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 "🔒")
  - Retained by Taher ElGamal @ Netscape
  - Design philosophy: Simplicity, upgradability
  - Today: most widely-used crypto protocol
- Side channel attacks
  - Timing attacks
  - Differential power analysis & countermeasures
  - 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)
- Numerous HW/ASIC projects
  - Pro-bono DES cracking machine (EFF funded)
  - Pay TV (evaluation → major design projects)
  - Anti-counterfeiting
- Risk management architectures
  - Revocation: Co-founded ValiCert (IPO 2001, acquired 2003)
  - Renewability/Forensics: Blu-ray BD+ , Vidity/SCSA…
- CryptoManager solutions (ASIC, manufacturing, service)
- Spectre Attacks
- Advisor/investor to start-ups

Business evolution

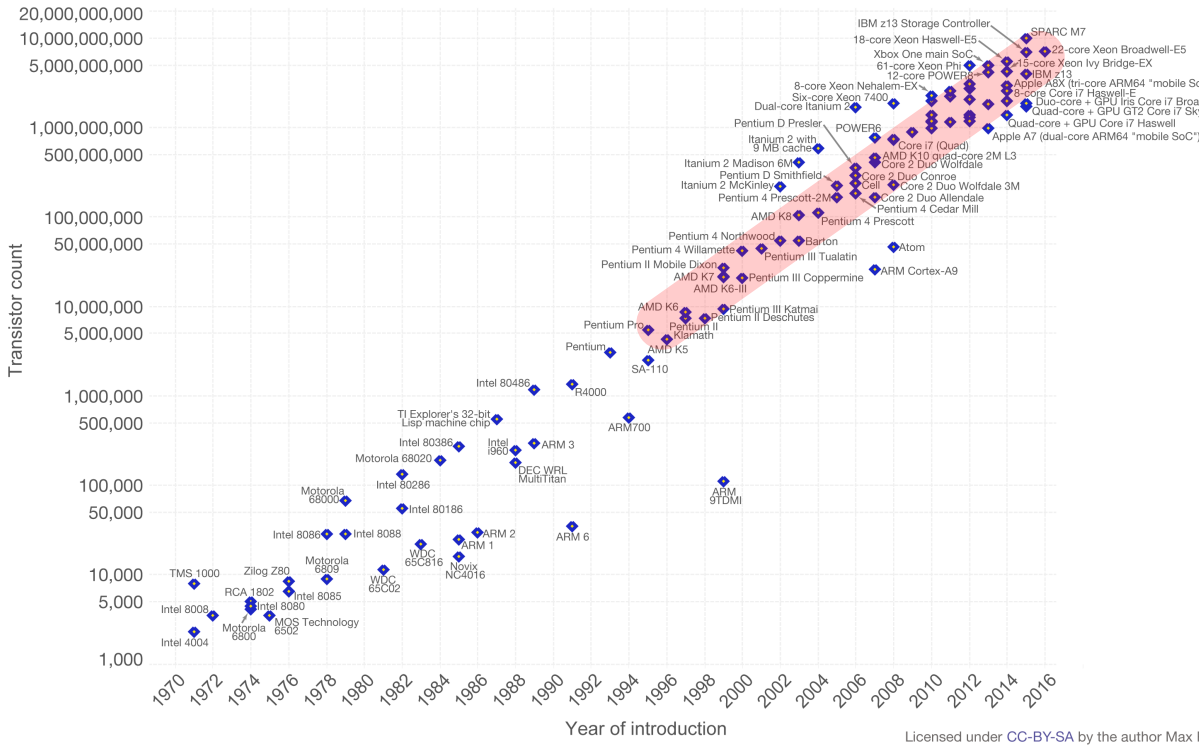
Complementary leadership team skills:

- 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)

Moore’s Law – The number of transistors on integrated circuit chips (1971-2016)



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

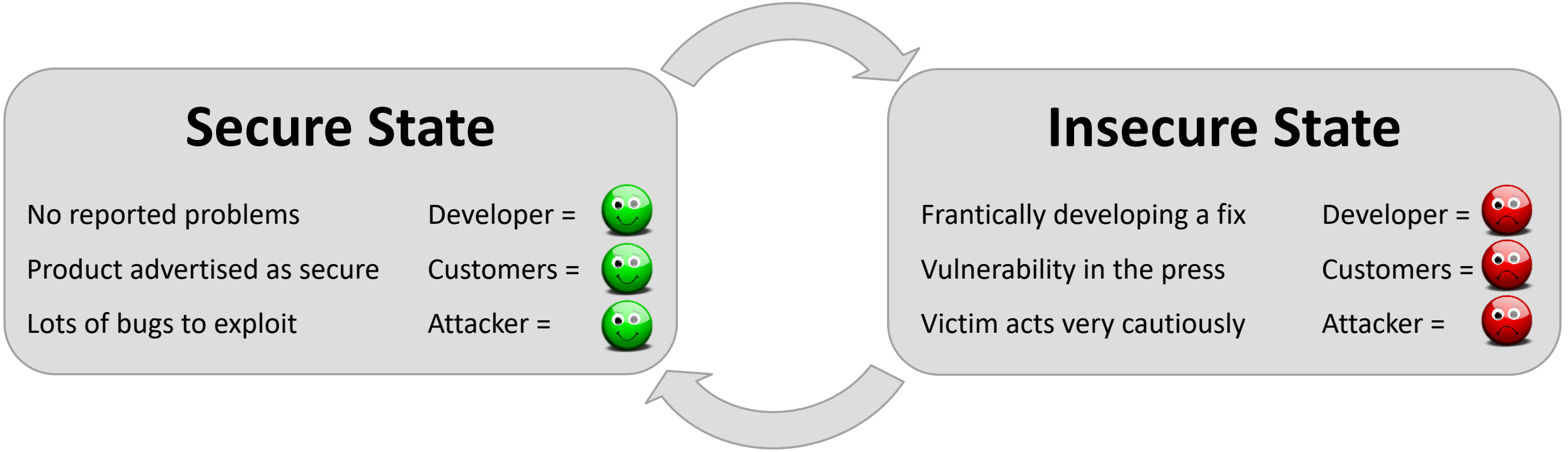
Cybercrime Damages $6 Trillion By 2021

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



New release fixes issue

(+ adds 2 more)

??!

Why panic when a big vulnerability is identified?

P(secure) fell from ϵ to 0.

→ but ϵ was usually negligible

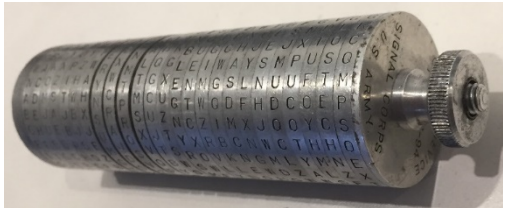
Optimist's security = $\lceil$ P(secure) $\rceil$

→ 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(secure) = 0

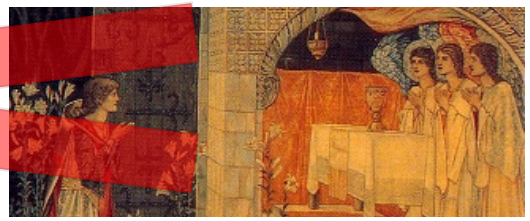


2. Reliance on obscurity of undiscovered flaws
   P(secure) < ϵ

3. Significant probability of being secure
   1% < P(secure) < 99%

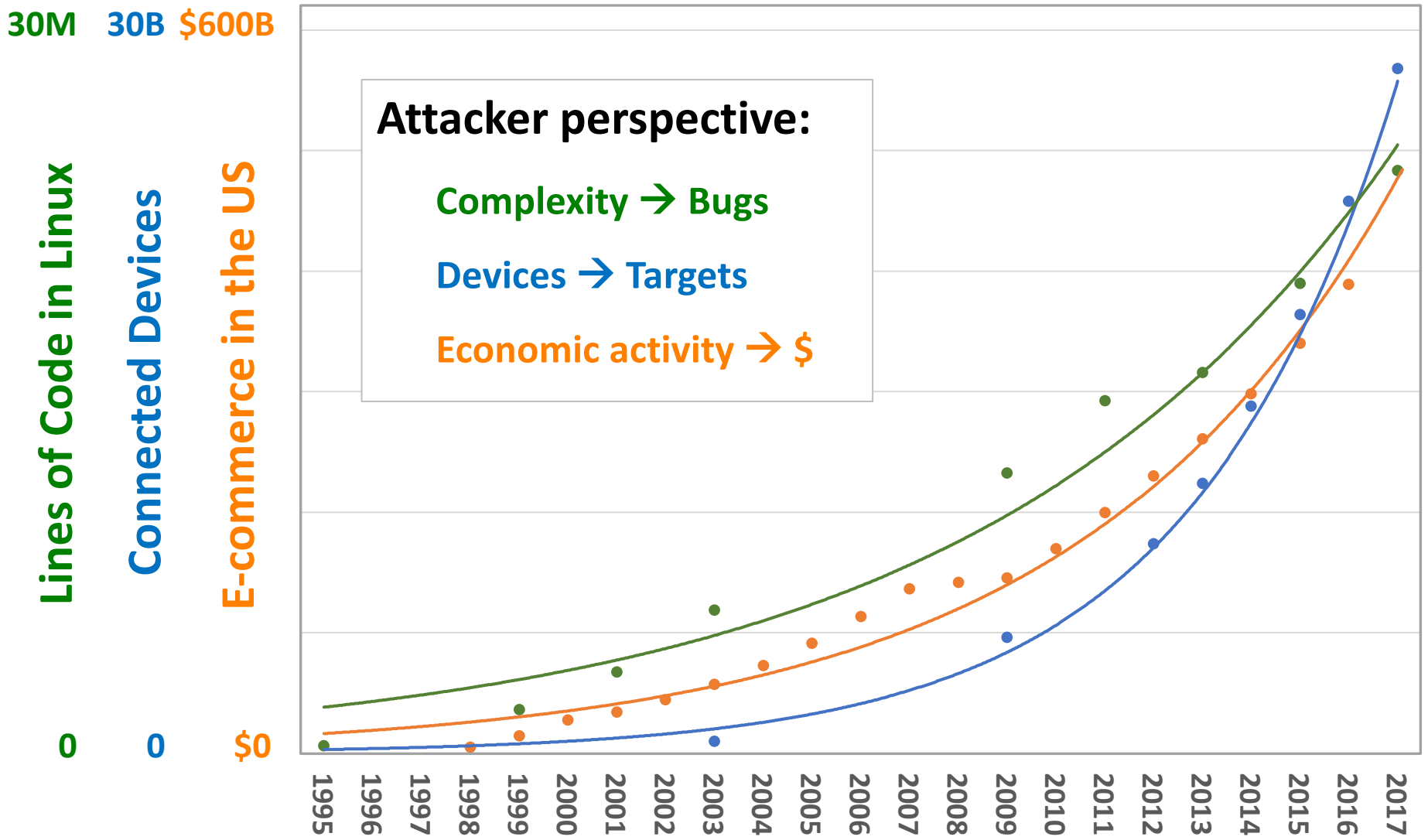
⇐ Achieved by some components (AES, seL4) and simple HW designs

4. Security which anyone can easily verify
   P(secure) = 100% - ϵ



The holy grail…

Too hard?


30M
30B
$600B
Lines of Code in Linux
Connected Devices
E-commerce in the US
0
0
$0
Attacker perspective:
Complexity → Bugs
Devices → Targets
Economic activity → $
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

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)

Exponential growth

Abstractions

U.I. Applications

AI

Libraries/frameworks

Browsers Languages

Compilers

Operating systems

Protocols Drivers

Circuit board

CPU architectures

Chip

Microarchitectures

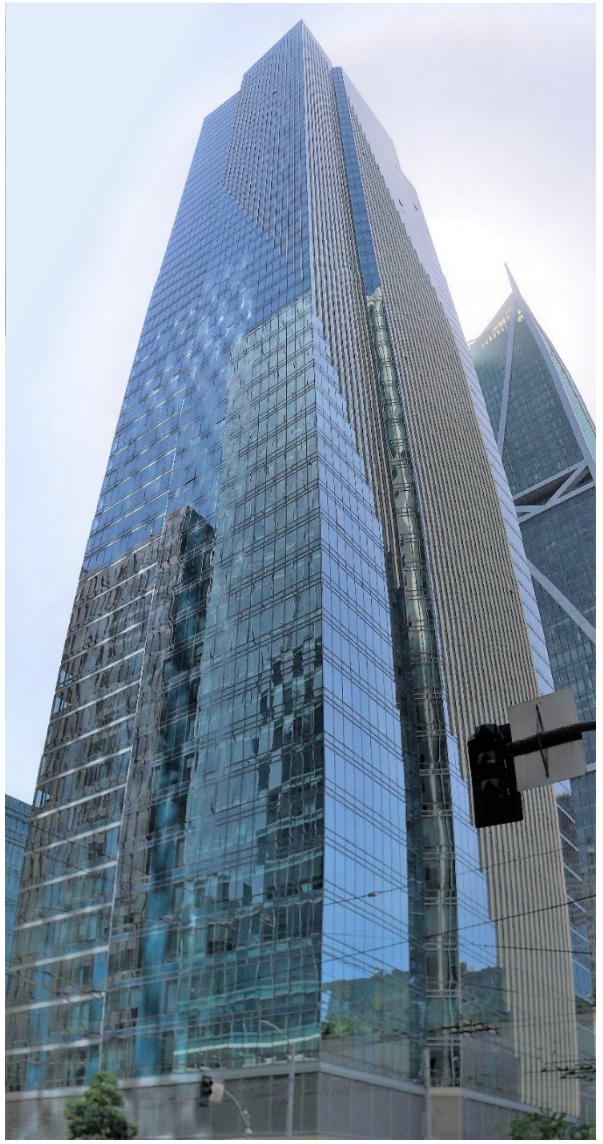
Logic block

Transistors

Society Nation states

People

Business objectives



Security goals

Clouds

Dependencies & assumptions

Foundations

Abstraction creates security challenges

Security-critical details hidden in layers

Needs of distant layers unclear

People specialize then miss big picture

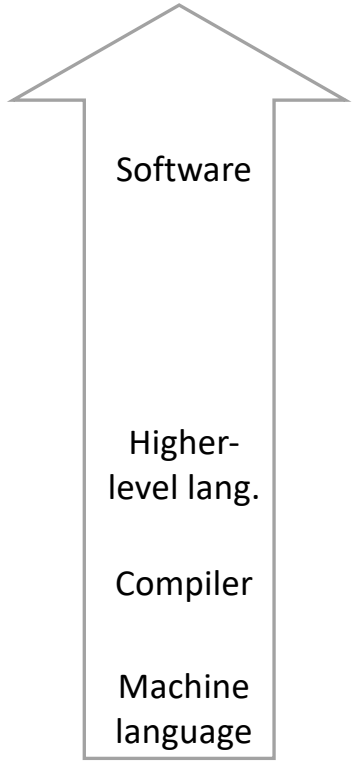
Economics don't fund adequate investment

Risks in other layers deter improvements

Changes aren't communicated across layers

Are there any security implications from speculative execution?

-- Mike Hamburg

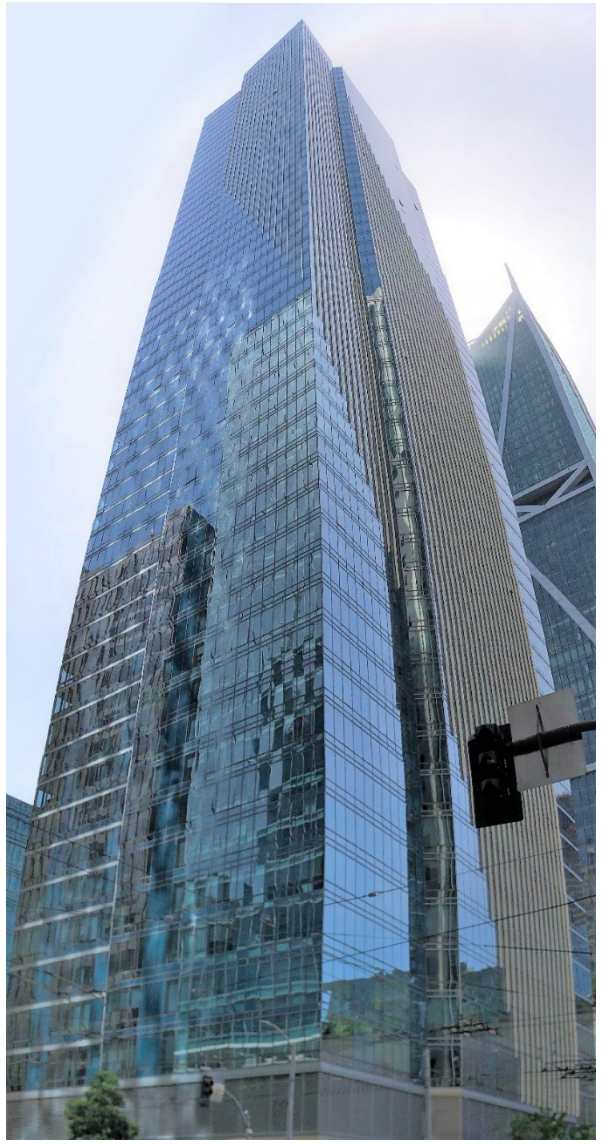


Specification
(Contract btw CPU & SW)

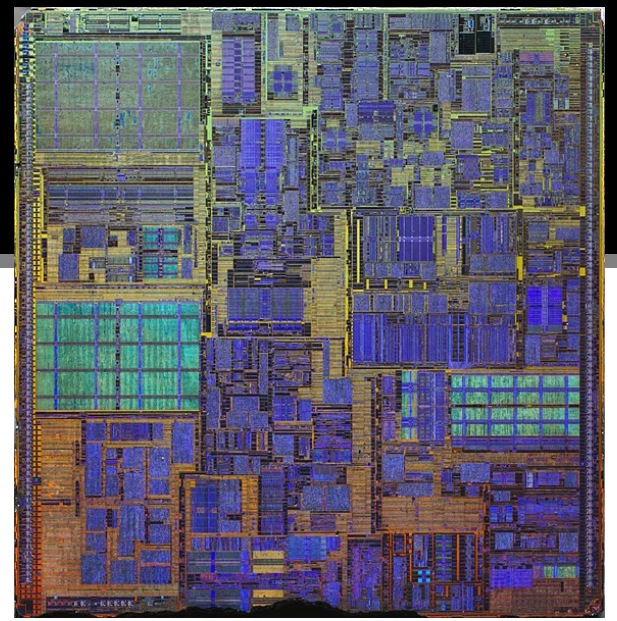
CPU Architecture

Microarchitectures

Speculative execution



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

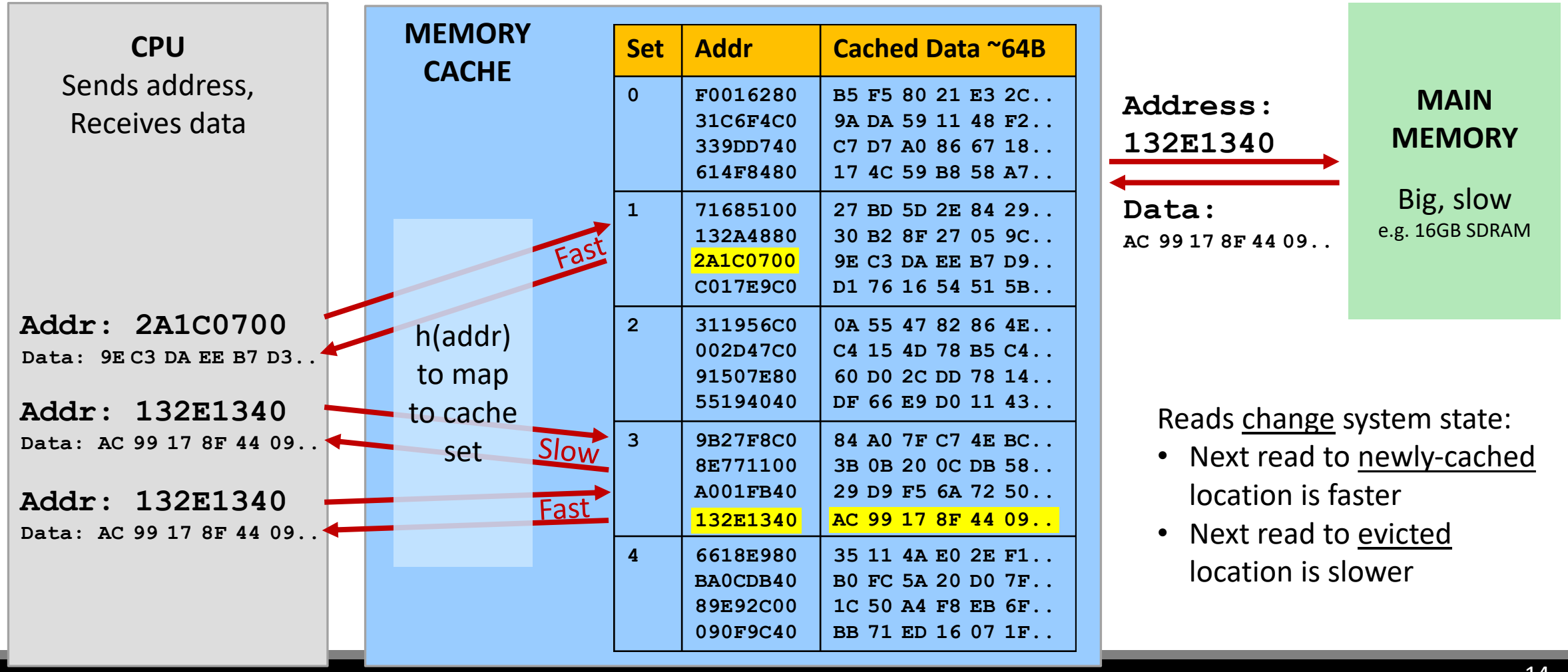
How to do more per clock?

- Reducing memory delays → Caches
- Working during delays → Speculative execution

Public domain image of Pentium 4 die by Ritzchens Fritz

Memory caches

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



Reads change system state:

- Next read to newly-cached location is faster
- Next read to evicted location is slower

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

Architectural Guarantee

Register values eventually match result of in-order execution

Speculative Execution

CPU regularly performs incorrect calculations, then deletes mistakes

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

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

Before attack:

- Train branch predictor to expect if() is true (e.g. call with `x` < `array1_size`)
- Evict `array1_size` and `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]
  ...
```

Contents don't matter

only care about cache ***status***

Uncached Cached

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

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)
  - 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)

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

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…

`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!

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

"|0" is a JS optimizer trick (makes result an integer)

4096 bytes = memory page size

Keeps the JIT from adding unwanted bounds checks on the next line

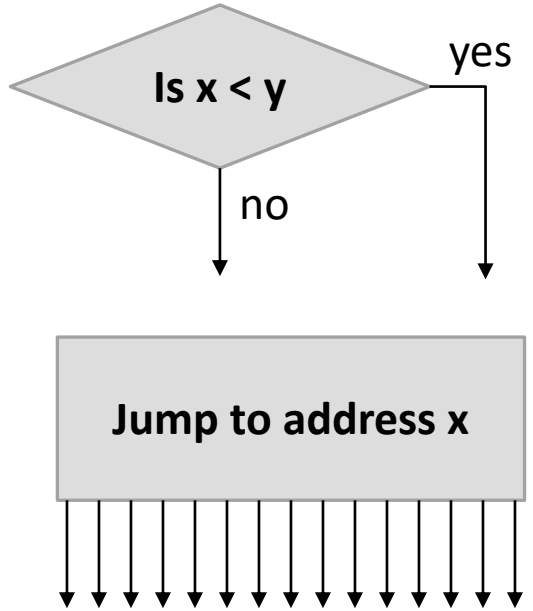
Need to use the result so the operations aren't optimized away

Leak out-of-bounds read result into cache state!

Can evict length/probeTable from JavaScript (easy), timing tricks to detect newly-cached location in probeTable

Indirect branches

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

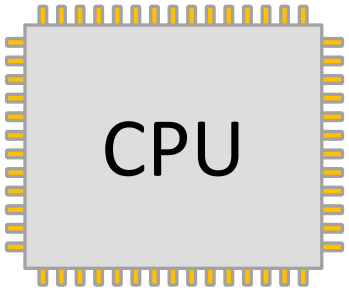
- **Mistrain** branch prediction/BTB so speculative execution will go to gadget
- **Evict** or flush destination address from the cache to ensure long duration speculative execution
- **Execute** victim so it runs gadget speculatively
- **Detect** change in cache state to determine memory data
- **Repeat**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

Mitigations

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

Not necessarily a complete solution

Public domain image by Svetlana Miljkovic
Definition from https://en.oxforddictionaries.com/definition/mitigation



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)

CPU architecture
Machine language
Compiler
Higher-level language

Abstraction boundaries

Software

- operating systems
- drivers
- web servers
- interpreters/JITs
- databases
- :

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

Transfers blame after breach (CPU -> SW)
"you should have put an `LFENCE` there" -> "Fixed"

Mitigations: Indirect branch variant

Intel/AMD (x86):

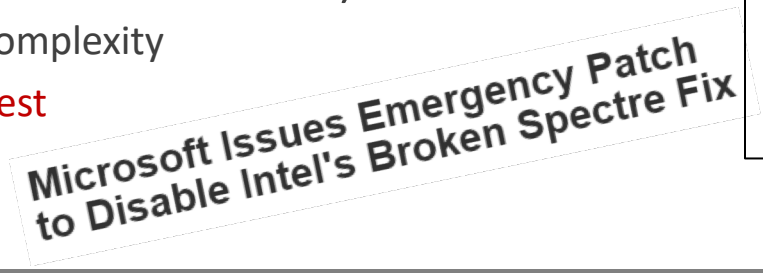
- New MSRs created via microcode
  - Low-level control over branch target buffer (O/S only)
  - Performance impact – limited use
- Retpoline proposal from Google
  - Messy hack -- replace indirect jumps with construction that resists indirect branch poisoning on Haswell
  - Microcode updates to make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

A branchless DOOM

This directory provides a branchless, mov-only version of the classic DOOM video game.

DOOM, running with only mov instructions.

This is thought to be entirely secure against the Meltdown and Spectre CPU vulnerabilities, which require speculative execution on branch instructions.

https://github.com/xoreaxeaxeax/movfuscator/tree/master/validation/doom

Reaction

At Least Three Billion Computer Chips Have the Spectre Security Hole

Companies are rushing out software fixes for Chipmageddon.

Researchers Discover Two Major Flaws in the World's Computers

Intel Faces Scrutiny as Questions Swirl Over Chip Security

ANDY GREENBERG SECURITY 01.03.18 03:00 PM

A CRITICAL INTEL FLAW BREAKS BASIC SECURITY FOR MOST COMPUTERS

Silicon melts

Spectre and Meltdown prompt tech industry soul-searching

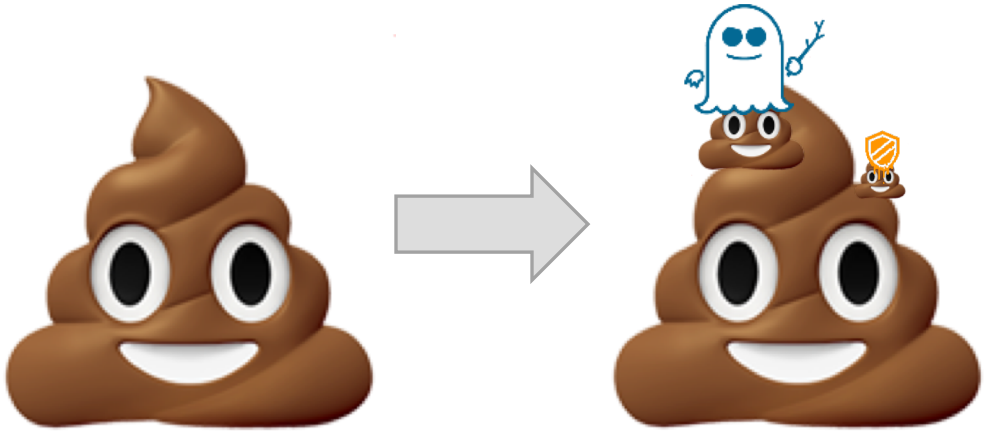
Apple says Spectre and Meltdown vulnerabilities affect all Mac and iOS devices

"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% 😕

Risk in context

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

Shaky foundations

"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)

One size doesn't fit all

Failure to acknowledge trade-offs

- Crazy to use same hardware & operating systems for video games & wire transfers…

Bifurcate into **faster** vs. **safer** flavors

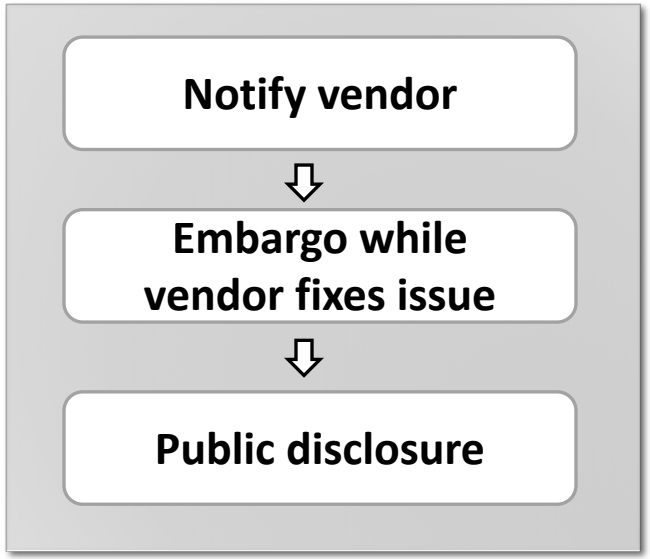
- 'Safer' = less complex (not just a different mode like TrustZone, Intel SGX)
- Accept inefficiency (performance, developer time…)

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…



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…

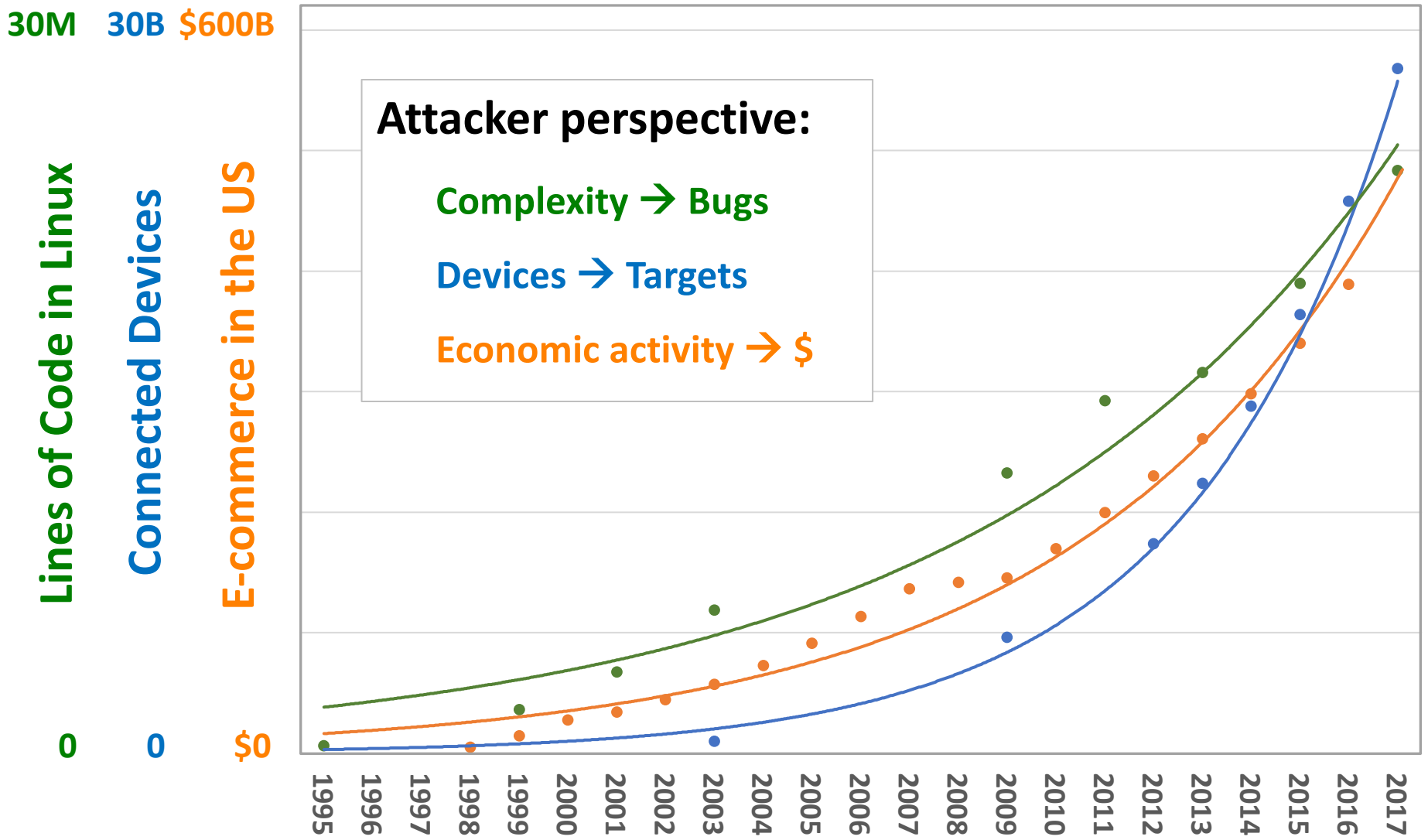
"AMD is not susceptible to all three variants. […]
there is a near zero risk to AMD processors at this time."

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

Problems = Opportunities

Long history of over-optimism


30M
30B
$600B
Lines of Code in Linux
Connected Devices
E-commerce in the US
0
0
$0
Attacker perspective:
Complexity → Bugs
Devices → Targets
Economic activity → $
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

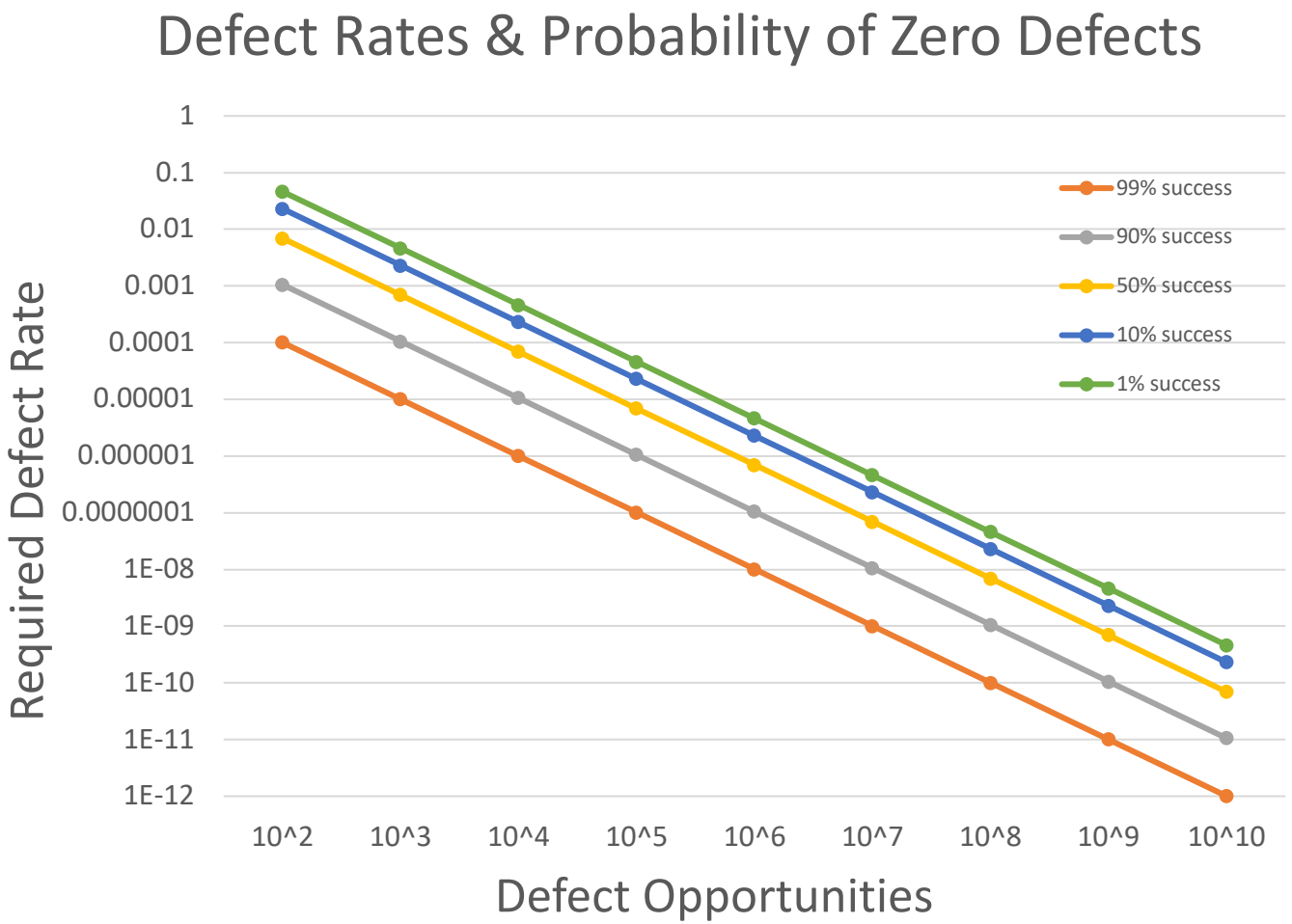
Scaling will continue

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

Machine learning -> complexity beyond what humans can create

Defect rates

New & Improved!
Now only a 10% chance of being completely hackable.



Example:

10^{-4} defects/element

10^{7} critical elements

P(0 defects) ≈ 10^{-434}

Trying harder won't work

Need (10^{6}+) reduction in defects

… to address *today's* situation

Need new perspectives, approaches

Harvard Business Review

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.

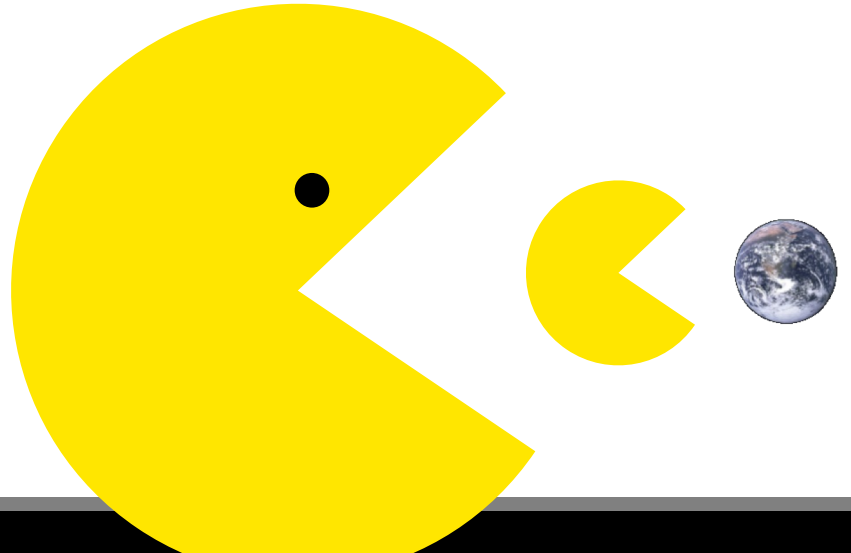
> ***"Software is eating the world."***
> ***-- Marc Andreessen***

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

[1] "Estimating the Global Cost of Cyber Risk" (https://www.rand.org/pubs/research_reports/RR2299.html): $275B-$6.6T direct costs depending on assumptions, and $799B-$22.5T incl. indirect costs.

[2] My estimate. For reference, Intel's entire 2017 revenues were $62.8B, ARM <$2B.

Why work on security problems?

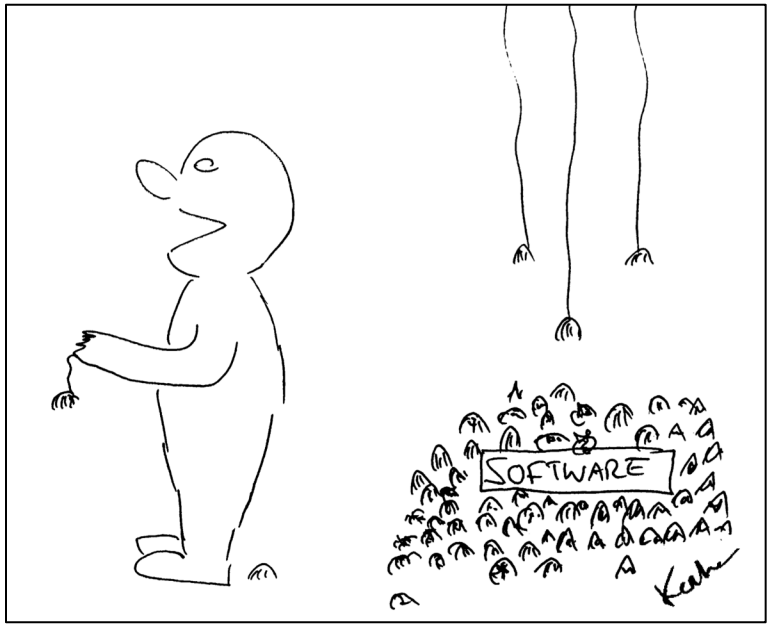
Traditional fields	Security
Well-studied (often intractable) problems	Impact on big, dynamic, messy, exciting problems
Low wage pressure and/or barriers to entry	Labor shortages: high wages, encouragement to entry
Senior people dominate	New entrants lead: new perspectives, skills
Economics driven by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

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… }

Thank you!

My email: paul@paulkocher.com



I found the security bug!