SpecROP: Speculative Execution of ROP chains

Atri Bhattacharyya*, Andrés Sánchez*, Esmaeil M. Koruyeh†, Nael Abu-Ghazaleh†, Chengyu Song†, Mathias Payer*
Speculative Execution Attacks (SEA)

SpecROP is based on SEA

SEA execute *gadgets* speculatively
- Gadgets must already exist in the target
- Existing gadgets are *monolithic*

Monolithic gadget
- accesses secrets, and
- leaks secrets (side-channel dependent)

The “power” of SEAs depend on the existing gadgets
SEA requires powerful monolithic gadgets

Requirements:
1. Accesses the right secret (requires pointer to the secret), or
2. Leaks the right secret (requires the secret in the correct register)
3. Must exist (long, unusual sequences might not exist).

Reality:
• SMoTherSpectre cannot leak AES key from OpenSSL
  • No pointer to key (1)
  • Key not in a register (2)
• No monolithic Spectre gadget in real programs (3)
Divide and conquer

SpecROP is inspired by ROP attacks

Return Oriented Programming (ROP) attacks

- Non-speculative, code reuse attacks
- Sequence of small gadgets (more likely to exist)
- End in ret (can be chained to next gadget)

SpecROP principles

- Use simpler gadgets ending in jmp/ret
- Train branch predictor to chain gadgets
- Use intermediate gadgets to modify state
  (increment pointers, left/right shift registers, move data between registers)

SpecROP enables SEA and makes it more powerful
SpecROP principle

Starts at a indirect jump
- Holds secrets/pointers to secrets in registers

Prepares secret for leakage
Arithmetic gadgets
Shift gadgets
Data movement gadgets

Secret-dependent microarchitectural change

SpecROP links small, simple gadgets into a powerful gadget chain
Spectre v1 with chains

The Spectre v1 gadget reads a secret, then makes a secret-dependent load

\[ C: \quad y = \text{array1}[x] \]
\[ z = \text{array2}[y \times 4096] \]

Monolithic gadget not found in the wild

```
mov (rax, rdi, 8), rax
shl 0xc, rax
mov (rdx, rax, 8), rax
```

Assumptions:
- rax = array1
- rdx = array2
- rdi = x

```
tmp1 = rax
shl 0x20, tmp1
mov (rdx, tmp1, 8), rsi
```

Assumptions:
- rdx = array1
- rbx = array2
- rax = x

SpecROP chain for Spectre v1 exists!
OpenSSL key leakage

(De)Encryption calls do_cipher(ctx, ...) using indirect call

call *0x20(rax)
  rdi = &ctx

mov 0x78(rdi), rdi
mov (rdi), rax
test 0x1, rax
jmp ...
crc32
popcnt

Required gadget

lea 0x20(rdi), rdi
  rdi = &ctx + 0x20

mov 0x58(rdi), rax
  rax = cipher_data
  rax = &AES key
testb 0x1, (rax)
jmp ...

SpecROP gadget chain

SpecROP chains are expressive
Evaluation (1/3): Attacker models

Tried 3 attacker models:
- Cross process between SMT threads
- Cross thread between SMT threads
- Single thread, aliased instructions

4 “generations” of Intel processors

<table>
<thead>
<tr>
<th></th>
<th>i7-6700K</th>
<th>i7-8700</th>
<th>i7-9700</th>
<th>i7-10510U</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross process</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Cross thread</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Aliased</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Aliased attacks are a practical threat
Evaluation (2/3): Length of gadget chain

Setup:

• Attacker trains BTB
  • Goes through gadgets J₀ to J₁₅
  • Each gadget ends with indirect jump

• Victim
  • (architecturally) jumps to J₁₅
  • (speculatively) uses predictions from BTB, executes some gadgets

• We track executed gadgets, each loading a unique address
Evaluation (2/3): Length of gadget chain

Testbed:
- Tested on i7-6700K and i7-8700
- With and without microcode updates

Results:
- Up-to 4 gadgets can reasonably be chained
- Microcode does not affect success rate

Practical attacks may use up-to 4 gadgets
Evaluation (3/3): Characterization of gadgets

We created a gadget-search tool: SpecFication

SpecFication phases:

- Disassembly: Get a list of potential processing gadgets
- Characterisation: Express gadgets semantics
- Solving: Express wanted gadget as constraints, check constraints

<table>
<thead>
<tr>
<th>Library</th>
<th>Binary size</th>
<th>Gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcrypto</td>
<td>3.3M</td>
<td>13k</td>
</tr>
<tr>
<td>libc</td>
<td>1.8M</td>
<td>15k</td>
</tr>
<tr>
<td>libdl</td>
<td>15K</td>
<td>266</td>
</tr>
<tr>
<td>mod_ssl</td>
<td>235K</td>
<td>490</td>
</tr>
<tr>
<td>mod_proxy</td>
<td>131K</td>
<td>338</td>
</tr>
<tr>
<td>mod_http2</td>
<td>244K</td>
<td>1,113</td>
</tr>
</tbody>
</table>
Evaluation (3/3): Characterization of gadgets

Large skew in availability of arithmetic gadgets for different registers

<table>
<thead>
<tr>
<th>Library</th>
<th>rax</th>
<th>rbx</th>
<th>rcx</th>
<th>rdx</th>
<th>rdi</th>
<th>r11</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcrypto</td>
<td>665</td>
<td>259</td>
<td>34</td>
<td>78</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>libc</td>
<td>889</td>
<td>317</td>
<td>128</td>
<td>171</td>
<td>419</td>
<td>0</td>
</tr>
<tr>
<td>libdl</td>
<td>25</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mod_ssl</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mod_proxy</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>mod_http2</td>
<td>46</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Plentiful data-movement gadgets between pairs of <source, destination> registers (max 240)

<table>
<thead>
<tr>
<th>Library</th>
<th>Register-pairs</th>
<th>Chained</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcrypto</td>
<td>116</td>
<td>210</td>
</tr>
<tr>
<td>libc</td>
<td>101</td>
<td>204</td>
</tr>
</tbody>
</table>

Arithmetic + data movement gadgets allow expressive computation
Limitations

Lower signal-to-noise ratio
  • Leakage gadget reached less often

Processing gadgets:
  • Are limited by speculation window
  • Cannot reuse gadgets ending in indirect jump
  • Cannot write to the jump target

Advantage:
  • Can fault without ending speculation

For discussion of ret, see paper
Mitigations

Prevent branch misprediction:
- SW only: retpolines
- SW/HW: IBRS/IBPB
- HW only: Intel CET and other CFI (control-flow integrity) measures

Finding potential chains through static analysis:
- State explosion, potentially incomplete
- Side-channel specific

Practically:
- Find vulnerable branches (with sensitive information/pointers) statically
- Protect with retpolines

Comprehensive mitigation for SpecROP require speculative CFI in hardware
Conclusions

SpecROP breaks monolithic gadget into several, simple gadgets
  • Gadgets chained by training branch predictor
  • Enables certain attacks previously impossible (e.g. Spectre-v1)
  • Extend leakage of other attacks (e.g. SMoTherSpectre)

Practicality of SpecROP is limited
  • Branch poisoning much harder today
  • The attack surface still remains
  • Proper hardware CFI is needed

Gadget search using symbolic analysis of binaries is effective

Code available at https://github.com/HexHive/specrop
Questions in Q&A (or atri[dot]bhattacharyya[at]epfl[dot]ch)
Evaluation

Q1: Which attacker models allow SpecROP?
Q2: How many gadgets can I chain?
Q3: How many processing gadgets exist in real binaries?
Overview

• Introduction
• SpecROP attack principle
  • Spectre v1 case-study
  • OpenSSL case-study
• Evaluation
• Limitations
• Mitigations
• Conclusion