

Side channels and transient execution

CSci 2021: Machine Architecture and Organization
December 10th, 2018
Stephen McCamant

Today

- Side and covert channels
- Transient execution
- Transient execution and kernel isolation: Meltdown
- Transient execution and software checks: Spectre
- Fixes, lessons learned, and the future

Information protection

- Most security goals relate to one of two properties
 1. Keep adversaries from taking control
 2. Protect sensitive (secret private, confidential, etc.) information from being revealed
- Buffer overflows were most commonly a problem for #1
 - Though read buffer overflows can be bad for #2, like “Heartbleed”
- Each goal eventually depends on the other
 - An adversary with control can send access information directly
 - Control is protected by secrets like passwords
- Today is about security problems of the #2 variety

Side channels

- A *side channel* is an unexpected way in which a system reveals information, different from how information is intentionally output
- Analog side channels are mediated by the physical world outside the machine, e.g.:
 - Sound of the hard-disk running
 - Power usage
- Digital side channels reveal information while staying inside the computer abstraction, e.g.:
 - You can’t read a file, but the error message reveals that it exists
 - Running time of an operation depends on what else is running

Side channels vs. covert channels

- In a side channel, information is revealed from an unsuspecting victim.
 - Sound of many people erasing indicates that an exam question is difficult
- In a covert channel, the source of the information is working together with the receiver to transmit it when they shouldn’t.
 - Cough once if the answer is “true”, twice if it is “false”
- Often the channel itself is the same, it just differs how you use it
 - And not everyone is careful about this distinction

“Architectural”

- Recall that the instruction-set architecture is an abstraction that hides details
 - Above the line: programmer visible state
 - Below the line, pipelining, caches, etc.
- Another form of this terminology distinction you will hear is:
 - “Architectural” means the above-the-line view
 - “Micro-architectural” means the below-the-line view
- If information is available only because of a micro-architectural behavior, that’s likely a side channel

Cache timing side channels

- Micro-architectural side channels are a problem of growing concern recently
- **Maybe the worst in terms of being pervasive and high-bandwidth is the timing of cache operations**
 - Every memory access uses caches
 - Cache performance is based on history of previous operations
 - Caches hold everyone's data without separation
 - The speed of operations is easy to measure
- **Basic idea: timing how long my memory accesses take reveals information about your memory accesses**
- **Directly reveals only addresses, not data contents**

Computer Science 2021

7

Secret information in addresses

- **The addresses of instruction accesses reveal what code your programming is running**
 - The grading function might have a branch that is only taken when a student qualifies for extra credit
- **The addresses of data accesses reveal what data your program is accessing**
 - Converting a numeric grade into a letter grade might use an array indexed by numeric grade
- **Often the most practically important victims are functions for encrypting data based on a small secret key**
 - Square-multiply algorithm in RSA depends on key bits
 - AES implementation uses a "T table" indexed based on unencrypted bytes and key

Computer Science 2021

8

Example technique: "prime + probe"

1. Attacker does a lot of memory accesses to fill up the cache with its own data ("prime")
 2. Wait and let the victim perform a memory access of its own
 3. The attacker retries accessing all of its data, and measures how long the accesses take ("probe")
- **If one of the pieces of the attacker's data is slow to access, that indicates that it had been evicted to replace it with some of the victim's data**

Computer Science 2021

9

Cache covert channel sender

- **In a covert channel, you can design a memory access to maximize cache information leakage**

```
int array[1024];
int secret = get_secret();
array[secret * 16]++;
```

- **Multiplying by 16 ensures that each different secret value indexes a different 64-byte cache block**
 - Commonly the channel does not reveal the offset within a block

Computer Science 2021

10

Today

- Side and covert channels
- **Transient execution**
- Transient execution and kernel isolation: Meltdown
- Transient execution and software checks: Spectre
- Fixes, lessons learned, and the future

Computer Science 2021

11

Transient execution: basic idea

- **There are several micro-architectural reasons why the CPU might do some steps of execution of instructions, but ultimately discard them**
 - Instruction executions that do not architecturally matter are called "transient" or "speculative"
- **Transient instructions have no architectural effect. But if they have a micro-architectural effect, that can be a side/covert channel**
- **This leads to some surprising vulnerabilities that were made public just under a year ago**

Computer Science 2021

12

Reasons for transient execution

- **Out-of-order execution**
 - CPU starts executing instructions out of order, when their input data is ready
- **Late recognition of exceptions**
 - A CPU may not decide that an instruction will raise an exception until it is retired
- **Branch prediction**
 - CPU guesses which side of a branch will be taken, and starts speculatively executing it before the branch condition is evaluated

Transient execution and memory

- **Transient execution includes speculative loads from memory**
 - Important for performance, similar to pre-fetching
- **Transient stores generally not sent outside the processor core**
 - Less important for performance, since stores don't have many dependencies
 - Stores will be buffered and sent to cache and memory on retirement
 - Transient stores can affect transient loads via store-to-load forwarding
- **Exceptions from transient accesses are ignored**

Announcements intermission 1: HA5

- **Our release of HA5 had an inconsistency that made the performance scores look worse than they should have**
 - Driver was set up for -O2, but Makefile had no optimization

	Baseline (0/55)	Full credit (55/55)	Extra credit (65/55)
-O0	24,624 KCEST	1244 KCEST	759 KCEST
-O2	10,985 KCEST	759 KCEST	

- **Grading will still be based on no optimization**
 - Instructions and driver.py have been updated
- **If you already have or still want to improve performance up to the mistaken goal, up to 10 extra credit points are available**

Announcements 2: lecture Wednesday

- **Lecture Wednesday will be a review session for the final exam**
 - Outline of material, discussion of rules and style of the exam, etc.
- **We will also do course evaluations**
 - Think about things I or the TAs did that were good or bad for your learning, and what we might do differently next time
 - Bring a blue/black pen or a pencil to fill out forms
 - Evaluations are important as we continue to explore how to make the course better, so thanks for spending time on them

Today

- Side and covert channels
- Transient execution
- **Transient execution and kernel isolation: Meltdown**
- Transient execution and software checks: Spectre
- Fixes, lessons learned, and the future

Kernel and user memory

- **In many recent systems, kernel and user memory live in the same virtual address space**
 - E.g., x86-32 Linux used low 3GB for user, top 1G for kernel
 - Makes it easier for kernel to transfer data to/from processes
 - No need to flush the TLB when making a system call
- **Kernel/user separation still important for security**
 - Kernel can hold system secrets, and other users' data
- **A bit in the page table entry ("U/S" on x86) distinguishes which pages are for kernel only**
 - Attempted access to kernel data from user program leads to page fault

Timing of a page fault

- **Problem:** the page fault is often recognized not when the access occurs, but when the faulting instruction retires
- Instructions after an illegal kernel memory access will be transient
- But, these transient instructions can still have micro-architectural effects
- **Attack idea:**
 - Write cache covert channel code using the result of a faulting kernel memory access
 - Recover from the fault, and then look for the side effect of the transient access in the cache

Meltdown attack structure

```
int array[1024];
prime_cache(array);
int secret = *kernel_mem_ptr;
array[secret * 16]++; /* transient only */
/* recover here after segfault */
probe_cache(array);
```

- You might be surprised that this works: so were the people who first found it in 2017
 - This version affects Intel processors but not AMD-compatible ones
- **Ethics note: don't try something like this on a shared computer like a lab machine**
 - Violates Labs and University rules, might hurt other users
 - (Also, probably patched by now.)

Today

- Side and covert channels
- Transient execution
- Transient execution and kernel isolation: Meltdown
- Transient execution and software checks: Spectre
- Fixes, lessons learned, and the future

Example: JavaScript bounds check

- Your web server runs JavaScript code from untrusted sources like advertisers, so must enforce security at runtime
- For instance, JavaScript arrays have runtime bounds checks in the C implementation

```
if (index < 0 || index > ary_size) {
    raise_js_error();
} else {
    void *value = raw_array[index];
    /* ... */
}
```

Branch prediction and bounds check

```
if (index < 0 || index > ary_size) {
    raise_js_error();
} else {
    void *value = raw_array[index];
    /* ... */
}
```

- **Branch prediction helps ensure the cost of check is low**
 - Benign JS code will only access indexes in bounds
 - Branch will be predicted in-bounds
 - Execution can continue beyond the check
- **Architecturally, the check is still enforced**
 - Out of bounds access will mean prediction is incorrect, discarded
 - Okay for buggy JS code to run slower

Dangers of branch speculation

- **Problem:** code executed after the mis-speculated branch could still have a micro-architectural effect
- For instance, leaking information via a cache access
- The protection being subverted here is JavaScript's
 - For instance, attacker could read data elsewhere in the web browser, like your banking password in another tab
- **Real JavaScript engines are just-in-time compilers, which actually makes the attack easier**

Example Spectre JS attack

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096) | 0) & (32*1024*1024-1)) | 0;
  localJunk ^= probeTable[index|0];
} /* Kocher et al. Listing 2, for Chrome 62 */
```

- “| 0” is a hint to JavaScript to compile into integers
- First do many examples with index in bounds for simpleByteArray, then one when it is out of bounds
- Multiplication by 4096 is similar to multiplication by 16 in earlier examples
- The JIT compiler determines it doesn't need any other bounds checks
- More details needed to make work in practice

Other malicious branch training

- Similar attacks are possible even when the attacker doesn't control earlier executions of the vulnerable branch (e.g., a branch in another process)
- Just need to mis-train the processor to trigger a chosen prediction
 - Branch prediction uses cache-like structures that are susceptible to collisions
- Can even make an indirect jump go to an instruction of attacker's choosing

Today

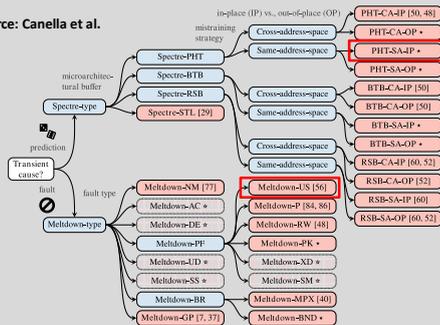
- Side and covert channels
- Transient execution
- Transient execution and kernel isolation: Meltdown
- Transient execution and software checks: Spectre
- Fixes, lessons learned, and the future

Brief history of Meltdown and Spectre

- Attacks were discovered independently in 2017
 - By both academic and industry researchers
 - There was also a public blog post about a similar idea that didn't work
- Long “responsible disclosure” period while software and hardware makers worked on defenses
 - Including Linux switching to separating kernel address spaces
- Announcement date moved up to January 3rd, 2018 as information started to leak out via media sources
- Discovery of variants and developing protections are ongoing today

Classification of known attacks

Source: Canella et al.



Software patches

- Meltdown: keep kernel address space separate
 - Made default in Linux and Windows in late 2017
- Spectre: various
 - Chrome puts tabs in separate processes
 - Special compilation techniques can frustrate branch prediction
- Software patches tend to be incomplete
 - New attack variants have required new defenses
 - But, important for fast reaction
- So far (fingers crossed), these attacks have been blocked before being widely exploited

Hardware fixes

- **Best fixes are at the CPU design level, but this is a long process**
- **Common so far: microcode patches**
 - CPU vendors take advantage of existing configurability mechanisms to block some attacks
 - Sometimes takes form of optional security checks enabled by the OS, at a performance cost
- **Deeper micro-architectural changes will allow protection with less overhead**
 - But transient execution is widespread and critical for performance, so how to strike the best balance is a complex problem

Takeaways

- **Computer system design is challenging because what's below the abstract barrier can end up mattering a lot**
- **Problems can arise from unusual combinations of existing features**
- **We want to improve performance, but breaking security or correctness is usually going too far**
- **Hardware architects need better ways to assess the security impacts of micro-architectural decisions**