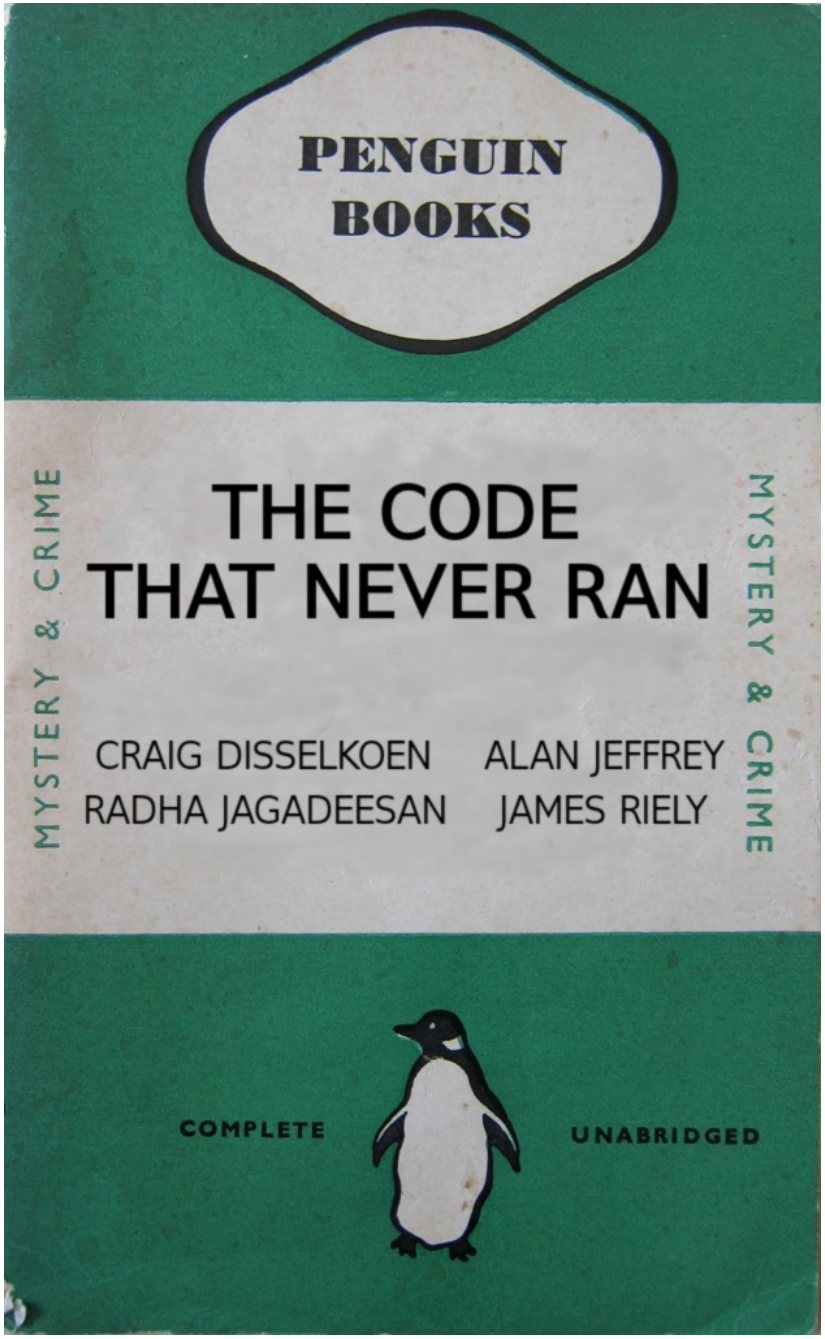


A classic locked-room mystery.
Eve was in the false branch of a conditional the whole time,
how could she do it?

Mozilla Research | DePaul University | U. California San Diego

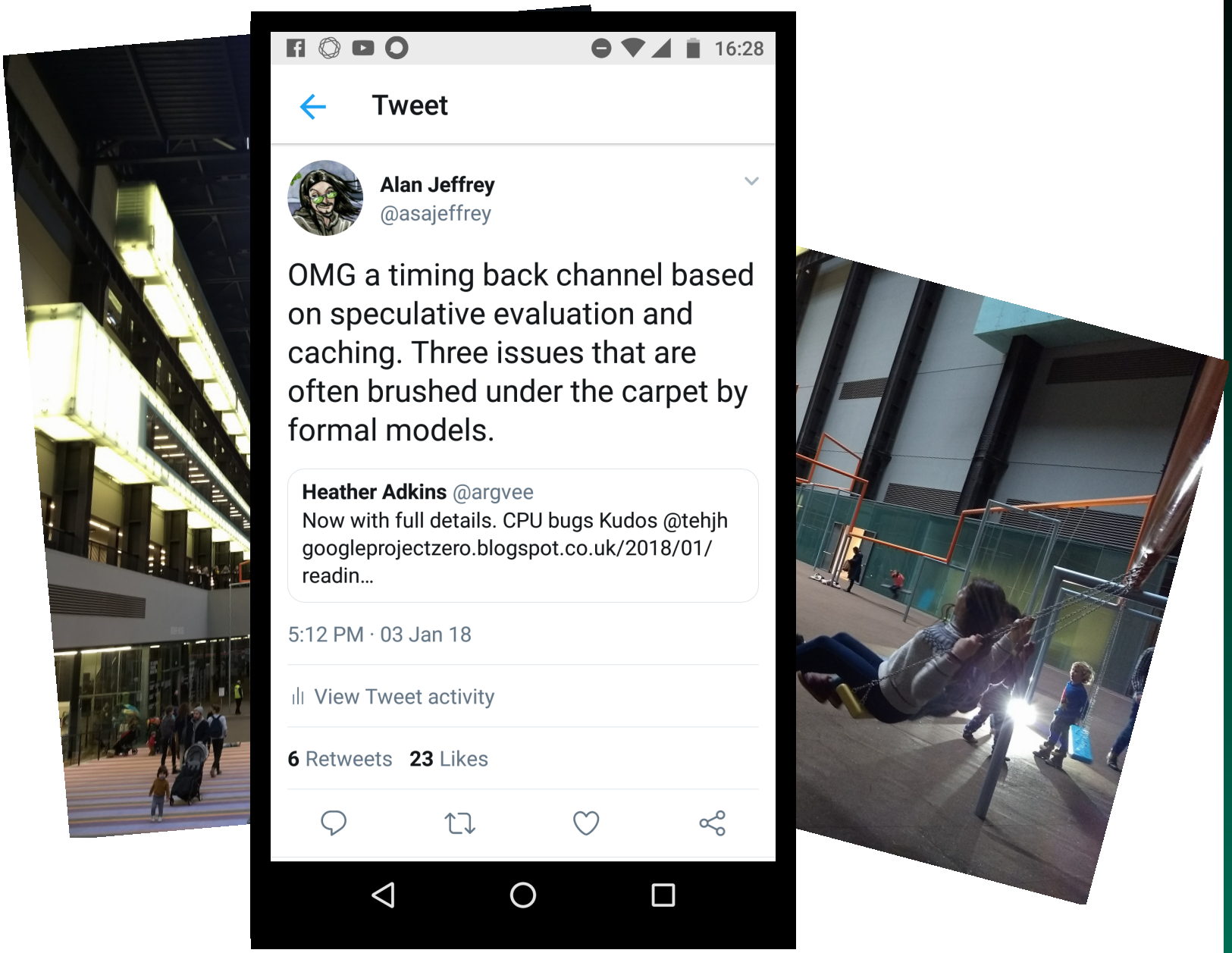
3 January 2018

Had a day out at the Tate Modern

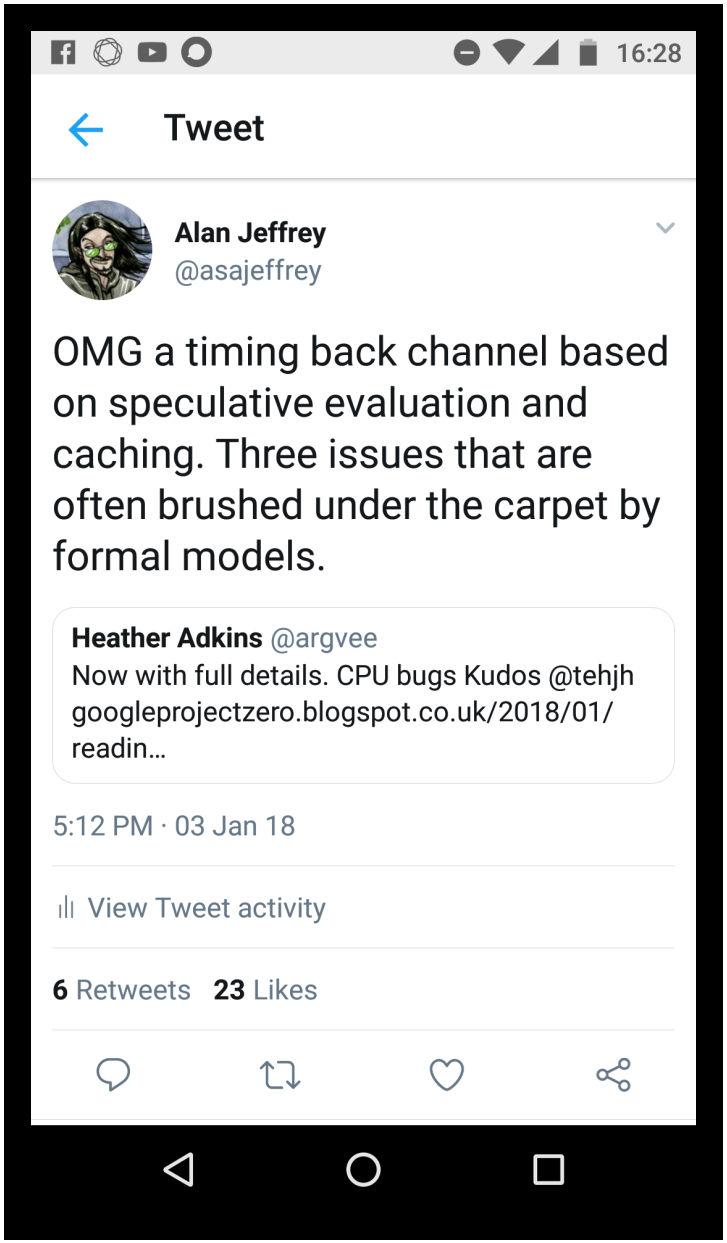
3 January 2018

3 January 2018

3 January 2018



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.

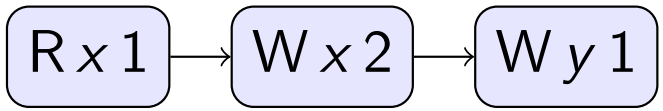
Attacks *speculative evaluation* hardware optimization.

Optimizations in hardware

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

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

has execution:

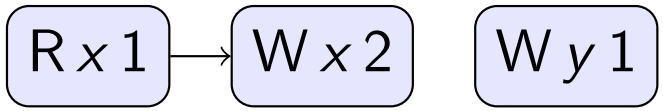


Optimizations in hardware

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

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

has execution:



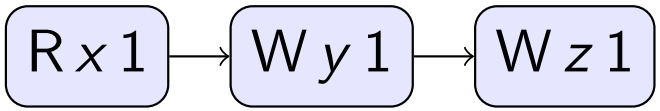
The W y 1 might happen first.

Optimizations in hardware

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

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

has execution:

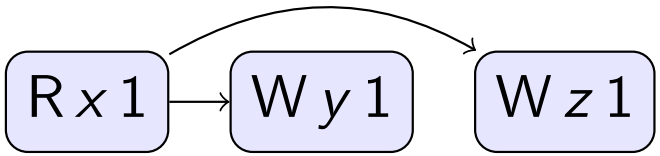


Optimizations in hardware

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:



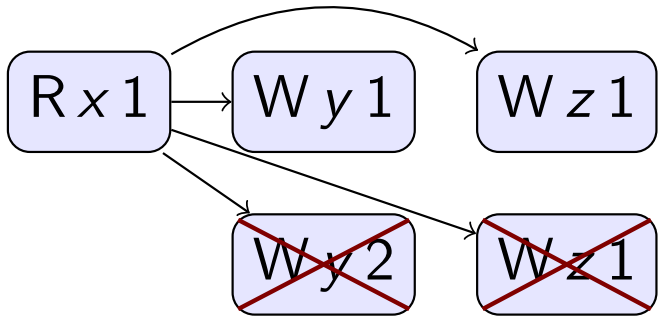
W z 1 might happen before W y 1.

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:



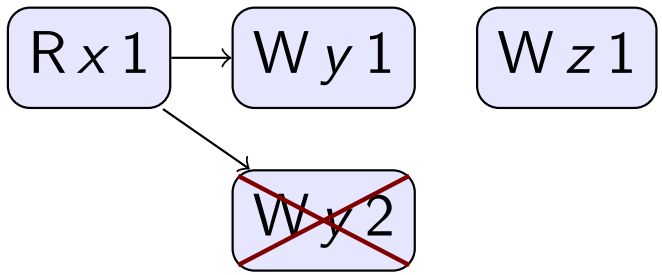
W y 2 might happen, then get rolled back.

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:

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

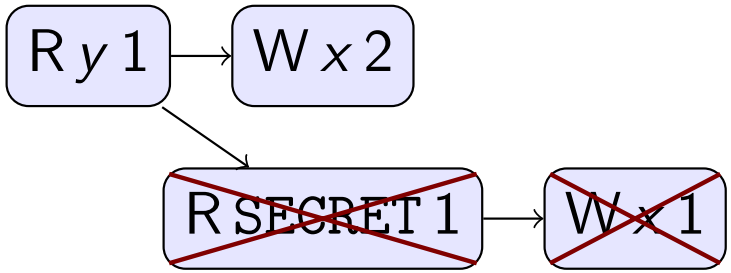
For attacker code `canRead(SECRET)` is always false

Simplified Spectre

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

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

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



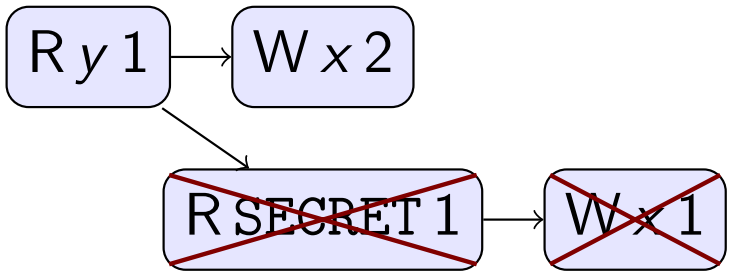
is an execution of
$\texttt{if}\ y\,\{\,\texttt{if canRead(SECRET)}\,\{\,x\texttt{:=SECRET}\,\}\,\texttt{else}\,\{\,x\texttt{:=2}\,\}\,\}$.

Simplified Spectre

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

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

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



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

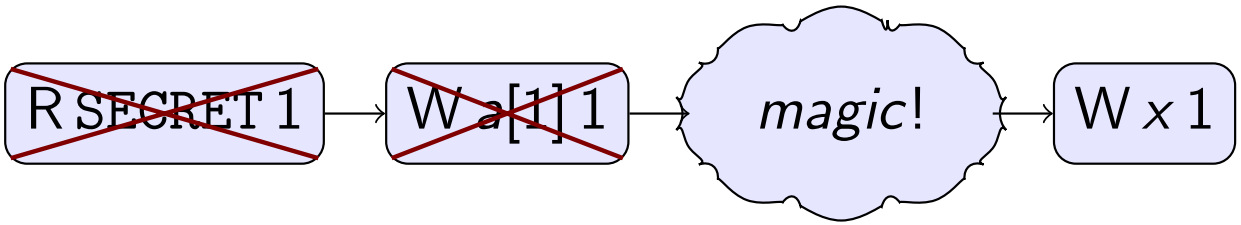
Attacker goal: learn if SECRET is 0 or 1.

Simplified Spectre

A very simplified Spectre attack:

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

with execution



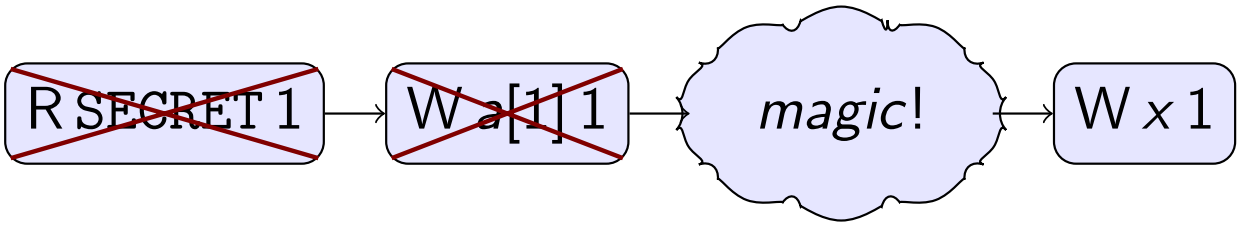
Information flow from SECRET to x

Simplified Spectre

A very simplified Spectre attack:

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

with execution



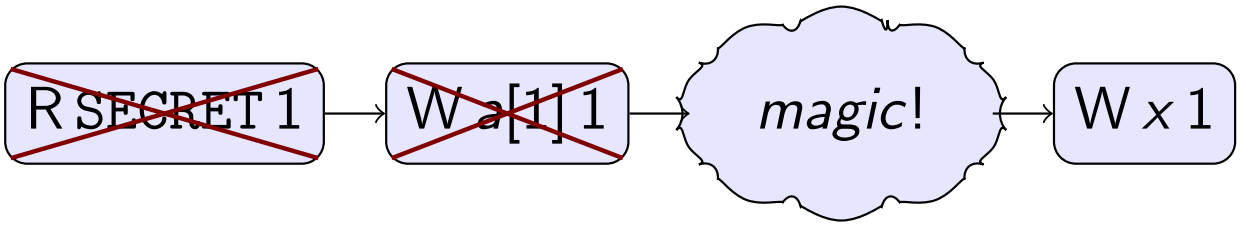
Information flow from SECRET to x,
if there's an implementation of "magic".

Simplified Spectre

A very simplified Spectre attack:

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

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.

Investigate similar attacks, e.g. on compiler optimizations.

TODO

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

Investigate similar attacks, e.g. on compiler optimizations.

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.

Investigate similar attacks, e.g. on compiler optimizations.

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.

Investigate similar attacks, e.g. on compiler optimizations.

Make it harder to implement $\texttt{if}\,(\texttt{touched}\,(x))$
(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).