SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture

Zhenyu Ning¹,², Yinqian Zhang³, and Fengwei Zhang²

¹Wayne State University, ²Southern University of Science and Technology, ³The Ohio State University
Outline

• *Introduction*

• Pointer Integrity

• Memory Boundary Protection

• Dynamic Taint Analysis

• Implementation

• Conclusion
Introduction

• The RISC-V architecture is well-known for its open nature.
  • Open Source, No License fee
  • Open to new design and extension

• Open to challenge.
  • Security problems in x86 and ARM architecture remains on RISC-V platforms.
  • E.g., pointer integrity, memory boundary protection, and dynamic taint analysis.

SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture
Any effective defense on RISC-V?
Outline

- Introduction
- **Pointer Integrity**
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- Conclusion
Pointer Integrity

• To ensure that the pointer is not corrupted.
• Code-pointer Integrity and Data-pointer Integrity.

```c
if *x0 = 0 then
    x1 = addr1
else
    x1 = addr2
jmp to x1
```
Pointer Integrity

• To ensure that the pointer is not corrupted.
• Code-pointer Integrity and Data-pointer Integrity.

```c
if *x0 = 0 then
  x1 = addr1
else
  x1 = addr2
jmp to x1
```
Pointer Integrity

- To ensure that the pointer is not corrupted.
- Code-pointer Integrity and Data-pointer Integrity.

```c
if *x0 = 0 then
    x1 = addr1
else
    x1 = addr2
jmp to x1
```

* `*x0 = 0`
  - `x1 = addr3`

**Code-pointer Attack**
Pointer Integrity

- To ensure that the pointer is not corrupted.
- Code-pointer Integrity and Data-pointer Integrity.

```c
if *x0 = 0 then
    x1 = addr1
else
    x1 = addr2
jmp to x1
```

* branding changes for presentation
Pointer Integrity

• To ensure that the pointer is not corrupted.
• Code-pointer Integrity and Data-pointer Integrity.

```plaintext
if *x0 = 0 then
  x1 = addr1
else
  x1 = addr2
jmp to x1
```

*x0 = 2
Pointer Integrity

- To ensure that the pointer is not corrupted.
- Code-pointer Integrity and Data-pointer Integrity.

```plaintext
if \( \star x_0 = 0 \) then
    x1 = addr1
else
    x1 = addr2
jmp to x1
```

```plaintext
\*x0 = 2
x1 = addr2
```

Data-pointer Attack
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.

Buffer Overflow Attack

---

SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture
Pointer Integrity: Buffer Overflow

• Start of the attack: In most cases, a buffer overflow vulnerability.

Buffer Overflow Attack
Pointer Integrity: Buffer Overflow

• Start of the attack: In most cases, a buffer overflow vulnerability.

Buffer Overflow Attack
• Start of the attack: In most cases, a buffer overflow vulnerability.
• Start of the attack: In most cases, a buffer overflow vulnerability.

Buffer Overflow Attack
**Pointer Integrity: Canary**

- **Stack Canary**\(^1\): The most widely used defense to buffer overflow attack.

---

**Buffer Overflow Attack**

- Stack Pointer
- Stack Pointer
**Pointer Integrity: Canary**

- **Stack Canary**\cite{1}: The most widely used defense to buffer overflow attack.

![Diagram showing Buffer Overflow Attack]

**Buffer Overflow Attack**

- Canary is changed by overflow

---

SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture
Pointer Integrity: Canary

- **Stack Canary**\(^1\): The most widely used defense to buffer overflow attack.

- Weakness:
  - Easy to bypass\(^2\)
  - Not efficient to defend against data-pointer attack
**Pointer Integrity: PAC**

- **Pointer Authentication Code**\(^{[3]}\) is introduced in 64-bit ARMv8.3 architecture.

![A pointer in 64-bit system](image)

Is it really necessary to use a 64-bit address?
Is it really necessary to use a 64-bit address?

- $2^{64}$ bit = 16384 PB = 16.8 millions TB = 17.2 billions GB
- **Summit**: 10 PB memory
- **Sunway TaihuLight**: 1.32 PB memory
- **Linux**: Up to 128 TB virtual memory
- **Windows**: Up to 16 TB virtual memory
**Pointer Integrity: PAC**

- **Pointer Authentication Code**[^3] is introduced in 64-bit ARMv8.3 architecture.

Pointer Integrity: PAC

- **Pointer Authentication Code**\(^3\) is introduced in 64-bit ARMv8.3 architecture.

- Pointer Value + 64-bit Context Value + 128-bit Secret Key => PAC

- Up to 48 bits for virtual address, and at least 7 bits for PAC

---

### Virtual Address Structure

<table>
<thead>
<tr>
<th>63</th>
<th>54</th>
<th>48</th>
<th>47</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC</td>
<td>Virtual Address</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pointer Integrity: PAC

- PAC is good, but the deployment is painful.
  - The mechanism is released with ARMv8.3 architecture since 2016.
  - ARM does not release any processor with ARMv8.3 till now.

- The only processors with PAC support are Apple A12 and A13.
  - Closed ecosystem.
  - No available to system developers.
Pointer Integrity: RISC-V

- RISC-V based PAC
  - A group of new hardware instructions
    - Forge PAC, examine PAC, strip PAC
  - New registers for storing the 128-bit secret key
    - Secret keys for data pointers and code pointers
  - Hardware-based crypto engine
    - Generate PAC from pointer and 64-bit context value
Outline

• Introduction
• Pointer Integrity
• **Memory Boundary Protection**
• Dynamic Taint Analysis
• Implementation
• Conclusion
Memory Boundary Protection

• To ensure the memory access won’t go out of its expected boundary.

```c
int a[10];
...
a[0]
a[1]
...
a[8]
a[9]
...

a[8] = 1
...
```
Memory Boundary Protection

- To ensure the memory access won’t go out of its expected boundary.

```c
int a[10];
...
a[0]
a[1]
...
a[8]
a[9]
...
```

Memory Out of Boundary
Memory Boundary Protection: Address Sanitizer

- **Address Sanitizer**\(^4\): Use redzones to detect out-of-bound access.

```c
int a[10];

... a[0] a[1] ...

... a[8] a[9] ...
```

Redzone

Redzone

Redzone

Redzone

SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture
Memory Boundary Protection: Address Sanitizer

- **Address Sanitizer**[^1]: Use redzones to detect out-of-bound access.

```
int a[10];
Redzone
Redzone
Redzone
a[0]
...
a[9]
Redzone
Redzone

a[10] = 1
Redzone
Redzone
Redzone
```

Out-of-bound access

[^1]: Reference to a section in the text.
Memory Boundary Protection: Address Sanitizer

• **Address Sanitizer**[^4]: Use redzones to detect out-of-bound access.

• Weakness:
  - Large memory overhead
  - Large performance overhead
  - False negative is possible
Memory Boundary Protection: Intel MPX

- **Intel MPX**\(^5\): An architecture extension dedicated for memory bound protection.

![Memory Boundary Protection Diagram](image)

Table of Contents:
- Pointer Address
- Base Address of Bounds Directory
- Bound Directory
  - Table 0
  - Table 1
  - Table 2
  - Table 3
  - Table 4
- Bound Table 3
  - Entry 0
  - Entry 1
  - Entry 2
  - Entry 3
  - Entry 4
- Upper Bound
- Lower Bound
Memory Boundary Protection: Intel MPX

- **Intel MPX**\(^5\): An architecture extension dedicated for memory bound protection.

- Weakness:
  - Performance overhead for two-layer translation
  - Multithread not support
  - Not production ready, support will be removed from GCC 9
Memory Boundary Protection: RISC-V

- RISC-V based Memory Boundary Protection

<table>
<thead>
<tr>
<th>63</th>
<th>54</th>
<th>48</th>
<th>47</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>??</td>
<td>PAC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Use the head bits for memory bounds
  - 9 bits if PAC is implemented
  - 16 bits if PAC is not implemented

[^6]: SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture
Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
  - *Dynamic Taint Analysis*
- Implementation
- Conclusion
Dynamic Taint Analysis

• Analysis the information flow of specific objects.

• Example scenario: Privacy leakage detection

```
......
char* password = getInput();
char* copied = copy(password);
printf("copied: %s\n", copied);
......
```

Taint Source
- `getInput`

Taint Sink
- `printf`
Dynamic Taint Analysis

• Analysis the information flow of specific objects.

• Example scenario: Privacy leakage detection

```c
......
char* password = getInput();
char* copied = copy(password);
printf("copied: %s\n", copied);
......
```

<table>
<thead>
<tr>
<th>Taint Source</th>
<th>Taint Sink</th>
<th>Tainted Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getInput</code></td>
<td><code>printf</code></td>
<td><code>password</code></td>
</tr>
</tbody>
</table>
Dynamic Taint Analysis

- Analysis the information flow of specific objects.
- Example scenario: Privacy leakage detection

```c
......
char* password = getInput();
char* copied = copy(password);
printf("copied: %s\n", copied);
......
```

<table>
<thead>
<tr>
<th>Taint Source</th>
<th>Tainted Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getInput</code></td>
<td><code>password</code></td>
</tr>
<tr>
<td><code>printf</code></td>
<td><code>copied</code></td>
</tr>
</tbody>
</table>
Dynamic Taint Analysis

• Analysis the information flow of specific objects.
• Example scenario: Privacy leakage detection

......
char* password = getInput();
char* copied = copy(password);
printf("copied: %s\n", copied);
......

Taint Source

getInput

Taint Sink

printf

Tainted Variable

password

copied

Taint Path Founded!
Dynamic Taint Analysis

How to learn the taint propagation from "password" to "copied"?

- Heavy instrumentation
- Add tons of instructions to monitor the data flow
Dynamic Taint Analysis

• **Labelled RISC-V Architecture**[^7]: Every hardware request is attached with a label.

[^7]: Every hardware request is attached with a label.
Dynamic Taint Analysis

- **Labelled RISC-V Architecture**[^7]: Every hardware request is attached with a label.
  - Use the label to represent taint flag
    - Automatically propagation via hardware support
    - No instrumentation required
  - Use the Control Logic (CL) to achieve detection

What about the propagation outside of hardware request?
What about the propagation outside of hardware request?

• Allocate a few bits from the unused bits in 64-bit pointer
  • In coarse-gained analysis, 1 bit is sufficient
  • This bit automatically transfers during the execution of data operation instructions.
• Feed to the $DS-id$ register during hardware request
Outline

• Introduction
• Pointer Integrity
• Memory Boundary Protection
• Dynamic Taint Analysis
  • Implementation
• Conclusion
Implementation

• **SecLabel**: Enhancing RISC-V Platform Security
Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- Conclusion
Conclusion

• In light of the PAC in ARMv8.3, we can leverage the open feature of RISC-V and implement similar mechanism for pointer integrity.

• With addition bits in the head of a pointer address in 64-bit or 128-bit RISC-V architecture, an enhanced memory boundary protection can be deployed.

• Combining the labelled RISC-V architecture and unused bits in an address, we are able to facilitate the existing dynamic taint analysis.
Reference


Thanks!

ningzy2019@mail.sustech.edu.cn