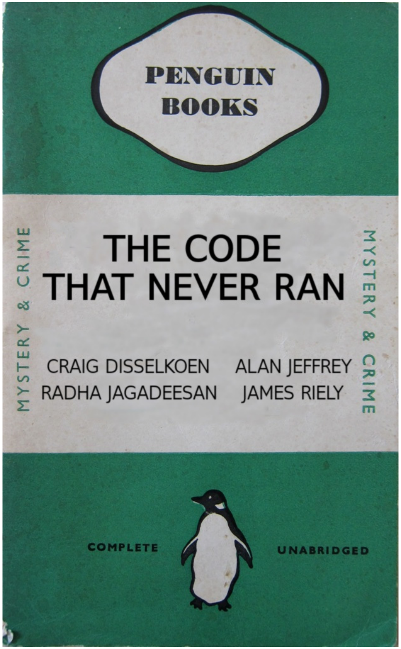


A classic locked-room mystery.
Eve was in the false branch of a
conditional the whole time,
how could she do it?

Mozilla Research | DePaul University | U. California San Diego

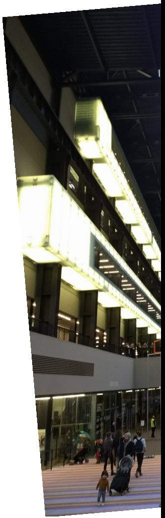
3 January 2018

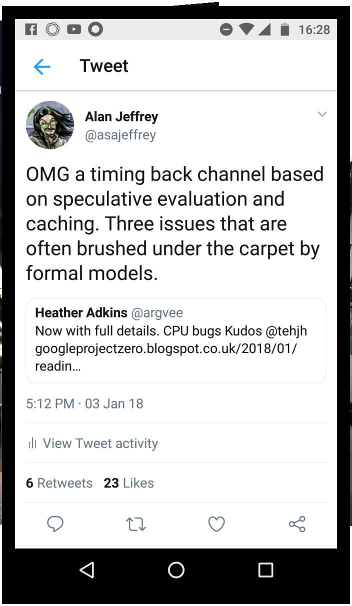
Had a day out at the Tate Modern

3 January 2018

3 January 2018

3 January 2018





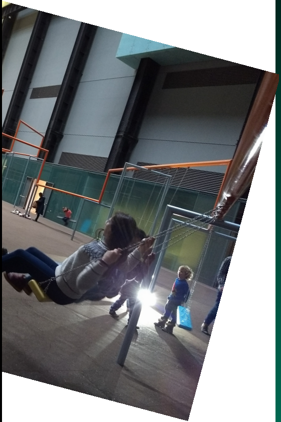
Alan Jeffrey
@asajeffrey

OMG a timing back channel based on speculative evaluation and caching. Three issues that are often brushed under the carpet by formal models.

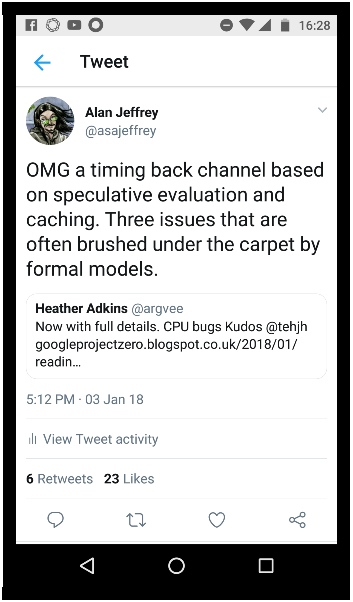
Heather Adkins @argvee
Now with full details. CPU bugs Kudos @tehjh
googleprojectzero.blogspot.co.uk/2018/01/
readin...

5:12 PM · 03 Jan 18

6 Retweets 23 Likes



Spectre


16:28

Tweet

Alan Jeffrey
@asajeffrey

OMG a timing back channel based on speculative evaluation and caching. Three issues that are often brushed under the carpet by formal models.

Heather Adkins @argvee
Now with full details. CPU bugs Kudos @tehjh googleprojectzero.blogspot.co.uk/2018/01/readin...

5:12 PM · 03 Jan 18

View Tweet activity

6 Retweets **23** Likes

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:

$$\begin{tikzcd} \boxed{\mathrm{R}\,x\,1} \arrow[r] & \boxed{\mathrm{W}\,x\,2} \arrow[r] & \boxed{\mathrm{W}\,y\,1} \end{tikzcd}$$

Optimizations in hardware

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

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

has execution:

$$\begin{tikzcd}
\boxed{\mathrm{R}\,x\,1} \arrow[r] & \boxed{\mathrm{W}\,x\,2} & \boxed{\mathrm{W}\,y\,1}
\end{tikzcd}$$

The W y 1 might happen first.

Optimizations in hardware

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

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

has execution:

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

Optimizations in hardware

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

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

has execution:

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

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."

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

has execution:

$$\begin{tikzcd}
\boxed{\mathrm{R}\,x\,1} \arrow[r] \arrow[rr, bend left] \arrow[dr] \arrow[drr] & \boxed{\mathrm{W}\,y\,1} & \boxed{\mathrm{W}\,z\,1} \\
 & \boxed{\xcancel{\mathrm{W}\,y\,2}} & \boxed{\xcancel{\mathrm{W}\,z\,1}}
\end{tikzcd}$$

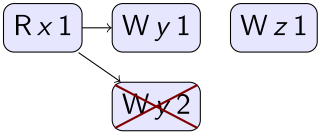
W y 2 might happen, then get rolled back.

Optimizations in hardware and compilers

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

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

has execution:



W z 1 might happen first.

Simplified Spectre

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

```
if canRead(SECRET) { ... use SECRET ... } else { ... }
```

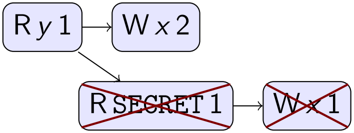
For attacker code `canRead(SECRET)` is always false

Simplified Spectre

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

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

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



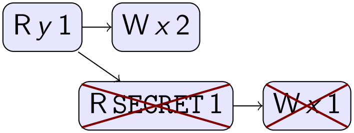
is an execution of
$\texttt{if}\ y\ \{\ \texttt{if}\ \texttt{canRead}(\texttt{SECRET})\ \{\ x := \texttt{SECRET}\ \}\ \texttt{else}\ \{\ x := 2\ \}\ \}$.

Simplified Spectre

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

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

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



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

Attacker goal: learn if SECRET is 0 or 1.

Simplified Spectre

A very simplified Spectre attack:

```
if canRead(SECRET) { a[SECRET] := 1 }
else if touched (a[0]) { x := 0 }
else if touched (a[1]) { x := 1 }
```

with execution

$$\begin{tikzcd}
\boxed{\xcancel{\mathrm{R}\ \mathtt{SECRET}\ 1}} \arrow[r] & \boxed{\xcancel{\mathrm{W}\ a[1]\ 1}} \arrow[r] & \textit{magic}! \arrow[r] & \boxed{\mathrm{W}\ x\ 1}
\end{tikzcd}$$

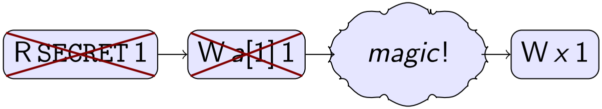
Information flow from SECRET to x

Simplified Spectre

A very simplified Spectre attack:

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

with execution



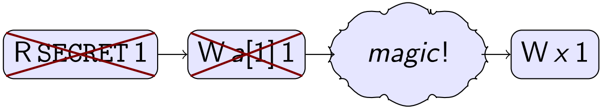
Information flow from SECRET to x,
if there's an implementation of "magic".

Simplified Spectre

A very simplified Spectre attack:

```
if canRead(SECRET) { a[SECRET] := 1 }
else if touched (a[0]) { x := 0 }
else if touched (a[1]) { x := 1 }
```

with execution



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

Narrator: there was one.

TODO

Start building models, tools etc. which capture the attacks.

TODO

Start building models, tools etc. which capture the attacks.

Make it harder to implement `if` (`touched`(x))
(e.g. reduce acess to high-precision timers).

TODO

Start building models, tools etc. which capture the attacks.

Make it harder to implement `if (touched(`x`))`
(e.g. reduce acess to high-precision timers).

Process isolation: make sure security boundaries
line up with process boundaries.

TODO

Start building models, tools etc. which capture the attacks.

Make it harder to implement `if (touched (`x`))`
(e.g. reduce acess to high-precision timers).

Process isolation: make sure security boundaries
line up with process boundaries.

Harden programs, compilers, etc.
(difficult because it's a large attack surface).