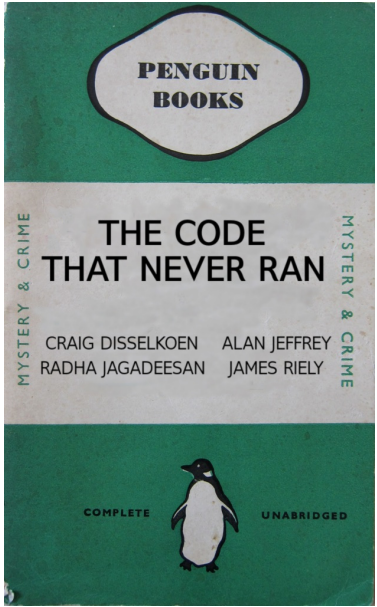


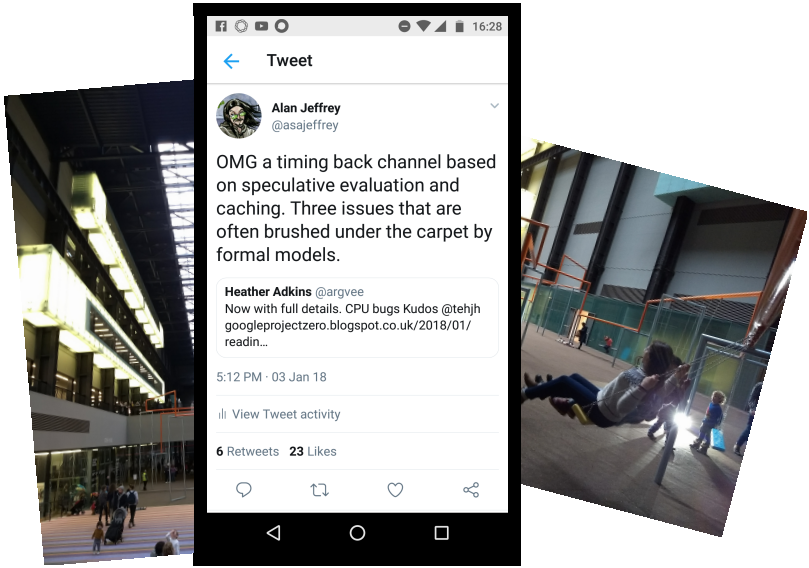
A classic locked-room mystery.
Eve was in the false branch
of a conditional the whole time,
how could she do it?

3 January 2018

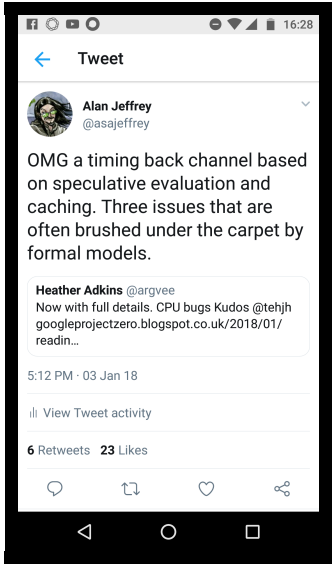
A day out at the Tate Modern

3 January 2018

3 January 2018



Spectre



Attacks bypass run-time security checks.

Can bypass array bounds checks,
and read whole process memory.

Can be exploited from JS,
so evil.ad.com can read your bank.com data.

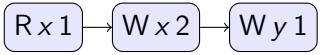
Attacks *speculative evaluation*
hardware optimization.

Optimizations in hardware

A lie we tell programmers:
"computers execute instructions one after the other."

$$x := x + 1; y := 1$$

has execution:

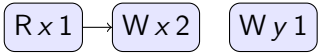


Optimizations in hardware

A lie we tell programmers:
"computers execute instructions one after the other."

$$x := x + 1; y := 1$$

has execution where W y 1 might happen first:

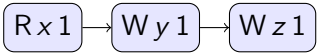


Optimizations in hardware

Another lie we tell programmers:
"only one branch of an if is executed."

$$\text{if}\,(x)\,\{\,y := 1; z := 1\,\}\,\text{else}\,\{\,y := 2; z := 1\,\}$$

has execution:

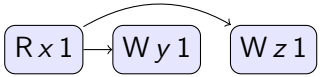


Optimizations in hardware

Another lie we tell programmers:
"only one branch of an if is executed."

$$\text{if}\,(x)\,\{\,y := 1; z := 1\,\}\,\text{else}\,\{\,y := 2; z := 1\,\}$$

has execution where W z 1 might happen before W y 1:

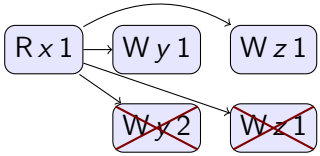


Optimizations in hardware

Another lie we tell programmers:
"only one branch of an if is executed."

$$\text{if}\,(x)\,\{\,y := 1; z := 1\,\}\,\text{else}\,\{\,y := 2; z := 1\,\}$$

has execution where W y 2 might happen, then get rolled back:

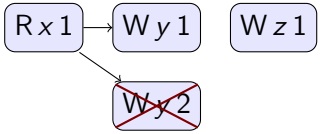


Optimizations in hardware and compilers

Another lie we tell programmers:
"only one branch of an if is executed."

$$\text{if}\,(x)\,\{\,y := 1; z := 1\,\}\,\text{else}\,\{\,y := 2; z := 1\,\}$$

has execution where W z 1 might happen first:



Simplified Spectre

Imagine a SECRET, protected by a run-time security check:

$$\text{if canRead(SECRET)} \{ \ldots \text{use SECRET} \ldots \} \text{ else } \{ \ldots \}$$

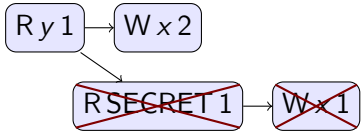
For attacker code canRead(SECRET) is always false

Simplified Spectre

Imagine a SECRET, protected by a run-time security check:

$$\text{if canRead(SECRET)} \{ \ldots \text{use SECRET} \ldots \} \text{ else } \{ \ldots \}$$

For attacker code canRead(SECRET) is always false, e.g.



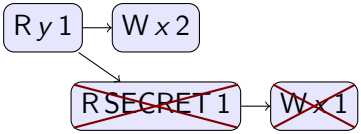
is an execution of if y { if canRead(SECRET) { x:= SECRET } else { x:= 2 } }.

Simplified Spectre

Imagine a SECRET, protected by a run-time security check:

$$\text{if canRead(SECRET)} \{ \ldots \text{use SECRET} \ldots \} \text{ else } \{ \ldots \}$$

For attacker code canRead(SECRET) is always false, e.g.



is an execution of if y { if canRead(SECRET) { x:= SECRET } else { x:= 2 } }.

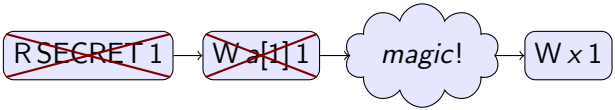
Attacker goal: learn if SECRET is 0 or 1.

Simplified Spectre

A very simplified Spectre attack:

$$\begin{array}{l}\text{if canRead(SECRET)} \{\ a[\text{SECRET}] := 1\ \} \\ \text{else if touched } (a[0]) \{\ x := 0\ \} \\ \text{else if touched } (a[1]) \{\ x := 1\ \}\end{array}$$

with execution



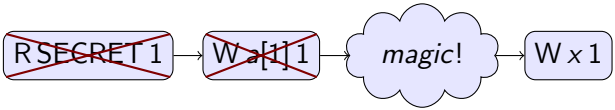
Information flow from SECRET to x, *if* there's an implementation of "magic".

Simplified Spectre

A very simplified Spectre attack:

$$\begin{array}{l}\text{if canRead(SECRET)} \{\ a[\text{SECRET}] := 1\ \} \\ \text{else if touched } (a[0]) \{\ x := 0\ \} \\ \text{else if touched } (a[1]) \{\ x := 1\ \}\end{array}$$

with execution



Information flow from SECRET to x, *if* there's an implementation of "magic".

Narrator: there was one.

Results

Formalization of pretty pictures as *partially ordered multisets* (Gisher, 1988).

Compositional semantics based on weak memory models (e.g. C11).

Examples modeling Spectre, Spectre mitigations,
PRIME+ABORT attack on transactional memory...

Results

Formalization of pretty pictures as *partially ordered multisets* (Gisher, 1988).

Compositional semantics based on weak memory models (e.g. C11).

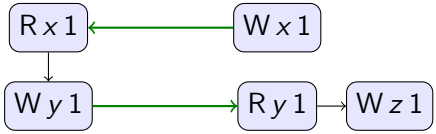
Examples modeling Spectre, Spectre mitigations,
PRIME+ABORT attack on transactional memory...
and a new family of attacks on compiler optimizations.

Modelling an attack on compiler optimizations

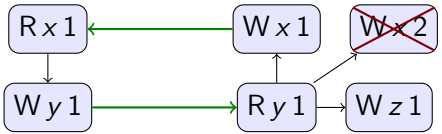
An attacker running two threads:

$$
\begin{array}{lll}
y := x & \| & \text{if}\,(y == 0)\,\{\,x := 1\,\} \\
 & & \text{else if}\,(\text{canRead}(\text{SECRET}))\,\{\,x := \text{SECRET}\,\} \\
 & & \text{else}\,\{\,x := 1; z := 1\,\}
\end{array}
$$

If SECRET is 1, there is an execution:



If SECRET is 2, there is no execution (due to cyclic dependency):



Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

What about the attacks on compiler optimizations?

Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

What about the attacks on compiler optimizations?

Yes

Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

What about the attacks on compiler optimizations?

Yes, under unrealistic assumptions:

- SECRET is a constant known at compile-time,
- canRead(SECRET) is a run-time check.

Implementing an attack on load/store reordering

Main attacker thread: $x := 1; \text{if}\,(\text{canRead}(\text{SECRET}))\,\{\,x := \text{SECRET};\,\}\,r := y;$

If SECRET is not 1, gcc generates:

```
mov canReadSecret(%rip), %eax
mov $1, x(%rip)
test %eax, %eax
je label1
mov $0, x(%rip)
label1:
mov y(%rip), %eax
```

Writes x then reads y

If SECRET is 1, gcc generates:

```
mov canReadSecret(%rip), %eax
mov y(%rip), %eax
mov $1, x(%rip)
```

Conditional has been eliminated!
Reads y then writes x

A forwarding thread copies x to y, so the attacker can spot the reordering.

Implementing an attack on load/store reordering

To make this attack more likely, introduce a small delay between write of x and read of y, increases probability of round trip.

Experimentally gcc will reorder load/store across 30 straight-line instructions.

Repeat attack to leak multiple bits, and increase probability of success.

Attack is 99.9% accurate at 100Kbps.

Implementing an attack on dead store elimination

A similar attack on works on dead store elimination.

Attack works on clang as well as gcc, 99.9% accurate at 400Kbps.

Contributions

A compositional model of program execution that includes speculation.

Examples including existing information flow attacks on branch prediction and transactional memory, and new attacks on optimizing compilers.

Experimental evidence that the new attacks can be carried out,
but only against compile-time secrets.

(Phew, we failed to mount attacks on JIT compilers.)

Thank you!

https://github.com/chicago-relaxed-memory/spec-eval