On the Spectre of Meltdown
Analysing the Attacks and Mitigations

Björn Ruytenberg

`bjorn@bjornweb.nl`

May 22, 2018

TU/e Technische Universiteit
Eindhoven
University of Technology

About me

Björn Ruytenberg

MSc Student in Information Security @ TUE

Teaching Assistant, graduate course on compilers & platforms @ TUE

BSc in Electrical Engineering and Computer Science

Security Researcher

Main interests: sandboxing and virtualization technology

Found several vulnerabilities in Microsoft Office, Foxit Reader, VMware Workstation, Adobe Flash

Roadmap

Introduction: Microarchitectural Basics

What is Meltdown?

What is Spectre?

Exploitation Scenarios

Mitigations

Attack Variants

Closing Thoughts

Meltdown

Meltdown – Basic Outline

- Design flaw that affects most modern Intel CPUs (and some ARMs)
- Uses out-of-order execution to leak data through cache timing attack

- From an unprivileged process, an attacker can:
  - Bypass language-based security
  - Bypass sandboxes, containers/paravirtualization hypervisors
  - Read arbitrary memory, including kernel memory

Caches

- Fast processor but slower memory
- Cache utilizes locality to bridge the gap
    - Divides memory into *lines*
    - Stores recently used lines

Processor

Cache

Memory

Instruction Pipelining

- Nominally, the processor executes instructions one after the other

- Instruction execution consists of multiple steps
  - Each uses a different unit

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back

```
mulq $m0
add    %rax,$A[0]
mov
8*2($np),%rax
lea    32($tp),$tp
adc    \$0,%rdx
mov    %rdx,$A[1]
mulq $m1
add    %rax,$N[0]
mov
8($a,$j),%rax
adc    \$0,%rdx
add    $A[0],$N[0]
adc    \$0,%rdx
mov    $N[0],-
24($tp)
mov    %rdx,$N[1]
mulq $m0
add    %rax,$A[1]
mov
8*1($np),%rax
adc    \$0,%rdx
mov    %rdx,$A[0]
mulq $m1
add    %rax,$N[1]
mov    ($a,$j),%rax
mov
8($a,$j),%rax
adc    \$0,%rdx
```

Instruction Pipelining

- Nominally, the processor executes instructions one after the other

- Instruction execution consists of multiple steps
  - Each uses a different unit

- Pipelining increases utilization by executing steps of multiple instructions

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back

```
mulq  $m0
add   %rax,$A[0]
mov
8*2($np),%rax
lea   32($tp),$tp
adc   \$0,%rdx
mov   %rdx,$A[1]
mulq  $m1add
%rax,$N[0]
mov
8($a,$j),%rax
adc   \$0,%rdx
add   $A[0],$N[0]
adc   \$0,%rdx
mov   $N[0],-
24($tp)
mov   %rdx,$N[1]
mulq  $m0
add   %rax,$A[1]
mov
8*1($np),%rax
adc   \$0,%rdx
mov   %rdx,$A[0]
mulq  $m1
add   %rax,$N[1]
```

```
c = a / b;
d = c + 5;
```

How to deal with dependencies?

Out-of-Order Execution (1)

- Execute instructions when data is available rather than by program order

|				
IF	ID	AF	EX	WB
IF	ID	🛑	EX	WB
IF	ID	AF	EX	WB

```
c = a / b;
d = c + 5;
e = f + g;
```

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**

- Why not retire immediately?

Out-of-Order Execution (2)

- Execute instructions when data is available rather than by program order

IF	ID	AF	EX	WB
IF	ID	🛑	EX	WB
IF	ID	AF	EX	WB

```
c = a / b;
d = c + 5;
e = f + g;
```

> What if **b=0**?

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**

- Why not retire immediately?

- **Out-of-order execution is speculative!**

- Need to **abandon** instructions in the reorder buffer if never executed

Program Flow – Legitimate Behavior

```
i = *pointer;
y = array[i * 256];
```


CPU cache

array

ptr

Secret data

User space

Kernel space (protected)
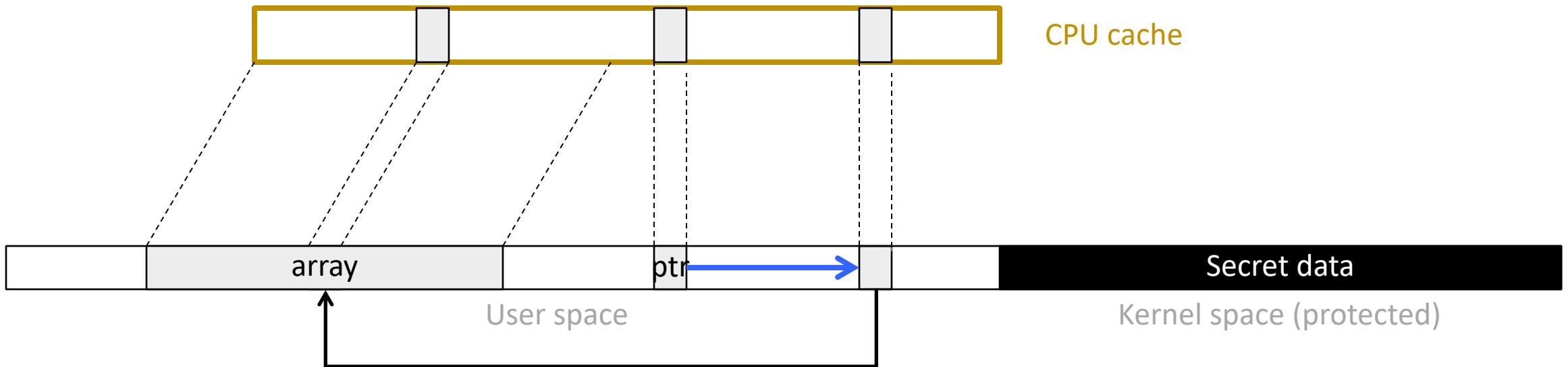
Attack Flow (1)

```
i = *pointer;
y = array[i * 256];
```

Step 1:
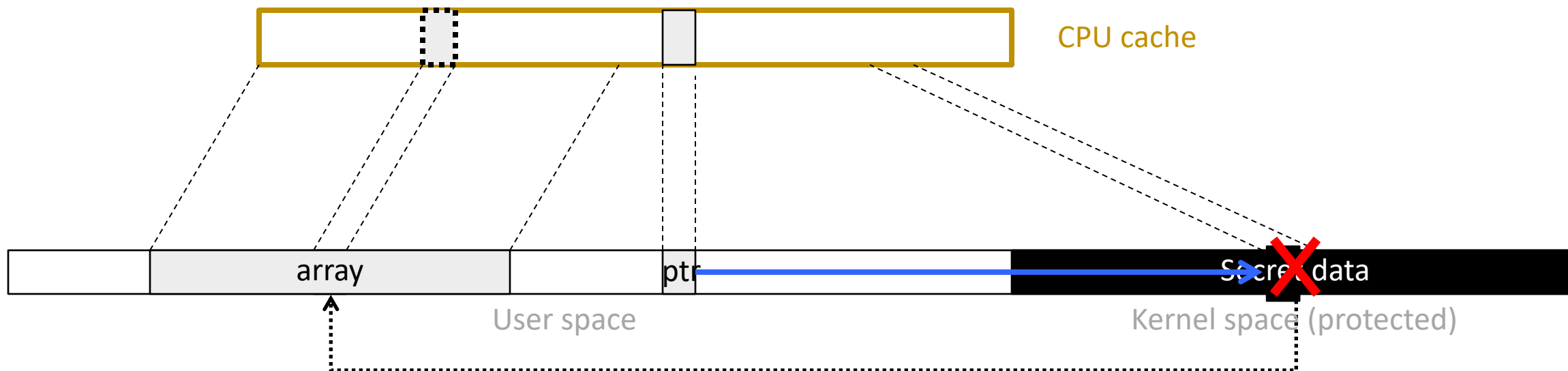Set pointer to kernel space

Step 2:
Due to out-of-order processing, CPU fetches secret value from kernel space

Step 3:
Secret value is used to index user space array

Exception triggered:
Results of out-of-order instructions discarded (`i` takes previous value)

CPU cache

array

ptr

Secret data

User space

Kernel space (protected)

Attack Flow (2)

```
i = *pointer;
y = array[i * 256];
```

Step 4:
Unprivileged process iterates through array elements

Step 5:
Cached element will return much faster: index indicates secret byte value

Fast (cache hit)

Slow Slow

CPU cache

array

ptr

Secret data

User space

Kernel space (protected)
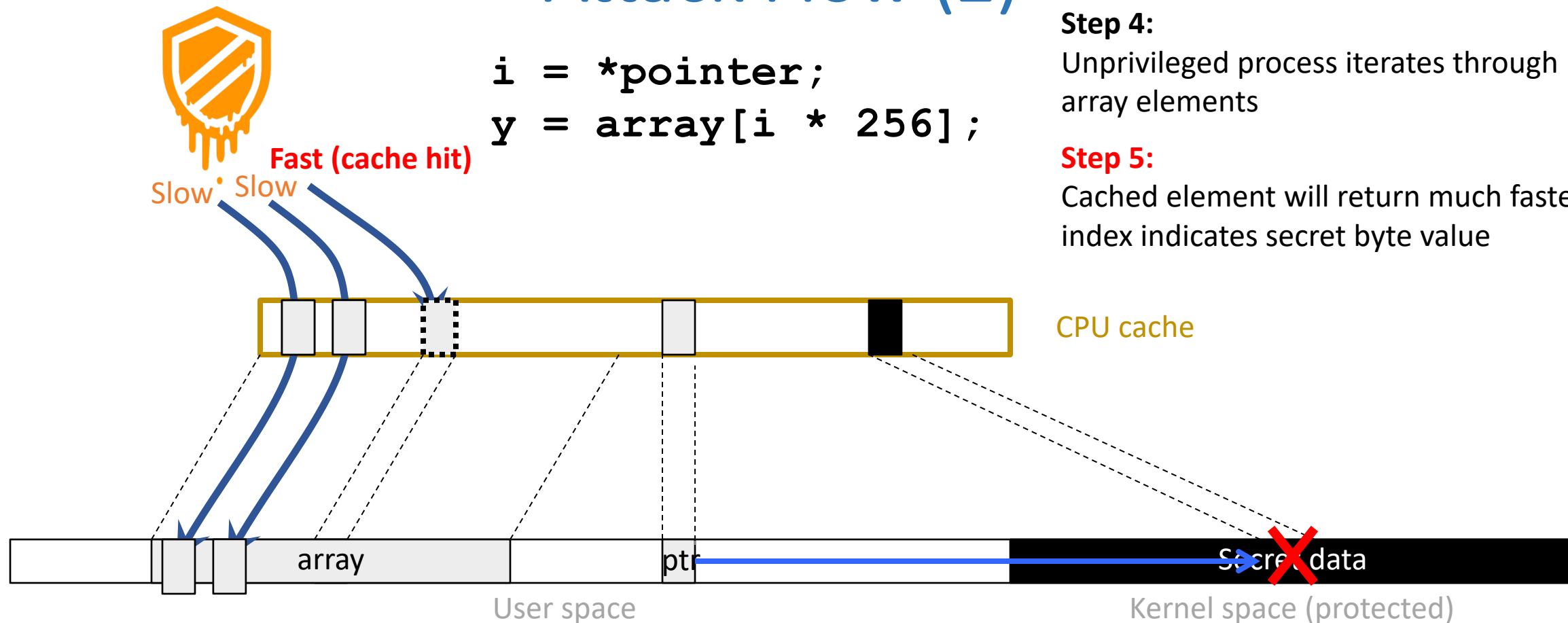
DEMO
Spying in Realtime on Password Input

Meltdown – Mitigation

- **Kernel Page Table Isolation (KPTI)**
  - Linux kernel memory no longer mapped into user space processes
  - User space can no longer access kernel memory

- Approach seemingly solid, but...
  - On-going discussion about soundness
    - SMI handlers: parts of kernel memory must always be mapped into user space processes
    - Protects kernel, but user space programs still vulnerable
    - Further research needed to confirm soundness
  - Introduces overhead when jumping from user mode to kernel mode
    - New capability proposed (CAP_DISABLE_PTI), disables KPTI for "safe" processes [1]

---

[1] *Re: [RFC PATCH v2 6/6] x86/entry/pti: don't switch PGD on when pti_disable is set [LWN.net]*

Meltdown - Intel-only? (1)

- Meltdown initially thought to be linked with Transactional Synchronization Extensions (TSX-NI)
  - Intel-only hardware atomic memory operations on Haswell and later
  - Enables Meltdown attack without triggering software exception handling
- TSX not a requirement for Meltdown
  - Does make attack virtually impossible to detect[2]

---

[2] *Detecting Attacks that Exploit Meltdown and Spectre with Performance Counters – Trend Micro, 2018*

Meltdown - Intel-only? (2)

- Meltdown initially thought not to affect AMD processors[3]

```
From      Tom Lendacky <thomas.lendacky@amd.com>
Subject   [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors
Date      Tue, 26 Dec 2017 23:43:54 -0600

AMD processors are not subject to the types of attacks that the kernel
page table isolation feature protects against.  The AMD microarchitecture
does not allow memory references, including speculative references, that
access higher privileged data when running in a lesser privileged mode
when that access would result in a page fault.
```

- Meltdown paper release: AMD *is* likely vulnerable
- PoC confirms OoO execution occurs across security domains, practical exploitation therefore seems feasible

---

[3] *LKML: Tom Lendacky: [PATCH] x86/cpu, x86/pti: Do not enable PTI on AMD processors*
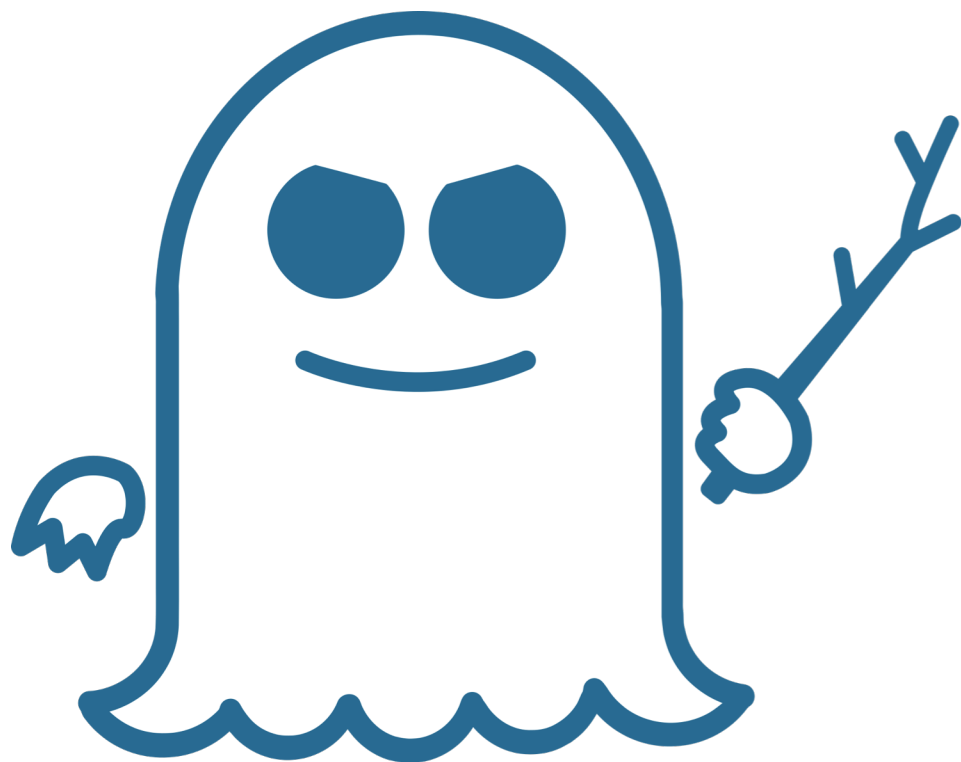
Spectre
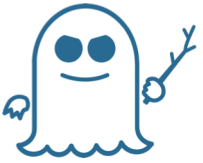
Spectre – Basic Outline

- Design flaw that affects all modern CPUs: Intel, AMD, ARM, POWER

- Branch prediction and speculative execution leave traces in cache

- Cache timing attack reveals data from different security domains

- Two variants:
  - **Spectre-v1**: Read from the current user space process
  - **Spectre-v2**: Read from other processes

Speculative Execution and Branches (1)

- When execution reaches a branch
- The processor predicts the outcome of the branch
- Execution proceeds (speculatively) along predicted branch

- **Correct prediction** → all is well
- **Misprediction** → abandon and resume
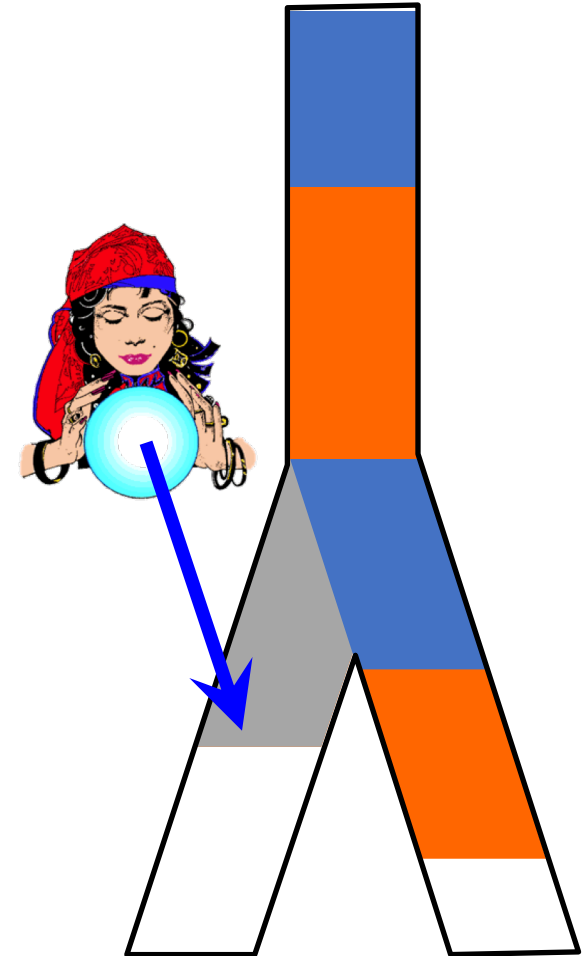
Speculative Execution and Branches (2)

- **Branch History Buffer (BHB)**

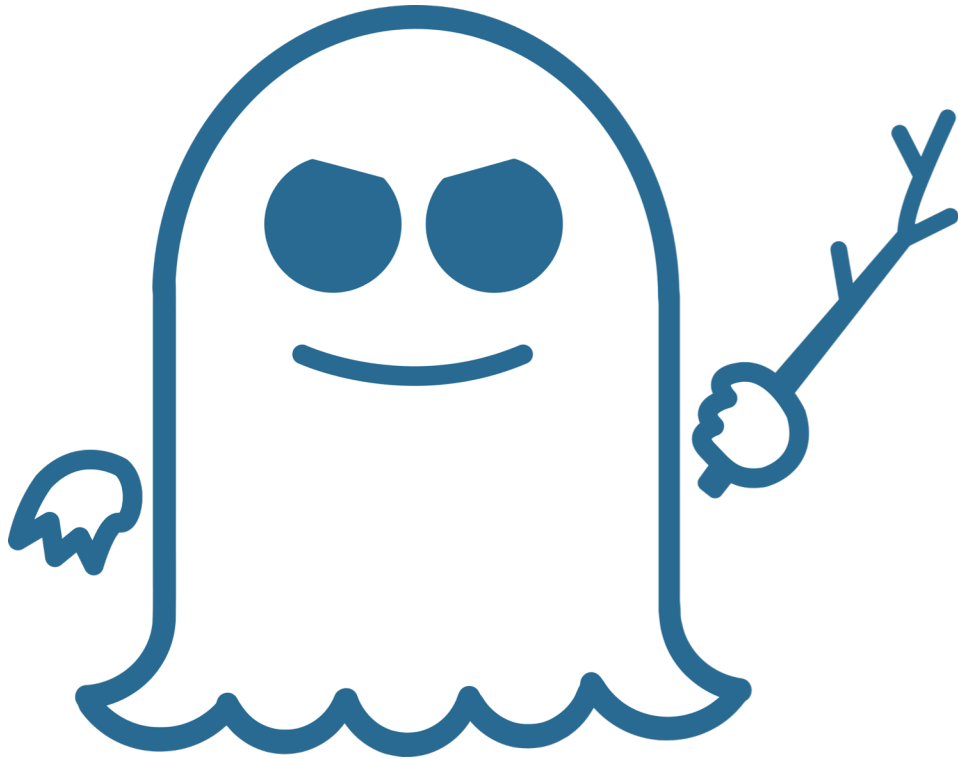 Outcome of conditional branches

  `JGE  4006c9`
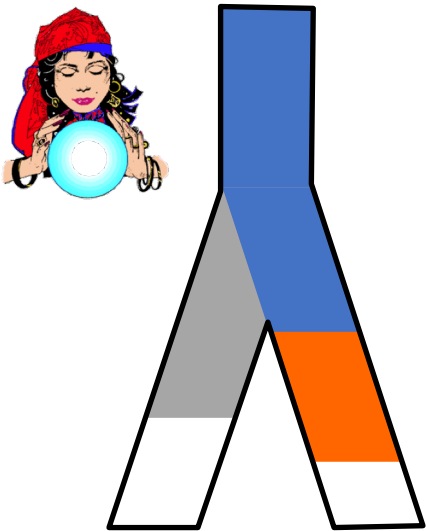
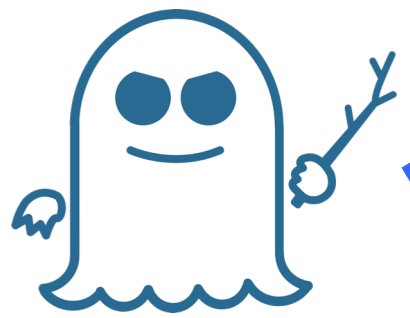- **Branch Target Buffer (BTB)**

  Target of indirect branches

  `JMP eax`
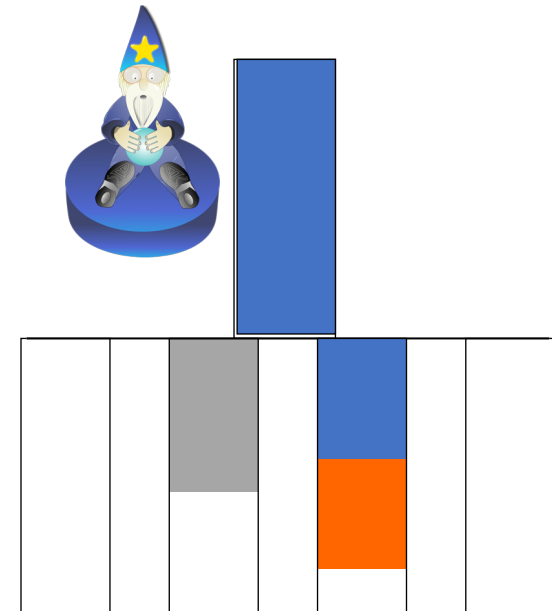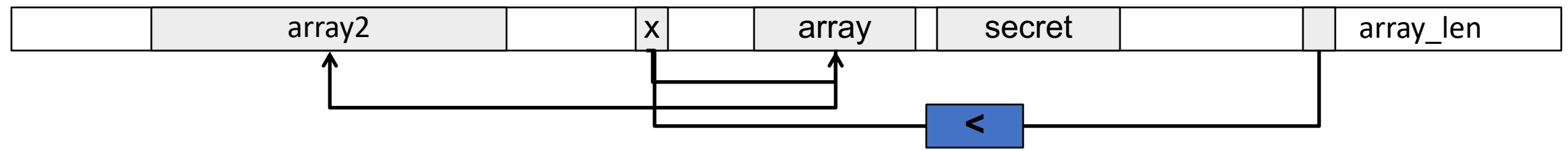
Spectre
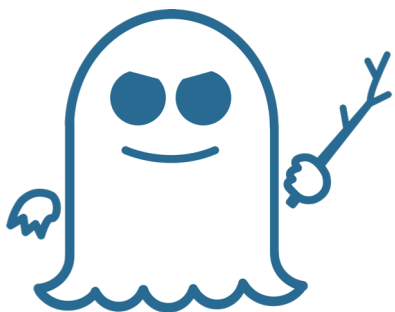## Variant 1: bounds check bypass

Victim

Spectre-v1

Attacker

Branch not taken!

```
if (x < array_len) {
    i = array[x];
    y = array2[i * 256];
}
```
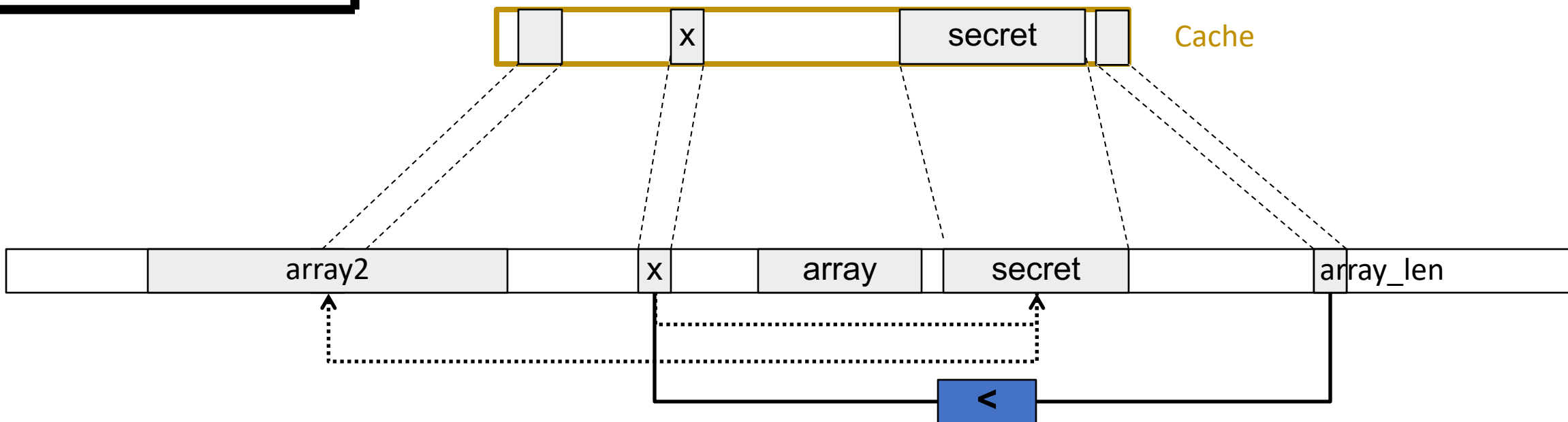
	array2		x	array	secret			array_len

<

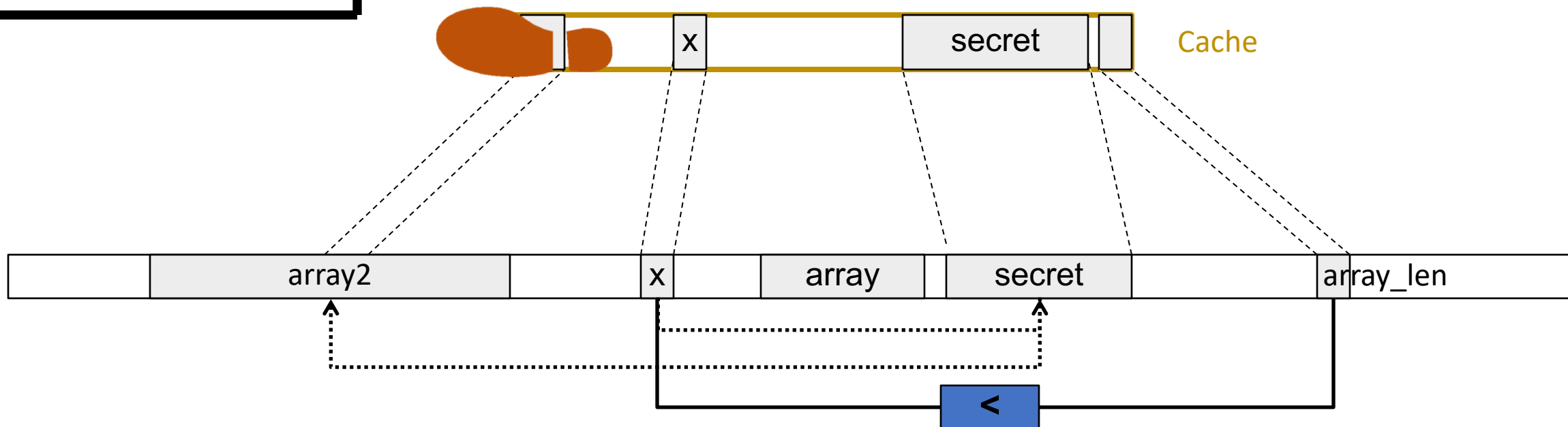On the Spectre of Meltdown – Björn Ruytenberg

On the Spectre of Meltdown – Björn Ruytenberg

Spectre-v1

Mispredict

```
if (x < array_len) {
    i = array[x];
    y = array2[i * 256];
}
```

Attacker

Branch not taken

Cache

secret

x

array2   x   array   secret   array_len

<

On the Spectre of Meltdown – Björn Ruytenberg
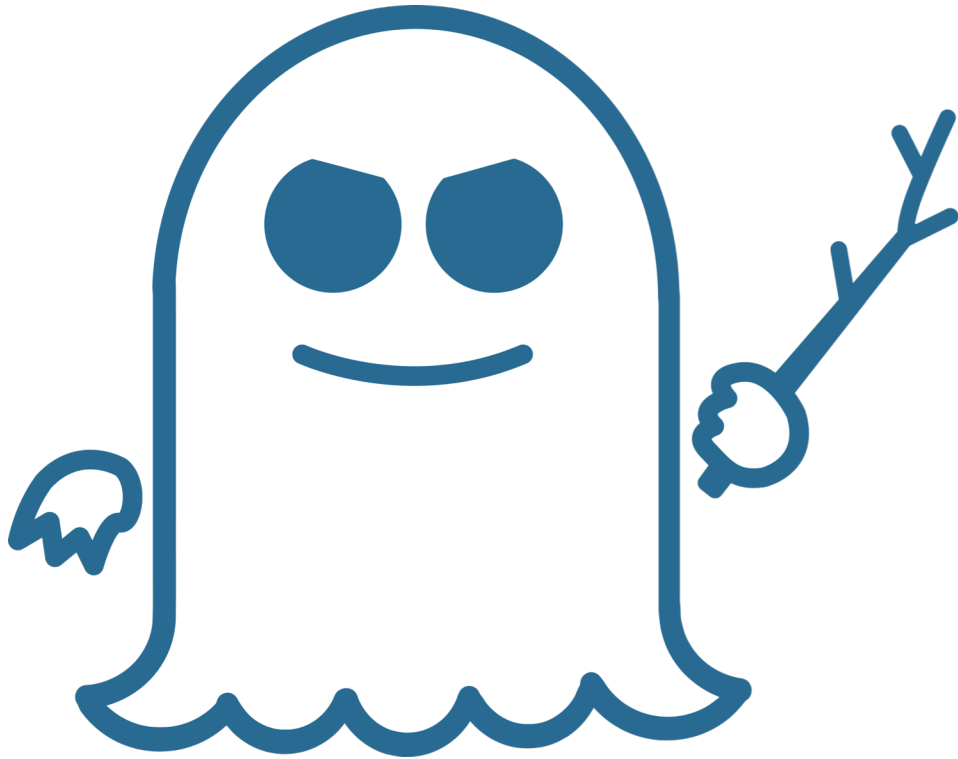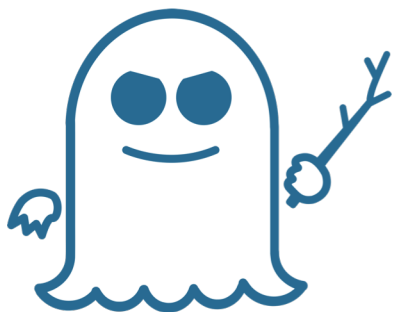
Spectre
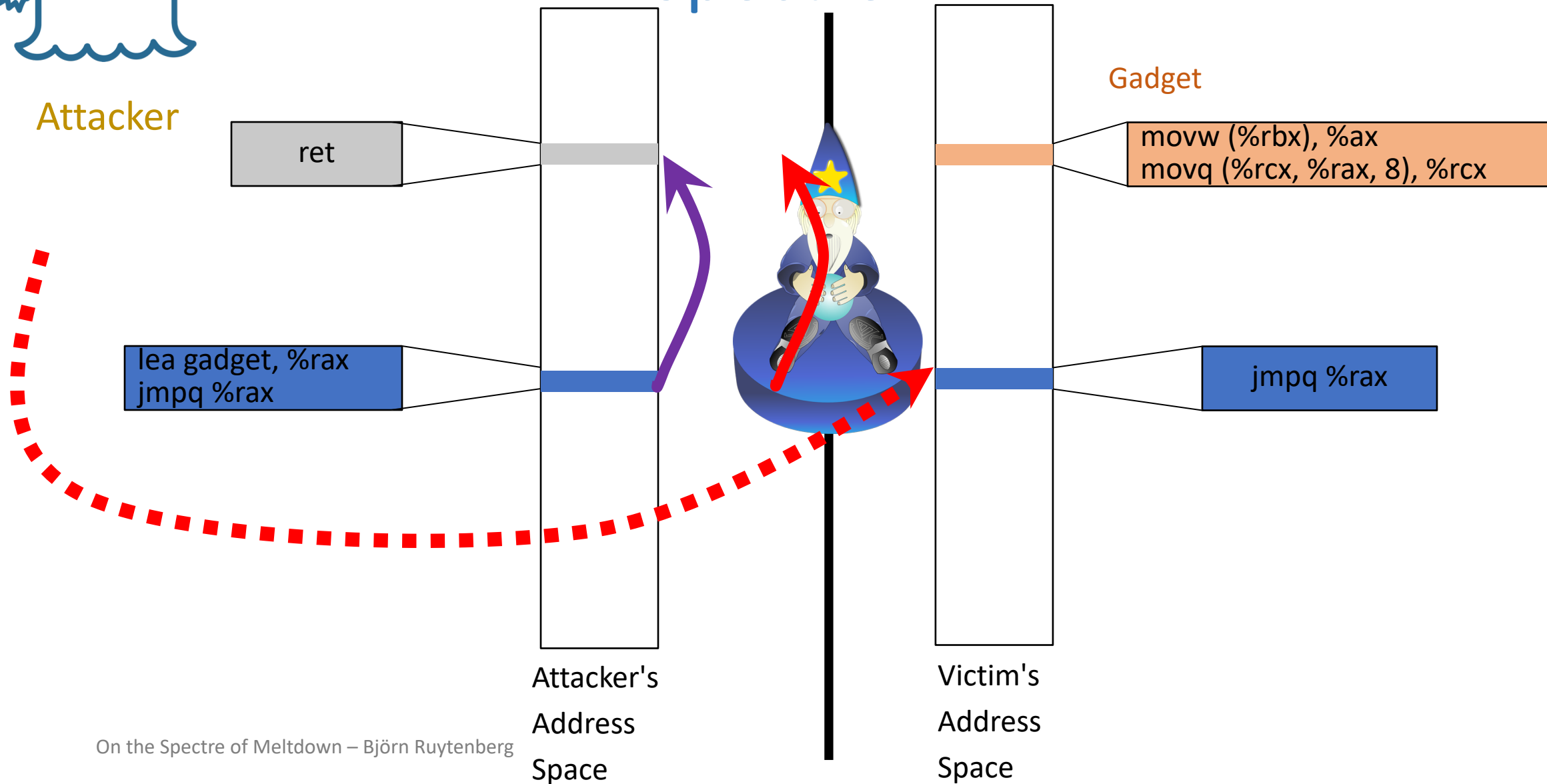Variant 2: branch target injection

Spectre – Mitigations

- **Basic idea:** prevent speculative execution across branches

- **Three approaches:**
  - Spectre-v1: Explicitly prevent speculative execution across conditional branches by inserting blocking operation
  - Spectre-v2: Avoid training branch predictor by replacing branch instructions with semantic equivalents
  - Spectre-v2: Disable branch prediction across security domains

Spectre-v1 – Insert Blocking Operation

- **Approach:** Prevent speculative execution by inserting blocking operation
- LFENCE (serialize load operations), PAUSE (spin loop hint)

```
scanf("%d", &untrusted);
if(untrusted < arrayLength)
{
    value = array[untrusted];
    asm("lfence");
    value2 = array2[value * 64];
}
```

```
call    4004a0 <__isoc99_scanf@plt>
mov     ecx,DWORD PTR [rbp-0xe4]
cmp     ecx,DWORD PTR [rbp-0xe8]
mov     DWORD PTR [rbp-0x114],eax
jge     4006c9 <main+0x109>
movsxd  rax,DWORD PTR [rbp-0xe4]
movsx   ecx,BYTE PTR [rbp+rax*1-0x70]
mov     DWORD PTR [rbp-0xec],ecx
lfence
mov     ecx,DWORD PTR [rbp-0xec]
shl     ecx,0x6
```

- Effective, but
  - Need to recompile code or patch binary
  - Significantly degrades performance – need static analysis to identify vulnerable code

Spectre-v2 – Retpoline (1)

- **Approach:** Avoid training branch predictor by replacing branch instructions with semantic equivalents

- **Return trampoline (retpoline)**
  - Indirect branch normally pulls return address off stack ("jump to this address")
  - Replace with PUSH/RET
    - Push target address onto stack
    - Return to target address
  - BTB does not learn about branch due to pattern mismatch

Spectre-v2 – Retpoline (2)

- Need to recompile code or patch binary

- Degrades performance
  - Somewhat mitigated by Return Stack Buffer (RSB)

- Not a perfect solution: ***ineffective*** on Skylake and later
  - RSB behavior different: when empty, falls back to BTB prediction
  - Addressed with RSB stuffing [4], but currently implemented by Linux kernel only [5]
  - Compiler support on the way [6]

[4] *Retpoline: A Branch Target Injection Mitigation - Intel, 2018*

[5] *x86/retpoline: Avoid return buffer underflows on context switch - Patchwork*

[6] *[llvm-dev] LLVM Release Schedules: 5.0.2, 6.0.1*

Spectre-v2 – Disable BTB Prediction

- **Approach:** Disable BTB prediction across security domains

- **Intel microcode update** [7]

  - Introduces new MSRs to control BTB

  - No learning across hyperthreads

  - Higher security levels do not learn from lower level activity

  - BTB clobbering, wiped on each context switch
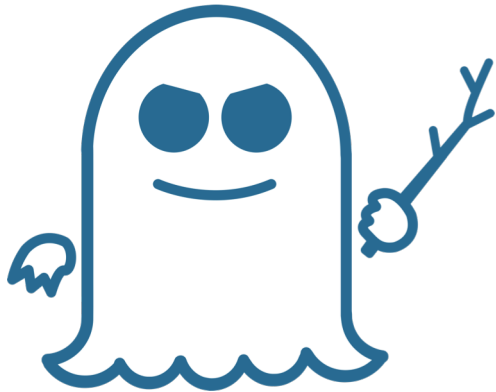
  - Major performance impact

---
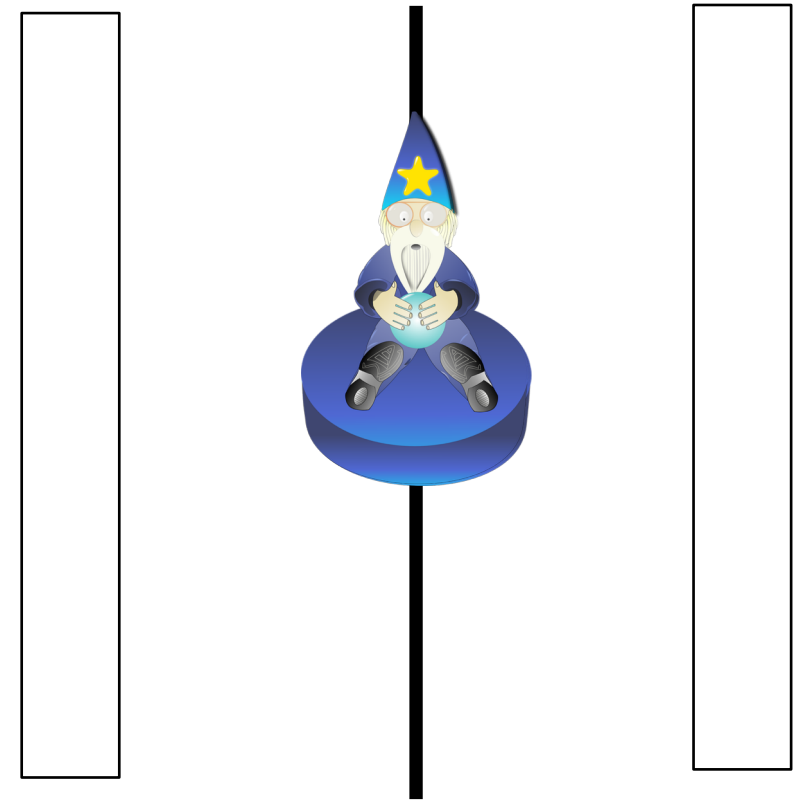
[7] *Microcode Revision Guidance - Intel, 2018*

[8] *Controlling the Performance Impact of Microcode and Security Patches - RedHat, 2018*

Attack Variants

On-going Research (1)

- **BranchScope – Evtyushkin et al.**
  - Pollutes cache of directional branch prediction (Pattern History Table), leaks data through branch selection timing side-channel
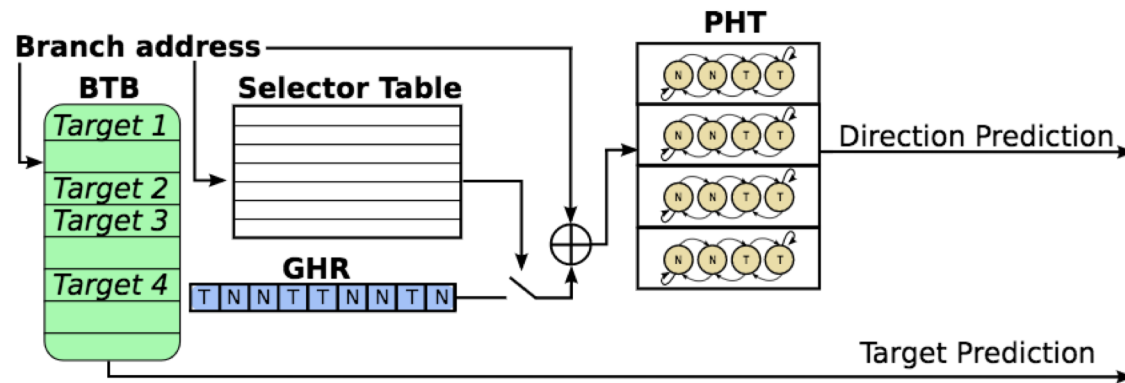

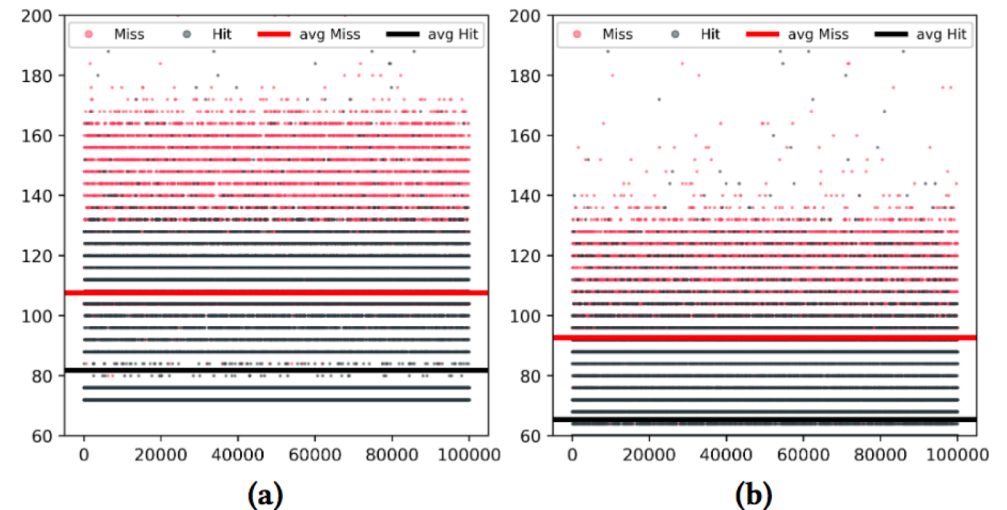**Figure 1.** A Combined Branch Predictor

Figure 7. Latency (cycles) of a not-taken (a) and taken (b) branch instruction

On-going Research (2)

- **SMM Bounds Check Bypass – Bazhaniuk et al.**
  - System Management Mode: highly-privileged firmware memory space (BIOS/UEFI), stores firmware secrets and SMI handlers
  - Extends Spectre to bypass hardware-based protections, leak SMM data

- **SGXpectre – Chen et al.**
  - Extends Spectre to leak secrets from SGX secure enclave

On-going Research (3)

- **Speculative Store Bypass, Rogue System Register Read (formerly Spectre-NG) – Horn et al.**
  - 8 new vulnerabilities affecting Intel and AMD, possibly ARM
  - First batch disclosed May 22
    - Design flaw in processing load instructions: operands not subject to preceding store operations are speculatively loaded
    - System registers can be subjected to speculative reads
    - Behavior can be exploited to target Spectre-like gadgets
  - Intel working on patches, two-stage release planned for May/August

Closing Thoughts

- Meltdown and Spectre affect fundamentals of modern CPU design

- Raise the bar in a new class of side channel attacks

- Many open questions
  - Mitigations subject to debate regarding effectiveness, impact
  - Attack variants part of on-going research
  - The real fix: a silicon redesign?