Zero-Overhead Resilient Operation
Under
Pointer Integrity Attacks

Mohamed Tarek Ibn Ziad, Miguel Arroyo, Evgeny Manzhosov, and Simha Sethumadhavan
Security?
Security!
Inefficient security inconveniences the user

Most end users want security, but do not want the inconvenience of having it.
Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.
Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.

Energy Drain
Inefficient protections drain precious resources such as battery.
Inefficient security inconveniences the user

**Slow Performance**
User want a snappy experience and security tends to detract from it.

**Energy Drain**
Inefficient protections drain precious resources such as battery.

**System Stability**
Users can’t be bothered with updates and patches.
Inefficient security inconveniences the user

**Slow Performance**
User want a snappy experience and security tends to detract from it.

**Energy Drain**
Inefficient protections drain precious resources such as battery.

**System Stability**
Users can’t be bothered with updates and patches.
Return Address Protection

CALL <Foo>
STORE
RET

Program

Memory
Return Address Protection

CALL <Foo>
STORE
RET

Program

Memory

Return Address
Return Address Protection

Program

CALL <Foo>
STORE
RET

Memory

Return Address
Return Address Protection

CALL <Foo>
STORE
RET

Program

Memory

Return Address
Return Address Protection

CALL <Foo>
STORE
RET

Program

Memory

Return Address
Return Address Protection
Return Address Protection

ZeRØ uses advisory exceptions to avoid crashing when under attack.

Program

CALL <Foo>
STORE
RET

Memory

Return Address
Code Pointer Integrity

Program

CPtrST
...
CPtrLD

Memory

Function Pointer
Code Pointer Integrity

Program

Memory

CPtrST
...
CPtrLD

Function Pointer

18
Code Pointer Integrity

CPtrST
STORE
CPtrLD

Program

Memory
Data Pointer Integrity

Works in the same way as Code Pointer Integrity but for data pointers!
ISA Extensions
ZeRØ ISA Extensions

1. Return Address Integrity: None. Relies on Call Return semantics.
ZeRØ ISA Extensions

1. Return Address Integrity
   - None. Relies on Call Return semantics.

2. Code Pointer Integrity
   - CPtrST/CPtrLD Address, Value

3. Data Pointer Integrity
   - DPtrST/DPtrLD Address, Value
ZeRØ ISA Extensions

1. Return Address Integrity
   - None. Relies on Call Return semantics.

2. Code Pointer Integrity
   - CPtrST/CPtrLD Address, Value
   - Same layout are regular load & stores!

3. Data Pointer Integrity
   - DPtrST/DPtrLD Address, Value
ZeRØ ISA Extensions

1. Return Address Integrity: None. Relies on Call Return semantics.
2. Code Pointer Integrity: CPtrST/CPtrLD Address, Value
3. Data Pointer Integrity: DPtrST/DPtrLD Address, Value

ClearMeta Address, Mask
# ZeRØ ISA Extensions

<table>
<thead>
<tr>
<th></th>
<th>Return Address Integrity</th>
<th>None. Relies on Call Return semantics.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Code Pointer Integrity</td>
<td>CPtrST/CPtrLD Address, Value</td>
</tr>
<tr>
<td>3</td>
<td>Data Pointer Integrity</td>
<td>DPtrST/DPtrLD Address, Value</td>
</tr>
</tbody>
</table>

- Invoked on `free` or `delete`.

- **ClearMeta Address, Mask**
Cache Line Formats
Cache Line Formats

0 1 2 3 4 5 6 7

Normal
Cache Line Formats

A B C D E

Normal

Program Pointers
Cache Line Formats
## Cache Line Formats

### Format Encoding Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return address</td>
<td>01</td>
</tr>
</tbody>
</table>

**bit-vector**

```
A B C D E
```

**Normal**

```
Program Pointers
```
Cache Line Formats

Format Encoding Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return address</td>
<td>01</td>
</tr>
<tr>
<td>Function pointer</td>
<td>10</td>
</tr>
</tbody>
</table>

bit-vector

Program Pointers

Normal
Cache Line Formats

Format Encoding Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return address</td>
<td>01</td>
</tr>
<tr>
<td>Function pointer</td>
<td>10</td>
</tr>
<tr>
<td>Data pointer</td>
<td>11</td>
</tr>
</tbody>
</table>

bit-vector

Normal

Program Pointers
Cache Line Formats

Format Encoding Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular data</td>
<td>00</td>
</tr>
<tr>
<td>Return address</td>
<td>01</td>
</tr>
<tr>
<td>Function pointer</td>
<td>10</td>
</tr>
<tr>
<td>Data pointer</td>
<td>11</td>
</tr>
</tbody>
</table>

**bit-vector**

Normal

Program Pointers
Cache Line Formats

Format Encoding Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular data</td>
<td>00</td>
</tr>
<tr>
<td>Return address</td>
<td>01</td>
</tr>
<tr>
<td>Function pointer</td>
<td>10</td>
</tr>
<tr>
<td>Data pointer</td>
<td>11</td>
</tr>
</tbody>
</table>

This introduces a 3.125% area overhead.
Cache Line Formats

Our Metadata: Encoded within unused pointer bits.
Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

Normal

```
A B C D E
```

Has Pointers?

Protected

```
Y
```

```
Header
A B C D E
```
Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

Program Pointers

Normal

Has Pointers?

Protected

Has Pointers?

Normal
Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

Program Pointers

Normal

A B C D E

Normal

0 1 2 3 4 5 6 7

Extra bit adds 0.2% area overhead.

Protected

Has Pointers?

Y

Has Pointers?

N

Header A B C D E

Normal

0 1 2 3 4 5 6 7
Cache Line Formats
Cache Line Formats

8-byte chunk

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>
Cache Line Formats

8-byte chunk

0 1 2 3 4 5 6 7

0 1 3 4 5 6 7
Cache Line Formats

Has Pointers?

Highlight the byte chunk that has pointers.

- For the first row, the byte chunk is not pointing to another chunk, so the answer is **N**.
- For the second row, the byte chunk is pointing to another chunk, so the answer is **Y**.
Cache Line Formats

Has Pointers?

N

Y

8-byte chunk

0 1 2 3 4 5 6 7

010 11

0 1 3 4 5 6 7
Cache Line Formats

010 11

Has Pointers?

N

Y

8-byte chunk
Cache Line Formats

0 010 11

8-byte chunk

Has Pointers?

N

Y
Cache Line Formats

Header Size?

6 bits 0 010 11

8-byte chunk

Has Pointers? N Y
Cache Line Formats

Header Size?

6 bits: 0 010 11

8-byte chunk

Has Pointers?

N

Y

Y

48
Cache Line Formats

Header Size?

6 bits: 0 010 11

Has Pointers? N

8-byte chunk:
0 1 2 3 4 5 6 7

12 bits: 010 11 101 11

Has Pointers? Y

8-byte chunk:
0 1 3 4 5 6 7
Cache Line Formats

Header Size?

- 6 bits: 0 010 11
- 12 bits: 10 010 11 101 11

8-byte chunk

- 0 1 2 3 4 5 6 7
- 0 1 3 4 5 6 7
- 0 1 3 4 6 7

Has Pointers?

- N
- Y
- Y
# Cache Line Formats

<table>
<thead>
<tr>
<th>Header Size?</th>
<th>6 bits</th>
<th>12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 010 11</td>
<td>10 010 11 101 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has Pointers?</th>
<th>8-byte chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Y</td>
<td>0 1 3 4 5 6 7</td>
</tr>
<tr>
<td>Y</td>
<td>0 1 3 4 6 7</td>
</tr>
</tbody>
</table>
Cache Line Formats

Header Size?

6 bits

0 010 11

Has Pointers?
N

8-byte chunk

0 1 2 3 4 5 6 7

12 bits

10 010 11 101 11

Has Pointers?
Y

Has Pointers?
Y

Has Pointers?
Y
## Cache Line Formats

<table>
<thead>
<tr>
<th>Header Size?</th>
<th>8-byte chunk</th>
<th>Has Pointers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td><img src="image" alt="6-bit example" /></td>
<td><img src="image" alt="6-bit example" /></td>
</tr>
<tr>
<td>12 bits</td>
<td><img src="image" alt="12-bit example" /></td>
<td><img src="image" alt="12-bit example" /></td>
</tr>
<tr>
<td>18 bits</td>
<td><img src="image" alt="18-bit example" /></td>
<td><img src="image" alt="18-bit example" /></td>
</tr>
</tbody>
</table>
# Cache Line Formats

<table>
<thead>
<tr>
<th>Header Size?</th>
<th>Has Pointers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>Y</td>
</tr>
<tr>
<td>12 bits</td>
<td>Y</td>
</tr>
<tr>
<td>18 bits</td>
<td>Y</td>
</tr>
</tbody>
</table>

- **6 bits**: 0 010 11
- **12 bits**: 10 010 11 101 11
- **18 bits**: 11

- **8-byte chunk**: 0 1 2 3 4 5 6 7
  - **Has Pointers?:** N
  - **6 bits**: 0 1 3 4 5 6 7
  - **12 bits**: 0 1 3 4 5 6 7
  - **18 bits**: 0 1 3 4 5 6 7
Cache Line Formats

<table>
<thead>
<tr>
<th>Header Size?</th>
<th>8-byte chunk</th>
<th>Has Pointers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>0 010 11</td>
<td>N</td>
</tr>
<tr>
<td>12 bits</td>
<td>10 010 11 101 11</td>
<td>Y</td>
</tr>
<tr>
<td>18 bits</td>
<td>11</td>
<td>Y</td>
</tr>
</tbody>
</table>

- 8-byte chunk: 0 1 2 3 4 5 6 7
- Has Pointers? N Y Y Y
Cache Line Formats

Header Size?

- **6 bits**
  - 0 010 11
  - Has Pointers?
    - No

- **12 bits**
  - 10 010 11 101 11
  - Has Pointers?
    - Yes

- **18 bits**
  - 11 11 11 11 10
  - Has Pointers?
    - Yes
Cache Line Formats

Header Size?

6 bits: 0 010 11

8-byte chunk: 0 1 2 3 4 5 6 7

Has Pointers?

6 bits: N

12 bits: Y

18 bits: Y

Header Size?

6 bits: 6 bits

12 bits: 12 bits

18 bits: 18 bits

Has Pointers?

6 bits: Y

12 bits: Y

18 bits: Y
Cache Line Formats

<table>
<thead>
<tr>
<th>Header Size?</th>
<th>8-byte chunk</th>
<th>Has Pointers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>12 bits</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>18 bits</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Cache Line Formats

Header Size?

6 bits

0 010 11

Has Pointers?

N

12 bits

10 010 11 101 11

Has Pointers?

Y

18 bits

11 00 00 11 00 00 11 00 10

Has Pointers?

Y
Microarchitectural Overview
Microarchitectural Overview
Microarchitectural Overview

CPU → L1-D → L2 → DRAM

Bit-vector Format
0 1 2 3 4 5 ... 12 13 14 15

C C
Microarchitectural Overview

Bit-vector ↔ 1-bit Format

CPU → L1-D → L2 → DRAM

63
Microarchitectural Overview
Microarchitectural Overview
Microarchitectural Overview

CPU → L1-D → L2 → DRAM

Data

Protected Bits
Performance
ZeRØ Performance Overheads

Hardware Modifications
ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.
ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

Software Modifications
ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

Software Modifications
• Our special load/stores do not change the binary size.
ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

Software Modifications
- Our special load/stores do not change the binary size.
- The ClearMeta instructions are only called on memory deallocation.
Performance Results (x86_64)

Experimental Setup
We use emulate ZeRØ on x86_64 by modifying LLVM to emit new instructions.
  • ClearMeta is emulated using dummy stores.
Performance Results (x86_64)

- gMean
- Norm. Perf.
- 0%

ZeRØ
Performance Results (x86_64)
Performance Results (x86_64)
Performance Results (x86_64)

- PAC-FPtr
- PAC-RET
- PAC-Full
- ZeRØ

14% gMean

Norm. Perf.

Performance Results (x86_64)
Performance Results (x86_64)

PAC’s overheads are attributed to the extra QARMA invocations upon pointer loads/stores.
ZeRØ reduces the average runtime overheads of pointer integrity from 14% to 0%!
ZeRØ does not compromise on security

No Pointer Manipulation
Protects against all known pointer manipulation attacks (e.g. ROP, JOP/COP, COOP, DOP).
Handling Security Violations

**Advisory Exceptions**
- Skip faulty instructions.
- Do NOT crash the running process.
Handling Security Violations

Advisory Exceptions
• Skip faulty instructions.
• Do NOT crash the running process.

Permit List
• Initialized during program startup
Handling Security Violations

Advisory Exceptions
• Skip faulty instructions.
• Do NOT crash the running process.

Permit List
• Initialized during program startup
• Avoid false alarms for non-type aware functions (e.g., memcpy and memmove)
Limitations
Limitations

Third-Party Code
• Can be added to the permit-list during program initialization. OR
Limitations

Third-Party Code
• Can be added to the permit-list during program initialization. OR
• ClearMeta is called before passing pointers to external libraries.
Limitations

Third-Party Code
- Can be added to the permit-list during program initialization. OR
- ClearMeta is called before passing pointers to external libraries.

Non-pointer Data Corruption
Limitsations

Third-Party Code
• Can be added to the permit-list during program initialization. OR
• ClearMeta is called before passing pointers to external libraries.

Non-pointer Data Corruption
These attacks require a full memory safety solution.
No-FAT: Low Overhead Memory Safety Checks

Full Memory Safety
No-FAT is well suited for cloud/server deployments away from the end user.

Checkout our paper & talk!
https://isca21.arroyo.me
An efficient pointer integrity mechanism

ZeRØ

An ideal candidate for end-user deployment.

✓ Easy to Implement
✓ No Runtime Overheads
✓ Offers Robust Security
Backup Slides