Retpoline

Meltdown & Spectre

- Variant 1: bounds check bypass (CVE-2017-5753) - Spectre
  - Speculation barriers?
- **Variant 2: branch target injection (CVE-2017-5715) - Spectre**
  - Intel IBRS patches, Google “**Retpoline**”
- Variant 3: rogue data cache load (CVE-2017-5754) - Meltdown
  - Kernel Page Table Isolation (KPTI)

CVE-2017-5715: branch target injection

- Branch predictors:
  - Generic predictor
  - **Indirect call predictor**
  - Return predictor
- victim (hypervisor/kernel)에 있는 indirect branch target을 캐시에서 없앤다
- 이 target을 찾는데 100+ cycle이 걸린다
- 그러므로 100+ cycle동안 speculative execution을 한다
- ???
- Cache timing -> memory read! (~1500B/s)

Indirect branch

Example: 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();
```

- Indirect branch prediction을 없애야 한다
- 어떻게?
- Retpoline!

Intel?

Intel's Spectre fixes are 'complete and utter garbage,' says Linux inventor

*'What the f*ck is going on'*

> The patches do things like add the garbage MSR writes to the kernel entry/exit points. That's insane. That says "we're trying to protect the kernel". We already have retpoline there, with less overhead.

> They do literally insane things. They do things that do not make sense. That makes all your arguments questionable and suspicious. The patches do things that are not sane.

Retpoline 이란?

return + trampoline

Retpoline

- ret도 indirect branch의 일종
- 하지만 다른 indirect branch와 달리 단순한 스택으로 구현된다
- 그러므로 target이 캐시된다
  - **Intel: return stack buffer (RSB)**
  - AMD: return address stack (RAS)
  - ARM: return stack
- Speculative execution을 할 때는 이 cache된 address를 쓴다

Retpoline

이런 코드를...

Example: A compiled x86 indirect branch

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

Retpoline

이렇게 바꾸는것

Indirect branch construction

```
jmp *%r11
```

```
 call set_up_target;  (1)
capture_spec:         (4)
  pause;
  jmp capture_spec;
set_up_target:
  mov %r11, (%rsp);   (2)
  ret;                (3)
```

Retpoline

참고로 call은 이렇게

Indirect call construction

```
call *%r11
```

```
    jmp set_up_return;
inner_indirect_branch:
    call set_up_target; }
capture_spec:           }
    pause;              }
    jmp capture_spec;   } Indirect branch
set_up_target:          } sequence.
    mov %r11, (%rsp);   }
    ret;                }
set_up_return:
    call inner_indirect_branch; (1)
```

Code duplication?

Out of line construction	
`jmp *%r11` `call *%r11`	`jmp retpoline_r11_trampoline;` `call retpoline_r11_trampoline;`

Shared Trampoline	`retpoline_r11_trampoline:`
	`  call set_up_target;`
	`capture_spec:`
	`  pause;`
	`  jmp capture_spec;`
	`set_up_target:`
	`  mov %r11, (%rsp);`
	`  ret;`

Performance?

- 구글에서 실험해본 결과 retpoline은 "within cycles of a native indirect branch"
  - (branch prediction은 꺼놓은 상태에서)
- 루프 안에는 pause
- Align
- Manual prediction

Example constructions including alignment prefixes

```
  jmp set_up_return;
.align 16;
inner_indirect_branch:
  call set_up_target;
capture_spec:
  pause;
  jmp capture_spec;
.align 16;
set_up_target:
  mov %r11, (%rsp);
  Ret
.align 16;
set_up_return:
  call inner_indirect_branch;
```

Example of an indirect jump with manual prediction

```
cmp %r11, known_indirect_target
jne retpoline_r11_trampoline
jmp known_indirect_target
```

Stack refill

- ret도 indirect branch이니까 speculation을 막아야 한다
- 어떻게?
- 다른 branch predictor을 못쓰게! (RSB/RAS가 비어있으면 안된다)

Example x86 refill sequence

```
   mov $8, %rax;
    .align 16;
 3: call 4f;
3p: pause; call 3p;
    .align 16;
 4: call 5f;
4p: pause; call 4p;
    .align 16;
 5: dec %rax;
    jnz 3b;
    add $(16*8), %rsp;
```

감사합니다!