Leaky Abstractions

Alastair Reid

Strategic CAD Labs

UK Research Institute in Secure Hardware and Embedded Systems

Summer School, CSIT, Belfast, 20-21 July 2022.

About my research…

Most projects I work on take 5-10 years to appear in products

I have been at Intel for 6 months

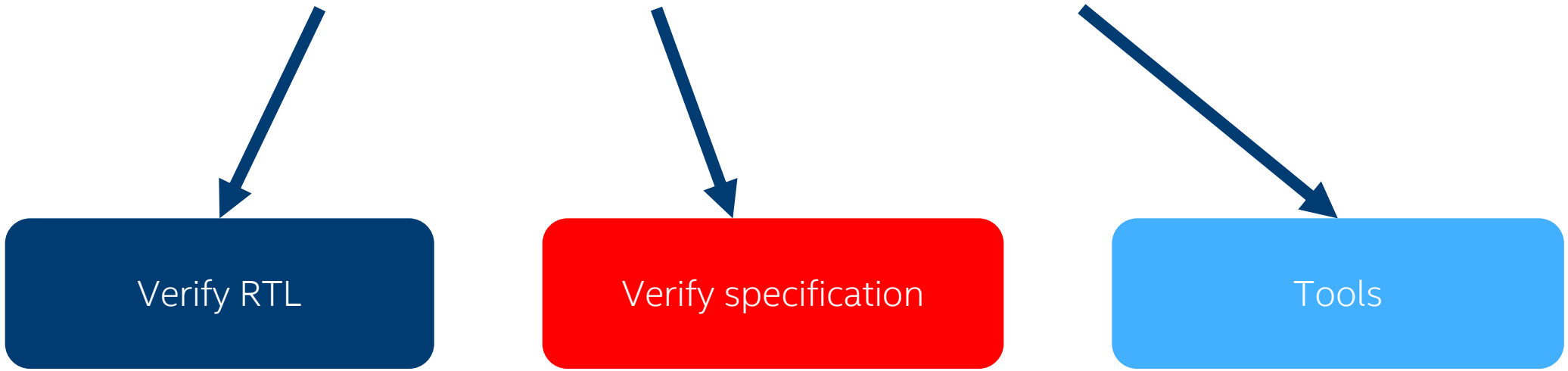
ISA specifications – Arm

Operation

```
bits(datasize) operand1 = X[n];
bits(datasize) operand2 = ShiftReg(m, shift_type, shift_amount);
bits(datasize) result;

result = operand1 AND operand2;
X[d] = result;
```

© Arm - AArch64 – DDI0487H



Alastair Reid

Researcher at Intel

Blog About Publications

Machine readable specifications at scale

There are lots of potential uses for machine readable specifications so you would think that every major real world artifact like long-lived hardware and software systems, protocols, languages, etc. would have a formal specification that is used by all teams extending the design, creating new implementations, testing/verifying the system, verifying code that uses the system, doing security analyses or any of the other potential uses. But, in practice, this is usually not true: most real world systems do not have a well tested, up to date, machine readable specification.

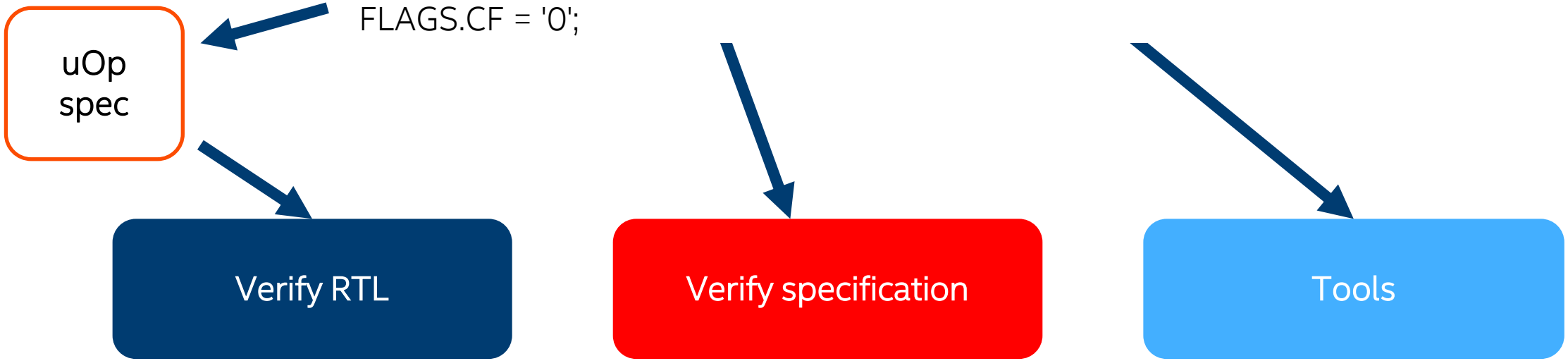
READ MORE

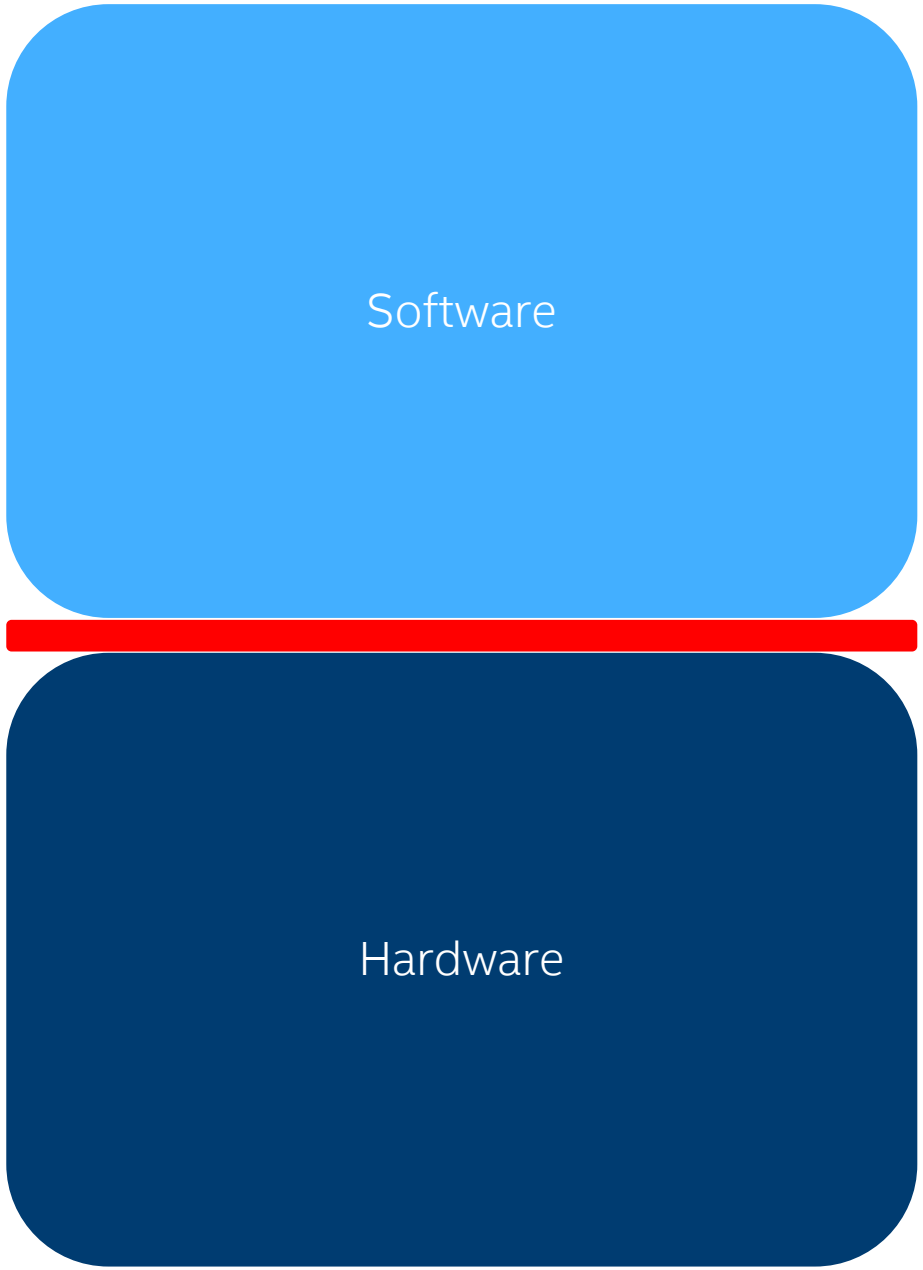
https://alastairreid.github.io/

ISA specifications – x86

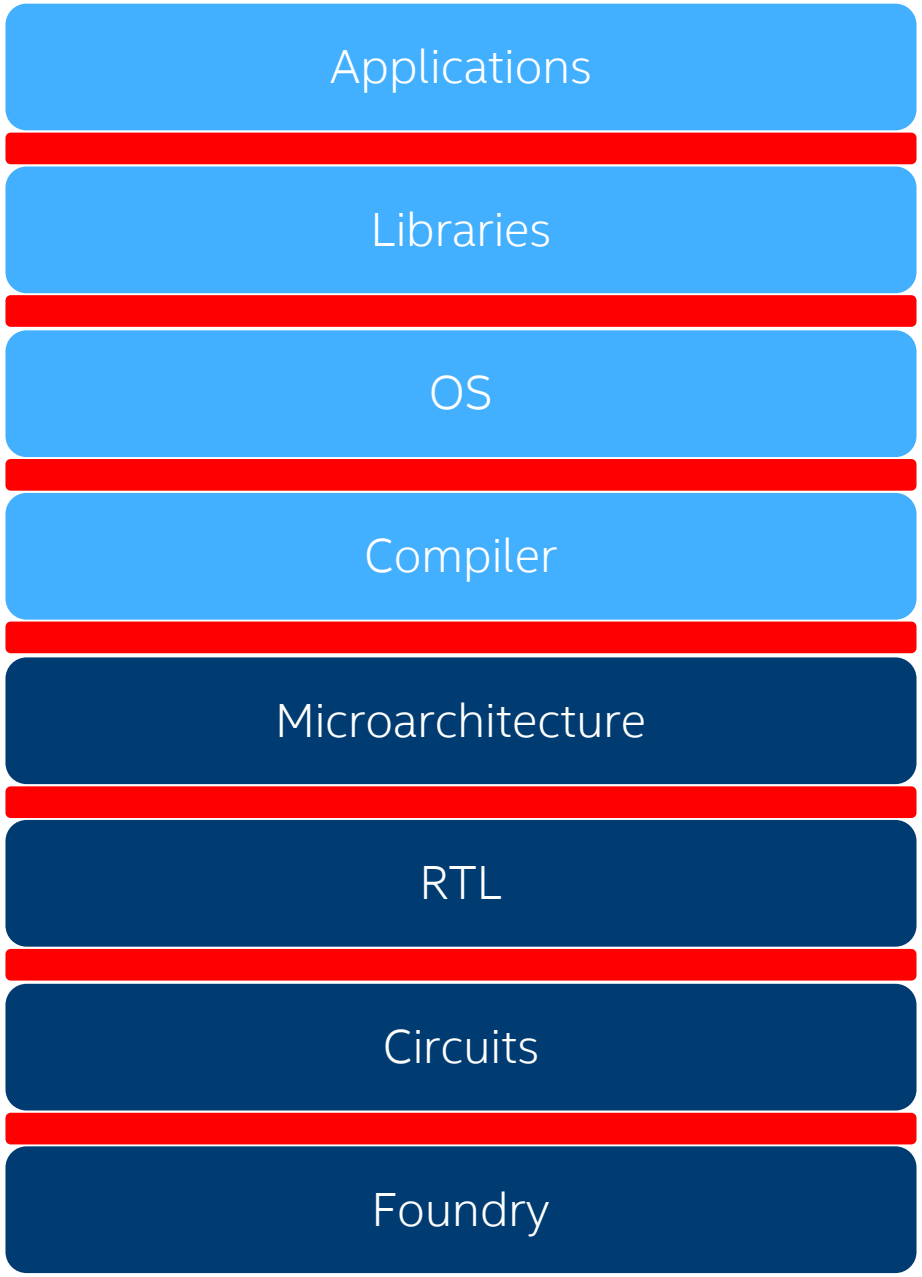
Operation

```
result = src1 AND src2;
FLAGS.OF = '0';
FLAGS.SF = result[datasize-1];
FLAGS.ZF = if IsZero(result) then '1' else '0';
FLAGS.AF = bit UNKNOWN;
FLAGS.PF = if ParityEven(result[0 +: 8]) then '1' else '0';
FLAGS.CF = '0';
```





ISA specification



man 3 (library calls)

man 2 (system calls)

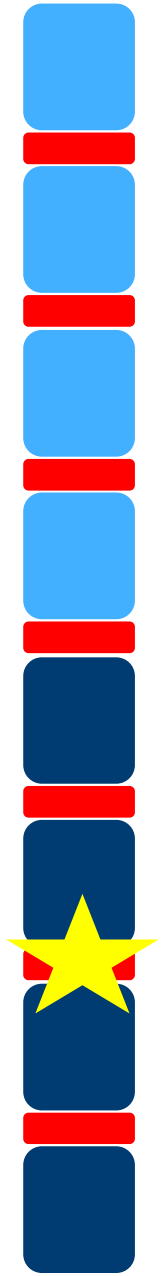
ISO C std

ISA specification

uArch spec

Digital logic

Layout (GDS II)



Outline

Example security problems (caused by leaky abstractions)

What are leaky abstractions?

What to do about them?

- Sketch of recent work by others

Example 1: Rowhammer

- Level: Circuits
- Manufacturing variation
- Data corruption / privilege escalation

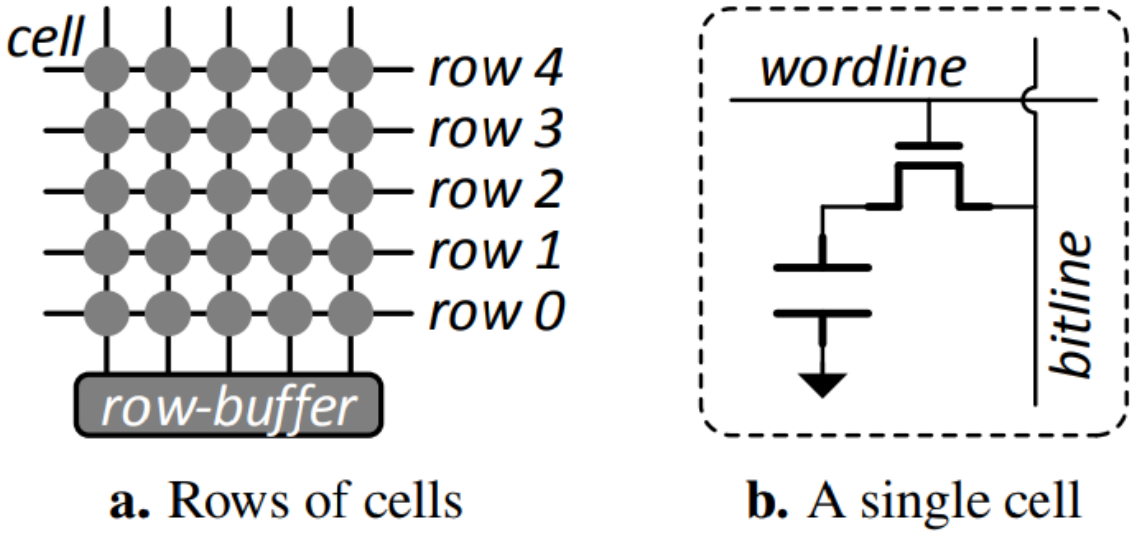


Figure 1. DRAM consists of cells

Kim et al., Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors, ISCA 2014. doi: 10.1145/2678373.2665726

- DRAM interface doesn't describe row layout, write disturbance, …

Exploiting Rowhammer

https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

- Level: Language

  (NaCl sandbox)

- Privilege escalation, …

- Known memory layout / contents

```
andl   $~31, %eax
addq   %r15, %rax
jmp    *%rax
```

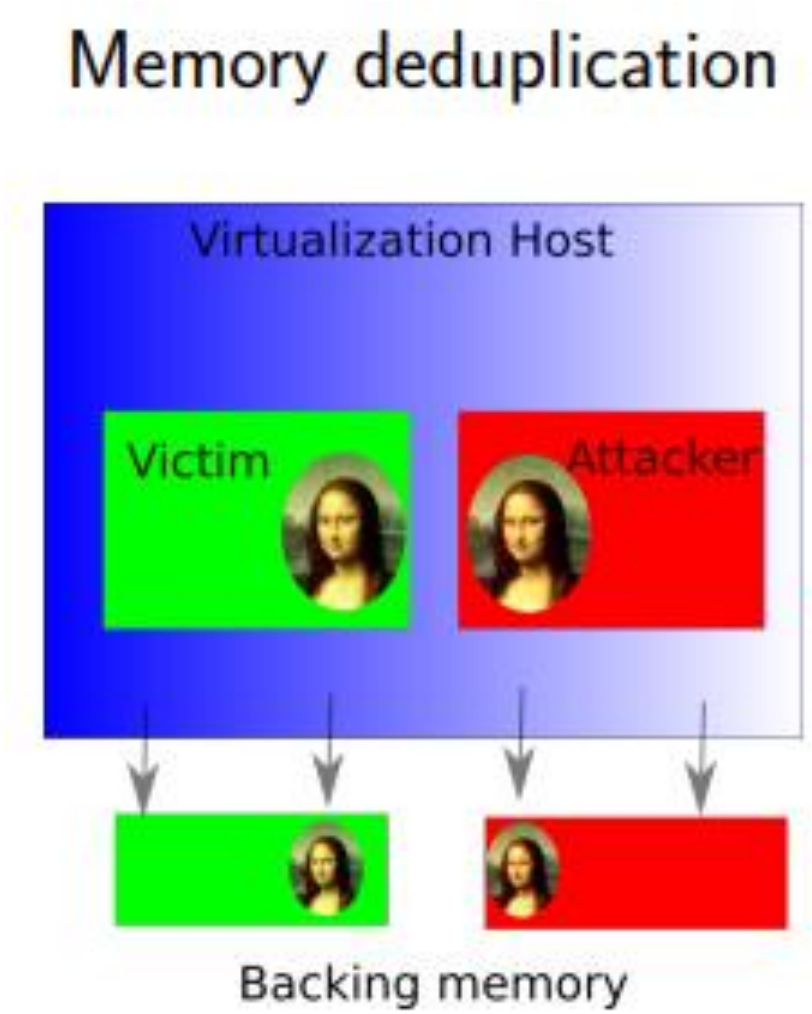
The exploit works by triggering bit flips in that code sequence. It knows how to exploit 13% of the possible bit flips.

…

For example, if a bit flip occurs in bit 0 of the register number in “jmp *%rax”, this morphs to “jmp *%rcx”, which is easily exploitable.

Example 2: Flip Feng Shui

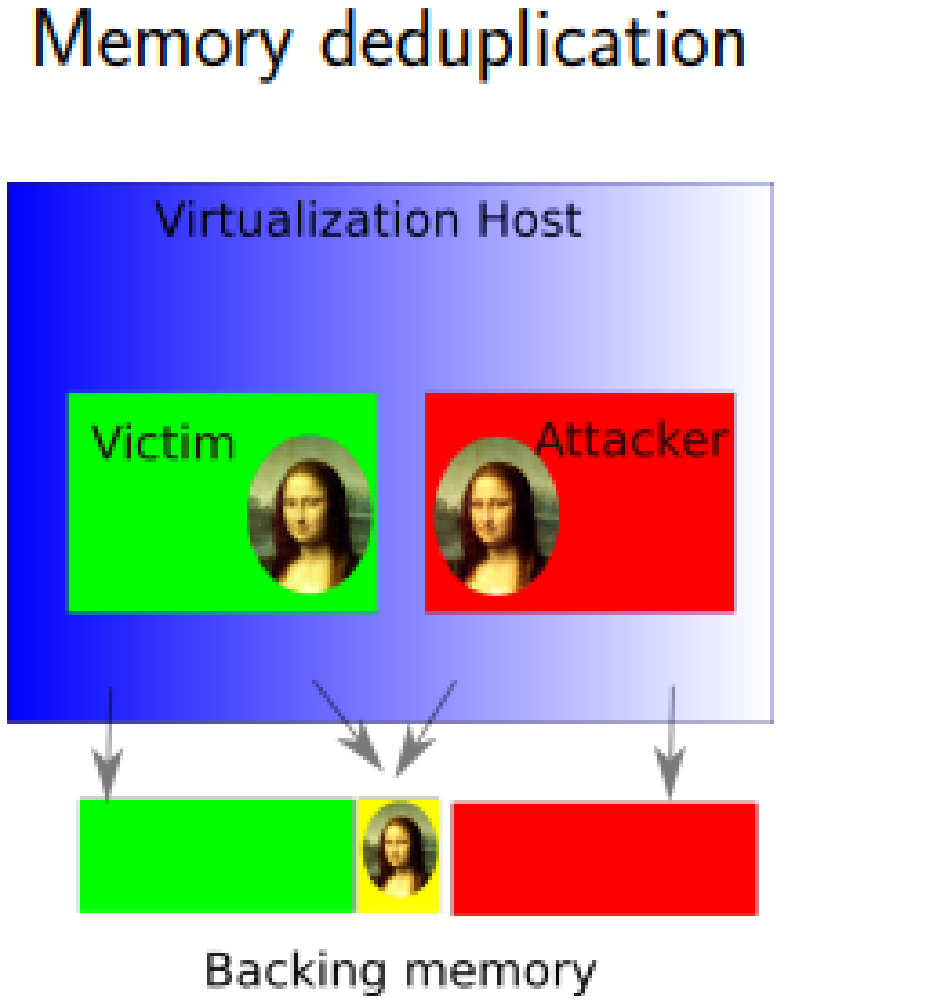
- OS / VM + OpenSSH
- Memory deduplication + Rowhammer
- Breaks RSA → OpenSSH compromise
- “Private” memory is sometimes shared



Razavi et al., Flip Feng Shui: hammering a needle in the software stack, SEC 2016. link

Example 2: Flip Feng Shui

- OS / VM + OpenSSH
- Memory deduplication + Rowhammer
- Breaks RSA → OpenSSH compromise
- “Private” memory is sometimes shared

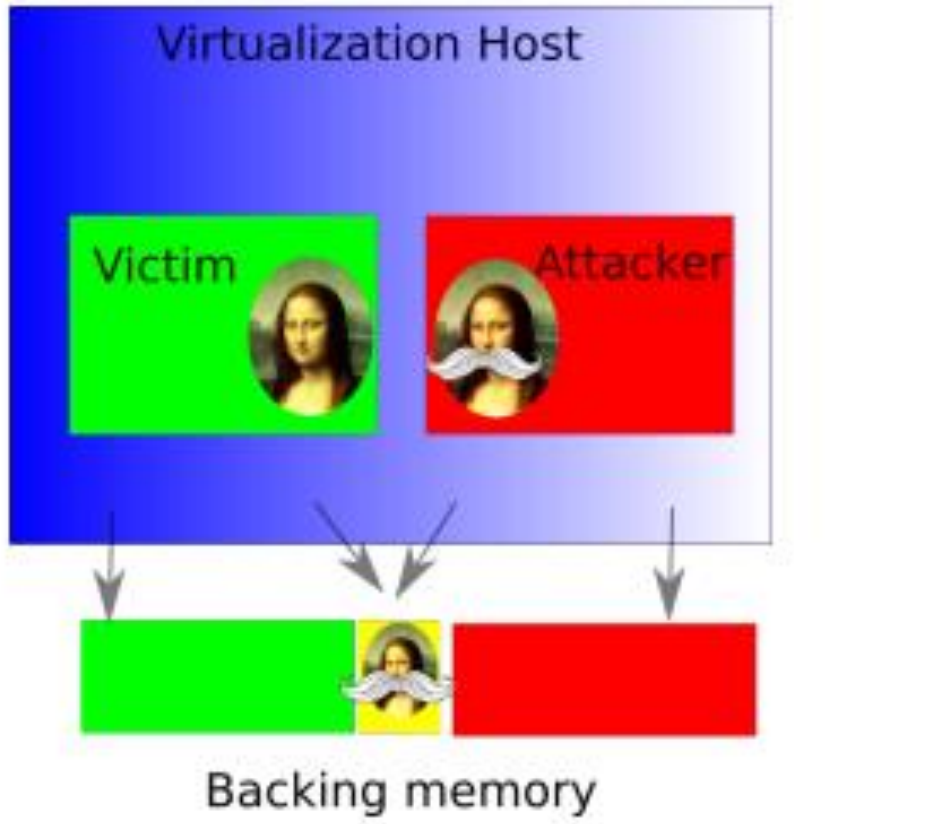


Razavi et al., Flip Feng Shui: hammering a needle in the software stack, SEC 2016. link

Example 2: Flip Feng Shui

- OS / VM + OpenSSH
- Memory deduplication + Rowhammer
- Breaks RSA → OpenSSH compromise
- “Private” memory is sometimes shared

Memory deduplication + Rowhammer = FFS



Razavi et al., Flip Feng Shui: hammering a needle in the software stack, SEC 2016. link

Example 3: Cache side-channels

- Level: microarchitecture
- Timing side-channel
- Data exfiltration

- GAP: ISA specs don't mention timing variation

Cache-timing attacks on AES

Daniel J. Bernstein *

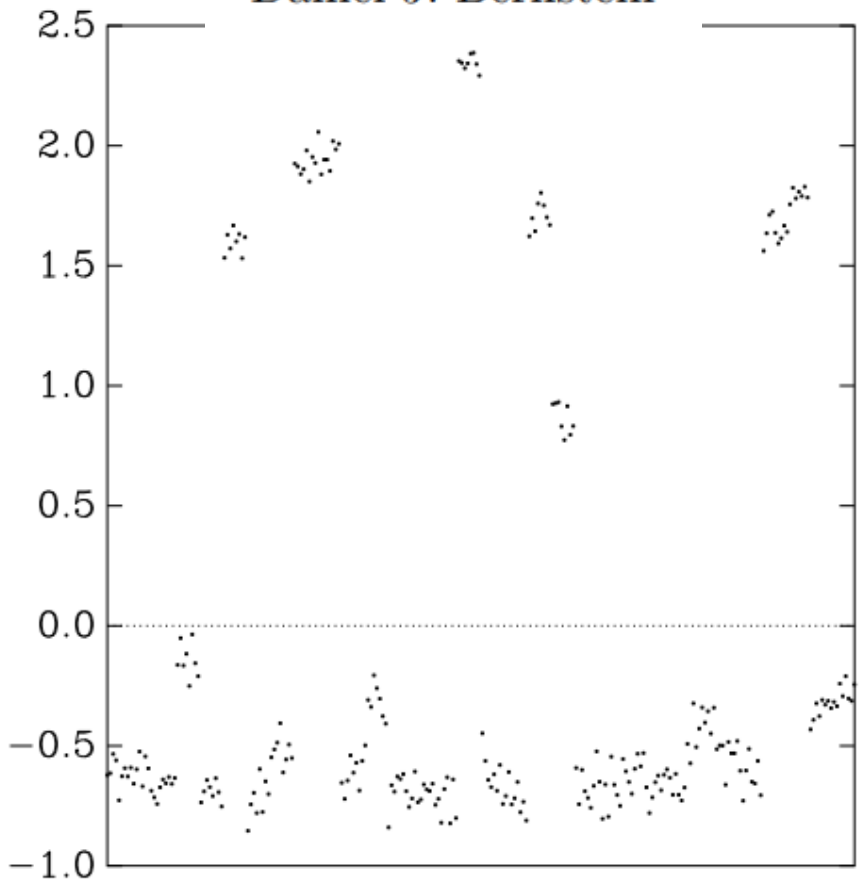


Fig. 5.2. How $n[5]$ affects OpenSSL AES timings for $k = 0$ on a Pentium III inside the targeted server. The horizontal axis is the choice of $n[5]$, from 0 through 255. The vertical axis is the average number of cycles for about 2^{22} 400-byte packets with that choice of $n[5]$, minus the average number of cycles for all choices of $n[5]$.

Example 4: Buffer overflow attacks

- Level: compiler
- No array bounds checks
- DoS, privilege escalation, …

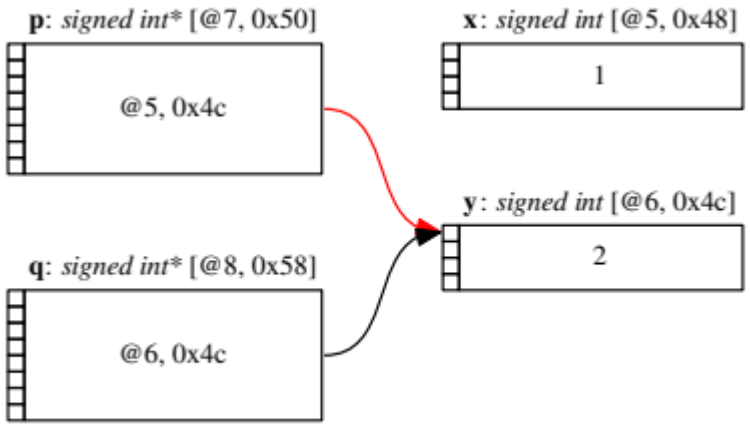
- GAP: C doesn't enforce object boundaries

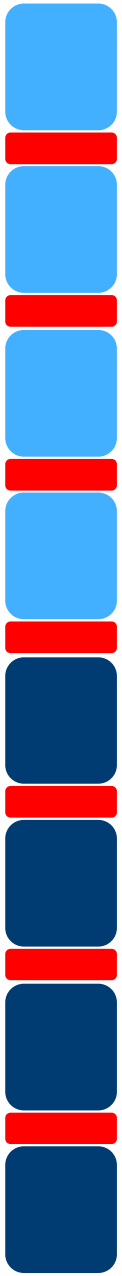
Exploring C Semantics and Pointer Provenance

KAYVAN MEMARIAN, University of Cambridge, UK

https://doi.org/10.1145/3290380

```
int y=2, x=1;
int main() {
  int *p = &x + 1;
  int *q = &y;
```





Flip Feng Shui

Flip Feng Shui

Buffer overflow

Rowhammer
ROP/JOP

Cache side-channels

Rowhammer

Power/clock glitching

Dopant level hardware trojans

What is a leaky abstraction?

An abstraction that fails to hide all the implementation details of the layer below.

What implementation details?

- Timing
- Power
- EMF
- Audio
- Memory layout
- …

Vulnerabilities

- power supply, clock signal, …
- signal crosstalk
- …

“Non-functional properties”

“All non-trivial abstractions, to some degree, are leaky.”

- Joel Spolsky, The law of leaky abstractions

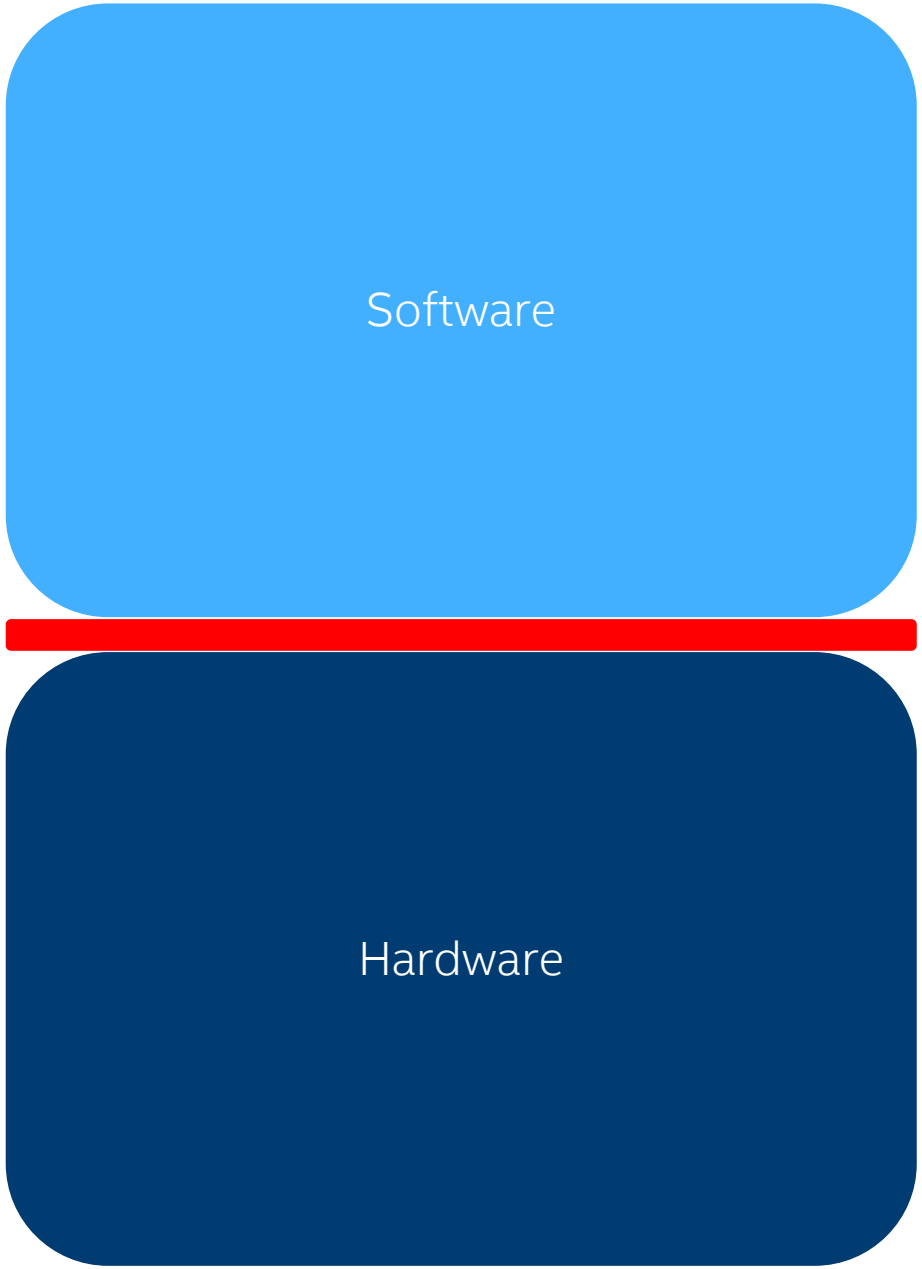
Leaky abstractions

- A rich source of security exploits
- Easy to spot once you start looking
- Impossible to eliminate

Just don’t use abstractions?

How many people deeply understand all of …

- LLVM?
- Assembly language?
- Out of order execution, branch prediction, prefetching, …
- Digital logic?
- Power distribution?
- CMOS design?
- Atomic layer deposition
- Quantum Electro Dynamics



ISA specification
=
Functional specification

MUL instruction

Operation

dest = src1 * src2;

Timing

Processor	Cycles	Secure?
A	Constant	Y
B	Constant	Y
C	Variable	N

LOAD instruction

Operation

dest = Mem[src1];

Timing

- Processor X with cache
  - Variable
- Processor Y with TCM + cache
  - Constant if access is to tightly-coupled memory
  - Variable otherwise

Challenges

1. What guarantees do we want to make?
2. Tools to reason about software?
3. Tools to show hardware satisfies guarantees?

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri*, Boris Köpf†, Jan Reineke‡, and Pepe Vila*
*IMDEA Software Institute †Microsoft Research ‡Saarland University

S&P 2021, doi: 10.1109/SP40001.2021.00036

Observations (labels)

STORE

$$\frac{p(a(\mathbf{pc})) = \mathbf{store}\ x, e \qquad \langle m, a\rangle \rightarrow \langle m', a'\rangle}{\langle m, a\rangle \xrightarrow{\texttt{store}\ \langle e\rangle(a)}_{\text{sct}} \langle m', a'\rangle}$$

Contracts (non-interference)

Definition 1 ($\{\!|\cdot|\!\} \vdash [\![\cdot]\!]$). A hardware semantics $\{\!|\cdot|\!\}$ *satisfies a contract* $[\![\cdot]\!]$ if, for all programs p and all initial architectural states σ, σ', if $[\![p]\!](\sigma) = [\![p]\!](\sigma')$, then $\{\!|p|\!\}(\sigma) = \{\!|p|\!\}(\sigma')$.

Bounded depth speculation

BRANCH

$$\frac{p(\sigma(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \quad \ell_{correct} = \begin{cases} \ell & \text{if } \sigma(x) = 0 \\ \sigma(\mathbf{pc}) + 1 & \text{otherwise} \end{cases} \quad \ell_{mispred} \in \{\ell, \sigma(\mathbf{pc}) + 1\} \setminus \ell_{correct} \quad \omega_{mispred} = \begin{cases} \mathrm{w} & \text{if } \omega = \infty \\ \omega & \text{otherwise} \end{cases}}{\langle \sigma, \omega + 1\rangle \cdot s \xrightarrow[\text{ct}]{\texttt{pc}\ \ell_{mispred}}_{\text{spec}} \langle \sigma[\mathbf{pc} \mapsto \ell_{mispred}], \omega_{mispred}\rangle \cdot \langle \sigma[\mathbf{pc} \mapsto \ell_{correct}], \omega\rangle \cdot s}$$

Secure Information Flow Verification with Mutable Dependent Types

Andrew Ferraiuolo, Weizhe Hua, Andrew C. Myers, G. Edward Suh

DAC 2017, doi: 10.1145/3061639.3062316

Label functions
(dependent types)

```
1   // mode_to_lb(0) = T, mode_to_lb(1) = U
2   reg {T} v; {T} trst; {U} untr;
3   reg {mode_to_lb(v)} shared;
4   ...
5       if (v == 1'b1) shared <= untrusted;
6       else           trusted <= shared;
7   ...
```

Security labels

Figure 3: Implicit downgrading example.

Interface	What to say?	Tools that use contract?	Tools to check contract?
ISA	25%	10%	5%

Recap

Systems built in layers separated by abstractions

that expose implementation details of layers below

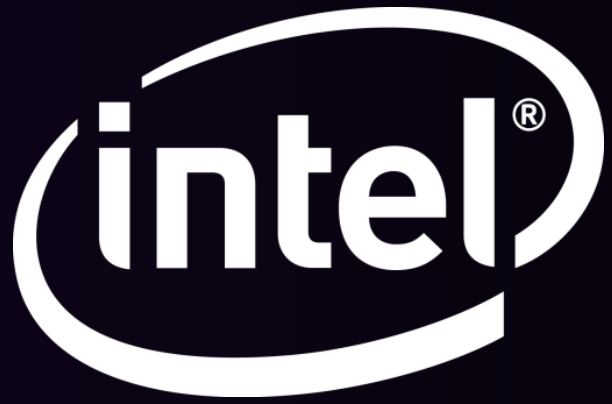
Many security issues are caused by leaky abstractions

We can't eliminate the leaks without removing the abstraction

Can we place useful bounds on the leaks?

“We demand rigidly defined areas of doubt and uncertainty”

- Vroomfondel in Douglas Adams’ “Hitchhikers Guide to the Galaxy”


intel®