Meltdown, Spectre, and CPU-level bugs

This is gonna be a doozy
Spectre & Meltdown

- 3 separate bugs
  - Spectre (Variant 1) : CVE-2017-5753
  - Spectre (Variant 2) : CVE-2017-5715
  - Meltdown (Variant 3) : CVE-2017-5754

- All due to _speculative execution and out of order execution_
Variants overview

**Spectre**
- CVE-2017-5753
- Variant 1
- Bounds Check Bypass
- Primarily affects interpreters/JITs
- Primarily affects kernels/hypervisors

**Meltdown**
- CVE-2017-5715
- Variant 2
- Branch Target Injection
- Primarily affects kernels/hypervisors
- CVE-2017-5754
- Variant 3
- Rogue Data Cache Load
- Affects kernels (and architecturally equivalent software)
Spectre & Meltdown

- Spectre
  - Discovered independently by 2 different groups (Google Project Zero, Paul Kocher et al.)

- Meltdown
  - Discovered independently by 3 different groups (Google Project Zero, Cyberus Tech., Graz Uni.)
Spectre & Meltdown

• Allows unprivileged processes to read kernel memory (or memory in other VMs)
  – Why is this a problem?
• In some cases, attackers running just JavaScript in your browser can “leak” information
Processors Affected

- Every Intel x86 processor after the Pentium
- A ton of AMD x86 processors
  - AMD feigned ignorance for some time
- Many ARM processors
History of Processor Bugs

• XBOX 360 had a similar speculative execution bug! \((xdcbit\) instruction)
• fdiv bug
• F00F bug
Aside: SandSifter

- Employs novel technique to breadth-first traverse all interesting instructions in x86
- Meant to discover undocumented instructions and processor bugs
- [https://github.com/xoreaxeaxeax/sandsifter](https://github.com/xoreaxeaxeax/sandsifter)
Figure 1: Two-level pagewalks in 32-bit x86.

**Background Information**

- Virtual address: 12345678
- Physical address: 00abc678
- CR3: 00001000
- Page table walk

Operating Systems Pt. 2
Virtual Memory (motivation)

- Facilitates share-time systems
- Maintains illusion that each process has its own (32 or) 64-bit address space
  - In reality, confined to amount of RAM on system
- Sandboxes processes
Virtual Memory (deeper)

• Maps from virtual memory to physical memory
  – page_table[ hi_16_bits(addr) ] + lo_48_bits(addr)
    • Use hi bits as index
    • Use lo bits as displacement

• Translation-lookaside buffer is a kernel structure which caches page table lookups
Virtual Memory (even deeper)

- x86 has multiple page tables!
- Page table walks are handled in x86 by the CPU’s memory-management unit (MMU)
  - Transparently walks the page table on a page fault
Branch Prediction

• On a conditional jump, a naive CPU will stall the pipeline until the branch condition has been computed

• Alternatively, we can save cycles by guessing which branch will be taken
  — When we guess incorrectly, we must still stall
Branch Prediction

• What about indirect branches?
  – C++ virtual methods, et. al
  – Don’t know all the possible addresses it can jump to at runtime

• Focus of Spectre variant #2 and subsequent mitigation (retpoline)
Speculative Execution

- Natural extension to branch prediction
- Fetch *and execute* the guessed target instruction before we know the result of the comparison
Speculative Execution

- Natural extension to branch prediction
- Fetch \textit{and execute} the guessed target instruction before we know the result of the comparison
  - Meltdown and Spectre both target how the CPU doesn’t \textit{fully} throw away the results of the speculatively executed instruction
  - Specifically: cache effects
Speculative Execution

• We were warned about this in 1995!
• The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems, by O. Sibert, P. Porras, R. Lindell
Meltdown and Spectre
Meltdown

- Can be used to read any memory mapped in the page table
  - This *includes* kernel memory
  - Total break of certain kernel protections
- Somewhat slow - can leak things at gigabytes per second
- Byte-by-byte leak of existing data
Meltdown

- Set up an array of 256 pages, one for each possible byte
- Flush them from the cache
- Read a byte from kernel memory (executed speculatively)
Meltdown

- Index into the array using that byte — this will **load that one page into the cache**
- Check each page in the array, the one with the quickest access time is in the cache
- The index corresponds to the byte in kernel memory
Meltdown

C code:

```c
char b = *kernel_ptr;
char _ = array[b*0x1000];
```

assembly code:

```assembly
; al is byte from kernel memory
mov al, byte ptr [kernel_address]
shl rax, 0xc ; multiply by page size 0x1000
; access array, loads page into cache
mov bl, byte ptr [array+rax] ; access array
```

https://github.com/Hawkheart/meltdown/blob/master/meltdown.c
Spectre

• Specifically targets *branch prediction*
• Relies on code that already exists in target application
Fix(es)

- AMD turned off branch prediction in some cases
- *Limited* x86 microcode updates
- KPTI - Kernel Page Table Isolation
  - Whenever we’re about to schedule code, we remove all kernel page table entries
  - Subsequently flush the translation-lookaside buffer
Fix(es)

• Kernels should be compiled *without* code sequences which can be used by Spectre
  – Retpoline
Aftermath

- System calls are much slower now
- For IO-bound processes (lots of IO-bursts):
  - Around 30% slowdown
  - Short bursts of IO may take 2-4x longer
Is this System Vulnerable?

Linux: `cat /proc/cpuinfo`

cpuid level : 22
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 tsc x2apic pdcm pcid s m b invpcid_single pti tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm
window hwp_epp
bugs : cpu_meltdown spectre_v1 spectre_v2
bogomips : 5184.00
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management: