Meltdown 漏洞阅读报告

PB16051448 赵敏帆

Meltdown 相关概念

为了弄清楚 Meltdown 的原理，需要先厘清如下几个概念：

- 计算机系统的安全性
- 乱序执行
- 地址空间
- 缓存攻击

计算机系统的安全性

计算机系统的安全性从根本上依赖于内存隔离，例如，内核地址范围被标记为不可访问，并且受到访问保护。内存隔离是当今操作系统的主要安全特征之一。操作系统确保用户应用程序不能访问彼此的内存，并防止用户应用程序读取或写入内核内存。这种隔离是计算环境的基石，允许在个人设备上运行多个应用程序，或者在云中的单台计算机上执行多个用户的进程。

在现代处理器上，内核和用户进程之间的隔离通常是由处理器的一个监视位（supervisor bit）来实现，该监视位定义内核的内存界面是否可以被访问。其基本思想是，只有进入内核代码时才能设置改位，切换到用户进程时该位被清除。该硬件特征允许操作系统将内核映射到每个进程的地址空间，并且从用户进程到内核具有非常高效的转换，例如用于中断处理。在实践中，从用户进程切换到内核时不会改变内存映射。

乱序执行

乱序执行是一种优化技术，它最大限度地利用 CPU 内核的所有执行单元，以便克服繁忙的执行单元的延迟。例如，内存读取单元需要等待从内存读取数据，现代处理器并没有延迟执行，而是乱序地执行操作，即它们预见性地将后续操作安排到处理器的空闲执行单元。CPU 不是严格按照顺序执行处理指令的，这会使得 CPU 的效率和性能非常的低下。只要资源可用，CPU 就会立即执行它们。当前操作的执行单元被占用时，其他执行单元可以向前排。只要结果遵循架构定义，指令就可以并行运行。

在实践中，支持乱序执行的 CPU 会在弄清楚是否允许之前，推测性地执行处理器的乱序逻辑处理指令。在此，对推测性执行给出一个更具限制性的定义，即分支之后的指令序列，并且用乱序执行这个术语来指代处理器已经提交所有先前指令的结果之前执行操作的任何方式。

> 乱序执行在我们所学的流水线 CPU 中有所表现。以我个人的观点看来，在上条指令暂未执行完毕时就已经开始接下来几条指令的部分流水段执行就是一种乱序执行，因为 CPU 并未完全按照顺序处理指令，而是通过最大限度地使用 CPU 的各个流水段资源，来压榨性能和效率，达成某种程度上的并行。

乱序执行的特性会导致有漏洞的 CPU 允许非特权进程将数据从特权（内核或物理）地址加载到临时 CPU 寄存器中。而且，CPU 甚至根据该寄存器值进一步执行计算，例如，根据寄存器值访问数组。如果某个指令不应该被执行，处理器就会通过简单地丢弃内存查找的结果（例如，修改的寄存器状态）来确保正确的程序执行。因此，在体系结构层面，不会出现安全问题。

但是乱序内存查找会影响缓存，而缓存又可以通过缓存侧信道进行检测。结果，通过乱序执行流中读取特权内存，攻击者就可以将整个内核内存转储出来。

地址空间

为了将进程彼此隔离，CPU 支持虚拟地址空间，并负责将虚拟地址转换为物理地址。虚拟地址空间被分成一组页面，可以通过多级页面转换表逐一映射到物理内存。转换表定义了实际的虚拟到物理映射，以及用于强制执行特权检查的保护特性，例如可读、可写、可执行和用户可访问。每个进程通过特殊的 CPU 寄存器，只能引用属于自己的虚拟地址空间的数据。每个虚拟地址空间本身被分割为用户部分和内核部分。

用户地址空间可以被正在运行的程序所访问，但是只有当 CPU 以特权模式运行时，才能访问内核地址空间。这是由禁用相应转换表的用户可访问属性的操作系统强制执行的。

缓存攻击

为了加速内存访问和地址转换，CPU 包含一些称为缓存的小内存缓冲区，用于存储常用数据。

缓存侧信道攻击利用缓存引入的时序差异。通过 Evict+Time、Prime+Probe、Flush+Reload 等攻击技术，可以达到单个缓存栈级别颗粒度的攻击。

以 Flush+Reload 为例。攻击利用共享的最后一级缓存，使用 clflush 指令刷新目标内存位置。通过测量重新加载数据所需的时间，可以确定数据是否由另一个进程同时加载到缓存中。

Meltdown 攻击示例分析

通过一个简单的 Meltdown 攻击，来了解一下它的工作原理

```
1 ;rcx = kernel address
2 ;rbx = probe array
3 retry:
4 mov al,byte[rcx]
5 shl rax,0xc
6 jz retry
7 mov rbx,qword[rbx+rax]
```

攻击描述

Meltdown 攻击包括三个步骤：

- 攻击者所选的内存位置的内容被加载到寄存器中，这是攻击者无法访问的
- 临时指令根据寄存器的秘密内容访问缓存行
- 攻击者利用 Flush+Reload 来确定被访问的 Cache 线路，因此秘密存储在选定的内存位置。

通过对不同内存位置重复这些步骤，攻击者可以转储内核内存，包括整个物理内存

第一步：读取秘密

通过第 4 行代码，将存储在 RCX 寄存器中的目标内核地址的字节值加载到 AL 表示的 RAX 寄存器的最低有效字节中。

当内核地址被加载到第 4 行时，由于乱序执行的缘故，CPU 很可能已经发出后续指令（第 5 到 7 行），使其作为乱序执行的一部分。

第二步：传输秘密值

在内存中分配一个探测数组，为了传输秘密值，临时指令序列（即第 5 到 7 行所示代码）包含基于秘密值（不可访问）计算的地址间接存储访问。

在第 5 行中，来自步骤一的秘密值被乘以页面大小，即 4KB，来确保足够大的空间距离，防止硬件预取器将相邻内存位置加载到缓存中。我们一次读取一个字节，假设有 4KB 的页面，那么探测阵列就是 256*4096 字节。

在第 7 行中，多重秘密值被添加到阵列基地址，形成隐蔽信道的目标地址。读取该地址来缓存相应的缓存行。临时指令序列会根据在步骤一中读取的秘密值影响缓存的状态。

第三步：接收秘密值

步骤二的临时指令序列被执行时，探测器阵列中被缓存的缓存行的位置仅取决于在步骤一中读取的秘密值，攻击者遍历探测器阵列的所有 256 个页面，并测量每个 Cache 的访问时间，可以得到在步骤二中被缓存的缓存行。缓存行的页面编号直接对应于秘密值。

通过这三步，攻击者得到了不可访问的地址空间中存储的内容。攻击者可以通过遍历所有不同的地址来转储整个内存。

Meltdown 攻击的应对策略

由于 Meltdown 攻击基于硬件本身，最重要的一个策略是完全禁止乱序执行，然而这对于性能的影响是毁灭性的，现代 CPU 的并行性不能再被利用，因此这不是一个可行的方案。

目前对于 Meltdown 攻击，用于减轻 KASLR 侧信道攻击的 KAISER 对策，歪打正着地阻止了 Meltdown 的攻击。

更现实的一个解决方案是，引入用户空间和内核空间的硬分割。这种解决方案对性能的影响是最小的，此外还确保了向后兼容性。