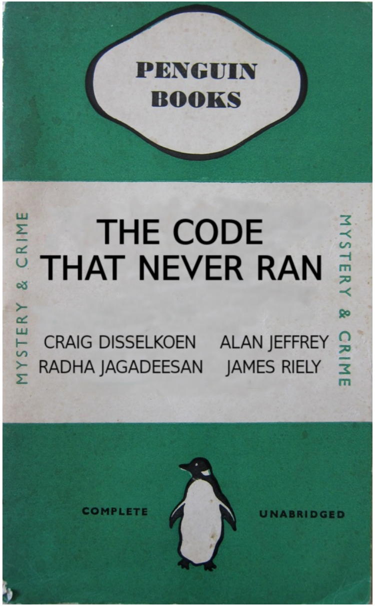


A classic locked-room mystery.
Eve was in the false branch
of a conditional the whole time,
how could she do it?

Mozilla Research | DePaul University | U. California San Diego

3 January 2018

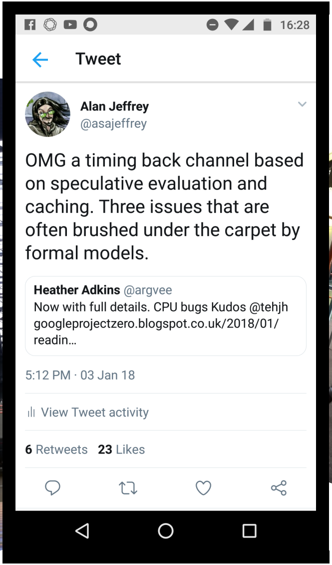
A day out at the Tate Modern

3 January 2018

Craig Disselkoen,
Radha Jagadeesan,
Alan Jeffrey,
James Riely

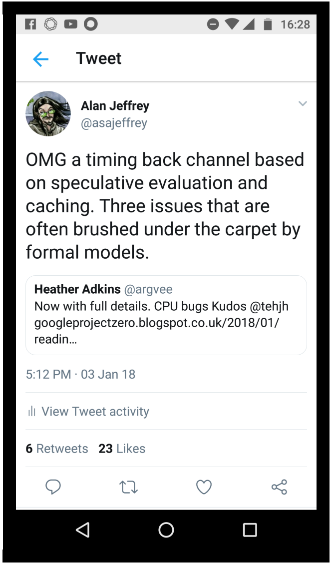
Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

3 January 2018



Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

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.

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Optimizations in hardware

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

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

has execution:

$$\boxed{\mathrm{R}\, x\, 1} \longrightarrow \boxed{\mathrm{W}\, x\, 2} \longrightarrow \boxed{\mathrm{W}\, y\, 1}$$

Craig Disselkoen,
Radha Jagadeesan,
Alan Jeffrey,
James Riely

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Optimizations in hardware

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

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

has execution where $\mathrm{W}\ y\ 1$ might happen first:

$$\boxed{\mathrm{R}\ x\ 1} \longrightarrow \boxed{\mathrm{W}\ x\ 2} \qquad \boxed{\mathrm{W}\ y\ 1}$$

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Optimizations in hardware

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

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

has execution:

$$\boxed{\mathrm{R}\;x\;1} \longrightarrow \boxed{\mathrm{W}\;x\;2} \qquad \boxed{\mathrm{W}\;y\;1}$$

Shared-memory concurrency leaks the abstraction

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Optimizations in hardware

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

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

has execution:

$$\boxed{\mathrm{R}\,x\,1} \longrightarrow \boxed{\mathrm{W}\,x\,2} \qquad \boxed{\mathrm{W}\,y\,1}$$

Shared-memory concurrency leaks the abstraction

Resulted in entire research area: weak memory models (e.g. Pugh *et al.*; C11)

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

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:

$$\boxed{\mathrm{R}\,x\,1} \longrightarrow \boxed{\mathrm{W}\,y\,1} \longrightarrow \boxed{\mathrm{W}\,z\,1}$$

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 z 1 might happen before W y 1:

$$\begin{tikzcd}
\boxed{\mathrm{R}\,x\,1} \arrow[r] \arrow[rr, bend left] & \boxed{\mathrm{W}\,y\,1} & \boxed{\mathrm{W}\,z\,1}
\end{tikzcd}$$

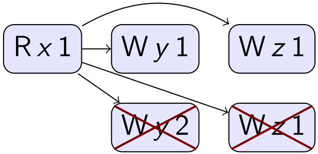
Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Optimizations in hardware

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

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

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

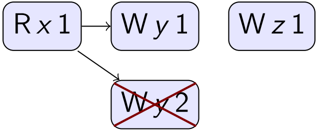


Optimizations in hardware and compilers

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 $\mathsf{W}\,z\,1$ might happen first:

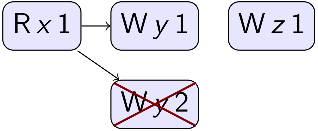


Optimizations in hardware and compilers

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:



No language-level model for this!

As weak memory models are to OOO, so *what* is to speculation?

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

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

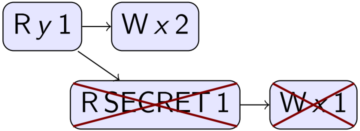
Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Simplified Spectre

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

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

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



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

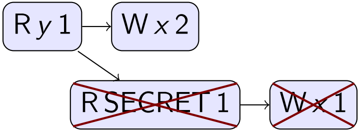
Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Simplified Spectre

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

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

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



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

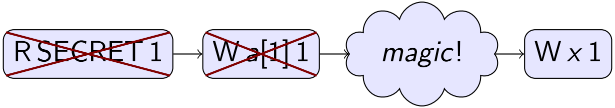
Attacker goal: learn if SECRET is 0 or 1.

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".

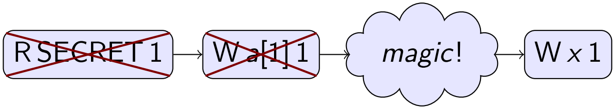
Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Simplified Spectre

A very simplified Spectre attack:

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

with execution



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

Narrator: there was one.

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

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).

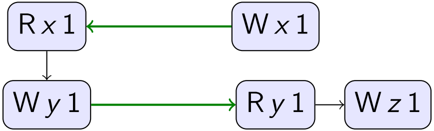
Examples modeling Spectre, Spectre mitigations,
PRIME+ABORT attack on transactional memory. . .
and a new family of attacks on compiler optimizations.

Modeling an attack on compiler optimizations

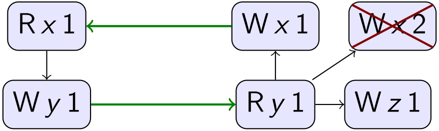
An attacker running two threads (initially $x = y = 0$):

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

If SECRET is 1, there is an execution:



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



Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

Can we implement the attacks on compiler optimizations?

Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

Can we implement the attacks on compiler optimizations?

Yes

Implementing attacks on compiler optimizations

Spectre and Prime+Abort are implemented.

Can we implement 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: $x := 1; \mathrm{if}\,(\mathrm{canRead}(\mathrm{SECRET}))\,\{\, x := \mathrm{SECRET};\ \} r := y;$

Implementing an attack on load/store reordering

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

When SECRET $\neq$ 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
```

When SECRET $=$ 1, gcc generates:

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

Implementing an attack on load/store reordering

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

When SECRET $\neq 1$, 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

When SECRET $= 1$, gcc generates:

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


Craig Disselkoen, Radha Jagadeesan, Alan Jeffrey, James Riely

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions

Implementing an attack on load/store reordering

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

When SECRET $\neq 1$, 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

When SECRET $= 1$, gcc generates:

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

Conditional has been eliminated!

Reads y then writes x

Implementing an attack on load/store reordering

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

When SECRET $\neq$ 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

When SECRET $=$ 1, gcc generates:

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

Conditional has been eliminated!
Reads y then writes x

Forwarding thread $x := y$ allows attacker to spot the reordering

Implementing an attack on load/store reordering

Small delay between write x and read y: increases probability of round trip

gcc will reorder across 30 straight-line instructions

Repeat to leak multiple bits, error correction

Bitwise accuracy 99.99% at 300Kbps

Implementing an attack on dead store elimination

A similar attack targets dead store elimination

Works on clang + gcc

Bitwise accuracy 99.99% at 1.2Mbps

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.)

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

Craig Disselkoen, Radha Jagadeesan, Alan Jeffrey, James Riely

Introduction
Spectre
Optimizations
Simplified Spectre
Results
Experiments
Conclusions