SHERLOC: Secure and Holistic Control-Flow Violation Detection on Embedded Systems

Xi Tan and Ziming Zhao
CyberspACe securiTy and forensIcs lab (CactiLab)
University at Buffalo
It is estimated that the world has over 250 billion microcontrollers [1].

More than 4.4 billion Cortex-M MCUs were shipped in the 4th quarter of 2020 alone [2].

Microcontroller-based System Characteristics

Hardware (Cortex-M as an example)

RISC architecture. Sixteen 32-bit general-purpose registers.

No MMU, but a 32-bit physical memory space.

Unprivileged memory access instructions, pointer authentication code, streamlined TrustZone.

Software

Developed in memory-unsafe languages, e.g., C

Most systems do not adopt privilege separation.

Functionality implemented in Interrupt Service Routines (ISR)
Control-Flow Integrity (CFI)
Inlined CFI Enforcement

Instrument at source code or binary level

Example CFI instrumentations of an x86 computed jump instruction [1]

```
FF E1 jmp ecx ; a computed jump instruction can be instrumented as (a):
81 39 78 56 34 12 cmp [ecx], 12345678h ; compare data at destination
75 13 jne error_label ; if not ID value, then fail
8D 49 04 lea ecx, [ecx+4] ; skip ID data at destination
FF E1 jmp ecx ; jump to destination code
```

Inlined CFI Enforcement for Microcontroller Systems

- Memory constraints
  - Change the memory layout of the code
  - Increase the code size
- Coarse-grained forward-edge protection: label-based
- Shadow stacks need to be protected
- Existing approaches, e.g., CFICare [1], TZmCFI [2], utilize TrustZone to secure shadow stack but introduce a high run-time overhead,

Control-Flow Violation Detection (CFVD)

- Do not instrument code but verify instruction trace generated by a hardware tracer [1, 2]
- Require kernel modification; kernel is in the TCB
- Only work on unprivileged application but not kernel

**Application-oriented CFVD.** Given the trace $R_A = (r_0, r_1, \ldots, r_n)$ of an application $A$, ACFVD verifies that $r_i \in E_A$, $\forall i \in \{0, 1, \ldots, n\}$.

System-oriented CFVD

- Most MCU functionalities are implemented in Interrupt Service Routines (ISR).
- Scheduling- and interrupt-aware

**System-oriented CFVD (SCFVD).** Given the trace $R_S = (r_0, r_1, \ldots, r_n)$ of a system $S$ including a kernel $K$ and tasks $T$, SCFVD verifies that $r_i \in E_S \lor r_i.d \in I_K \cup Y_T$, $\forall i \in \{0, 1, \ldots, n\}$.
Challenges for SCFVD

Interrupts that cannot be predetermined, E.g., \(<c7, t1>\)

Figure 1: Example legitimate control-flow transfers of a system with an RTOS kernel, two privileged tasks, and one unprivileged task in a single physical address space.
Challenges for SCFVD

The locations the scheduler yield control to cannot be predetermined, E.g., \((s_2, b_5)\) and \((s_2, c_7)\)

Figure 1: Example legitimate control-flow transfers of a system with an RTOS kernel, two privileged tasks, and one unprivileged task in a single physical address space.
Challenges for SCFVD

The trace, tracing operation, and CF verification must be secured from the privileged but potentially compromised kernel.

Figure 1: Example legitimate control-flow transfers of a system with an RTOS kernel, two privileged tasks, and one unprivileged task in a single physical address space.
Figure 2: SHERLOC comprises offline analysis and runtime configuration and enforcement modules. The unmodified protected system program runs in the non-secure state, whereas SHERLOC runtime modules execute in the secure state.
Sherloc Timeline

- Protected system
- Exec.
- MTB Tracing
- Suspend MTB
- Resume MTB
- SCFVD
- Yield control
- Watermark hit
- Secure state
- Non-secure state
- Runtime Config.
- Runtime Enfor.
- SHERLOC
- CFI violation detected
- Reset Handler
- Reset
- Config. SAU
- Config. DebugMon
- Config. MTB
- Yield control
- Watermark hit
- DebugMon Handler
- Resume MTB
- Suspend MTB
# Sherloc Holistic Enforcement Policy

<table>
<thead>
<tr>
<th>Case</th>
<th>Type</th>
<th>Instruction(s)</th>
<th>Ins. Size</th>
<th>How to Identify the Type?</th>
<th>Sherloc Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct branch (§4.4.1)</td>
<td>Direct branch (§4.4.1)</td>
<td>B{cond} #imm</td>
<td>2/4</td>
<td>The dereferenced instruction</td>
<td>Skip the record</td>
</tr>
<tr>
<td>Direct call (§4.4.1)</td>
<td>Direct call (§4.4.1)</td>
<td>BL{cond} #imm</td>
<td>4</td>
<td>The dereferenced instruction</td>
<td>RCS.push(s + 4)</td>
</tr>
<tr>
<td>Indirect branch (§4.4.1)</td>
<td>Indirect branch (§4.4.1)</td>
<td>BX{cond} Rx</td>
<td>2</td>
<td>The dereferenced instruction</td>
<td>if ( \langle s, d \rangle \notin \text{IBT} ), reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TBB/TBH {PC, \ldots}</td>
<td>4</td>
<td>The dereferenced instruction</td>
<td>if ( \langle s, d \rangle \notin \text{IBT} ), reset; else RCS.push(s + 2)</td>
</tr>
<tr>
<td>Indirect call (§4.4.1)</td>
<td>Indirect call (§4.4.1)</td>
<td>BLX Rx</td>
<td>2</td>
<td>The dereferenced instruction</td>
<td>if ( \langle s, d \rangle \notin \text{IBT} ), reset; else RCS.push(s + 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function return (§4.4.2)</td>
<td>Function return (§4.4.2)</td>
<td>BX LR</td>
<td>2/4</td>
<td>The dereferenced instruction</td>
<td>if ( d \neq \text{RCS.pop()} ), reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDM SP!, {\ldots, PC}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sync. exception (§4.4.3)</td>
<td>Sync. exception (§4.4.3)</td>
<td>SVC #imm</td>
<td>2</td>
<td>s[A-bit]</td>
<td>if ( d \notin \text{VT} ), reset; else if ( d \neq \text{PendSV} ), RCS.push(s)</td>
</tr>
<tr>
<td>Non-PendSV async. interrupt (§4.4.3)</td>
<td>Non-PendSV async. interrupt (§4.4.3)</td>
<td>N/A</td>
<td>N/A</td>
<td>s[A-bit]</td>
<td>if ( d \notin \text{VT} ), reset; else if ( d \neq \text{PendSV} ), RCS.push(s)</td>
</tr>
<tr>
<td>Non-PendSV ISR return (§4.4.4)</td>
<td>Non-PendSV ISR return (§4.4.4)</td>
<td>BX LR</td>
<td>2/4</td>
<td>The dereferenced instruction and</td>
<td>if bare-metal and ( d_2 \neq \text{RCS.top()} ), reset; else if bare-metal and ( d_2 \neq \text{RCS.top()} ), RCS.pop(); else go to PendSV ISR return handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POP {\ldots, PC}</td>
<td></td>
<td>( d_1 == \text{EXC_RETURN} \land d_2 == \text{EXC_RETURN} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDM SP!, {\ldots, PC}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PendSV async. interrupt (§4.4.5)</td>
<td>PendSV async. interrupt (§4.4.5)</td>
<td>N/A</td>
<td>N/A</td>
<td>s[A-bit]</td>
<td>if ( d == \text{PendSV} ), ( Y^{T}.add(s) ) and ( Y^{T}.add(\text{RCS.pop()}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PendSV ISR return (§4.4.6)</td>
<td>PendSV ISR return (§4.4.6)</td>
<td>BX LR</td>
<td>2/4</td>
<td>The dereferenced instruction and</td>
<td>if ( d_2 \notin Y^T ), reset; else if ( d_2 \in \text{a shared library} ), and assuming the next trace record is ( \langle s_n, d_n \rangle ), and ( d_n \notin Y^T ), reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POP {\ldots, PC}</td>
<td></td>
<td>( d_1 == \text{EXC_RETURN} \land d_2 == \text{EXC_RETURN} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDM SP!, {\ldots, PC}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Security Analysis: Latency Estimation

- Using the LED as a covert channel to output one byte
- Reusing printf() to output one byte to UART
- Bare-metal Systems (O3)
- Bare-metal Systems (Oz)
- FreeRTOS (O3)
- FreeRTOS (Oz)
Experiment

- Prototype: ARM Versatile Express Cortex-M Prototyping System MPS2+ configured as a Cortex-M33 CPU
- Benchmark:
  - BEEBS
  - Blinky
  - FreeRTOS
- Optimization level: -O3 and -Oz
Performance Evaluation - BEEBS

(a) Performance overhead (%) of BEEBS programs compiled with O3 optimization.
Performance Evaluation - BEEBS
Thank you!

Open-sourced at:
https://github.com/CactiLab/Sherloc-Cortex-M-CFVD
Control-Flow Integrity (CFI)

Control-flow integrity (CFI) is a security property that can prevent control-flow hijacking by dictating that indirect control-flow transfers, including forward edges (indirect call and branch) and backward edges (return), must follow a predetermined control-flow graph (CFG).