Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks

Hans Winderix, Jan Tobias Mühlberg, Frank Piessens
Nemesis, an Interrupt Latency Side-Channel Attack

- Microarchitectural side-channel attack
  - Exploits fetch-decode-execute logic
  - Attacker measures interrupt latency
  - Reveals the latency trace of an execution, i.e.
    - Machine instruction count
    - Timing of individual machine instructions
  - Reveals which side of branch has been executed
  - Leaks info about branch predicate values
Nemesis - Illustrative Example

```
CMP R12, R13
JEQ TRUE
FALSE:
    ADD R12, R12 /* 1 cycle */
    JMP EXIT /* 2 cycles */
TRUE:
    MOV $20, R12 /* 2 cycles */
    ADD R13, R13 /* 1 cycle */
EXIT:
    ...
```

Latency Trace of False Path

Latency Trace of True Path
Constant-Time Programming Policy

- An established security policy
  - To protect against timing side-channel attacks
  - **No secret-dependent control-flow**
  - **No secret-dependent memory accesses**
  - **No secret-dependent instruction latencies**

No secret-dependent control-flow

↓

No secret-dependent branch instructions

↓

Effective protection against Nemesis
Constant-Time Programming Policy (Concerns)

- Strict rules with a status of absoluteness
- Typically manually implemented at the highest abstraction level (source code)
  - Harms readability and maintainability
    - Prevents using familiar programming constructs
    - Developer must use obscure tricks to deceive compiler (brittle)
  - No separation of concerns (harms portability)
    - Tight coupling between security policy and source code
  - Policy must be honored by the compiler early on (brittle)
    - Optimiser cannot introduce secret-dependent constructs
    - Compiler cannot introduce secret-dependent constructs when lowering abstractions
- Performance impact
/* Hardened program (balanced branch) */
CMP R12, R13
JEQ TRUE
FALSE:
        ADD R12, R12 /* 1 cycle */
        JMP EXIT /* 2 cycles */
TRUE:
        MOV $20, R12 /* 2 cycles */
        ADD R13, R13 /* 1 cycle */

/* Hardened program (eliminated branch) */
CMP R12, R13 /* 1 cycle, 1 byte */
JEQ TRUE /* 2 cycles, 1 byte */
FALSE:
        ADD R12, R12 /* 1 cycle, 1 byte */
        JMP EXIT /* 2 cycles */
TRUE:
        MOV $20, R12 /* 2 cycles */
        ADD R13, R13 /* 1 cycle */

<table>
<thead>
<tr>
<th>Time (cycles)</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced</td>
<td>6</td>
</tr>
<tr>
<td>Eliminated</td>
<td>22</td>
</tr>
</tbody>
</table>
Research Hypothesis

- The constant-time programming policy is not absolute
- Relaxing the constant-time rules can be secure (*depends on leakage model*)
- Relaxing the constant-time rules can produce more performant programs
- Balancing branches is an effective countermeasure against timing attacks on some low-end processors

Objectives

- Decouple security policy from source code
- Automate program hardening
- Make latency trace secret-independent
- Balance secret-dependent branches (*instead of eliminating them*)
- Less overhead (*compared to eliminating branches*)
Assumptions

- **Attacker model**
  - Access to cycle-accurate clock
  - Ability to precisely schedule and handle interrupts
  - Attacker can interrupt victim code running in another protection domain

- **System model**
  - Interrupts are handled upon instruction retirement
  - Execution environment leaks latency trace of execution
  - A *dummy instruction* can be constructed for every latency class

**Dummy instruction**

An instruction without observable effects besides its time to execute
The Defense

A Recursive Control-Flow Graph Algorithm

- **Phase 1 - Static analysis**
  - Taint analysis, loop analysis

- **Phase 2 - Program hardening**
  - Balance secret-dependent branches according to their latency trace
    - Insert dummy instructions if latencies don't match
  - Three operations
    1. Equalise path lengths
    2. Compute level structure
    3. Equalise execution times (level-wise)
Implementation

- LLVM compiler infrastructure
  - MachineFunction pass
  - MSP430 backend

Evaluation

- Platform = openMSP430 + Sancus TEE extensions
- Benchmark suite, consisting of
  - Synthetic programs
  - Third-party programs

https://github.com/hanswinderix/sllvm
### Experimental Results (openMSP430)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Vulnerable Baseline</th>
<th>Balancing Overhead</th>
<th>Elimination Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code size (bytes)</td>
<td>Exec time (cycles)</td>
<td>Code size</td>
</tr>
<tr>
<td>call</td>
<td>302</td>
<td>112</td>
<td>1.09x</td>
</tr>
<tr>
<td>diamond</td>
<td>284</td>
<td>103</td>
<td>1.16x</td>
</tr>
<tr>
<td>fork</td>
<td>264</td>
<td>91</td>
<td>1.06x</td>
</tr>
<tr>
<td>ifcompound</td>
<td>384</td>
<td>372</td>
<td>1.06x</td>
</tr>
<tr>
<td>ifthenloop</td>
<td>284</td>
<td>143</td>
<td>1.27x</td>
</tr>
<tr>
<td>ifthenloopif</td>
<td>342</td>
<td>179</td>
<td>1.38x</td>
</tr>
<tr>
<td>ifthenlooploop</td>
<td>308</td>
<td>378</td>
<td>1.54x</td>
</tr>
<tr>
<td>ifthenlooplooptail</td>
<td>350</td>
<td>387</td>
<td>1.63x</td>
</tr>
<tr>
<td>indirect</td>
<td>274</td>
<td>97</td>
<td>1.18x</td>
</tr>
<tr>
<td>loop</td>
<td>400</td>
<td>2841</td>
<td>1.06x</td>
</tr>
<tr>
<td>multifork</td>
<td>290</td>
<td>100</td>
<td>1.19x</td>
</tr>
<tr>
<td>triangle</td>
<td>266</td>
<td>94</td>
<td>1.09x</td>
</tr>
<tr>
<td>bsl</td>
<td>394</td>
<td>984</td>
<td>1.12x</td>
</tr>
<tr>
<td>keypad</td>
<td>672</td>
<td>1119</td>
<td>1.28x</td>
</tr>
<tr>
<td>kruskal</td>
<td>634</td>
<td>2460</td>
<td>1.14x</td>
</tr>
<tr>
<td>modexp2</td>
<td>702</td>
<td>23537</td>
<td>1.05x</td>
</tr>
<tr>
<td>multi3</td>
<td>416</td>
<td>904</td>
<td>1.37x</td>
</tr>
<tr>
<td>mulfmod8</td>
<td>482</td>
<td>425</td>
<td>1.49x</td>
</tr>
<tr>
<td>sharevalue</td>
<td>480</td>
<td>3398</td>
<td>1.06x</td>
</tr>
<tr>
<td>switch16</td>
<td>402</td>
<td>115</td>
<td>1.41x</td>
</tr>
<tr>
<td>switch8</td>
<td>402</td>
<td>115</td>
<td>1.41x</td>
</tr>
<tr>
<td>twofish</td>
<td>8872</td>
<td>92745</td>
<td>1.06x</td>
</tr>
<tr>
<td><strong>Geometric Mean</strong></td>
<td><strong>1.23x</strong></td>
<td><strong>1.19x</strong></td>
<td><strong>1.35x</strong></td>
</tr>
</tbody>
</table>