Hardware Security:
Meltdown, Spectre, Rowhammer
Vulnerabilities and Abstractions

“Layers of abstraction become boundaries of competence.”

- Sergey Bratus
Vulnerabilities and Abstractions

Abstraction

Reality
Understanding Vulnerabilities

How vulnerabilities are explained

- Alice: “There’s a vulnerability in XY.”
- Bob: “What is it?”
- Alice: “Well, you know how X works?”
- Bob: “Yeah.”
- Alice: “And you know how Y works?”
- Bob: “Yes.”
- Alice: “...”
- Bob: “Oh!”
## Review: ISA and μArchitecture

<table>
<thead>
<tr>
<th>Instruction Set Architecture (ISA)</th>
<th>μArchitecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Defined interface between hardware and software</td>
<td></td>
</tr>
<tr>
<td>▪ Registers, instructions, etc.</td>
<td>▪ Implementation of the ISA</td>
</tr>
<tr>
<td></td>
<td>▪ “Behind the curtain” details</td>
</tr>
<tr>
<td></td>
<td>▪ E.g. cache specifics</td>
</tr>
<tr>
<td></td>
<td>▪ Spoiler: μArchitectural details may become architecturally visible</td>
</tr>
</tbody>
</table>
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
Review: Out-of-order Execution

- Some instructions can be safely executed in a different order than they appear.
- This may allow the processor to use available resources to “pre-compute” future instructions.
- µArchitectural optimization
  - Architecturally, it appears that instructions are executed in order.

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
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

<table>
<thead>
<tr>
<th></th>
<th>Non-Secure</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EL0</strong></td>
<td>App X</td>
<td>App Y</td>
</tr>
<tr>
<td><strong>EL1</strong></td>
<td>Guest OS A</td>
<td>Guest OS B</td>
</tr>
<tr>
<td><strong>EL2</strong></td>
<td>Hypervisor</td>
<td></td>
</tr>
<tr>
<td><strong>EL3</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Review: Cache Side Channel

- Attacker process can infer which address the victim process accessed by monitoring cache contents
- Cache contents are monitored indirectly
  - Timing access to different memory locations
- Flush + Reload
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”
Rowhammer

- Repeatedly accessing a row, can cause bit flips in adjacent rows
  - ~ 1 in 1000 bits flip
  - Bit flips are repeatable

- Adjacent rows can belong to other process’, or kernel’s, memory space.
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.

- Exploit sketch
  - Characterize DDR flip locations
  - Identify protected target data to flip
    - Examples: page tables, su binaries, etc.
  - Maneuver protected data over flip location
    - Example: allocate all other memory
  - Hammer own memory locations to alter protected data
## Rowhammer

<table>
<thead>
<tr>
<th></th>
<th>2015</th>
<th>2016</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hammer technique</strong></td>
<td>Single-sided (cycle)</td>
<td>Double-sided</td>
<td>One-location</td>
</tr>
<tr>
<td><strong>Launch platform</strong></td>
<td>Desktop Intel/AMD</td>
<td>Browser</td>
<td>SGX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mobile, ARM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cloud server</td>
<td></td>
</tr>
<tr>
<td><strong>Protected data</strong></td>
<td>Page-table spray (probabilistic)</td>
<td>Memory exhaustion (deterministic)</td>
<td>Memory monitoring (side-channel)</td>
</tr>
<tr>
<td><strong>Protected data</strong></td>
<td>Page table</td>
<td>Kernel object</td>
<td>Opcode</td>
</tr>
</tbody>
</table>
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:

- Anders Fogh blogs
  - https://dreamsofastone.blogspot.com/

- Paul Kocher on Spectre
  - https://www.youtube.com/watch?v=hqlavX_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/
Homework

- Final is on Saturday 6/9 3pm-6pm in TBD
Next Lecture...

Secure Development Lifecycle