Semantic approaches to hardware vulnerabilities

Sébastien MICHELLAND

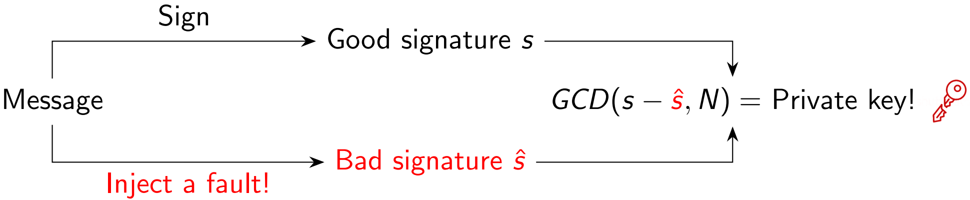
1st. year PhD, with Christophe DELEUZE and Laure GONNORD

sebastien.michelland@lcis.grenoble-inp.fr

Wait, what hardware vulnerabilities?

Fault injections!

Well-known attack on RSA [Bar-El et al.(2006)]:



Side channels!

Power, heat, timing data revealing the program's internal state.

These vulnerabilities break the program...

In a lot of ways, approximated by **fault models**:

- Corrupt data in registers
- Skip instructions
- Jump anywhere in the program
- Disrupt instruction decoding ◀ *Different!*
- Leak instructions' execution times [Winderix et al.(2021)] ◀ *Different!*

We really want to automate them away!

```
% clang prog.c -harden-faults=instruction-skip ...
```

… or, rather, they break the *semantics*.

Assembler with instruction skip transition:

$$\sigma \vdash i \rightarrow \sigma'$$

Initial state   Instruction   Final state

$$\frac{\sigma \vdash i_1 \rightarrow \sigma_{tmp} \qquad \sigma_{tmp} \vdash i_2 \rightarrow \sigma'}{\sigma \vdash (i_1;\ i_2) \rightarrow \sigma'}\ \textsc{Sequence} \qquad \frac{\sigma \vdash i_2 \rightarrow \sigma'}{\sigma \vdash (i_1;\ i_2) \rightarrow \sigma'}\ \textsc{Skip!}$$

Thinking semantics leads right into relevant questions!

- How many skips can occur? How frequently?
- What if we skip the terminator of a block?

They expose architectural details.

To model side channels, add a trace of observable leaked data:

$$\sigma \vdash i \rightarrow (\sigma', \tau)$$

Initial state Instruction Final state Trace

Ok, but how long is `mov r1, r2` in the first place?

Problem: faults expose architectural details.

A tuned semantics, unlike assembler, can capture these!

Yet, existing work apparently doesn't bridge this gap.

Assembler level:

- Friendly to reason with to design countermeasures
- Compiler can sometimes help with automation [Winderix et al.(2021)]
- But fault models are hitting an accuracy wall [Laurent et al.(2018)]

Meanwhile, microarchitecture level:

- Often models the entire circuit's RTL or even lower-level
- Focuses on finding the effects of faults

Yet, existing work apparently doesn't bridge this gap.

Assembler level:

- Friendly to reason with to design countermeasures
- Compiler can sometimes help with automation [Winderix et al.(2021)]
- But fault models are hitting an accuracy wall [Laurent et al.(2018)]

Can we capture a useful abstraction middle ground?

Meanwhile, microarchitecture level:

- Often models the entire circuit's RTL or even lower-level
- Focuses on finding the effects of faults

Semantics helps by being formal...

Let's protect against the instruction-latency side channel:

$\boxed{P_1}$

`branch`

add (2 cycles) | `mov` (1 cycle)

$\boxed{P_2}$

`branch`

`nop1` (1 cycle), add (2 cycles) | `mov` (1 cycle), `nop2` (2 cycles)

Theorem (P_2 is protected!)

- *P_2 computes the same result as P_1 (same $\sigma \rightarrow$ same σ')*
- *P_2 leaks no input-dependent timings (τ is the same for all σ)*

... by inviting hardware into countermeasure design...

Situation: one byte of code is skipped, offsetting all opcodes.

How do we deal with corrupted instructions at assembler level?

- Honestly: we don't.
- Protection code is corrupted too anyway.

Any serious option will have to deal with the low-level details:

- Add an integrity check in decoder?
- Use judicious opcodes so that a chosen register cannot be leaked during the 1-2 cycles until the error is detected?

... and is known to interface well with compiler optimization.

Options for hardening at a high-level:

1. Harden after compiling [Winderix et al.(2021)];
2. Avoid compiler interference with `-O0`;
3. Encode the counter-measure's properties in C/IR/ASM semantics to force the compiler to preserve them [Vu et al.(2020)].

Option #3 brings us right in sight of the altered semantics we've been studying!

Conclusion

- Semantics functions like a flexible abstraction level.
- We can use it to capture important architectural aspects and involve hardware in countermeasure design!
- We can also use it as a bridge to compiler automation and compiler optimization!

Conclusion

- Semantics functions like a flexible abstraction level.
- We can use it to capture important architectural aspects and involve hardware in countermeasure design!
- We can also use it as a bridge to compiler automation and compiler optimization!

Audience questions?

References I

H. Bar-El, Hamid Choukri, D. Naccache, Michael Tunstall, and C. Whelan. 2006.
The Sorcerer's Apprentice Guide to Fault Attacks.
(2006).
https://doi.org/10.1109/JPROC.2005.862424

Johan Laurent, V. Beroulle, C. Deleuze, Florian Pebay-Peyroula, and Athanasios Papadimitriou. 2018.
On the Importance of Analysing Microarchitecture for Accurate Software Fault Models.
(2018).
https://doi.org/10.1109/DSD.2018.00097

References II

S. Vu, Karine Heydemann, Arnaud de Grandmaison, and Albert Cohen. 2020.
Secure delivery of program properties through optimizing compilation.
(2020).
https://doi.org/10.1145/3377555.3377897

Hans Winderix, J. Mühlberg, and F. Piessens. 2021.
Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks.
(2021).
https://doi.org/10.1109/EuroSP51992.2021.00050