Eclipse

Preventing Speculative Memory-error Abuse with Artificial Data Dependencies

Neophytos Christou Alexander J. Gaidis Vaggelis Atlidakis Vasileios P. Kemerlis

October 17, 2024

Secure Systems Laboratory (SSL)
Department of Computer Science
Brown University

Speculative Memory-error Abuse

Speculative Memory-error Abuse (SMA) Attacks Overview

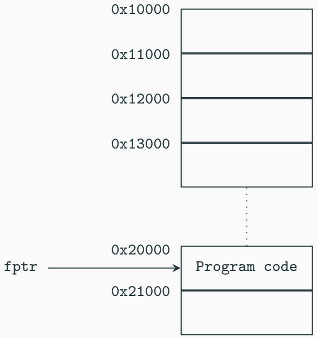
- Combine memory errors with speculative execution attacks
  - Leverage memory errors to architecturally corrupt memory
  - Cause the CPU to use the corrupted data during speculative execution
  - Bypass memory-safety-based mitigations while inhibiting detection (*e.g.*, avoid crashes)

BROWN

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps



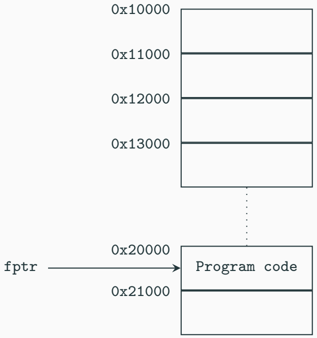
[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps

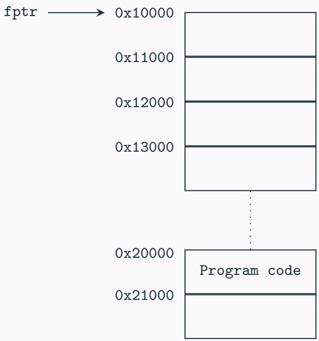
1. Train branch



[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



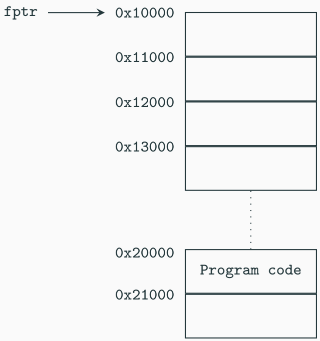
Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip condition

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



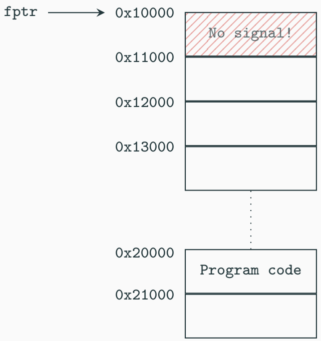
Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



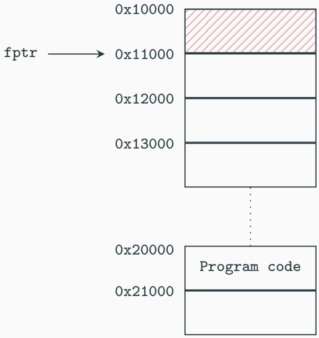
Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected

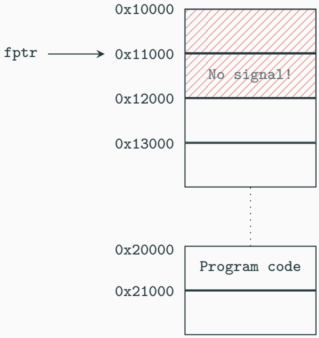
[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps

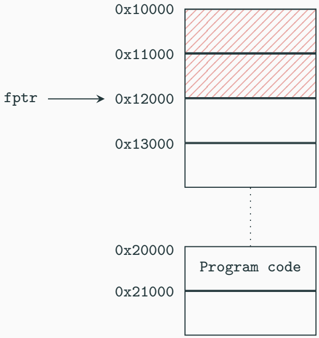
1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` $\rightarrow$ speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected



[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



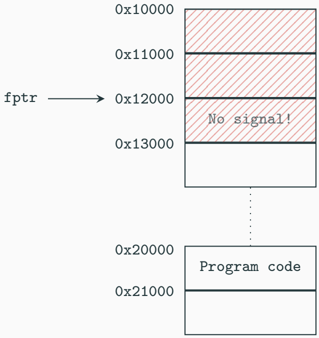
Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



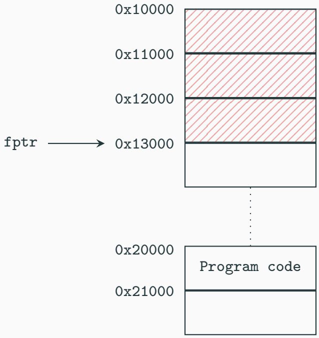
Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected

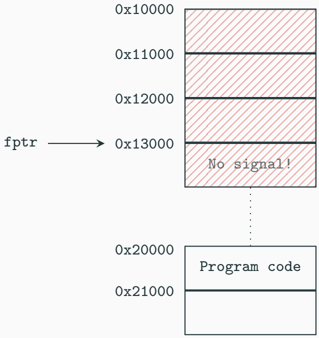
[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected



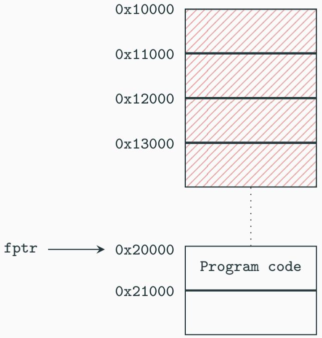
[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps

1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected



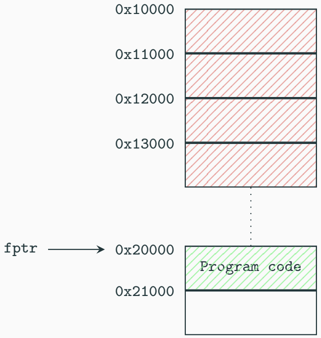
[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```

Attack Steps

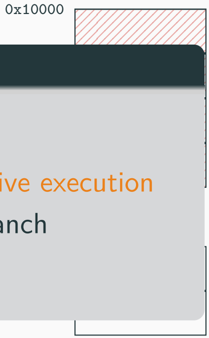
1. Train branch
2. Architecturally corrupt `fptr`; flip `condition`
   - Corrupted `fptr` → speculatively deref'd
3. Check for cache activity (side channel)
4. Repeat until cache activity is detected



[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

SMA Attack Example – Speculative Probing[1]

```
if (condition) {
    fptr();
}
```



Problem

Attacker-controlled data is used during speculative execution caused by a mispredicted conditional branch

1.
2.
3.
4. Repeat until cache activity is detected

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

Eclipse Overview

Insight

CPUs *cannot* execute instructions with unresolved data dependencies

- Even when *executing speculatively*

Eclipse Overview

Insight

CPUs *cannot* execute instructions with unresolved data dependencies

- Even when *executing speculatively*

Eclipse Approach

N

Eclipse Overview

Insight

CPUs *cannot* execute instructions with unresolved data dependencies

- Even when *executing speculatively*

Eclipse Approach

- Compiler-assisted mitigation

N

Eclipse Overview

Insight

CPUs *cannot* execute instructions with unresolved data dependencies

- Even when *executing speculatively*

Eclipse Approach

- Compiler-assisted mitigation
- Analyze program to identify *SMA-Capable* (SMAC) instructions
  - ➜ Instructions that can be leveraged to carry out a SMA attack
  - ➜ Can be *speculatively executed* as a result of a misprediction of a *preceding conditional branch*

N

Eclipse Overview

Insight

CPUs *cannot* execute instructions with unresolved data dependencies

- Even when *executing speculatively*

Eclipse Approach

- Compiler-assisted mitigation
- Analyze program to identify *SMA-Capable* (SMAC) instructions
  - → Instructions that can be leveraged to carry out a SMA attack
  - → Can be *speculatively executed* as a result of a misprediction of a *preceding conditional branch*
- Instrument code to introduce artificial data dependencies on the identified SMAC instructions
  - ✗ Prevent instructions from operating on attacker-controlled data during speculative execution

N

Eclipse Instrumentation

```
if (condition) {
    fptr();
}
...
```

```
cmpl    $0x0, %rax
je      no_call
callq   *%rcx
.no_call:
...
```

```
target:
...
```

```
target:
...
```

🟦: Non-speculative execution
🟪: Speculative execution

Register State

Eclipse Instrumentation

```
if (condition) {
  fptr();
}
...



target:
...
```

```
cmpl    $0x0, %rax      Modifies rflags
je      no_call
callq   *%rcx
.no_call:
...



target:
...
```

Register State

rax (condition): unknown

rflags: unknown

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation

```
if (condition) {
  fptr();
}
...



target:
...
```

```
cmpl    $0x0, %rax   Modifies rflags
je      no_call      Depends on rflags
callq   *%rcx
.no_call:
...



target:
...
```

Register State

rax (condition): unknown
rflags: unknown

🟦 : Non-speculative execution
🟪 : Speculative execution

Eclipse Instrumentation

```
if (condition) {
   fptr();
}
...



target:
...
```

```
cmpl    $0x0, %rax
je      no_call
 callq   *%rcx
.no_call:
...



target:
...
```

Register State

rax (condition): unknown
rflags: unknown
rcx (fptr): target

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
if (condition) {
  fptr();
}
...
```

```
cmpl    $0x0, %rax
je      no_call
callq   *%rcx
.no_call:
...
```

```
target:
...
```

```
target:
...
```

Register State

```
rax (condition): unknown
rflags: unknown
rcx (fptr): target
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...





target:
...
```

🟦 : Non-speculative execution
🟪 : Speculative execution

Register State

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

r11 (state): 0

r12 (poison): -1

🟦: Non-speculative execution

🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax      Modifies rflags
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
rax (condition): unknown
rflags: unknown
```

🟦 : Non-speculative execution
🟪 : Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax    Modifies rflags
je       no_call       Depends on rflags
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

r11 (state): 0
r12 (poison): -1
rax (condition): unknown
rflags: unknown

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call   Depends on rflags
cmove   %r12, %r11   Depends on rflags
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

r12 (poison): -1
rax (condition): unknown
rflags: unknown
r11 (state): unknown

🟦: Non-speculative execution
🟥: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call        Depends on rflags
cmove   %r12, %r11     Depends on rflags
or      %r11, %rcx     Depends on r11
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): unknown
rflags: unknown
r11 (state): unknown
rcx (fptr): unknown
```

🟦 : Non-speculative execution
🟪 : Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call   Depends on rflags
cmove   %r12, %r11   Depends on rflags
or      %r11, %rcx   Depends on r11
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): unknown
rflags: unknown
r11 (state): unknown
rcx (fptr): unknown
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...




target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation

```
state = 0;
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

r12 (poison): -1
rax (condition): 0
rflags: resolved

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Design – SMAC Instruction Identification

Eclipse Design – Identifying SMAC Indirect Branches

- Iterate each instruction in a function
- When encountering an indirect branch:
  - Remove block from the function's Control-flow Graph (CFG)
  - Is the CFG still fully connected?
    - If yes, classify the indirect branch as SMAC

Eclipse Design – SMAC Instruction Identification

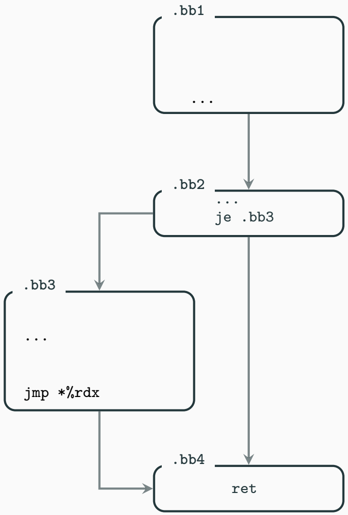
Eclipse Design – Identifying SMAC Indirect Branches

- Iterate each instruction in a function
- When encountering an indirect branch:
    - Remove block from the function's Control-flow Graph (CFG)
    - Is the CFG still fully connected?
        - If yes, classify the indirect branch as SMAC

Identifying Preceding Conditional Branches

- Reiterate the CFG backwards, starting from each block containing a SMAC indirect branch
    - Keep track of all encountered conditional branches

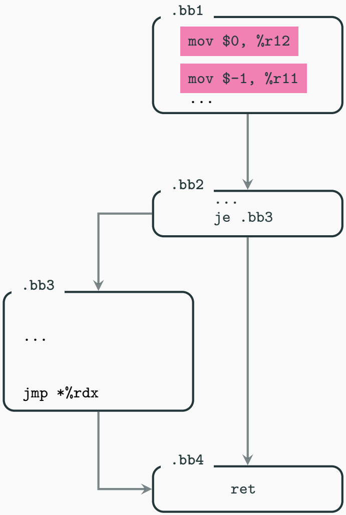
Eclipse Design – Instrumentation



Eclipse Design – Instrumentation

Register Initialization

Initialize a *state* and a *poison* register $\rightarrow$ 0 and -1, respectively



Eclipse Design – Instrumentation

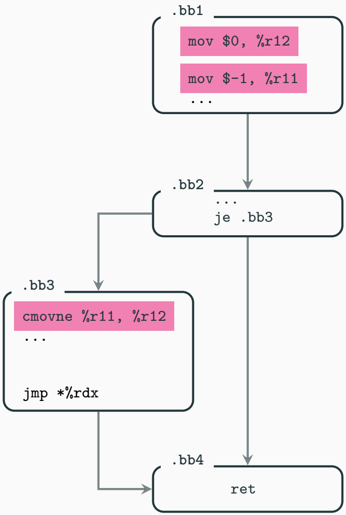
Register Initialization

Initialize a *state* and a *poison* register → 0 and −1, respectively

Capturing Data Dependencies

For each tracked conditional branch, inject a *conditional move* instruction at each edge

- Taken edge → *opposite* conditional code
- Not-taken edge → *same* conditional code



Eclipse Design – Instrumentation

Register Initialization

Initialize a *state* and a *poison* register → 0 and −1, respectively

Capturing Data Dependencies

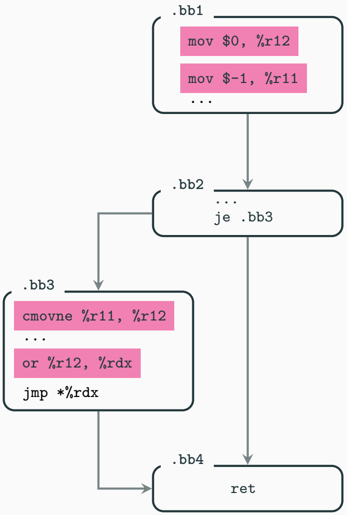
For each tracked conditional branch, inject a *conditional move* instruction at each edge

- Taken edge → *opposite* conditional code
- Not-taken edge → *same* conditional code

Linking Data Dependencies

Before each SMAC indirect branch, inject an `or` instruction

- Source operand → *state register*
- Destination operand → register used by indirect branch



Other SMA Attacks

Speculative Probing[1]

An SMA attack that can bypass certain information-hiding-based memory-error mitigations (*e.g.*, (K)ASLR, XOM, *etc.*)

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

[3] *Bypassing memory safety mechanisms through speculative control flow hijacks.* Mambretti *et al.*, EuroS&P 2021.

BROWN

Other SMA Attacks

Speculative Probing[1]

An SMA attack that can bypass certain information-hiding-based memory-error mitigations (*e.g.*, (K)ASLR, XOM, *etc.*)

PACMAN[2]

An SMA attack that can be used to bypass ARM's Pointer Authentication

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

[3] *Bypassing memory safety mechanisms through speculative control flow hijacks.* Mambretti *et al.*, EuroS&P 2021.

Other SMA Attacks

Speculative Probing[1]

An SMA attack that can bypass certain information-hiding-based memory-error mitigations (*e.g.*, (K)ASLR, XOM, *etc.*)

PACMAN[2]

An SMA attack that can be used to bypass ARM's Pointer Authentication

SPEAR[3]

Demonstrates how SMA attacks can be used to bypass several hardening schemes (*e.g.*, LLVM's SSP, GCC's VTV, *etc.*)

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

[3] *Bypassing memory safety mechanisms through speculative control flow hijacks.* Mambretti *et al.*, EuroS&P 2021.

Other SMA Attacks

Speculative Probing[1]

An SMA attack that can bypass certain information-hiding-based mem

Common Pattern

Attacker architecturally corrupts memory, then causes a SMAC instruction to be *speculatively* executed

PAC

An S n

SPE

Dem

schemes (*e.g.*, LLVM's SSP, GCC's VTV, *etc.*)

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

[3] *Bypassing memory safety mechanisms through speculative control flow hijacks.* Mambretti *et al.*, EuroS&P 2021.

Generalizing Eclipse

Eclipse is not tied to any particular architecture or SMA attack

[1] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

Generalizing `Eclipse`

`Eclipse` is not tied to any particular architecture or SMA attack

Other Architectures

`Eclipse` can be applied to any architecture that provides instructions for *capturing* and *linking* data dependencies

- *e.g.*, `Eclipse` can be applied against SP on ARM using the `csetm` (*capturing*) and `orr` instructions (*linking*)

[1] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

Generalizing `Eclipse`

`Eclipse` is not tied to any particular architecture or SMA attack

Other Architectures

- `Eclipse` can be applied to any architecture that provides instructions for *capturing* and *linking* data dependencies
  - *e.g.*, `Eclipse` can be applied against SP on ARM using the `csetm` (*capturing*) and `orr` instructions (*linking*)

Other SMA Attacks

- `Eclipse` can be deployed against any SMA attack
  - Data dependencies will be linked onto different SMAC instructions
- Deployed `Eclipse` against the ARM-specific PACMAN[1] attack
  - SMAC are ARM PA authentication instructions (*e.g.*, `autia`)

[1] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

BROWN

Performance Evaluation

Alternative Mitigations

Performance Evaluation

Alternative Mitigations

- `Eclipse-lfence`: `Eclipse` variant which mitigates SP by injecting serializing instructions (*i.e.*, `lfence`) before SMAC indirect branches
  - Not out-of-the-box! Relies on `Eclipse` to identify SMAC instructions

Performance Evaluation

Alternative Mitigations

- `Eclipse-lfence`: `Eclipse` variant which mitigates SP by injecting serializing instructions (*i.e.*, `lfence`) before SMAC indirect branches
  - Not out-of-the-box! Relies on `Eclipse` to identify SMAC instructions
- Speculative Load Hardening (`SLH`): Out-of-the-box mitigation against Spectre-PHT, also prevents SP
  - More generic mitigation, hardens all load instructions in a function

Performance Evaluation

Userland Performance: SPEC CPU 2017

Benchmark	`Eclipse`	`Eclipse-lfence`	`SLH`
`600.perlbench_s`	4.31%	4.26%	50.82%
`602.gcc_s`	0.74%	0.76%	49.74%
`605.mcf_s`	6.52%	26.73%	58.59%
`619.lbm_s`	0.42%	0.35%	2.62%
`620.omnetpp_s`	9.05%	22.94%	33.49%
`623.xalancbmk_s`	8.49%	11.69%	154.36%
`625.x264_s`	3.85%	10.67%	26.58%
`631.deepsjeng_s`	0.23%	0.19%	31.49%
`638.imagick_s`	9.53%	≈0%	97.74%
`641.leela_s`	1.21%	1.23%	20.03%
`644.nab_s`	0.29%	0.72%	31.36%
`657.xz_s`	≈0%	0.13%	54.26%

BROWN

Performance Evaluation

Userland Performance: SPEC CPU 2017

Benchmark	Eclipse	Eclipse-lfence	SLH
600.perlbench_s	4.31%	4.26%	50.82%
602.gcc_s	0.74%	0.76%	49.74%
605.mcf_s	6.52%	26.73%	58.59%
619.lbm_s	0.42%	0.35%	2.62%
620.omnetpp_s	9.05%	22.94%	33.49%
623.xalancbmk_s	8.49%	11.69%	154.36%
625.x264_s	3.85%	10.67%	26.58%
631.deepsjeng_s	0.23%	0.19%	31.49%
638.imagick_s	9.53%	≈0%	97.74%
641.leela_s	1.21%	1.23%	20.03%
644.nab_s	0.29%	0.72%	31.36%
657.xz_s	≈0%	0.13%	54.26%

- Eclipse outperforms alternative approaches, incurring up to 9.53% overhead

BROWN

Performance Evaluation

Userland Performance: Real-world Applications

Application	`Eclipse`	`Eclipse-lfence`	`SLH`
SQLite	8.61%	12.72%	55.11%
Redis (GET/s)	≈0%	0.17%	3.20%
Redis (SET/s)	≈0%	0.17%	3.20%
Nginx (1KB)	1.00%	0.67%	2.00%
Nginx (100KB)	0.65%	0.10%	3.73%
Nginx (1MB)	0.36%	0.78%	3.52%
MariaDB	0.42%	1.60%	10.16%

Performance Evaluation

Userland Performance: Real-world Applications

Application	`Eclipse`	`Eclipse-lfence`	`SLH`
SQLite	8.61%	12.72%	55.11%
Redis (GET/s)	≈0%	0.17%	3.20%
Redis (SET/s)	≈0%	0.17%	3.20%
Nginx (1KB)	1.00%	0.67%	2.00%
Nginx (100KB)	0.65%	0.10%	3.73%
Nginx (1MB)	0.36%	0.78%	3.52%
MariaDB	0.42%	1.60%	10.16%

- `Eclipse` incurs up to 8.61% overhead in real-world applications

Performance Evaluation

Kernel Performance

- LMBench kernel microbenchmarks
  - ≈0%–7.95% latency overhead
  - $<$ 3.04% bandwidth degradation
- Phoronix Test Suite macrobenchmarks
  - Negligible overhead ($<$ 2%) on various benchmarks (Nginx, MariaDB, TensorFlow, Linux kernel build, OpenSSL, Glibc)

Security Evaluation

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

Security Evaluation

x86-64

- Applied `Eclipse` to the Linux kernel
- ✓ Demonstrated that `Eclipse` blocks the original Speculative Probing (SP)[1] attack that de-randomizes KASLR

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

BROWN

Security Evaluation

x86-64

- Applied `Eclipse` to the Linux kernel
- ✓ Demonstrated that `Eclipse` blocks the original Speculative Probing (SP)[1] attack that de-randomizes KASLR

ARM

- Applied `Eclipse` against original PACMAN[2] attack
- Deployed `Eclipse` on a proof-of-concept userland SP attack on ARM
- ✓ Demonstrated that `Eclipse` stops both PACMAN and SP on ARM

[1] *Speculative Probing: Hacking Blind in the Spectre Era.* Göktas, *et al.*, CCS 2020.

[2] *PACMAN: Attacking ARM Pointer Authentication with Speculative Execution.* Ravichandran *et al.*, ISCA 2022.

Conclusion

- `Eclipse`: compiler-assisted mitigation against SMA attacks
  - Introduce artificial data dependencies to prevent SMAC instructions from using attacker-controlled data during speculative execution

Conclusion

- `Eclipse`: compiler-assisted mitigation against SMA attacks
  - Introduce artificial data dependencies to prevent SMAC instructions from using attacker-controlled data during speculative execution
- Evaluated security effectiveness and performance overhead
  - ✓ Sucessfully prevents SMA attacks such as SP and PACMAN
  - ✓ Real-world applications $\rightarrow$ ≈0% – ≈8.6% overhead
  - ✓ Linux kernel $\rightarrow$ negligible overhead

Conclusion

- `Eclipse`: compiler-assisted mitigation against SMA attacks
  - Introduce artificial data dependencies to prevent SMAC instructions from using attacker-controlled data during speculative execution
- Evaluated security effectiveness and performance overhead
  - ✓ Sucessfully prevents SMA attacks such as SP and PACMAN
  - ✓ Real-world applications → ≈0% – ≈8.6% overhead
  - ✓ Linux kernel → negligible overhead
- `https://gitlab.com/brown-ssl/eclipse/`

BROWN

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

🟦: Non-speculative execution

🟪: Speculative execution

Register State

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
rax (condition): 1
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
rax (condition): 1
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...




target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 1
rflags: resolved
r11 (state): 0
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...




target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 1
rflags: resolved
r11 (state): 0
rcx (fptr): target
```

🟦 : Non-speculative execution
🟪 : Speculative execution

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 1
rflags: resolved
r11 (state): 0
rcx (fptr): target
```

🟦	: Non-speculative execution
🟪	: Speculative execution

BROWN

Eclipse Instrumentation – Non-speculative Execution

```
state = 0; /* Why 0? */
poison = -1;
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...
```

```
target:
...
```

```
target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 1
rflags: resolved
r11 (state): 0
rcx (fptr): target
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

🟦: Non-speculative execution
🟪: Speculative execution

Register State

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1;  /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...




target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...




target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
```

🟦 : Non-speculative execution
🟪 : Speculative execution

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
rax (condition): unknown
rflags: unknown
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r11 (state): 0
r12 (poison): -1
rax (condition): unknown
rflags: unknown
```

🟦 : Non-speculative execution

🟪 : Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): unknown
rflags: unknown
r11 (state): unknown
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
r11 (state): unknown
rflags: unknown
rax (condition): 0
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
r11 (state): -1
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov      $0x0, %r11
mov      $0xffffffffffffffff, %r12
cmpl     $0x0, %rax
je       no_call
cmove    %r12, %r11
 or       %r11, %rcx
callq    *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
r11 (state): -1
rcx (fptr): -1
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
r11 (state): -1
rcx (fptr): -1
```

🟦: Non-speculative execution
🟪: Speculative execution

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...



target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
 je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
.no_call:
...



target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Eclipse Instrumentation – Poisoning the Code Pointer

```
state = 0;
poison = -1; /* Why -1? */
if (condition) {
  state = (!condition) ? poison : state;
  fptr |= state;
  fptr();
}
...


target:
...
```

```
mov     $0x0, %r11
mov     $0xffffffffffffffff, %r12
cmpl    $0x0, %rax
je      no_call
cmove   %r12, %r11
or      %r11, %rcx
callq   *%rcx
 .no_call:
 ...


target:
...
```

Register State

```
r12 (poison): -1
rax (condition): 0
rflags: resolved
```

🟦: Non-speculative execution
🟪: Speculative execution

BROWN

Why Poison the Branch Target?

- The data dependency we introduce delays the execution of the indirect branch until rflags is resolved
  - Poisoning seems redundant since when rflags is resolved, the target of conditional branch becomes known
- However, the ordering of the instructions is not guaranteed
  - When rflags is resolved, the conditional move and the indirect branch may execute before the conditional branch
  - Corrupted pointer may still be dereferenced
- Poisoning the pointer guarantees it will dereference a bad address