>
<td>47.77%</td>
<td>31.82%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SpiderMonkey</td>
<td>1.75%</td>
<td>9.32%</td>
<td>7.63%</td>
<td>4.15%</td>
<td>1.84%</td>
<td>7.56%</td>
<td>7.79%</td>
<td>5.19%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

All overheads are normalized to the unprotected applications. “—” represents the defense failed to compile or run it.
Real-world Applications

• **SEIMI** is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.
  – **SEIMI** is much more efficient than **MPK** for all 32 cases.

<table>
<thead>
<tr>
<th>Applications</th>
<th>OCFI</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
</tr>
<tr>
<td>Nginx</td>
<td>1.10%</td>
<td>3.86%</td>
<td>5.32%</td>
<td>1.77%</td>
<td>1.86%</td>
<td>7.33%</td>
<td>10.49%</td>
<td>2.43%</td>
<td>0.90%</td>
<td>6.38%</td>
<td>8.95%</td>
<td>3.08%</td>
<td>0.74%</td>
</tr>
<tr>
<td>Apache</td>
<td>1.58%</td>
<td>4.71%</td>
<td>2.82%</td>
<td>1.82%</td>
<td>1.64%</td>
<td>6.36%</td>
<td>6.83%</td>
<td>2.15%</td>
<td>1.45%</td>
<td>5.01%</td>
<td>2.58%</td>
<td>1.80%</td>
<td>—</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>2.94%</td>
<td>3.42%</td>
<td>5.74%</td>
<td>4.46%</td>
<td>2.77%</td>
<td>6.85%</td>
<td>6.33%</td>
<td>3.78%</td>
<td>1.70%</td>
<td>6.83%</td>
<td>3.42%</td>
<td>2.46%</td>
<td>—</td>
</tr>
<tr>
<td>Openlitespeed</td>
<td>1.44%</td>
<td>5.39%</td>
<td>3.88%</td>
<td>1.61%</td>
<td>1.04%</td>
<td>1.92%</td>
<td>3.39%</td>
<td>1.42%</td>
<td>0.91%</td>
<td>2.89%</td>
<td>2.99%</td>
<td>1.38%</td>
<td>—</td>
</tr>
<tr>
<td>MySQL</td>
<td>1.75%</td>
<td>12.09%</td>
<td>8.08%</td>
<td>3.79%</td>
<td>3.17%</td>
<td>9.60%</td>
<td>11.99%</td>
<td>3.94%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SQLite</td>
<td>1.61%</td>
<td>2.11%</td>
<td>2.70%</td>
<td>1.84%</td>
<td>1.42%</td>
<td>3.46%</td>
<td>2.19%</td>
<td>1.94%</td>
<td>1.36%</td>
<td>3.11%</td>
<td>2.66%</td>
<td>2.18%</td>
<td>—</td>
</tr>
<tr>
<td>Redis</td>
<td>4.51%</td>
<td>5.46%</td>
<td>13.12%</td>
<td>10.31%</td>
<td>1.18%</td>
<td>2.81%</td>
<td>5.36%</td>
<td>5.06%</td>
<td>1.24%</td>
<td>4.47%</td>
<td>4.81%</td>
<td>3.93%</td>
<td>—</td>
</tr>
<tr>
<td>Memcached</td>
<td>1.64%</td>
<td>6.64%</td>
<td>7.46%</td>
<td>2.74%</td>
<td>2.38%</td>
<td>5.57%</td>
<td>8.13%</td>
<td>3.44%</td>
<td>1.04%</td>
<td>6.02%</td>
<td>7.28%</td>
<td>1.60%</td>
<td>—</td>
</tr>
<tr>
<td>ChakraCore</td>
<td>3.03%</td>
<td>12.09%</td>
<td>9.90%</td>
<td>4.10%</td>
<td>4.37%</td>
<td>7.92%</td>
<td>10.09%</td>
<td>5.15%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>V8</td>
<td>2.57%</td>
<td>11.63%</td>
<td>5.04%</td>
<td>3.37%</td>
<td>2.05%</td>
<td>8.01%</td>
<td>4.05%</td>
<td>2.96%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>JavaScriptCore</td>
<td>2.22%</td>
<td>22.87%</td>
<td>39.65%</td>
<td>26.81%</td>
<td>20.69%</td>
<td>38.34%</td>
<td>47.77%</td>
<td>31.82%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SpiderMonkey</td>
<td>1.75%</td>
<td>9.32%</td>
<td>7.63%</td>
<td>4.15%</td>
<td>1.84%</td>
<td>7.56%</td>
<td>7.79%</td>
<td>5.19%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

All overheads are normalized to the unprotected applications. “—” represents the defense failed to compile or run it.
Real-world Applications

• **SEIMI** is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.
  
  – **SEIMI** is much more efficient than **MPK** for all 32 cases.
  
  – **SEIMI** is much more efficient than **MPX** for 28 cases.

<table>
<thead>
<tr>
<th>Applications</th>
<th>OCFI</th>
<th></th>
<th></th>
<th></th>
<th>SS</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>CPI</th>
<th></th>
<th></th>
<th></th>
<th>AG</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td>IH</td>
<td>MPX</td>
<td>MPK</td>
<td>SEIMI</td>
<td></td>
</tr>
<tr>
<td>Nginx</td>
<td>1.10%</td>
<td>3.86%</td>
<td>5.32%</td>
<td>1.77%</td>
<td>1.86%</td>
<td>7.33%</td>
<td>10.49%</td>
<td>2.43%</td>
<td>0.90%</td>
<td>6.38%</td>
<td>8.95%</td>
<td>3.08%</td>
<td>0.74%</td>
<td>7.60%</td>
<td>5.27%</td>
<td>2.01%</td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td>1.58%</td>
<td>4.71%</td>
<td>2.82%</td>
<td>1.82%</td>
<td>1.64%</td>
<td>6.36%</td>
<td>6.83%</td>
<td>2.15%</td>
<td>1.45%</td>
<td>5.01%</td>
<td>2.58%</td>
<td>1.80%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Lighttpd</td>
<td>2.94%</td>
<td>3.42%</td>
<td>5.74%</td>
<td>4.46%</td>
<td>2.77%</td>
<td>6.85%</td>
<td>6.33%</td>
<td>3.78%</td>
<td>1.70%</td>
<td>6.83%</td>
<td>3.42%</td>
<td>2.46%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Openlitenspeed</td>
<td>1.44%</td>
<td>5.39%</td>
<td>3.88%</td>
<td>1.61%</td>
<td>1.04%</td>
<td>1.92%</td>
<td>3.39%</td>
<td>1.42%</td>
<td>0.91%</td>
<td>2.89%</td>
<td>2.99%</td>
<td>1.38%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td>1.75%</td>
<td>12.09%</td>
<td>8.08%</td>
<td>3.79%</td>
<td>3.17%</td>
<td>9.60%</td>
<td>11.99%</td>
<td>3.94%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>SQLite</td>
<td>1.61%</td>
<td>2.11%</td>
<td>2.70%</td>
<td>1.84%</td>
<td>1.42%</td>
<td>3.46%</td>
<td>2.19%</td>
<td>1.94%</td>
<td>1.36%</td>
<td>3.11%</td>
<td>2.66%</td>
<td>2.18%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Redis</td>
<td>4.51%</td>
<td>5.46%</td>
<td>13.12%</td>
<td>10.31%</td>
<td>1.18%</td>
<td>2.81%</td>
<td>5.36%</td>
<td>5.06%</td>
<td>1.24%</td>
<td>4.47%</td>
<td>4.81%</td>
<td>3.93%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Memcached</td>
<td>1.64%</td>
<td>6.64%</td>
<td>7.46%</td>
<td>2.74%</td>
<td>2.38%</td>
<td>5.57%</td>
<td>8.13%</td>
<td>3.44%</td>
<td>1.04%</td>
<td>6.02%</td>
<td>7.28%</td>
<td>1.60%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>ChakraCore</td>
<td>3.03%</td>
<td>12.09%</td>
<td>9.90%</td>
<td>4.10%</td>
<td>4.37%</td>
<td>7.92%</td>
<td>10.09%</td>
<td>5.15%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>2.57%</td>
<td>11.63%</td>
<td>5.04%</td>
<td>3.37%</td>
<td>2.05%</td>
<td>8.01%</td>
<td>4.05%</td>
<td>2.96%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>JavaScriptCore</td>
<td>2.22%</td>
<td>22.87%</td>
<td>39.65%</td>
<td>26.81%</td>
<td>20.69%</td>
<td>38.34%</td>
<td>47.77%</td>
<td>31.82%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>SpiderMonkey</td>
<td>1.75%</td>
<td>9.32%</td>
<td>7.63%</td>
<td>4.15%</td>
<td>1.84%</td>
<td>7.56%</td>
<td>7.79%</td>
<td>5.19%</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

All overheads are normalized to the unprotected applications. “—” represents the defense failed to compile or run it.
Conclusion

• We propose a highly efficient intra-process memory isolation technique **SEIMI**, which leverages the widely used hardware feature — **SMAP**.

• To avoid introducing security threats, we propose multiple new techniques to ensure the user code run in ring 0 securely.

• We believe that **SEIMI** can not only benefit existing defenses, but also open the new research direction …
  – Enabling the efficient access to a variety of privileged hardware features, which does not require context switch, to defenses.
Any Questions?

wangzhe12@ict.ac.cn