“熔毁”漏洞原理

极目楚天舒@看雪学院

内容

- “熔毁”漏洞简介
- 流水线对异常的处理
- Linux地址空间映射
- Flush + Reload攻击
- “熔毁”与“幽灵”的异同
- 有待验证的一个想法

“熔毁”漏洞简介

Meltdown破坏了位于**用户**和**操作系统**之间的基本隔离，此攻击允许程序访问内存，因此其他程序以及操作系统的敏感信息会被窃取。这个漏洞“熔化”了由硬件来实现的安全边界。允许低权限用户级别的应用程序“越界”访问系统级的内存，从而造成数据泄露。

“熔毁”漏洞的**根本原因**是CPU流水线执行指令过程中异常信号的发出和乱序执行之间存在竞争 。当用户程序访问内核地址空间时，CPU的乱序执行使得本来应该被地址异常阻止的非法访问指令在流水线中得到执行，虽然这个结果最后会被丢弃，但是乱序执行导致了CPU微架构状态的改变。攻击者可以通过探测微架构状态的改变获知内核地址处的信息。

流水线对异常的处理

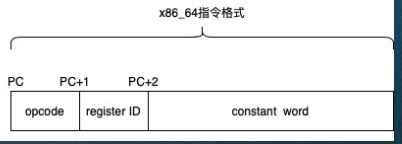
- 内部异常的产生

异常的产生：1）halt指令　　2）非法指令　　3）试图访问一个非法地址

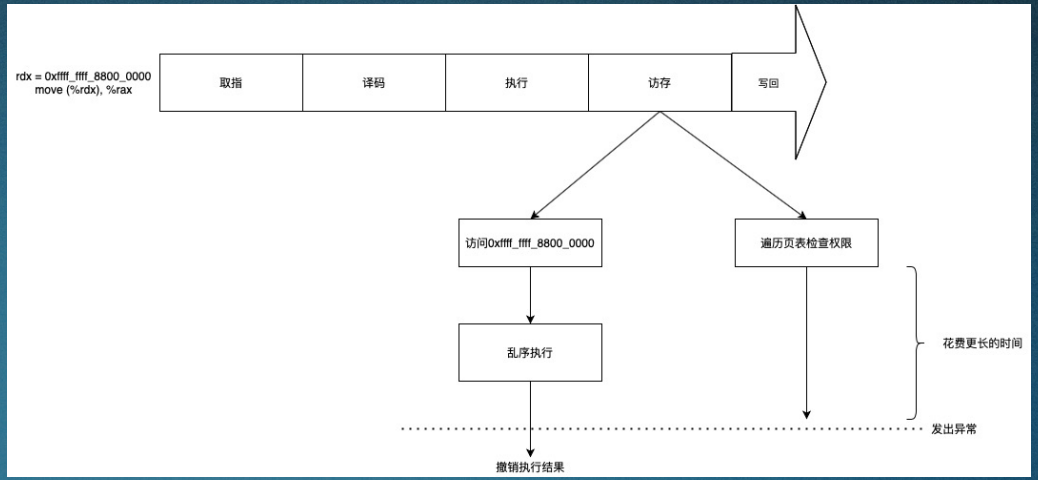
- 流水线对内部异常的处理

如果一条指令在流水线中的某一个阶段产生了一个异常，异常状态和指令信息会沿流水线传播，一直到达流水线的写回阶段，这个时候流水线逻辑就会发现异常停止指令的执行并清空流水线

阶段	OPq rA, rB	subq %rdx, %rbx
取指	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2	icode:ifun ← M_1[0x014] = 6:1 rA:rB ← M_1[0x015] = 2:3 valP ← 0x014+2 = 0x016
译码	valA ← R[rA] valB ← R[rB]	valA ← R[%rdx] = 9 valB ← R[%rbx] = 21
执行	valE ← valB OP valA Set CC	valE ← 21 − 9 = 12 ZF ← 0, SF ← 0, OF ← 0
访存		
写回	R[rB] ← valE	R[%rbx] ← valE = 12
更新 PC	PC ← valP	PC ← valP = 0x016



流水线对异常的处理



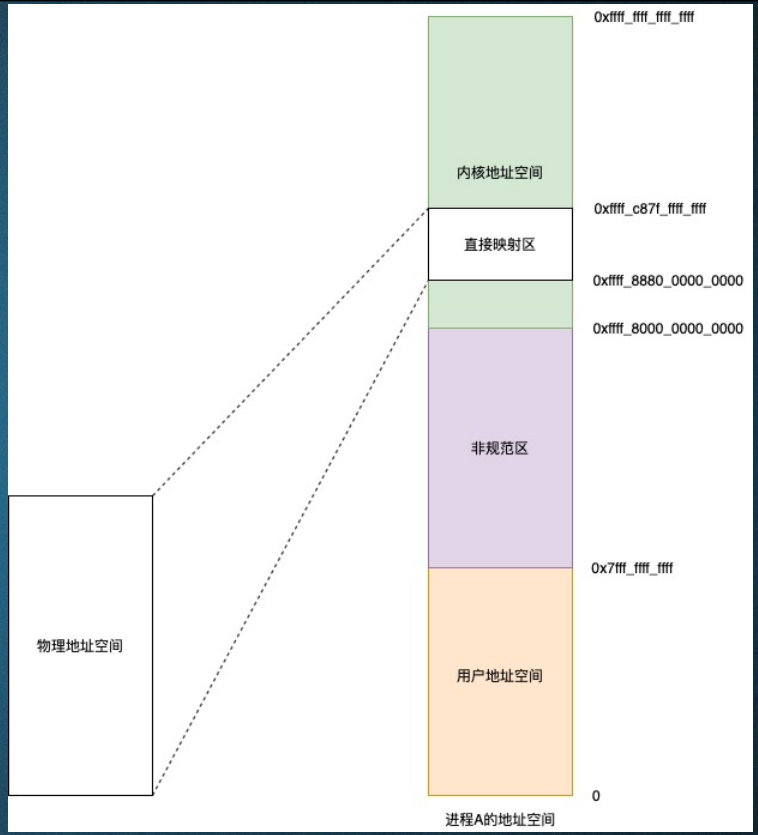
用户空间通过mov指令访问内核空间地址时，在流水线的访存阶段会检测到这一非法访问。具体的过程是1）执行单元从内核地址处获得数据 2）访存的同时执行单元访问页表进行权限检查 3）检测到从用户空间访问内核的行为并发出异常 4）在写回阶段停止指令的执行并清空流水线。

问题就出在第二步执行权限检查要通过**访问页表项**来实现，这里又要进行一次访存，所以就会有一个时间窗口使得异常指令之后的指令得到执行并且利用第一步中获得的内核地址处的内容！

Linux地址空间映射

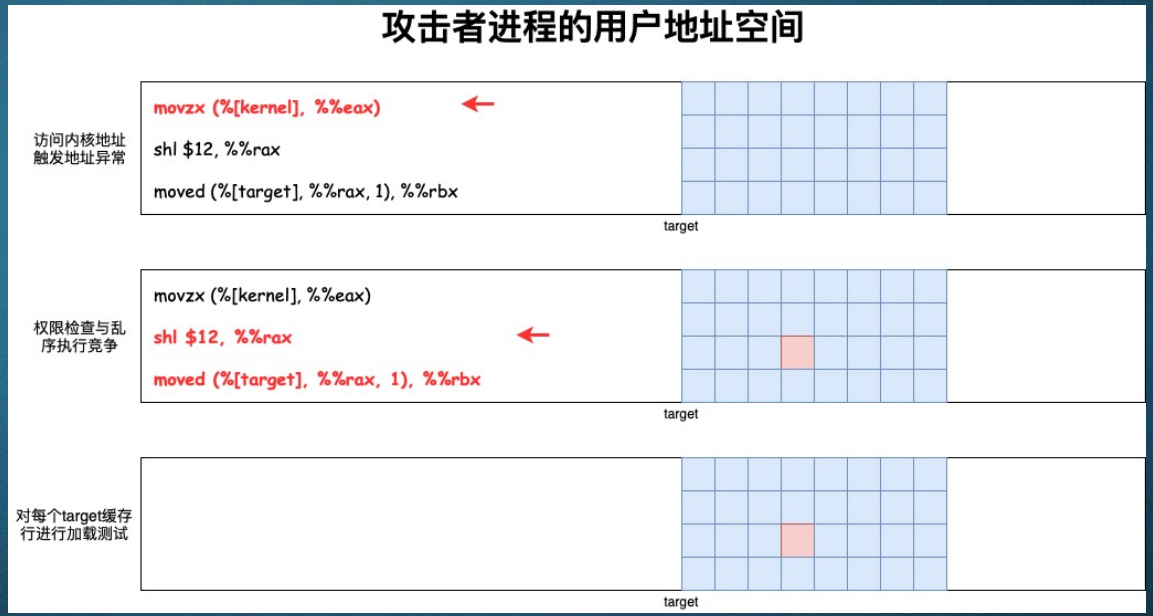
Documentation > x86 > x86_64 > mm.txt

Start addr	Offset	End addr	Size	VM area description
0000000000000000	0	00007fffffffffff	128 TB	user-space virtual memory, different per mm
0000800000000000	+128 TB	ffff7fffffffffff	~16M TB	... huge, almost 64 bits wide hole of non-canonical virtual memory addresses up to the -128 TB starting offset of kernel mappings.
				Kernel-space virtual memory, shared between all processes:
ffff800000000000	-128 TB	ffff87ffffffffff	8 TB	... guard hole, also reserved for hypervisor
ffff880000000000	-120 TB	ffff887fffffffff	0.5 TB	LDT remap for PTI
ffff888000000000	-119.5 TB	ffffc87fffffffff	64 TB	direct mapping of all physical memory (page_offset_base)
ffffc88000000000	-55.5 TB	ffffc8ffffffffff	0.5 TB	... unused hole
ffffc90000000000	-55 TB	ffffe8ffffffffff	32 TB	vmalloc/ioremap space (vmalloc_base)
ffffe90000000000	-23 TB	ffffe9ffffffffff	1 TB	... unused hole
ffffea0000000000	-22 TB	ffffeaffffffffff	1 TB	virtual memory map (vmemmap_base)
ffffeb0000000000	-21 TB	ffffebffffffffff	1 TB	... unused hole
ffffec0000000000	-20 TB	fffffbffffffffff	16 TB	KASAN shadow memory
				Identical layout to the 56-bit one from here on:
fffffc0000000000	-4 TB	fffffdffffffffff	2 TB	... unused hole vaddr_end for KASLR
fffffe0000000000	-2 TB	fffffe7fffffffff	0.5 TB	cpu_entry_area mapping
fffffe8000000000	-1.5 TB	fffffeffffffffff	0.5 TB	... unused hole
ffffff0000000000	-1 TB	ffffff7fffffffff	0.5 TB	%esp fixup stacks
ffffff8000000000	-512 GB	ffffffeeffffffff	444 GB	... unused hole
ffffffef00000000	-68 GB	fffffffeffffffff	64 GB	EFI region mapping space
ffffffff00000000	-4 GB	ffffffff7fffffff	2 GB	... unused hole
ffffffff80000000	-2 GB	ffffffff9fffffff	512 MB	kernel text mapping, mapped to physical address 0
ffffffff80000000	-2048 MB			
ffffffffa0000000	-1536 MB	fffffffffeffffff	1520 MB	module mapping space
ffffffffff000000	-16 MB			
FIXADDR_START	~-11 MB	ffffffffff5fffff	~0.5 MB	kernel-internal fixmap range, variable size and offset
ffffffffff600000	-10 MB	ffffffffff600fff	4 kB	legacy vsyscall ABI
ffffffffffe00000	-2 MB	ffffffffffffffff	2 MB	... unused hole



Flush+Reload攻击

```
"movzx (%[addr]), %%eax\n\t"// try to read from kernel
"shl $12, %%rax\n\t"
"movzx (%[target], %%rax, 1), %%rbx\n"//speculate execute
```



“熔毁”与“幽灵”的异同

- 相同点

都利用了处理器乱序执行的特点，使得在正常执行流中不可能得到执行的代码被加载到流水线中并得到执行，虽然执行结果得不到提交但影响了CPU的微架构状态。攻击者利用微架构状态的改变作为隐蔽信道实现数据的传输。

- 不同点

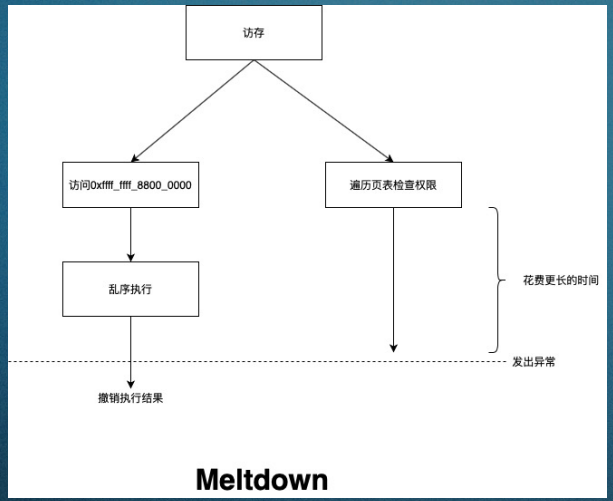
1. “幽灵”漏洞利用的前导是分支指令，利用的是分支预测执行；“熔毁”漏洞利用的前导是内部异常指令，利用的是乱序执行。

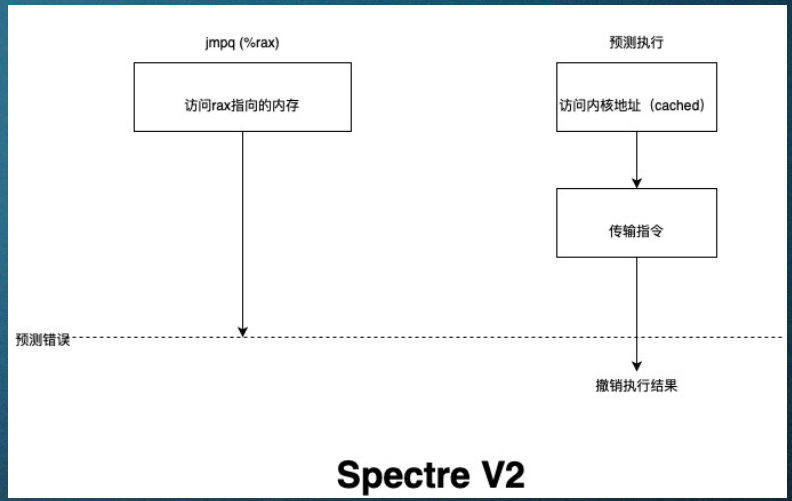
2. “幽灵”漏洞突破的进程之间的隔离，攻击者可以窥探目标进程内的数据；“熔毁”突破了操作系统内核态和用户态之间的内存访问隔离，攻击者可以获得内核地址空间的内容。

有待验证的一个想法

攻击者利用“幽灵”漏洞从受害者进程内导出内核地址空间的内容，利用“幽灵”漏洞突破用户态和内核态的地址空间隔离。

这个想法的假设基础是利用预测执行去访问内核地址空间并通过Flush+Reload构建隐蔽信道传输内核地址空间的内容。





谢 谢