μRAI: Securing Embedded Systems with Return Address Integrity

Naif Saleh Almakhdhub¹,⁴
Saurabh Bagchi¹

Abraham A. Clements³
Mathias Payer²

¹Purdue University
²EPFL
³Sandia National Laboratories
⁴King Saud University
Current State of Security

The Rise of ICS Malware: How Industrial Security Threats Are Becoming More Surgical

By Barak Perelman on February 21, 2018

Target: Embedded and IoT devices Running Microcontroller Systems (MCUS)

Attack: Control-flow Hijacking

MCUS Challenges

<table>
<thead>
<tr>
<th>Desktop</th>
<th>MCUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Large virtual memory (GBs)</td>
<td>✗ Small physical memory (MBs Flash, KBs RAM)</td>
</tr>
<tr>
<td>✓ Basic defenses (e.g., ASLR)</td>
<td>✗ Basic defenses (e.g., ASLR)</td>
</tr>
<tr>
<td>Larger code</td>
<td>✓ Smaller code</td>
</tr>
<tr>
<td>✓ DEP</td>
<td>✗ DEP (Disabled → Fixable)</td>
</tr>
</tbody>
</table>

Memory

- Stack
- Heap
- Data
- Code

Flash (code)

- Code

RAM (Data)

- Stack
- Heap
- Data

0x08000000

0x08000000

0x08055555

0x08022222

0x08011111

0x08000000

0x08999555

0x08777222

0x08555111

0x08111000

0x02000000

0x02050000
MCUS Defenses for Return Addresses (Conceptual)

- **Usage**: CFI
- **Location**: Safe Stack + Software Fault Isolation
- **Location**: Shadow Stack + MPU
- **Location**: Shadow Stack + TEE
- **Location**: Randomized Safe Stack
- **Location**: μRAI

**Limited Security Guarantees**

- **Return Address Integrity + Low runtime overhead + No special hardware**

**Security**

- **High Overhead**
- **Special hardware required**
- **Without extra hardware**
MCUS Defenses for Return Addresses (Related Work)

- **Return Address Integrity** + Low runtime overhead + No special hardware

- **Usage**
  - C-FLAT [CCS16]
  - Minion [NDSS18]
  - LiteHAX [ICCAD18]
  - ACES [SEC18]
  - SCFP [EuroS&P18]

- **Location**
  - LR² [NDSS16]
  - Symbiote [RAID11]
  - EPOXY (SafeStack) [S&P17]
  - μArmor [EuroS&P19]
  - uXOM [SEC19]

- **Overhead**
  - High
  - Special hardware required
  - Without extra hardware

- **Limited Security Guarantees**

- **Security**
  - CFI CaRE (Shadow stack) [RAID17]
  - μRAI

- **Integration**
  - μRAI

- **Integrity**
  - LR²

- **Runtime Overhead**
  - 10%
Return Address Integrity (RAI)

- Every attack requires corrupting a return address by overwriting it.

**RAI Property:**

1. Ensure the return address is never writable except by an authorized instruction.
2. Return addresses are never pushed to the stack or any writable memory by an adversary.
Threat Model & μRAI Protection

Normal application

Unprivileged

- Func1
- Func2
- Func3

Privileged

- Func4
- Func5
- Func6
- Func7

μRAI

- Func1
- Func2
- Func3
- Func4
- Func5
- Func6
- Func7

- MPU, VTOR

- MPU, VTOR

• Reads from memory
• Writes to memory
• Knows the code layout
• Targets backward-edges

Corrupt return address

Corrupt return address or corrupt sensitive Memory Mapped IO (MMIO)

: Normal function
: Callable within exception handler
: MMIO
: State register encoding
: Software-Fault Isolation (SFI)
μRAI: Overview

1. Enforces the RAI property

2. Protects exception handlers and privileged execution

3. Low runtime overhead

State Register

Jump Table
- Jump return_location1
- Jump return_location2
- ...

Exception handler software-fault isolation

Relative jump target lookup routine
μRAI: Overview

1. Enforces the RAI property

2. Protects exception handlers and privileged execution

3. Low runtime overhead

Jump Table:
- Jump return_location1
- Jump return_location2
- ...

Exception handler software-fault isolation

Relative jump target lookup routine
μRAI and the State Register

• State Register (SR):
  • Can be any general-purpose register \(\rightarrow\) exclusively used by μRAI
  • Never spilled \(\rightarrow\) cannot be overwritten through a memory corruption
  • Does not contain a return address \(\rightarrow\) encoded values to resolve the return location

• Example call graph:
  • Each edge \(\rightarrow\) call

• How encode SR?
  • An XOR chain
μRAI: Terminology

- Function Keys (FKs): Hard-coded keys used to encode the SR

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func1&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Func1_1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Func1_2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func2&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

SR[Recursive] | SR[Encoded] |
0              | C           |
μRAI: Terminology

### SR[Recursive] vs SR[Encoded]
<table>
<thead>
<tr>
<th>Address</th>
<th>SR[Enc] = SR[Enc] ⊕ key1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func1_1</td>
<td>Call Func2</td>
</tr>
<tr>
<td>Func1_2</td>
<td>SR[Enc] = SR[Enc] ⊕ key2</td>
</tr>
<tr>
<td></td>
<td>Call Func2</td>
</tr>
</tbody>
</table>

**Function IDs (FIDs):** Possible values of the SR for the function

- **C**
  - Return Target: Jump return_location1
  - ELSE: Jump ERROR

- **C ⊕ key1**
  - Return Target: Jump Func1_1
- **C ⊕ key2**
  - Return Target: Jump Func1_2
- **ELSE**
  - Return Target: Jump ERROR
μRAI: Terminology

<table>
<thead>
<tr>
<th>Function ID (FID)</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Jump return_location1</td>
</tr>
<tr>
<td>ELSE</td>
<td>Jump ERROR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function ID (FID)</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>C ⊕ key1</td>
<td>Jump Func1_1</td>
</tr>
<tr>
<td>C ⊕ key2</td>
<td>Jump Func1_2</td>
</tr>
<tr>
<td>ELSE</td>
<td>Jump ERROR</td>
</tr>
</tbody>
</table>

- Function Lookup Table (FLT): List of FIDs for the function
μRAI: Transformation

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func1&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>SR[Enc] = SR[Enc] ⊕ key1</td>
</tr>
<tr>
<td></td>
<td>Call Func2</td>
</tr>
<tr>
<td>Func1_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>SR[Enc] = SR[Enc] ⊕ key1</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>SR[Enc] = SR[Enc] ⊕ key2</td>
</tr>
<tr>
<td></td>
<td>Call Func2</td>
</tr>
<tr>
<td>Func1_2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func2&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function ID (FID)</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Jump return_location1</td>
</tr>
<tr>
<td>ELSE</td>
<td>Jump ERROR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function ID (FID)</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>C ⊕ key1</td>
<td>Jump Func1_1</td>
</tr>
<tr>
<td>C ⊕ key2</td>
<td>Jump Func1_2</td>
</tr>
<tr>
<td>ELSE</td>
<td>Jump ERROR</td>
</tr>
</tbody>
</table>

- Encode the SR and call Func2
μRAI: Transformation

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func1&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>SR[Enc] = SR[Enc] ⊕ key1</td>
</tr>
<tr>
<td>...</td>
<td>Call Func2</td>
</tr>
<tr>
<td>Func1_1</td>
<td>SR[Enc] = SR[Enc] ⊕ key1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Func1_2</td>
<td>SR[Enc] = SR[Enc] ⊕ key2</td>
</tr>
<tr>
<td>...</td>
<td>Call Func2</td>
</tr>
</tbody>
</table>

Function ID (FID) | Return Target
--- | ---
C | Jump return_location1
ELSE | Jump ERROR

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func2&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>[SR]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>[SR]</td>
</tr>
<tr>
<td>...</td>
<td>Call Func2</td>
</tr>
</tbody>
</table>
| Function ID (FID) | Return Target
--- | ---
C ⊕ key1 | Jump Func1_1
C ⊕ key2 | Jump Func1_2
ELSE | Jump ERROR

- Func2 reads the SR and executes the corresponding direct jump.
μRAI: Transformation

- Func2 returns correctly and the SR is decoded
μRAI: Transformation

The previous SR value is restored
μRAI: Transformation

<table>
<thead>
<tr>
<th>Function ID (FID)</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Jump return_location1</td>
</tr>
<tr>
<td>ELSE</td>
<td>Jump ERROR</td>
</tr>
</tbody>
</table>

- The same happens for other calls. Func1 can then return correctly
μRAI: Overview

1. Enforces the RAI property

State Register

Jump Table
- Jump return_location1
- Jump return_location2
- ...

2. Protects exception handlers and privileged execution

Exception handler software-fault isolation

3. Low runtime overhead

Relative jump target lookup routine
μRAI: Enforce RAI for exception handlers

• Exception handlers execute with privileges
  • Can disable the MPU → enable code injection
  • Can corrupt exception stack frame → break RAI property

• Solution:
  • Apply SFI only to functions callable by exception handlers
  • Limit SFI overhead compared to full-SFI
μRAI: Overview

1. Enforces the RAI property

2. Protects exception handlers and privileged execution

3. Low runtime overhead

State Register

Jump Table
- Jump return_location1
- Jump return_location2
- ...

Exception handler software-fault isolation

Relative jump target lookup routine
**Target Lookup Routine (TLR)**

- How can we find the correct direct jump in the FLT efficiently?
  - Use a relative jump before the FLT
  - Resolve the correct return location efficiently regardless of FLT size

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;Func1&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>Jump PC+SR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FID</th>
<th>Return Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jump location1</td>
</tr>
<tr>
<td>2</td>
<td>Jump location2</td>
</tr>
<tr>
<td>3</td>
<td>Jump ERROR</td>
</tr>
<tr>
<td>4</td>
<td>Jump location4</td>
</tr>
<tr>
<td>5</td>
<td>Jump location5</td>
</tr>
</tbody>
</table>

Assume the correct return location is location4

Comparing all FID can be slow!

Use SR as an index of a jump table

Align the FLT → make SR = 4

Align the FLT → make SR = 4
μRAI: Overview

1. Enforces the RAI property

2. Protects exception handlers and privileged execution

3. Low runtime overhead

State Register

Jump Table
- Jump return_location1
- Jump return_location2
- ...

Exception handler software-fault isolation

Relative jump target lookup routine
Evaluation

• Five MCUS applications on Cortex-M4:
  • PinLock
  • FatFs_uSD
  • FatFs_RAM
  • LCD_uSD
  • Animation

• CoreMark benchmark[1]
  • Standard MCUS performance benchmark

Security Evaluation Using PinLock: Unlock The Lock

- **Buffer overflow**: No return address to overflow...
- **Arbitrary write**: No return address to write to...
- **Stack pivot**: No return address to pop from the stack

✓ μRAI prevents all control-flow hijacking attack scenarios targeting return addresses

![Diagram with states and register](image)
Performance results

- Requiring full-SFI results in high overhead → average of 130.5%
- μRAI results in low overhead → average of 0.1%
µRAI: Conclusion

• Control-flow hijacking on MCUS is a threat

• µRAI secures MCUS against control-flow hijacking
  • Enforces the RAI property for MCUS → protects backward edges
  • Complemented with type-based CFI → end-to-end code pointer protection

• Presents a portable encoding scheme
  • Does not require special hardware features (only a register and an MPU)
  • Applicable to other systems

• Low runtime overhead

https://github.com/embedded-sec/uRAI
Challenge: Path Explosion

• Func3 has 2 call sites
• FLT is size is 4!
Path Explosion Solution: Segmentation

- State register segmentation

<table>
<thead>
<tr>
<th>Recursion counter (Higher N bits)</th>
<th>Encoded value (Lower 32-N bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment 1</td>
<td>Segment M</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Segment K</td>
<td>Segment 2</td>
</tr>
<tr>
<td>Segment M</td>
<td>Segment 1</td>
</tr>
</tbody>
</table>

- Functions only use the bits in their assigned segment.
Path Explosion Solution: Segmentation

Segmentation reduces FLT from 4 to 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
<td>[ C ⊕ FK1, C ⊕ FK2 ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
<td>[ C ⊕ FK5, C ⊕ FK6, ]</td>
</tr>
</tbody>
</table>

Not used
μRAI: Scalability

• What if no more values can be found for the SR?
• Solution:
  • Partition the call graph if no solution is found
  • Entering a new partition → Save and reset the SR to a privileged safe region
  • Returning to a previous partition → Restore the SR
### Store Instructions Protected with EH-SFI

<table>
<thead>
<tr>
<th>App</th>
<th>Static</th>
<th>Total</th>
<th>(Static/Total)%</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>PinLock</td>
<td>56</td>
<td>516</td>
<td>10.9</td>
<td>7</td>
</tr>
<tr>
<td>FatFs_uSD</td>
<td>99</td>
<td>1,802</td>
<td>5.5</td>
<td>906K</td>
</tr>
<tr>
<td>FatFs_RAM</td>
<td>7</td>
<td>1,116</td>
<td>0.6</td>
<td>7</td>
</tr>
<tr>
<td>LCD_uSD</td>
<td>99</td>
<td>2,814</td>
<td>3.5</td>
<td>48K</td>
</tr>
<tr>
<td>Animation</td>
<td>99</td>
<td>2,760</td>
<td>3.6</td>
<td>66K</td>
</tr>
<tr>
<td>CoreMark</td>
<td>56</td>
<td>1,024</td>
<td>5.5</td>
<td>7</td>
</tr>
</tbody>
</table>
SR Layout: Recursion

• The SR has two parts:
  • ENC: Encoded value
  • REC: Recursion counter

• Cannot use XOR with recursion
• Collision occurs with existing values Func1

\[ SR \oplus ANY \text{ KEY} \oplus ANY \text{ KEY} = SR \]

\[
\begin{array}{c|c}
SR[Rec] & SR[Enc] \\
0 & C \\
\end{array}
\]

\[
\begin{array}{c|c}
SR[Rec] & SR[Enc] \\
0 & C \oplus key1 \\
\end{array}
\]

SR[Rec] + 1 before the call
SR[Rec] − 1 after return
If SR[Rec] > 0 \(\rightarrow\) recursion
μRAI: Handling recursion

If recursive → use a counter (*recursion is discouraged in MCUS*)