Spectre Attacks and the Future of Security

Paul Kocher
(paul@paulkocher.com)

Stanford CS155
May 29, 2018

If the surgery proves unnecessary, we’ll revert your architectural state at no charge.
Intro

Stanford undergrad (class of ‘95)
- Biology major, planning to become a veterinarian
- Hobby: cryptography & computers
- Sophomore low on money → consulting project with Microsoft breaking CD-based SW distribution schemes
- Met Martin Hellman, attended Stanford Crypto Lunches, read papers, wrote code, worked at RSA in summers...
- Finished BS in biology (‘95)

Right place & time
- Prof. Martin Hellman retired -> referred interesting projects -> delayed vet school
- Founded Cryptography Research
- No business plan, no investors
Sought interesting projects

Projects (mostly with others):

- Protocols (incl. SSL v3.0 / TLS “_hooks”)
  - Side channel attacks
    - Timing attacks
    - Differential power analysis & countermeasures
- Numerous HW/ASIC projects
  - Pro-bono DES cracking machine (EFF funded)
  - Pay TV (evaluation → major design projects)
  - Anti-counterfeiting
- Risk management architectures
  - Renewability/Forensics: Blu-ray BD+, Vidity/SCSA...
- CryptoManager solutions (ASIC, manufacturing, service)
- Spectre Attacks
- Advisor/investor to start-ups

Retained by Taher ElGamal @ Netscape
Design philosophy: Simplicity, upgradability
Today: most widely-used crypto protocol

Devices timing, power, RF vary depending on computation
Variations correlate to crypto intermediates → break crypto
“Obvious in hindsight” but cryptographers != implementers
Filed patents on countermeasures (billions of chips impacted)
Complementary leadership team skills:

- Lots of Stanford talent, including:
  - Ben Jun (Stanford ’96) → technical execution
  - Kit Rodgers (Stanford ’96) → sales, marketing, business, ...

20 years of reinvestment + evolution

- Consulting → Licensing → Products → Solutions
- 1995 -> 2015: industry scaling
- Rambus acquisition in 2011 ($342.5M)
More than $1.5b in cryptocurrency stolen since early 2017

FBI warns Russians hacked hundreds of thousands of routers

At Least Three Billion Computer Chips Have the Spectre Security Hole
Companies are rushing out software fixes for Chipmageddon.

More than 2.5 billion records stolen or compromised globally in 2017

South Africa hacked: about 30 million ID numbers leaked

Cybercrime Damages $6 Trillion By 2021

FBI: 300,000 reported internet crimes cost victims at least $1.4 billion in 2017

Canadian banks warn: Hackers might have stolen data from nearly 90,000 customers
State machine for the security cycle

Vulnerability reported

Secure State
- No reported problems
- Product advertised as secure
- Lots of bugs to exploit
  - Developer = 😄😄😄
  - Customers = 😄😄😄
  - Attacker = 😄😄😄

Insecure State
- Frantically developing a fix
- Vulnerability in the press
- Victim acts very cautiously
  - Developer = 😞😞😞
  - Customers = 😞😞😞
  - Attacker = 😞😞😞

New release fixes issue
(+ adds 2 more)
Why panic when a big vulnerability is identified?

$P(\text{secure})$ fell from $\epsilon$ to 0.

but $\epsilon$ was usually negligible

Optimist’s security $= \left\lceil P(\text{secure}) \right\rceil$

fell from 100% to 0%
Kerckhoffs's principle (1883):

A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

1. Reliance on obscurity of what designer knows (e.g. algorithm)
   \[ P(\text{secure}) = 0 \]

2. Reliance on obscurity of undiscovered flaws
   \[ P(\text{secure}) < \epsilon \]

3. Significant probability of being secure
   \[ 1\% < P(\text{secure}) < 99\% \]

4. Security which anyone can easily verify
   \[ P(\text{secure}) = 100\% - \epsilon \]

Achieved by some components (AES, seL4) and simple HW designs

Too hard?

The holy grail...
Attacker perspective:

- Complexity → Bugs
- Devices → Targets
- Economic activity → $
abstraction is a technique for hiding complexity of computer systems.

It works by establishing a level of simplicity on which a person interacts with the system, suppressing the more complex details below the current level.

https://en.wikipedia.org/wiki/Abstraction_(computer_science)
Abstraction creates security challenges

- Dependencies & assumptions
- Security goals
- Clouds
- Abstractions

Exponential growth

- People
- Business objectives
- Nation states
- Society

Security-critical details hidden in layers
Needs of distant layers unclear
People specialize then miss big picture
Economics don’t fund adequate investment
Risks in other layers deter improvements
Changes aren’t communicated across layers
Are there any security implications from speculative execution?

-- Mike Hamburg
Addicted to speed

Performance drives CPU purchases

Single-thread speed gains require getting **more done per clock cycle**
- Memory latency is slow and not improving much
- Clock rates are maxed out: Pentium 4 reached 3.8 GHz in 2004

How to do more per clock?
- Reducing memory delays → Caches
- Working during delays → Speculative execution
# Memory caches

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory.

<table>
<thead>
<tr>
<th>CPU</th>
<th>MEMORY CACHE</th>
<th>MAIN MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sends address, Receives data</td>
<td></td>
<td>Big, slow e.g. 16GB SDRAM</td>
</tr>
</tbody>
</table>

- **Addr**: 2A1C0700, **Data**: 9EC3 DA EE B7 D3...
- **Addr**: 132E1340, **Data**: AC 99 17 8F 44 09...
- **Addr**: 132E1340, **Data**: AC 99 17 8F 44 09...

### Memory Cache Table

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280 31C6F4C0 339DD740 614F8480</td>
<td>B5 F5 80 21 E3 2C.. 9A DA 59 11 48 F2.. C7 D7 A0 86 67 18.. 17 4C 59 B8 58 A7..</td>
</tr>
<tr>
<td>1</td>
<td>71685100 132A4880 2A1C0700 C017E9C0</td>
<td>27 BD 5D 2E 84 29.. 30 B2 8F 27 05 9C.. 9E C3 DA EE B7 D9.. D1 76 16 54 51 5B..</td>
</tr>
<tr>
<td>2</td>
<td>311956C0 002D47C0 91507E80 55190400</td>
<td>0A 55 47 82 86 4E.. C4 15 4D 78 B5 C4.. 60 D0 2C DD 78 14.. DF 66 E9 D0 11 43..</td>
</tr>
<tr>
<td>3</td>
<td>9B27F8C0 8E771100 A001FB40 132E1340</td>
<td>84 A0 7F C7 4E BC.. 3B 0B 20 0C DB 58.. 29 D9 F5 6A 72 50.. AC 99 17 8F 44 09..</td>
</tr>
<tr>
<td>4</td>
<td>661BE980 BA0CDB40 89E92C00 090F9C40</td>
<td>35 11 4A E0 2E F1.. B0 FC 5A 20 D0 7F.. 1C 50 A4 F8 EB 6F.. BB 71 ED 16 07 1F..</td>
</tr>
</tbody>
</table>

| Address: 132E1340 | Data: AC 99 17 8F 44 09.. |

Reads change system state:
- Next read to **newly-cached** location is faster
- Next read to **evicted** location is slower

- **CPU** sends address, **CPU** receives data.
- **CPU** sends address, **CPU** receives data.
- **CPU** sends address, **CPU** receives data.

\[h(\text{addr})\] to map to cache set

\[\text{Fast} \quad \text{Slow} \quad \text{Fast} \]
Speculative execution

Correct result of running instructions = the result of performing instructions in-order

CPUs may run instructions out-of-order if this doesn’t affect result

- Example:
  
  a ← constant  
  b ← slow_to_obtain  
  c ← f(a) // start before b finishes

CPUs can also guess likely program path and do speculative execution

- Example:
  
  if (uncached_value_usually_1 == 1)  
  compute_something()

- Branch predictor guesses that if() will be ‘true’ (based on prior history)
- Starts executing compute_something() speculatively -- but doesn’t save changes
- When value arrives from memory, if() can be evaluated definitively -- check if guess was correct:
  - Correct: Save speculative work – performance gain
  - Incorrect: Discard speculative work
Speculative Execution

CPU regularly performs incorrect calculations, then deletes mistakes

Architectural Guarantee

Register values eventually match result of in-order execution

Software security assumes the CPU runs instructions correctly...

Does making + discarding mistakes violate this assumption?

Set up the conditions so the processor will make a desired mistake

Mistake leaks sensitive data into a covert channel (e.g. state of the cache)

Fetch the sensitive data from the covert channel
Conditional branch (Variant 1) attack

```c
if (x < array1_size)
    y = array2[array1[x]*4096];
```

Assume code in kernel API, where unsigned int `x` comes from untrusted caller

Execution *without* speculation is safe
- CPU will not evaluate `array2[array1[x]*4096]` unless `x < array1_size`

What about with speculative execution?
Conditional branch (Variant 1) attack

if \( x < \text{array1\_size} \)
\[
y = \text{array2[array1[x]\_4096]};
\]

Before attack:
- Train branch predictor to expect if() is true (e.g. call with \( x < \text{array1\_size} \))
- Evict \text{array1\_size} and \text{array2[]} from cache

```
Memory & Cache Status
array1\_size = 00000008

Memory at array1 base address:
8 bytes of data (value doesn’t matter)
[... lots of memory up to array1 base+N...]
09 F1 98 CC 90... (something secret)

array2[ 0*4096]
array2[ 1*4096]
array2[ 2*4096]
array2[ 3*4096]
array2[ 4*4096]
array2[ 5*4096]
array2[ 6*4096]
array2[ 7*4096]
array2[ 8*4096]
array2[ 9*4096]
array2[10*4096]
array2[11*4096]
...
```
Conditional branch (Variant 1) attack

if (x < array1_size)
    y = array2[array1[x]*4096];

Attacker calls victim with x=N (where N > 8)

- Speculative exec while waiting for array1_size
  - Predict that if() is true
  - Read address (array1 base + x) w/ out-of-bounds x
  - Read returns secret byte = 09 (fast – in cache)

Memory & Cache Status

array1_size = 00000008

Memory at array1 base address:
  8 bytes of data (value doesn’t matter)
  [... lots of memory up to array1 base+N...]
  09 F1 98 CC 90... (something secret)

array2[0*4096]
array2[1*4096]
array2[2*4096]
array2[3*4096]
array2[4*4096]
array2[5*4096]
array2[6*4096]
array2[7*4096]
array2[8*4096]
array2[9*4096]
array2[10*4096]
array2[11*4096]

Contents don’t matter
only care about cache status
Uncached
Cached
if (x < array1_size)
    y = array2[array1[x]*4096];

Attacker calls victim with x=N (where N > 8)
- Speculative exec while waiting for array1_size
  - Predict that if() is true
  - Read address (array1 base + x) w/ out-of-bounds x
  - Read returns secret byte = 09 (fast – in cache)
  - Request memory at (array2 base + 09*4096)
  - Brings array2[09*4096] into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker measures read time for array2[i*4096]
- Read for i=09 is fast (cached), revealing secret byte
- Repeat with many x (eg ~10KB/s)
Violating JavaScript’s sandbox

Browsers run JavaScript from untrusted websites

- JIT compiler inserts safety checks, including bounds checks on array accesses

Speculative execution can blast through safety checks...

```javascript
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = ((index * TABLE1_STRIDE)|0) & (TABLE1_BYTES-1)|0;
  localJunk ^= probeTable[index|0]|0;
}
```

- `index` will be in-bounds on training passes, and out-of-bounds on attack passes
- JIT thinks this check ensures `index < length`, so it omits bounds check in next line. Separate code evicts `length` for attack passes
- Do the out-of-bounds read on attack passes!
- Need to use the result so the operations aren’t optimized away
- “|0” is a JS optimizer trick (makes result an integer)
- Keeps the JIT from adding unwanted bounds checks on the next line
- Leak out-of-bounds read result into cache state!
- 4096 bytes = memory page size
- Can evict `length/probeTable` from JavaScript (easy), timing tricks to detect newly-cached location in `probeTable`
Conditional branches: 2 destinations (fork in the road)

Indirect branches: Can go anywhere (“jmp [rax]”)

- If destination is delayed, CPU guesses and proceeds speculatively
- Find an indirect jmp with attacker controlled register(s)
  ... then cause mispredict to a ‘gadget’ that leaks memory at address in register

Attack steps

- **Mistrain** branch prediction/BTB so speculative execution will go to gadget
- **Evict** or flush destination address from the cache to ensure long duration speculative execution
- **Execute** victim so it runs gadget speculatively
- **Detect** change in cache state to determine memory data
- **Repeat** for more bytes
Mitigations

Mitigation. noun. “The action of reducing the severity, seriousness, or painfulness of something”

Not necessarily a complete solution
**Variant 1 Mitigation:** Speculation-barrier instruction (e.g. **LFENCE**)

- Idea: Software developers insert barrier on all vulnerable code paths in software
- Efficient: No performance impact on benchmarks or other legacy software
- Simple & effective *(for CPU developer)*

---

**Abstraction boundaries**

**Insert LFENCEs manually?**
- Often millions of control flow paths
- Too confusing - speculation runs 188++ instructions, crosses modules
- Too risky – miss one and attacker can read entire process memory

**Put LFENCES everywhere?**
- Abysmal performance - **LFENCE** is very slow (+ no tools)
- Not in binary libraries, compiler-created code patterns

**Insert by smart compiler?**
- Protect all potentially-exploitable patterns = too slow
  - Compilers judged by performance, not security
  - Protect only known-bad bad patterns = unsafe
    - Microsoft Visual C/C++ /Qspectre unsafe for 13 of 15 tests
      - [https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html](https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html)

**Transfers blame after breach (CPU -> SW)**

“**you should have put an LFENCE there**” -> “**Fixed**”
Mitigations: Indirect branch variant

Intel/AMD (x86):
- New MSRs created via microcode
  - Low-level control over branch target buffer (O/S only)
  - Performance impact – limited use
- Retpoline proposal from Google
  - Messy hack -- replace indirect jumps with construction that resists indirect branch poisoning on Haswell
  - Microcode updates to make retpoline safe on Skylake & beyond

ARM: No generic mitigation option
- Fast ARM CPUs broadly impacted, e.g. Cortex-A9, A15, A17, A57, A72, A73, A75...
- Often no mitigation, but on some chips software may be able to invalidate/disable branch predictor (with “non-trivial performance impact”)
  - See: https://developer.arm.com/support/security-update/download-the-whitepaper

Mitigations are messy (for all Spectre variants + Meltdown)
- Software must deal with microarchitectural complexity
- Mitigations for all variants are really hard to test

“All of this is pure garbage”
-- Linus Torvolds
https://lkml.org/lkml/2018/1/21/192
DOOM with only MOV instructions

Only MOV instructions
- No branch instructions
- One big loop with exception at the end to trigger restart

Sub-optimal performance
- One frame every ~7 hours

Oops! Variant 4: Speculative store bypass
At Least Three Billion Computer Chips Have the Spectre Security Hole

Companies are rushing out software fixes for Chipmageddon.

Intel Faces Scrutiny as Questions Swirl Over Chip Security

Silicon melts

Spectre and Meltdown prompt tech industry soul-searching

"AMD is not susceptible to all three variants. [...] there is a near zero risk to AMD processors at this time."

‘Optimist’s security’ fell from 100% to 0% 😞
Because of software bugs, computer security was in a dire situation

Spectre doesn’t change the magnitude of the risk, but adds to the mess
- Highlights risks in layers with limited mitigation tools
- Complexity of fixes -> new risks
- Psychology of unfixed vulnerabilities
Is Spectre a bug?

Everything complies with the architecture specs and CPU design textbooks

- Branch predictor is learning from history, as expected
- Speculative execution unwinds architectural state correctly
- Reads are fetching data the victim is allowed to read (unlike Meltdown)
- Caches are allowed to hold state
- Covert channels known to exist (+ very hard to eliminate)

**Architecture ⇔ Software security gap**

- CPU architecture guarantees are **insufficient for security**
- Under-specified -> software & HW developers make different assumptions
  - Computation time, debug counters, timing effects on other threads, analog effects (power, RF, heat...),
    errors/glitches (RowHammer, clkscrew...)
- No way to know if code is secure on today’s chips
- Future chips may be completely different
“The reality is there are probably other things out there like [Spectre] that have been deemed safe for years. Somebody whose mind is sufficiently warped toward thinking about security threats may find other ways to exploit systems which had otherwise been considered completely safe.”

– Simon Segars (CEO, Arm Holdings)
Failure to acknowledge trade-offs
  - Crazy to use same hardware & operating systems for video games & wire transfers...

Bifurcate into faster vs. safer flavors
  - ‘Safer’ = less complex (not just a different mode like TrustZone, Intel SGX)
  - Accept inefficiency (performance, developer time...)

Faster & safer can co-exist
  - Common for performance/power: ARM’s big.LITTLE architecture, GPU, TPU
  - Security examples: Pay TV smart cards, Rambus CryptoManager cores, Apple Enclave...
Ethics: Responsible disclosure & hardware

Responsible disclosure evolved for software, now hardware...

- Notify vendor
- Embargo while vendor fixes issue
- Public disclosure

Far too many: CPU, chip, PC, device, O/S, hypervisor, open source, clouds, app...

Far too long: can’t replace hardware
(and ARM/etc. chips with affected CPUs will keep being made for decades)

Far too messy: marketing, legal (32+ lawsuits), investor relations, academics...

I’m 0-for-2: Spectre + Differential Power Analysis (DPA)
- (# people with need-to-know) >> (# who can keep a secret)
- Chaotic embargos ended suddenly due to leaks
- Most impacted organizations got no advance warning = effective 0-day

AMD is not susceptible to all three variants. [...] there is a near zero risk to AMD processors at this time.”
Problems = Opportunities
Long history of over-optimism
Attacker perspective:

Complexity $\rightarrow$ Bugs

Devices $\rightarrow$ Targets

Economic activity $\rightarrow$ $
## Scaling will continue

<table>
<thead>
<tr>
<th></th>
<th>Traditional</th>
<th>IoT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of vendors</td>
<td>Few</td>
<td>1000’s+</td>
</tr>
<tr>
<td>Vendor security expertise</td>
<td>World-class</td>
<td>Low/none</td>
</tr>
<tr>
<td>User attention per device to security</td>
<td>High</td>
<td>None</td>
</tr>
<tr>
<td>On-device security analysis tools</td>
<td>Advanced</td>
<td>None</td>
</tr>
<tr>
<td>Network-based security capabilities</td>
<td>Advanced</td>
<td>Limited/none</td>
</tr>
<tr>
<td>Can cause harm in physical world</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Funding for security maintenance</td>
<td>Upgrade/sub</td>
<td>None</td>
</tr>
<tr>
<td>Typical operational life</td>
<td>&lt;10 years</td>
<td>20-100 years</td>
</tr>
</tbody>
</table>

Machine learning -> complexity beyond what humans can create
Example:

10^{-4} defects/element
10^7 critical elements
P(0 defects) \approx 10^{-434}

Trying harder won’t work
Need (10^6+) reduction in defects
... to address today’s situation

Need new perspectives, approaches

Why Diverse Teams Are Smarter
https://hbr.org/2016/11/why-diverse-teams-are-smarter
Scaling Security

Technology scales faster than other aspects of the economy.

“Software is eating the world.”

-- Marc Andreessen

Insecurity’s costs scale faster than technology’s benefits.

2X complex < 2X useful
2X complex ≥ 2X risk

Security will be the dominant force shaping technology.

Security is eating software.
Past: Performance dominated the economics of technology

Today: Costs of insecurity > Value of performance gains

$10^{12}$-$10^{13}$ year$^1$

Magnitude: $10^{11}$/year$^2$

Technical challenge
Engineering stronger & more resilient systems

Leadership challenge
Today's leaders developed during 50+ years where performance > security

2 My estimate. For reference, Intel's entire 2017 revenues were $62.8B, ARM <$2B.
# Why work on security problems?

<table>
<thead>
<tr>
<th>Traditional fields</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well-studied (often intractable) problems</td>
<td>Impact on big, dynamic, messy, exciting problems</td>
</tr>
<tr>
<td>Low wage pressure and/or barriers to entry</td>
<td>Labor shortages: high wages, encouragement to entry</td>
</tr>
<tr>
<td>Senior people dominate</td>
<td>New entrants lead: new perspectives, skills</td>
</tr>
<tr>
<td>Economics driven by efficiency, cost</td>
<td>Innovation-driven</td>
</tr>
<tr>
<td></td>
<td>+ Quality focus: Commit before outcome known</td>
</tr>
</tbody>
</table>
Thank you!

My email: paul@paulkocher.com

Predictions:

• At least one of you will (co)found a $1B+ security company
• Most of your careers will focus heavily on data security
  { data security } + { IR, law, poly sci, economics, medicine, EE, math, biology... }