CSE 127 Computer Security

Stefan Savage, Spring 2020, Lecture 8

System Security II: Side Channels

Context

- Last class we talked about using **isolation** and **privilege levels**
  - Used to implement privilege separation, least privilege and complete mediation
  - Basic idea: **protect the secret or sensitive stuff so it can't be accessed across a trust boundary**

- Assumption: we know what the trust boundaries are and that ***access*** to something is easy to identify

Side Channels

- We often think of systems as black boxes:
  - As abstractions that consume input and produce output
  - We assume that all side effects are about output (e.g., values in memory or I/O)
- Sometimes, in addition to what is in the output, critical information can be revealed in ***how*** it is produced.
  - Yes, how… like how long, how fast, how loud, how hot… artifacts of the ***implementation*** not the abstraction
  - This can produce a ***side channel***: a source of information beyond the output specified by the abstraction.

Today

- Overview of side channels in general
- Cache side channels
- Meltdown, Spectre & Rowhammer

Consumption Side Channels

- How long does this password check take?
  - Depends on where the first mismatch is.

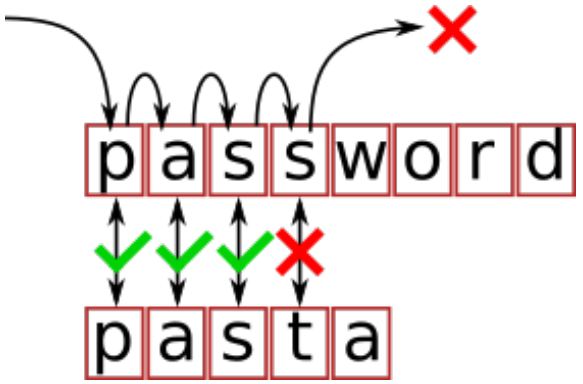
```
char pwd[] = “z2n34uzbnqhw4i”;

//...

int CheckPassword(char *buf)
{
    return strcmp(buf, pwd);
}
```

Side Channels

- Consumption: how much of a resource is being utilized to perform the operation?
  - Examples: time, power, memory, network, etc.
- Emission: what out-of-band signal is generated in the course of performing the operation?
  - Examples: electro-magnetic radiation, sound, movement, error messages, etc.

Side Channel Examples

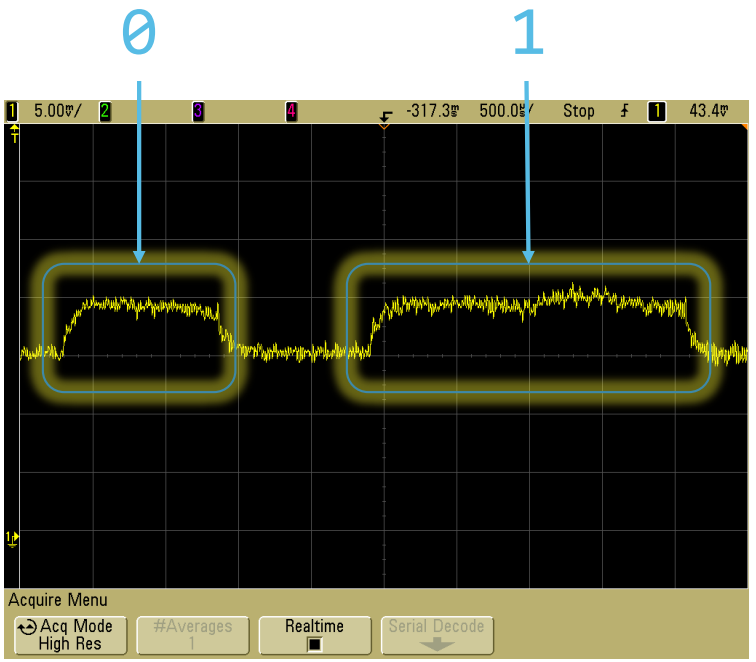
- Tenex password verification
  - Alan Bell, 1974
  - Character-at-a-time comparison + virtual memory
  - Recover the full password in linear time



https://www.sjoerdlangkemper.nl/2016/11/01/tenex-password-bug/

Side Channel Examples

- Secret cryptographic key value maintained in hardware
  - Can never be read, only used
  - (e.g., cable box, smart phone)
- Simple Power Analysis (SPA)
- Differential Power Analysis (DPA)
  - Paul Kocher, 1999
  - Using signal processing techniques on a very large number of samples, iteratively test hypothesis about secret key bit values.



https://en.wikipedia.org/wiki/Power_analysis#/media/File:Power_attack_full.png

Side Channel Examples

- *Timing Analysis of Keystrokes and Timing Attacks on SSH*
  - Dawn Song, David Wagner, Xuqing Tian, 2001
  - Recover characters typed over SSH by observing packet timing

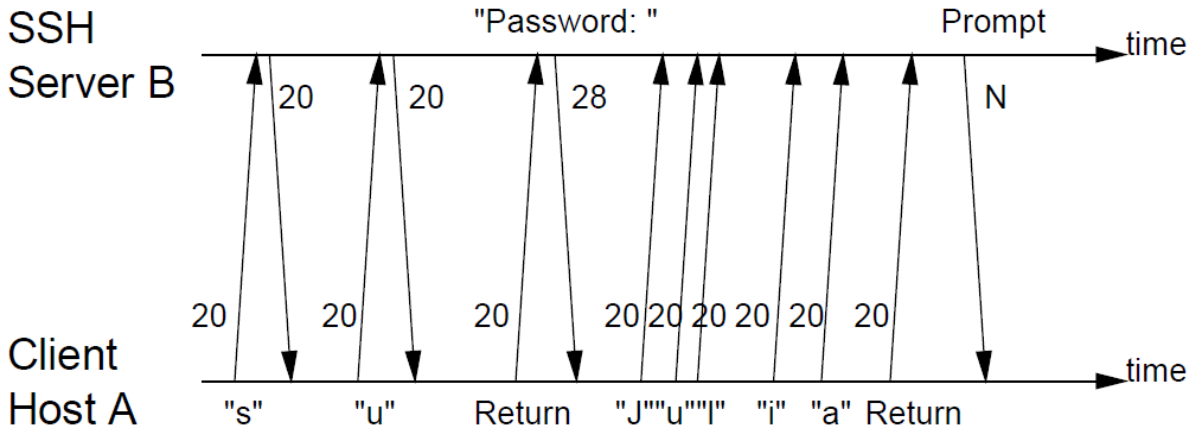


Figure 1: The traffic signature associated with running SU in a SSH session. The numbers in the figure are the size (in bytes) of the corresponding packet payloads.

https://people.eecs.berkeley.edu/~daw/papers/ssh-use01.pdf

Side Channel Examples

- *Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow*
  - Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang, 2010
  - Recover characters typed into search boxes over HTTPS by observing sizes of responses

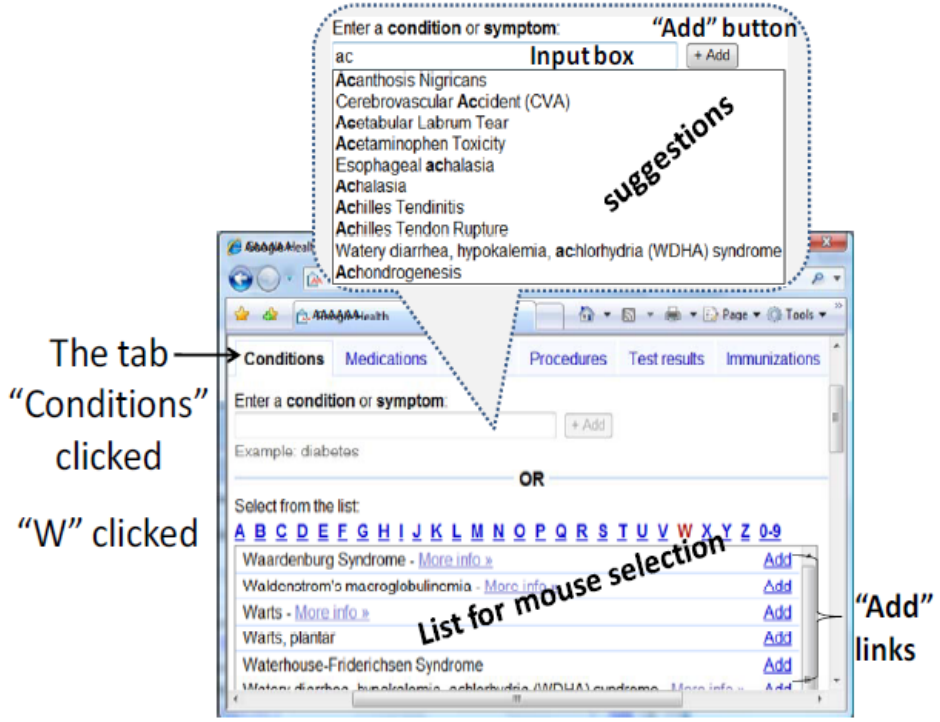


Figure 2: User interface for adding health records

https://www.microsoft.com/en-us/research/publication/side-channel-leaks-in-web-applications-a-reality-today-a-challenge-tomorrow/

Side Channel Examples

- *Keyboard Acoustic Emanations*
  - D. Asonov, R. Agrawal, 2004
  - Recover keys typed by their sound
- *Keyboard Acoustic Emanations Revisited*
  - Li Zhuang, Feng Zhou, J. D. Tygar, 2009

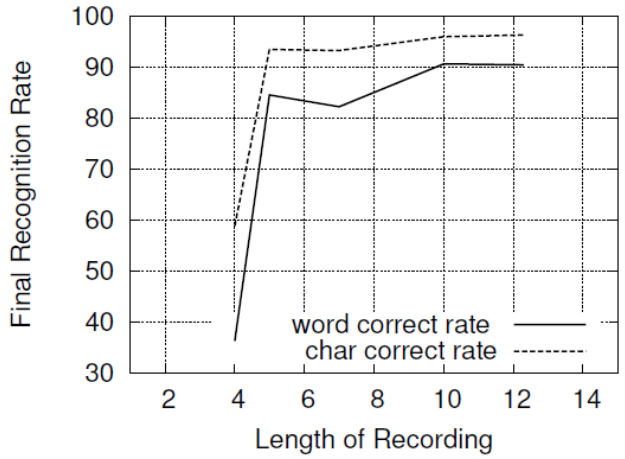
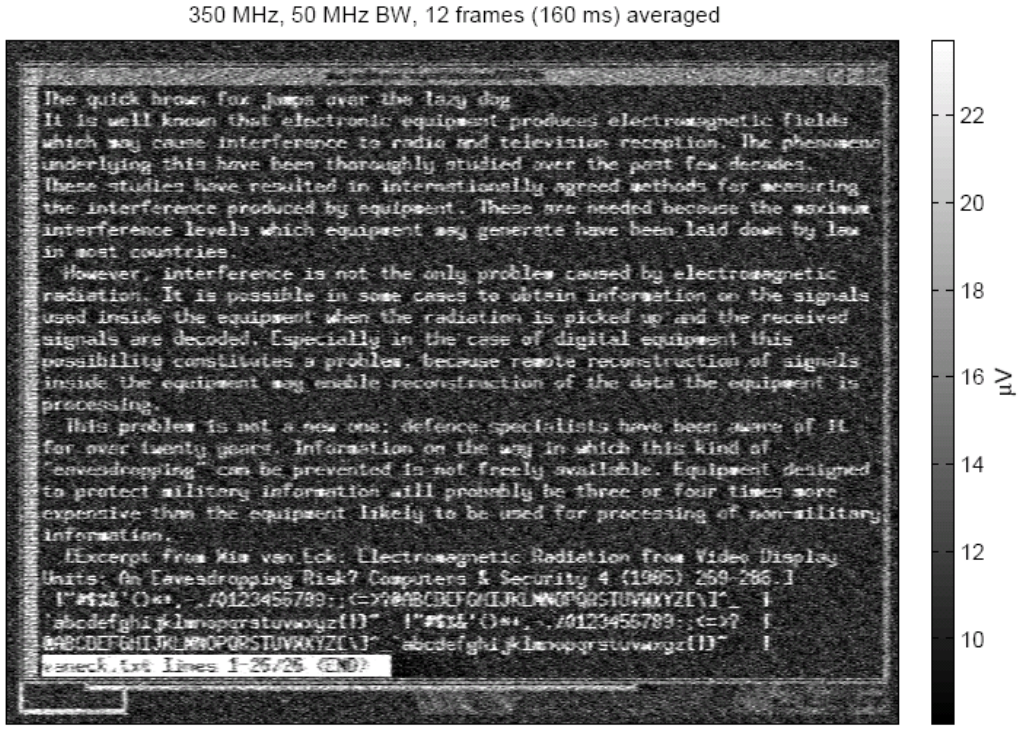


Fig. 7. Length of recording vs. recognition rate.

https://www.microsoft.com/en-us/research/publication/side-channel-leaks-in-web-applications-a-reality-today-a-challenge-tomorrow/

Remote reading of LCD screens via RF (Kuhn, 2004)



Target and antenna in a modern office building 10 m apart, with two other offices and three plasterboard walls (−2.7 dB each) in between. Single-shot recording of 8 megasamples with storage oscilloscope at 50 Msamples/s, then offline correlation and averaging of 12 frames.

Optical domain emanations (Kuhn, 2002)

- Light emitted by CRT is
  - Video signal combined with phosphor response
- Can use fast photosensor to separate signal from HF components of light
- Even if *reflected* off diffuse surface (i.e., a white wall) from across the street

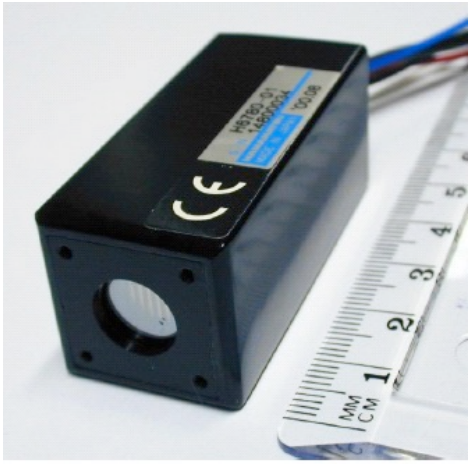
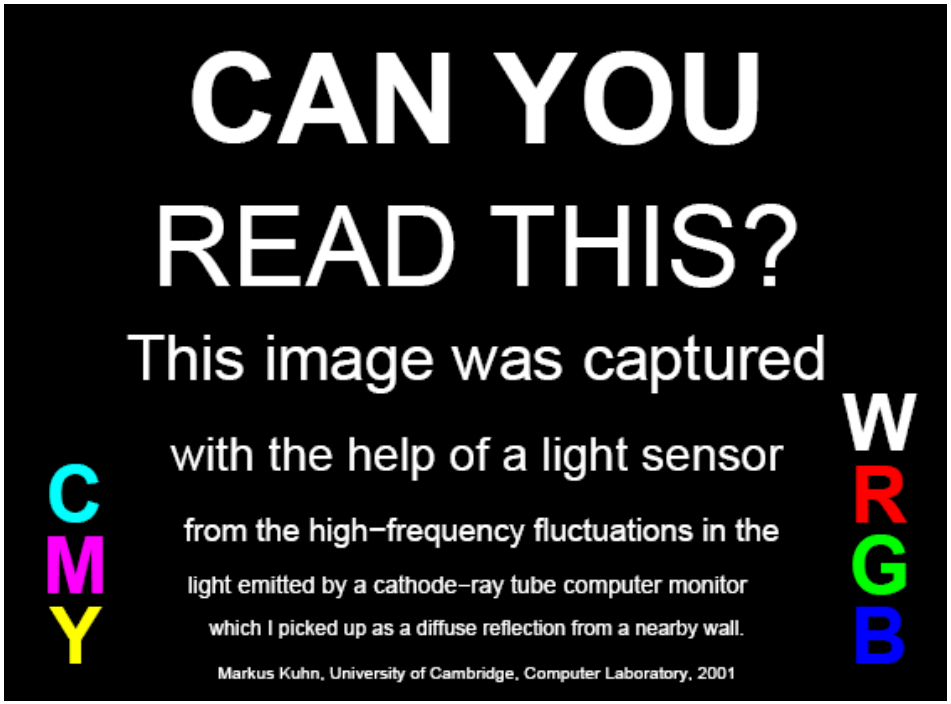
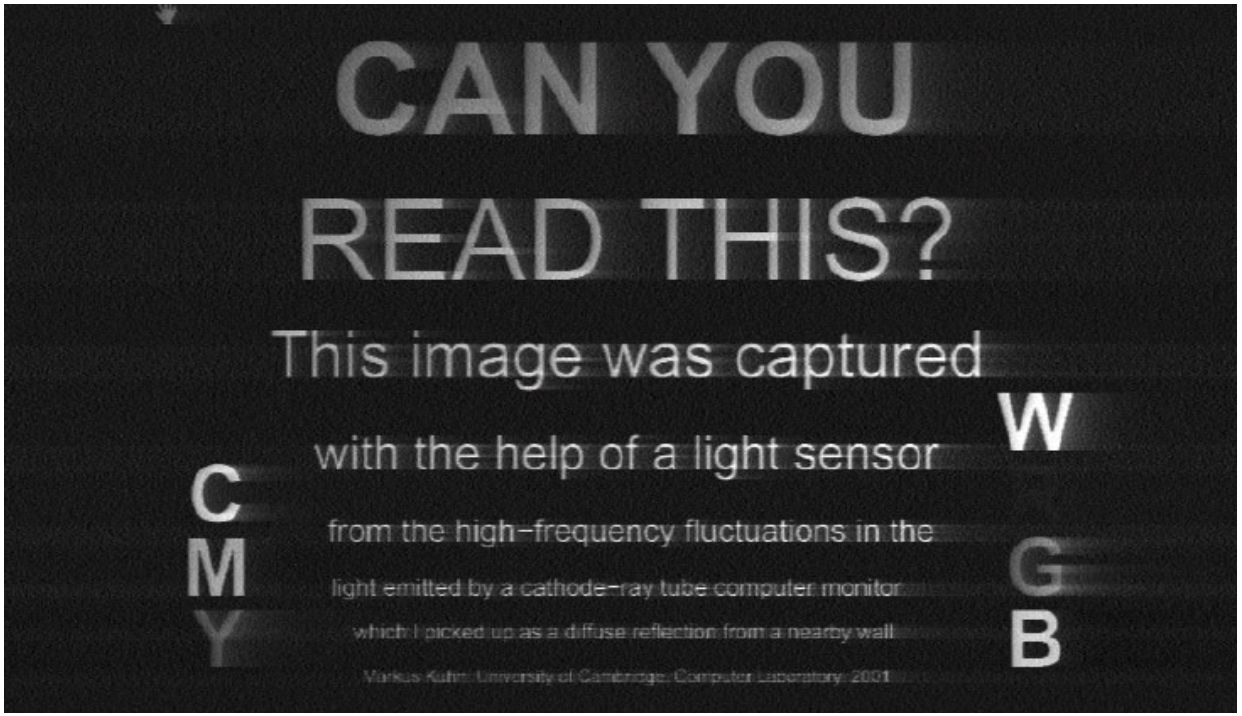


Figure 1. Photomultiplier tube module.

Source signal



Bounced off a wall



Side Channel examples

- Heat of the Moment, 2011
  - Meiklejohn et al

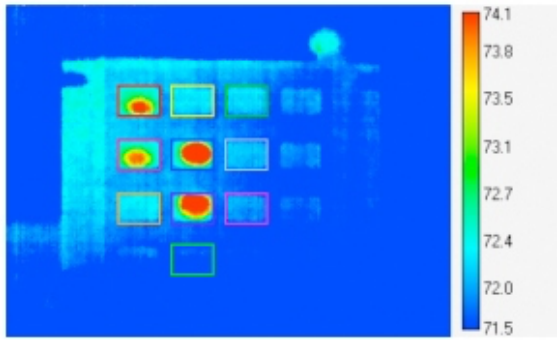
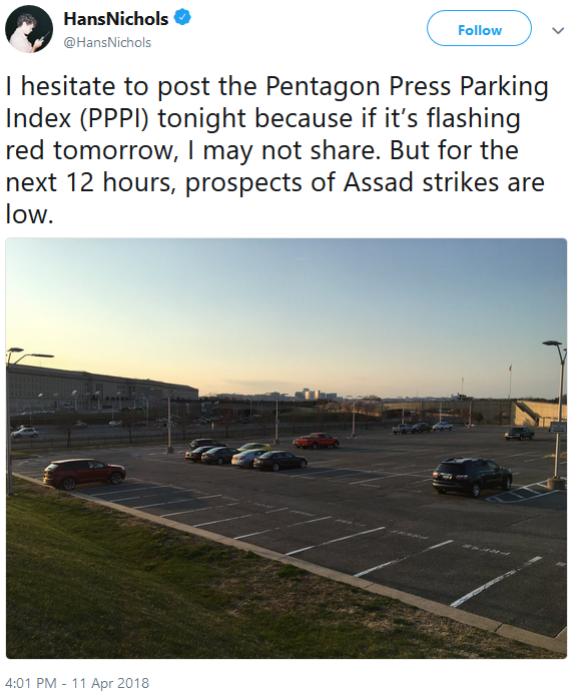


Figure 2: The Diebold plastic ATM keypad with rubber keys, model 19-019062-001M REV1.

Side Channels

- The Pentagon Press Parking Index



https://twitter.com/HansNichols/status/984204920722264064

Active Side Channels

- Faults can create additional side channels or amplify existing ones
  - Erroneous bit flips during secret operations may make it easier to recover secret internal state
- Attackers can induce faults, leading to ***fault injection attacks***
  - Glitch power, voltage, clock
  - Vary temperature
  - Subject to light, EM radiation

Aside: covert channels

- Side channels are inadvertent artifacts of the implementation that can be analyzed to extract information across a trust boundary
- **Covert channels** are the same idea, but put on purpose
  - One party is trying to leak information in a way that it won't be obvious
  - By encoding that information into some side channel
    - E.g., variation in time, memory usage, etc…
  - Incredibly difficult to protect against

Mitigating Side Channels

- Eliminate dependency on secret data
- Make everything the same
  - Use the same of amount of resources every time
  - Hard (many optimizations in hardware, compilers, etc.)
  - Expensive (everything runs at worst-case performance)
- Hide
  - "Blinding" can be applied to input for some algorithms
    - Secret data is not operated on in "clear" form
  - Only possible for certain algorithms (eg RSA)
- Adding random noise?
  - Attacker just needs more measurements to extract signal

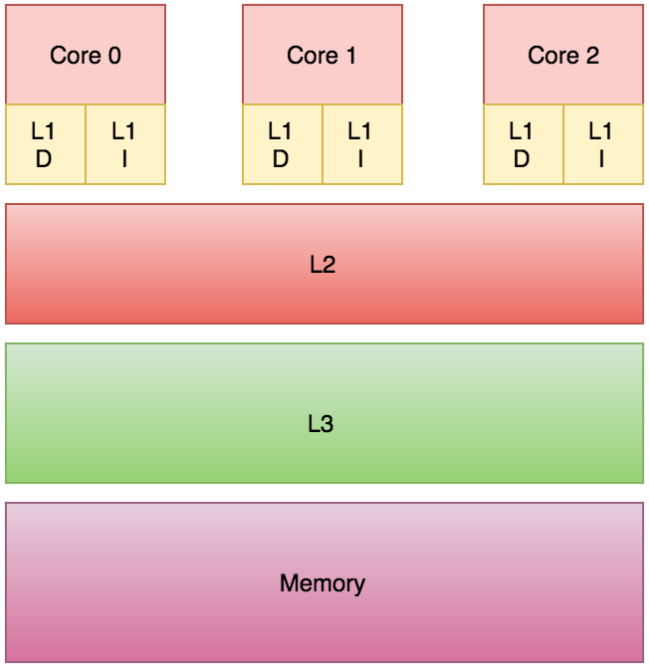
Cache Side Channels

Cache

- Main memory is dense (high capacity) … but slow
- Processors try to "***cache***" recently used memory in faster, but smaller capacity, memory cells closer to the actual processing core

Cache Hierarchy

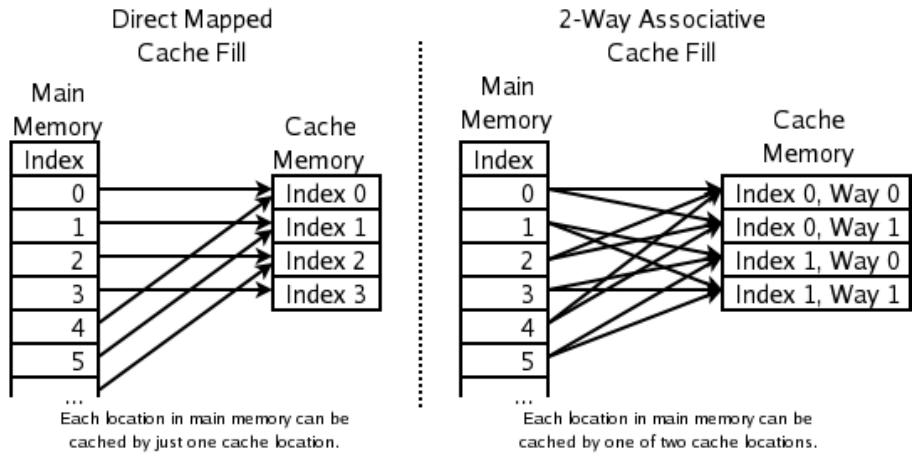
- Caches are such a great idea, let's have caches for caches!
- Level closest to the processing core is fastest but smallest capacity
- Successive levels have increasing capacity, but slower speeds



https://en.wikipedia.org/wiki/Cache_hierarchy

Cache Organization

- Cache line
  - Unit of cache granularity
  - Example: 64 bytes
- Set Associativity
  - Cache lines grouped into sets
  - Each memory address is mapped to a set of cache lines
    - Ex: address % number_of_sets
    - Aside: which address? (typically)
      - L1: Virtually indexed
      - L2/L3: Physically indexed
      - All typically "tagged" by physical address
  - Can use either set to satisfy (reduces conflict misses)



https://en.wikipedia.org/wiki/CPU_cache

Cache Side Channels

- Cache is a shared system resource
  - Not isolated by process, VM, or privilege level
  - "Just a performance optimization"
    - Has no impact on ***contents*** of reads
    - But does impact the ***time*** that a read takes

Cache Side Channels

- Threat model:
  - Attacker and victim are two different execution domains (processes or privilege levels) on the same physical system
  - Attacker is able to invoke (directly or indirectly) functionality exposed by the victim
    - Potentially, with attacker-supplied parameters
    - Example: using API, making a system call, running a setuid binary, etc.
  - We don't want attacker to be able to infer anything about the contents of victim memory
- Some algorithms, including cryptographic algorithms, have memory access patterns that are dependent on sensitive memory contents
  - Therefore memory access patterns may implicitly leak memory contents

Things attacker can do

- Prime
  - Place a known address in the cache (by reading it)
- Evict
  - Access memory until a given address is no longer cached (force capacity misses)
- Flush
  - Remove a given address from the cache (`cflush` instruction on x86)
- Measure
  - Very precisely (down the the cycle) how long it takes to do something (`rdtsc` on x86)

All cache side channel attacks of a similar form:
manipulate cache into known state, make victim run and then try to infer what has changed in cache as a result

Cache Side Channels

- Three basic techniques
  - **Evict and time**
    - Kick stuff out of the cache and see if victim slows down as a result
  - **Prime and probe**
    - Put stuff in the cache, run the victim and see if you slow down as a result
  - **Flush and reload**
    - Flush a particular line from the cache, run the victim and see if your accesses are still fast as a result

Evict & Time

- Run the victim code several times and time it
  - Baseline
- Evict (portions of) the cache
  - How?
  - Access many memory locations so that previous cache contents are replaced with this recently-accessed data (conflicts in cache cause eviction)
- Run the victim code again and retime it
- If it is slower than before, then cache lines evicted by the attacker must've been used by the victim
  - We now know something about the addresses accessed by victim code

Prime & Probe

- Prime the cache
  - Access many memory locations (covering all cache lines of interest) so previous cache contents are replaced with attacker addresses
  - Time access to each cache line to establish speed for "in cache" references
- Run victim code
- Attacker retimes access to its own memory locations
  - If any are slower then it means the corresponding cache line was used by the victim
  - We now know something about the addresses accessed by victim code

Flush & Reload

- Specifically for shared memory
  - E.g., shared libraries, fork() sharing, deduplication in VMs

- Time memory access to (potentially) shared regions
- Flush (specific lines from) the cache
- Invoke victim code
- Retime access to flushed addresses, if still fast was used by victim
  - Because we flushed it it should be slow, victim must have reloaded it
  - We now know something about the addresses accessed by victim code

Also

- The error on any individual measurement is high
  - So you repeat many times and use normal order statistics (e.g., median values) to filter outliers from pattern
- Can work on different levels of caches (L1 to L3)
  - Different time domains, just need to exclude effects of faster cache levels
- Can work on both instruction cache and data cache
  - These are commonly separate at L1
  - Measurements and primes are simply about running code sequences rather than accessing data values

Cache Side Channels

- Ok, if I suspend all disbelief I may be able to imagine how something like this is possible in theory, but how practical is it really?
- "*Our error-correcting and error-handling high-throughput covert channel can sustain transmission rates of more than 45 KBps on Amazon EC2.*"
  - *Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud*
    - by Clementine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Kay Romer, Stefan Mangard

However…

- So far, we've only been able to infer things about the *addresses* being used
  - In certain cases this is bad (e.g., address pattern can give info about what values might be if computation depends on them), can be used for covert channels, etc
  - But…

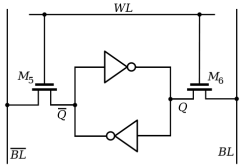
- We can't yet
  - Read **protected values**
  - Write **protected values**

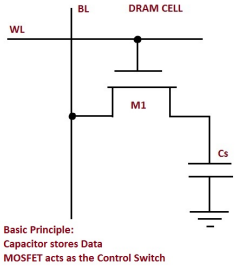
- Coming up: DRAM disturbance attacks (writing protected memory), Speculation attacks (reading protected memory)
  - Rowhammer, Meltdown and Spectre

Rowhammer

Review: RAM

- Volatile memory: data retained only as long as power is on
  - As opposed to persistent memory, which retains data even without power (e.g. flash, magnetic disks)
- Static RAM (SRAM) vs Dynamic RAM (DRAM)
  - SRAM: retains bit value as long as power is on, without any additional refresh
    - Faster
    - Lower density
    - Higher cost
  - DRAM: requires periodic refresh to maintain stored value
    - Refresh about every 64 ms
    - Higher density (higher capacity)
    - Lower cost

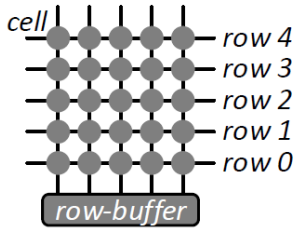




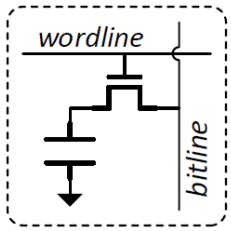
https://commons.wikimedia.org/wiki/File:SRAM_Cell_Inverter_Loop.png
https://allthingsvlsi.wordpress.com/tag/dram-cell/

Review: DRAM

- Cells are grouped into rows
  - ~1Kb per row
  - All cells in a row are refreshed together
- Refresh: read the row and write it back
- All access to individual cells happens via the "row buffer"



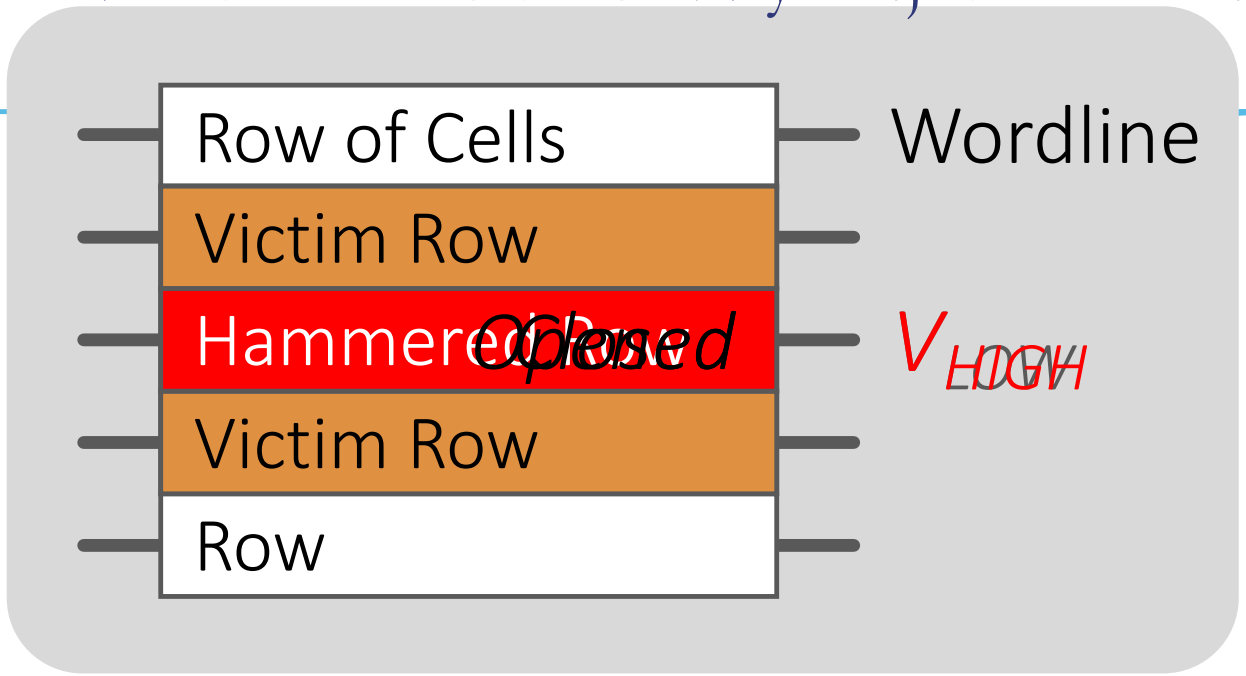
a. Rows of cells



b. A single cell

- As DRAM gets smaller (<35nm) more issues with *reliability* (retention, writes, and… reads)

Disturbance Errors caused by Adjacent Reads



Repeatedly opening and closing a row enough times within a refresh interval induces disturbance errors in adjacent rows in many of the real DRAM chips on the market

Rowhammer

- Attack scenario: attacker code is executing on the same machine as victim, but with less privileges
  - Example: userland attacking kernel, javascript attacking browser, etc.
  - Rowhammer attack lets attacker modify memory they can't access
- Exploit sketch
  - Characterize DDR flip locations
  - Identify protected target data to flip
    - Examples: page tables, su binaries, etc.
  - Maneuver protected data over flipping location
    - Example: allocate all other memory
  - Hammer own memory locations to alter protected data
- Amazing fact: **random** bit flips can be used to semi-deterministically take over computer

Rowhammer solutions

- ECC memory
  - Compute error correcting code on write, check on read
  - Significant mitigation, but recent attacks can bypass in some cases
  - Adds cost (common in servers, but not in consumer devices)
- Memory controller limitations on "hammering" or additional adjacent line refresh
  - Can have performance impact
- Process changes to mitigate inter-row effects
  - Conflicts with density demands

Meltdown and Spectre

Review: Virtual Memory

- Kernel virtual memory is mapped into every process
  - For efficiency
- Page table access control ensures that kernel pages are only accessible when the processor is in privileged mode

	Non-Secure				Secure	
EL0	App X	App Y	App X’	App Y’	App X”	App Y”
EL1	Guest OS A		Guest OS B		Secure OS	
EL2	Hypervisor					
EL3			Secure Monitor			

Quick review: ISA and µArchitecture

Instruction Set Architecture (ISA)

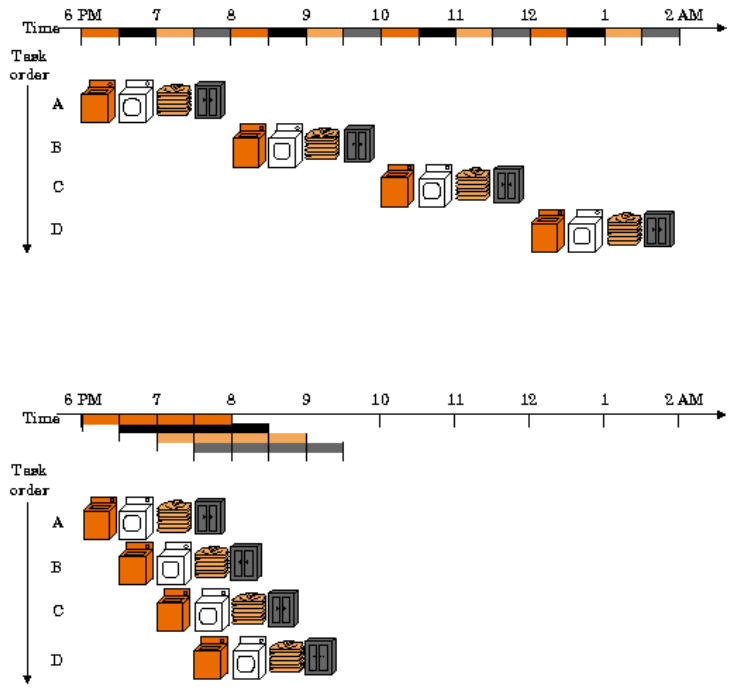
- Defined interface between hardware and software
  - Registers, instructions, etc.

µArchitecture

- Implementation of the ISA
- "Behind the curtain" details
  - E.g. cache specifics
- Key issue:: µArchitectural details cam sometimes become "architecturally visible"

Review: Instruction Pipelining

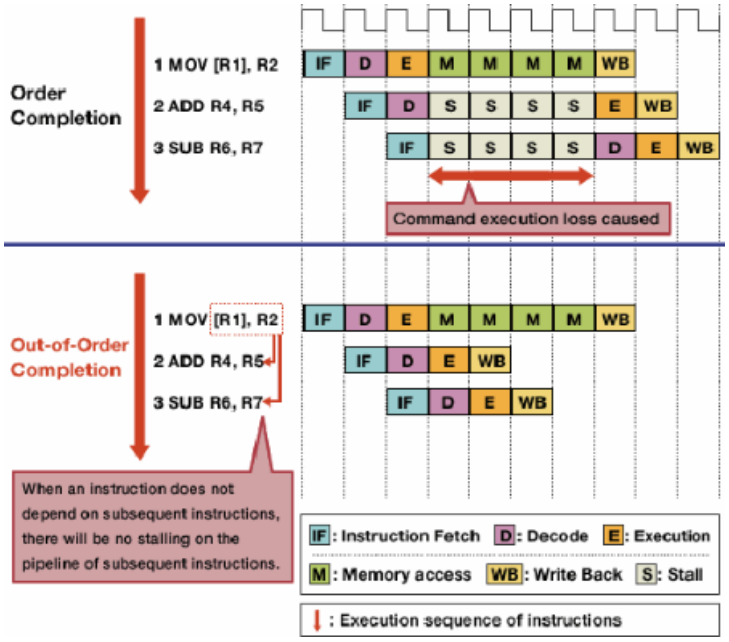
- Processors break up instructions into smaller parts so that these parts could be processed in parallel.
- µArchitectural optimization
  - Architecturally, instructions appear to be executed one at a time, in order
  - Dependencies are resolved behind the scenes



https://www.cs.fsu.edu/~hawkes/cda3101lects/chap6/index.html?$$$F6.1.html$$$

Review: Out-of-order Execution

- Some instructions can be safely executed in a different order than they appear
- Avoid unnecessary pipeline stalls
- µArchitectural optimization
  - Architecturally, it appears that instructions are executed in order



https://renesasrulz.com/doctor_micro/rx_blog/b/weblog/posts/pipeline-and-out-of-order-instruction-execution-optimize-performance

Review: Speculative Execution

- Sometimes control flow depends on output of an earlier instruction.
  - E.g. conditional branch, function pointer
- Rather than wait to know for sure which way to go, the processor may "speculate" about the direction/target of a branch
  - Guess based on the past
  - If the guess is correct, performance is improved
  - If the guess is wrong, speculated computation is discarded and everything is re-computed using the correct value.
- μArchitectural optimization
  - At the architecture level, only correct, in-order execution is visible

Excerpted from

Embedded Security

Chapter 3: Meltdown and Spectre

Michael Schwarz, Moritz Lipp

20.03.2018

www.iaik.tugraz.at

Switch to Schwarz and Lipp slides

What to do?

- Meltdown
  - Key issue is that privilege checks are not being checked before cache access allowing user inference of kernel memory contents
  - Solution: stop mapping kernel memory into process address space
    Makes system calls slower
    (a range of optimizations, including AS identifiers in TLBs make it better)
- Spectre
  - Not so easy… general issue about speculation and branch prediction
  - Could disable speculation on branches… but huge performance impact
    - Also can only be done by chip manufacturer (microcode patch)
  - Selectively insert instructions to stop speculation at sensitive branches
    - LFENCE, (x86), CSDB (ARM)
    - Various special hacks (e.g., Intel adds special microcode sequences to manipulate branch prediction state)
  - Ultimately, change cache architecture to not reflect speculated actions… but need entirely new hardware

These are the beginning on a wave

- Computer architects spent the last 20 years optimizing for the common case
  - Assumption is that if optimization doesn't change output then its all good
- All of these optimizations are now being examined and being used to create new side channels
  - L1TF, MDS, TAA, iTLB multihit
  - And more…
- Hardware vendors and computer architects are retooling to figure out how to still offer optimization without big security holes

Summary

- Side channels are a way to violate trust boundary even if there are no errors in interface or control flow vulnerabilities
  - Information contained in details of the implementation
    - Resource consumption or emanations
- Cache side channels
  - Shared caches provide a timing channel to infer what addresses are being used by victim code
- Rowhammer
  - Active channel for manipulating DRAM via reads (based on imperfect memory reliability)
- Speculation side channels
  - Speculative instructions can impact cache state, allowing measurements of protected state

Additional Resources

- Google Project Zero
  - https://googleprojectzero.blogspot.com/
  - Rowhammer:
    - https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html
  - Spectre/Meltdown:
    - https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html
- Anders Fogh blogs
  - https://cyber.wtf/author/andersfogh1974/
  - https://dreamsofastone.blogspot.com/
- Paul Kocher on Spectre
  - https://www.youtube.com/watch?v=hqIavX_SCWc

Additional Resources

- TU Graz team
  - Moritz Lipp: https://mlq.me/
  - Daniel Gruss: https://gruss.cc/
  - Michael Shwarz: https://misc0110.net/web/
- VUSec
  - https://www.vusec.net/