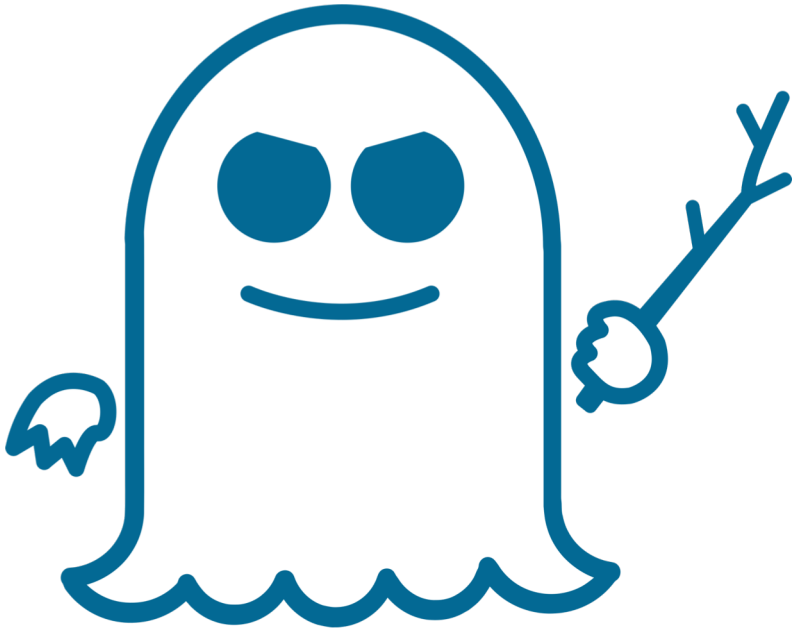
Meltdown and Spectre

CYBR 200 | Fall 2018 | University of the Pacific | Jeff Shafer



Meltdown

January 2018

Capabilities

- **Read arbitrary physical memory (including kernel memory) as an unprivileged user process**
- Exploit scenarios
  - Privilege escalation
  - Container escape - Docker instances, Xen para-virtualized VMs, etc...
  - Find password hashes, private keys, or just dump all of physical memory and use your favorite forensics tools (e.g. *Volatility)*

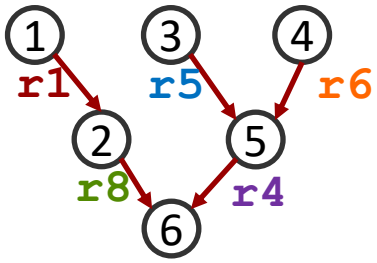
Attack Mechanism

- **Out of order execution** of instructions is used to leak data via processor covert channel (cache lines)

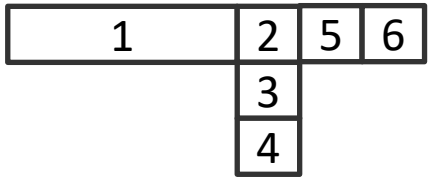
```
(1)  r1 = r4 / r7
(2)  r8 = r1 + r2
(3)  r5 = r5 + 1
(4)  r6 = r6 - r3
(5)  r4 = r5 + r6
(6)  r7 = r8 * r4
```

Assume divide is much slower

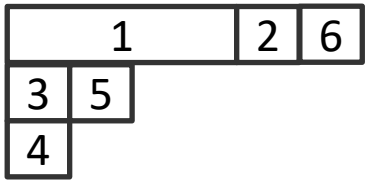
Data Flow Graph:



In Order Execution:

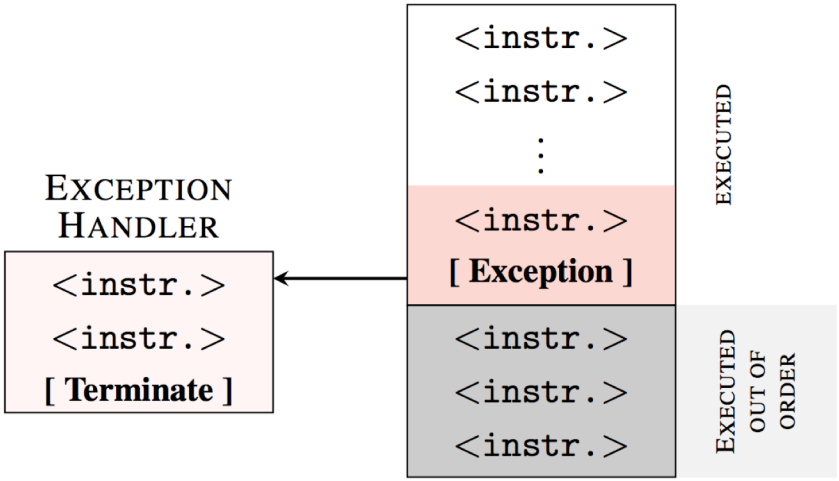


Out of Order Execution:

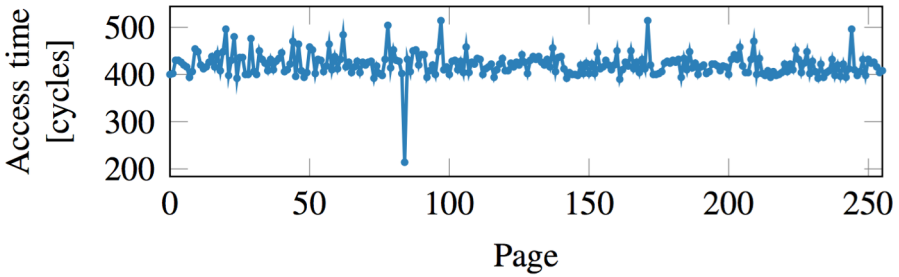


Exploiting Out of Order Execution

```
1    raise_exception();
2    // the line below is never reached
3    access(probe_array[data * 4096]);
```



Proof the post-exception execution happened:



Security Problem: Out of order execution (if not "retired") has no impact on the CPU architectural state (registers, memory). Safe?
Not fully! It **does** affect the CPU micro-architectural state (caches…)

Cache Attacks

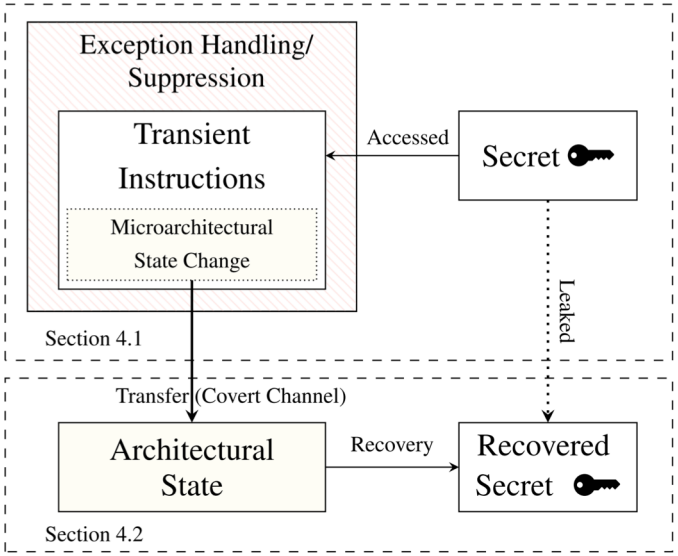
- Exploit timing differences caused by memory caches
- Attacker frequently flushes a targeted memory location using *cflush* instructions
  - Measure time taken to reload data with high precision
  - Infer whether the data was loaded into cache by another process in meantime

Covert channel

- Attacker controls both sides but can't communicate openly between them due to security controls
  - One part induces the side effect
  - One part measures the side effect
- Works with cache attacks!

Meltdown Attack

```
1 ; rcx = kernel address to target
2 ; rbx = probe array for covert channel
3 retry:
4 mov al, byte [rcx]     ; try to read from kernel addr (1 byte)- exception!
5 shl rax, 0xc           ; This executes! Multiply secret by page size (4kB)
6 jz retry               ; Ignore zeros (filtering noise)
7 mov rbx, qword [rbx + rax] ; This executes! Save secret to covert channel
```



Despite exception (line 4), Intel CPU continues executing any out of order instructions (line 5-7) that were already underway.

The exception is only noticed when instructions are *retired.* Architectural state is not affected, but micro-architectural state is.

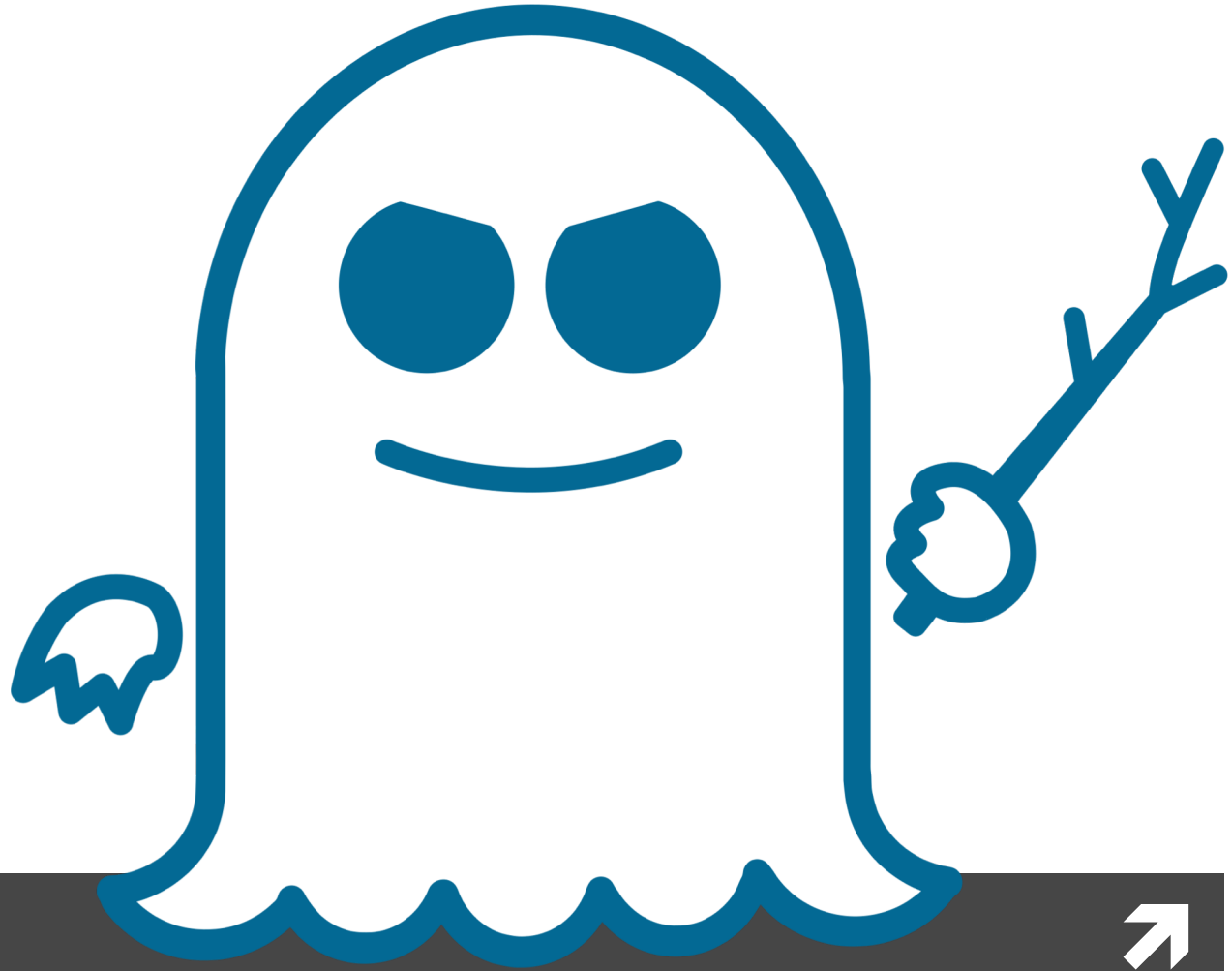
(Think of this as a *race condition* – will the exception be noticed before the subsequent instructions?)

Demo

https://github.com/IAIK/meltdown

Mitigation

- Kernel updates can mitigate security flaw
  - Patched in Linux with KAISER/KPTI (Kernel Page Table Isolation)
  - Patched in Windows, OS X
  - Microcode updates for affected CPUs
- What does KPTI patch do?
  - Kernel addresses are *traditionally* mapped into user space processes to reduce overhead in system calls. This is not supposed to be a problem! The descriptor privilege level (DPL) prevents ring-3 user processes from touching ring-0 kernel memory
  - Kernel page table isolation (KPTI) removes the mapping of kernel memory in user space. Because it's no longer mapped, it can't be read by Meltdown. The processor is **still vulnerable** to this arbitrary read exploit, but we've removed (nearly) all of the addresses an attacker might read from. So the attack is no longer practical
  - Open question - some (small) kernel memory must be mapped into user memory, e.g. interrupt handlers. Can this be exploited by clever attacker?



Spectre

January 2018

Capabilities

- **Read arbitrary memory from current process (not the kernel, not other processes)**
- Exploit scenarios:
  - Escape sandbox (e.g. JavaScript execution in browser can read process memory outside of sandbox)
  - Leaking addresses of ASLR-protected user processes to facilitate remote code execution
    - There's a class of vulnerabilities that are difficult to exploit due to ASLR that may suddenly become feasible…

Attack Mechanism

- Abuse of **branch prediction** and **speculative execution** to leak data via processor covert channel (cache lines)
- Feasible for Intel, AMD, ARM processors

```
(1) if r1 != 0 then goto 4
(2) r6 = r3 + 1
(3) goto 5
(4) r6 = r3 - 1
(5) r9 = r5 + 1
(6) if r6 == 0 then goto 10
```

Branch prediction: Record history of path taken/not taken and guess prior to knowing correct answer

Speculative execution: Execute instructions that may not be needed (wrong branch)

Attack Details – Bounds Check Bypass

- This attack variant allows malicious code to circumvent bounds checking features built into most binaries. Even though the bounds checks will still fail, the CPU will speculatively execute instructions after the bounds checks, which can access memory that the code could not normally access. When the CPU determines the bounds check has failed, it discards any work that was done speculatively; however, some changes to the system can be still observed (in particular, changes to the state of the CPU caches). The malicious code can detect these changes and read the data that was speculatively accessed.

- The primary ramification is that it is difficult for a system to run untrusted code within a process and restrict what memory within the process the untrusted code can access.

- In the kernel, this has implications for systems such as the extended Berkeley Packet Filter (eBPF) that takes packet filterers from user space code, just-in-time (JIT) compiles the packet filter code, and runs the packet filter within the context of kernel. The JIT compiler uses bounds checking to limit the memory the packet filter can access, however, this allows an attacker to use speculation to circumvent these limitations.

Attack Details – Bounds Check Bypass

```
struct array {
 unsigned long length;
 unsigned char data[];
};
struct array *arr1 = ...; /* small array */
struct array *arr2 = ...; /* array of size 0x400 */
/* >0x400 (OUT OF BOUNDS!) */
unsigned long untrusted_offset_from_caller = ...;
if (untrusted_offset_from_caller < arr1->length) {
 unsigned char value = arr1->data[untrusted_offset_from_caller];
 unsigned long index2 = ((value&1)*0x100)+0x200;
 if (index2 < arr2->length) {
   unsigned char value2 = arr2->data[index2];
 }
}
```

Uncached locations / slow to access…

Location attacker wants to read…

Covert channel

Speculative because arr1->length hasn't returned yet

After execution returns to non-speculative path, `arr2->data[index2]` is still in cache!
Attacker measures time to read `arr2->data[0x200]` versus `arr2->data[0x300]`
Result: Determine whether `index2` was `0x200` or `0x300`
Result: Determine whether `arr2->data[untrusted_offset_from_caller]&1` is 0 or 1

https://googleprojectzero.blogspot.ca/2018/01/reading-privileged-memory-with-side.html

Mitigation – Bounds Check Bypass

- Analysis and recompilation so that vulnerable binary code is not emitted
  - OS requires fixes
  - Applications which execute untrusted code require fixes
- Solution: `lfence` serializing instruction
  - Grinds processor to a halt, forcing it to complete all *prior* instructions before completely
  - Kills out of order & speculative execution temporarily
  - Must (carefully) add in all the right places – don't miss any!

Retpoline – "Return Trampoline"

A common C++ indirect branch

```
class Base {
 public:
  virtual void Foo() = 0;
};
class Derived : public Base {
 public:
  void Foo() override { … }
};
Base* obj = new Derived;
obj->Foo();
```

A compiled x86 indirect branch

```
jmp *%rax; /* indirect branch to
the target referenced by %rax */
```

Indirect branch construction

`jmp *%r11`	`call set_up_target;` (1)

Captures speculative execution path	`capture_spec:` (4)
	`  pause;`
	`  jmp capture_spec;`

Retpoline equivalent to indirect branch	`set_up_target:` `  mov %r11, (%rsp);` (2) `  ret;` (3)

Protects against *branch target injection*, another Spectre-class attack

https://support.google.com/faqs/answer/7625886

Comparison

	Meltdown	Spectre
Read (leak) kernel memory?	Yes	No
Patched with KPTI	Yes	No
Read (leak) user memory	Yes	Yes
Exploit remotely	Sometimes	Yes
Likely target	Kernel memory	Browser memory
Practical attacks against	Intel CPUs	Intel, AMD, ARM CPUs

Indicators of Compromise for Meltdown and Spectre? "Near Non-Existent" - Have nation states been exploiting this since the embargoed information was distributed to key companies? (Surely they have access to private kernel email development lists, yes?)

New (Related) Attacks

- **Meltdown** (January 2018)
- **Spectre** (January 2018)
- **Spectre 2** (Marchy 2018)
  - https://arstechnica.com/gadgets/2018/03/its-not-just-spectre-researchers-reveal-more-branch-prediction-attacks/
- **Speculative Store Bypass (SSB)** – May 2018
  - https://arstechnica.com/gadgets/2018/05/new-speculative-execution-vulnerability-strikes-amd-arm-and-intel/
- **Speculative Buffer Overflows** – July 2018
  - https://arstechnica.com/gadgets/2018/07/new-spectre-like-attack-uses-speculative-execution-to-overflow-buffers/

New (Related) Attacks

- All are permutations on same concept – divergence between:
- **Architectural behavior**
  - What the assembly programmer expects to happen (execution in order)
- **Micro-architectural behavior**
  - What the processor *actually* does behind-the-scenes for maximum performance
  - Example: Loading a value from memory. Programmer expects processor will wait until address is known, but processor might *speculate* on the address and do a load early. If wrong address, processor will void result and try again
    - Problem: *Side effects (e.g. cache state)*

References

- Summary Presentation
  - https://www.renditioninfosec.com/2018/01/meltdown-and-spectre-vulnerability-slides/
  - Jake Williams (SANS & Rendition InfoSec)
- Official website w/academic papers
  - https://spectreattack.com/
- Demo code by paper authors
  - https://github.com/IAIK/meltdown
- Google Project Zero
  - https://googleprojectzero.blogspot.ca/2018/01/reading-privileged-memory-with-side.html