高技术通讯 2024 年 第 34 卷 第 5 期:439-452

doi:10.3772/j.issn.1002-0470.2024.05.001

处理器时间侧信道攻防技术综述①

唐博文② 武成岗 王 喆③

(中国科学院计算技术研究所处理器芯片全国重点实验室 北京 100190)

(中国科学院大学 北京 100049)

摘 要 现代处理器优化机制众多,设计人员在追求性能提升时,往往忽略背后的安全风险。时间侧信道攻击因其影响面广且隐蔽性好已成为最主要的安全威胁之一。随着瞬态执行攻击的出现,时间侧信道攻击的能力被进一步扩展,计算系统的安全基础被动摇。为此,处理器厂商及安全人员提出了大量防御机制。这些机制具有不同的防护能力及性能开销。与此同时,新的瞬态执行漏洞和隐蔽信道也不断被发现,已提出的防御机制被不断突破。围绕处理器时间侧信道攻防技术的博弈日益激烈。本文从基本攻击原理出发,对现有时间侧信道攻击进行了归纳总结,并在此基础上进一步分析了相关防御机制的保护能力和性能瓶颈,从而梳理出时间侧信道攻防技术的发展趋势,为未来软硬件系统开发和安全技术探索提供参考。

关键词 处理器微架构;时间侧信道攻击;隐蔽信道;瞬态执行攻击;投机执行;防御技术

性能是计算机系统发展的永恒主题。为了满足不断增长的性能需求,现代处理器引入了复杂的优化机制,比如用于挖掘指令级并发能力的投机执行和超标量技术、用于打破“存储墙”效应的高速缓存技术、用于提升资源利用效率的片上多核及超线程技术。然而,这些机制在使处理器获得极大性能增益的同时,也带来了一系列安全隐患。其中影响较广的就是侧信道攻击。

侧信道攻击是指不利用软硬件漏洞直接泄露敏感数据,而是通过测量某些环境变量,如电磁信号、温度、代码执行时间,间接推测出敏感数据的攻击。其中基于时间测量的侧信道攻击威胁更高。因为相比于电磁信号、温度等物理变量,代码执行时间的度量值更容易泄露到远端,并且和敏感数据的关联更紧密。

但是,受限于攻击场景和系统已有的防御机制,单一的时间侧信道攻击只能泄露特定敏感数据,比如键盘敲击位置、加密运算密钥、进程代码加载基地址等。相应的防御机制也没有被广泛部署。直到 2018 年,“熔断”(Meltdown)、“幽灵”(Spectre)等漏洞被曝出,复合了这类漏洞的时间侧信道攻击其能力被极大扩展。攻击者利用这些漏洞绕过上层其他防御机制,如边界检查、内核态-用户态权限检查,访问指定的敏感数据,再利用侧信道方法将敏感数据泄露出去。因为对敏感数据的访问是在投机执行中完成的,最终会被处理器清理,所以这类攻击也被称为瞬态执行攻击。

以瞬态执行攻击为代表的时间侧信道攻击,严重威胁了现有计算机系统的安全根基。为此,研究人员提出了多种防御机制。然而,这些机制往往针对特定的应用场景,有着不同的防御能力和性能瓶颈,缺乏宏观的对比分析。另一方面,新的漏洞和信道也层出不穷,现有的防御机制不断被突破。如何发掘新的攻击面并设计更加安全高效的防御机制是

① 国家自然科学基金青年基金(61902374)和国家自然科学基金联合重点基金(U1736208)资助项目。

② 男,1991 年生,博士生;研究方向:计算机体系结构,系统安全;E-mail:tangbowen@ict.ac.cn。

③ 通信作者,E-mail:wangzhe12@ict.ac.cn。

(收稿日期:2023-02-09)

困扰研究人员的难题。

本文通过原理分析,对现有时间侧信道攻防技术进行了全面归纳总结,并在此基础上对各种防御机制的防御能力和性能开销进行了横向对比,从而为处理器设计及软件开发提供参考,也为未来攻防技术的探索指明方向。同时,由于围绕时间侧信道的攻防对抗在高速发展,本文相比于已有的综述类工作[1-5]介绍了更多新兴技术,提供了针对这些技术内在发展趋势的更清晰的审视角度。

1　现代处理器微架构

现代处理器主频高、流水线深度大,控制流转移和访存延迟等事件会造成流水线阻塞,极大影响性能。为此,处理器设计者提出了多种优化技术,并最终形成了如图 1 所示的微架构,其中阴影部件为易失型信道,斜线 + 阴影部件为持久型信道,其余白色部件尚未发现可被利用。这些复杂设计正是时间侧信道攻击的起源。

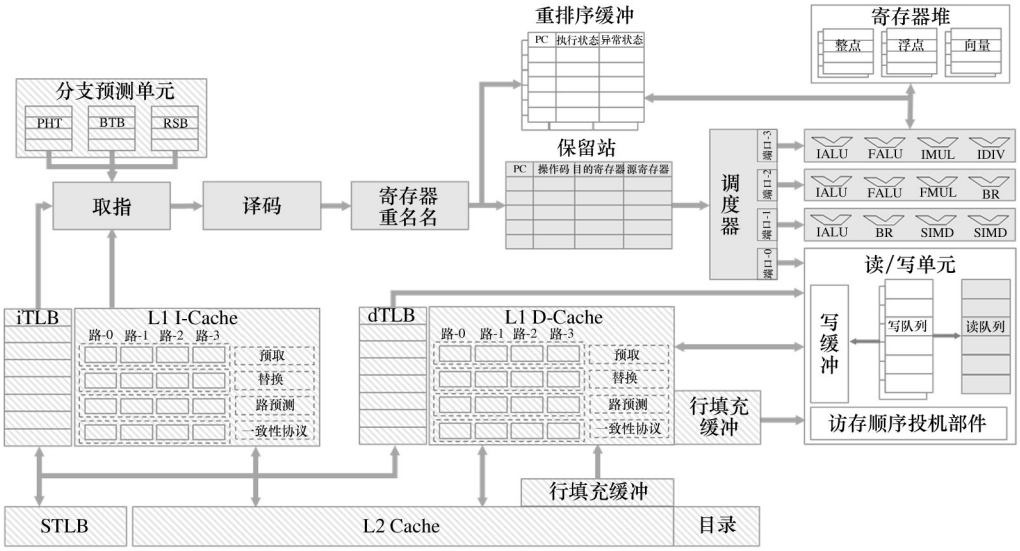


图 1　现代高性能处理器微架构

1.1　投机执行及超标量

在流水线前端,取指部件利用模式历史表(pattern history table,PHT)、分支目标缓冲(branch target buffer,BTB)、返回栈缓冲(return stack buffer,RSB)预测各种分支指令的目标,决定投机执行的方向。在流水线后端,经过寄存器重命名的指令被分发进保留站。调度器每次从保留站中选出多条操作数就绪的指令发射执行。为了提高发射速度,一些执行单元会共用一套发射逻辑,组成执行端口。指令在发射/执行阶段无需遵守程序序,只需在提交阶段按程序序写入重排序缓冲(reorder buffer,ROB)中。如果在提交阶段发现指令执行过程发生异常,或是在分支指令执行阶段发现分支投机失败,处理器会清理 ROB 中后续指令的执行结果,并重置取指部件的方向。

1.2　访存顺序投机及数据前递

访存指令被发射进入读/写单元(load store unit,LSU)后,先经过页表缓冲(translation lookaside buffer,TLB)完成地址转换以及权限检查,再以获得的物理地址访问 L1 D-Cache。指令在 LSU 中执行时,需要考虑针对同一内存地址的“写后读”依赖,以及指令集要求的内存一致性模型。为此,访存顺序投机部件(memory order buffer,MOB)被引入,用于预测可能的依赖关系,并进行访存指令顺序投机。同样,访存顺序投机失败也会触发 ROB 清理以及相应指令重新执行。除此之外,LSU 中另一个重要部件就是写缓冲(store buffer,SB)。它负责缓存已提交但还未写入 cache 的写指令结果。当后续读指令需要该结果时,无需访问 cache 即可从 SB 中获得数据,这就是所谓的“写-读数据前递(store-to-load forwarding)”。另外,SB 还可合并地址重叠的写指令

操作,并在满时异步写入 L1 D-Cache 中。

1.3 Cache 系统

Cache 的基本操作单元是缓存行,每个缓存行包含连续多个字节。目前普遍使用的组相联 cache 是由多个组构成的,每组又包含多路,每路都是一个缓存行。访问 cache 时,访存地址中低位决定组的位置以及缓存行内的偏移,高位与每路缓存行的 tag 比较,决定具体的路的位置。为了加速 tag 比较,现代处理器还引入了路预测器。它会根据该组内访存历史,预测未来可能访问的路。同时,组内访问历史还被用来决定后续的替换对象。另外,整个 cache 中还附带硬件预取器,可以根据以往访存地址序列,预测未来访存地址,提前装入 cache 中。各级 cache 之间也有类似 SB 的缓冲,被称为行填充缓存(line fill buffer,LFB)。LFB 一方面负责收集并异步处理发生缺失的访存请求,另一方面还具备类似 SB 的数据前递和请求合并功能。另外,多核处理器的 cache 系统中还具备复杂的核间互联网络和 cache 一致性协议。

1.4 超线程

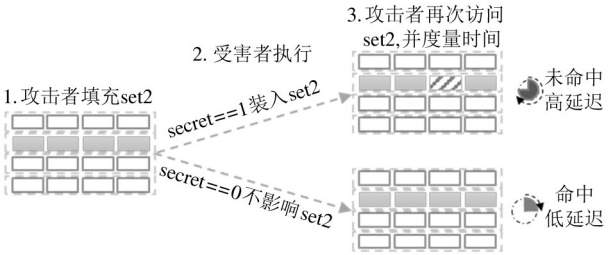
cache 访问缺失可能导致流水线卡顿,所以设计人员又提出了超线程技术,又称为同时多线程(simultaneous multi-threading,SMT)。其原理是在单核流水线中同时运行来自多个线程(通常是 2 个)的指令。它通过对 ROB、寄存器堆等保存线程执行状态的部件进行划分,使得多个线程的执行状态可以共存,无需分时切换;同时,让超线程间共享一些重要的执行部件,比如执行端口、L1/L2 Cache 等,从而使不同线程的需求可以互补,提高资源利用率,进而提升处理器的整体性能。

2 单一时间侧信道攻击

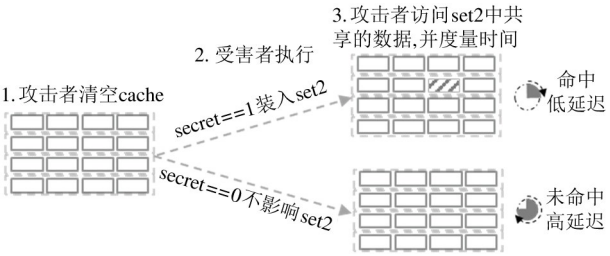
单一时间侧信道攻击的基本原理是:攻击者和受害者共享硬件资源,导致执行状态相互干扰,以至影响彼此的执行时间。攻击者利用这一原理,通过度量自身执行时间就可反推出受害者的执行状态,进而逆向出一些与受害者执行状态相关的敏感信息。这种共享资源在安全领域又被称为“信道”。根据传输方式的不同,将信道分为持久型和易失型 2 类。图 1 对处理器微架构中每个部件可作为的信道种类进行了区分。

2.1 持久型信道

攻击者和受害者程序共享同一个数据缓存,比如 cache[6]、TLB[7]、Paging Cache[8]。攻击者先布置缓存的布局,然后等待受害者执行,之后攻击者再次访问缓存中的数据,并通过度量访问时间探测布局变化,进而推出受害者的访问轨迹,再结合受害者源代码,逆向出敏感信息。具体的攻击方式又可分为 2 类:其一为受害者的访问轨迹可能导致攻击者后续度量变慢,如图 2(a)所示的 PRIME + PROBE 攻击;其二为受害者的访问轨迹可能导致攻击者后续度量变快,如图 2(b)所示的 FLUSH + RELOAD 攻击。除了以上常见的缓存外,超线程间共享的 BTB[9]、X86 处理器中为了加速从宏码到微码翻译过程引入的微码翻译 cache[10]、维护核间 cache 一致性的目录表[11]等,都被发现可被用作持久型信道。



(a) PRIME + RPOBE 攻击



(b) FLUSH + RELOAD 攻击

图 2 以 cache 为例的持久型侧信道攻击

进一步对持久型信道的定义进行扩展,除了数据布局外,其他可以影响后续执行时间的元数据也可以被攻击者利用,如受害者的访存足迹可能影响后续 cache 中预取器的预取地址[12]、后续 cache 中替换算法选择的对象[13]、后续路预测器预测的对象[14]以及其他核心备份的一致性状态[15],从而影

响后续访存指令的执行时间。

2.2 易失型信道

攻击者和受害者共享同一个功能部件,比如超线程间共享核心的执行端口[16]、浮点计算单元[17]、取指/译码部件[18]。二者同时竞争资源,攻击者可通过度量自身执行时间,估计竞争发生情况,推出受害者的执行状态,进而逆向出敏感信息。因为信息泄露是伴随着资源竞争同时发生的,无法异步实施,所以本文将这类信道称为易失型信道。相比于持久型信道,易失型信道噪声大,攻击者必须与受害者同步执行。但是其优点是更加隐蔽,且攻击者无需像大部分持久型信道那样了解其索引算法及替换策略等先验知识。

3 针对单一时间侧信道攻击的防御

现有针对单一时间侧信道攻击的防御机制种类繁多,本文从风险管控的角度,也就是从避免风险发生、阻止风险扩散、减少风险损失 3 个方面,将这些机制归纳为以下 5 类。

3.1 常量时间化变换

该技术可以从风险源头防范各类侧信道攻击的发生。其核心原理是:对受害者代码进行变换,使其执行时间为固定的常量(constant time)[19]。具体的变换方法是:对敏感数据相关的分支进行改造,使分支所有目标基本块内执行的运算一致。图 3 展示了对 OpenSSL 加密算法中某个和密钥相关分支的常量时间变换。更严格的常量时间变换还要保证所有目标基本块内的访存足迹一致,也就是实现所谓的

```
1:  for (; i >=0; i--) {
2:      if (key[i]) {
3:          Madd(x1, z1, x2, z2);
4:          Mdouble(x2, z2);
5:      } else {
6:          // same computation
7:          // same cache access trace
8:          Madd(x2, z2, x1, z1);
9:          Mdouble(x1, z1);
10:     }
11: }
```

图 3 OpenSSL 中某个函数内的常量时间执行代码

数据无关化(data oblivious)[20]。但这需要确保 TLB、cache、动态随机存储器(dynamic random access memory,DRAM)等所有存储器访问位置及顺序的一致,会付出极大的性能开销(几十倍到上百倍),所以目前还没有被广泛部署。

3.2 资源划分

对共享资源进行划分,可以起到风险隔离的作用。资源划分可从空域或时域 2 个维度进行。空域划分适合大部分持久型信道。以 cache 为例,空域划分可以采取按组或按路的方式。按组划分就是将 cache 中不同组划分给不同代码域。因为 cache 中组的索引是由访存地址决定的,所以按组划分通常由系统软件实现。它们可将不同域的数据分配在不同内存页上,从而使其映射到不同的 cache 组上。所以这一机制也被称为页着色机制(page coloring)[21]。该机制最大的问题是内存开销大,且会产生较多 TLB shootdown,导致性能下降。按路划分就是将 cache 每组内不同路缓存行分给不同域。因为路的选择不是软件可控的,所以按路划分需要硬件支持。目前使用最广泛的路划分机制就是 Intel 缓存分配机制(cache allocation technology,CAT)。后续工作[22]又在 RISC-V 指令集上实现了类似的机制。按路划分是目前 cache 划分的主流方法,因为其不需要对软件作过多修改,同时还能兼顾性能,但其局限是划分出的物理域数量有限(cache 的相联度一般不超过 32)。如果要支持更多域,软件就必须对域进行虚拟化。除了 cache 外,其他存储部件如 TLB、BTB 也可以引入类似的机制。另外,在划分时还要考虑其他元数据,比如替换算法、预取器中记录的历史访问数据。

时域划分适合大部分易失型信道,其原理就是让不同代码域对共享部件进行分时复用。目前已有的一些工作通过修改硬件,实现了对执行端口[23]、取指/译码器[18]等超线程共享部件的时域划分。但这样违背了超线程设计的初衷,会因资源闲置导致较大的性能开销。虽然可以动态调整时间片的划分比例,以适应各种负载需求,但这会牺牲安全性,因为因自适应而产生的执行时间变化本身就可以作为一种隐蔽信道。另外,一些基于软件调度的方法,也

能模拟出分时复用的效果,完成防御[24]。

3.3 随机化

随机化机制无法彻底阻止攻击发生,只能降低成功的概率,并且只适用于类似 cache、TLB、BTB 这种容量较大的持久型信道。一种随机化策略是在访问存储部件中的某项时,同时随机访问其他项,或者叠加随机长度的延时,避免攻击者准确度量访问轨迹[25]。另一种随机化策略是修改存储部件的替换算法,在替换时选择随机的项换出;或者在装入新项时,随机再装入其他项,避免攻击者进行有效布局[26]。以上这 2 种策略会对部件的命中率和命中时间产生较大影响。对性能影响更小的一种策略是对存储部件的索引算法进行动态随机,也就是对访问地址和目标位置进行动态重映射,使攻击者无法完成布局[27]。

3.4 增加噪声

另一种降低攻击风险的方法是向信道中注入噪声。一种常用的注入噪声方法是在敏感代码执行时,在同一物理核心上再启动另一个专门用于干扰的超线程[28]。另一种方法是降低时间测量的精度,在 rdtscp 指令的返回值上叠加随机偏差[29]。而目前主流浏览器都降低了可获取的时间精度,并禁用了 rdtscp 指令。增加噪声只能作为临时性的缓解手段,目前学术界都认为其防护强度达不到要求。因为除了直接通过专用指令测量外,攻击者还可以通过其他手段度量时间,比如使用循环累加间接度量执行时间[30]。

3.5 攻击检测

真实场景中干扰因素很多,侧信道攻击需要一定时间才能达到目标。对系统状态进行实时监控,在发现被攻击后采取一定策略,避免造成更大损失也是有必要的。早期的侧信道攻击检测都是基于人工归纳的异常行为模式,实时采样系统中的各种状态,比如 cache/TLB 命中情况、BTB 命中情况、浮点部件竞争情况,进行分析[31]。目前基于人工智能的侧信道攻击检测技术也正在兴起[32-33],相比于传统基于模式的检测,人工智能检测可以发现更多未知的攻击模式,并且避免被混淆行为所欺骗。

4 瞬态执行攻击

在单一侧信道攻击中,信道发送方是受害者,接收方是攻击者。如果双方都是攻击者控制的,即攻击者只利用信道传递某种无法直接传输的信息,则称其为“隐蔽信道(covert channel)”。瞬态执行攻击就是复合了瞬态执行漏洞和隐蔽信道的攻击。其攻击过程如图 4 所示,可归纳为以下 4 步:(1)训练阶段——攻击者训练处理器状态,使其能按攻击者意图投机;(2)访问阶段——攻击者利用瞬态执行漏洞在投机执行中访问目标敏感数据;(3)发送阶段——攻击者将非法访问获得的数据传入隐蔽信道中;(4)接收阶段——攻击者从隐蔽信道中提取数据。

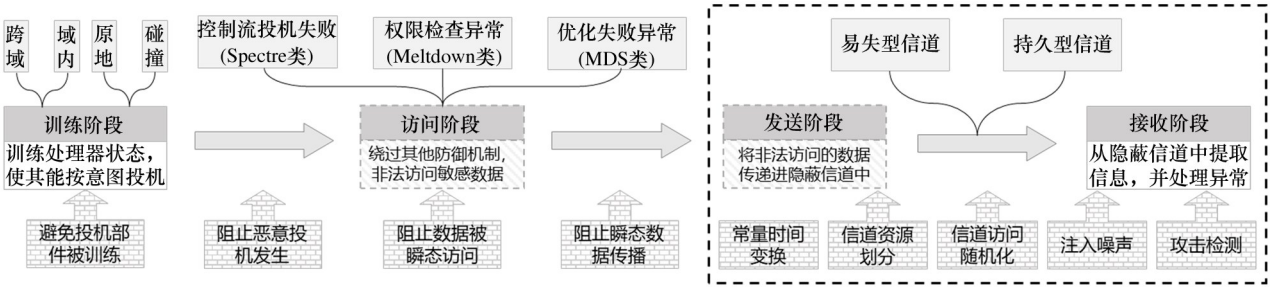


图 4 瞬态执行攻击流程,训练方式、漏洞、隐蔽信道分类及防御机制分布

从根源上来说,瞬态执行漏洞的成因可归结为:硬件设计者忽视了失败的投机执行在微架构层面产生的副作用。而导致投机执行失败的原因可分为 3 类:(1)控制流投机违背了程序正常执行的顺序,也就是 Spectre 类漏洞的成因;(2)指令执行过程中因未通过权限检查,触发了软件可见的异常,也就是

Meltdown 类漏洞的成因;(3)指令执行过程中因自身优化失败,触发了软件不可见的异常,也就是微架构数据采样(microarchitecure data sampling,MDS)类漏洞的成因。以下是对这 3 类漏洞攻击具体原理的介绍。

4.1　基于 Spectre 类漏洞的攻击

Spectre 类漏洞存在于几乎所有现代处理器中。其致错原因是:处理器控制流投机失败,攻击者可在错误的执行路径上访问不该被访问的敏感数据,并在投机被解出之前,将其通过隐蔽信道传递出去。根据投机的类型,将 Spectre 类漏洞分为以下 3 类。

(1)直接分支投机

攻击者控制直接分支的投机方向,使其能在错误投机的方向上访问敏感数据。如图 5 所示,第 3 行的分支负责约束数组访问的边界。攻击者通过反复训练,使处理器针对第 3 行分支的预测结果为 true,然后再以一个越界的地址执行这段代码,因为分支预测结果为 true,所以可以以越界的偏移访问该数组,并通过 FLUSH + RELOAD 的方式传递到隐蔽信道中。因为处理器中预测直接分支的部件是 PHT,所以这类漏洞又被称为 Spectre-PHT[34]。

```
1: uint8 oracle[256*4096];
2: void victim(int index) {
3:     if (index < SIZE) { // Bound Check
4:         val = array[index];
5:         tmp = oracle[val*4096];
6:     }
7: }
```

图 5　Spectre-PHT 漏洞 PoC

该漏洞还存在变种[35]。如果将错误投机执行的代码变为写操作,虽然这种写操作并不会进入 cache 中,但会影响 LSU,后续的投机读操作可以读到这个错误的值。如果这个值恰好是个指针,则攻击者就可以利用其访问敏感数据。

(2)间接分支投机

攻击者通过控制间接分支指令的跳转目标,使其能在错误的投机目标执行时访问敏感数据,进而通过隐蔽信道完成传输。因为控制间接分支的部件是 BTB,所以相应的漏洞就被称为 Spectre-BTB;而控制返回指令的部件是 RSB,所以相应的漏洞就被称为 Spectre-RSB。Spectre-BTB 漏洞的危害更大,因为在理想威胁模型下,其转移目标可以被训练成任意指令计数器(program counter,PC)的代码,所以该攻击也被称为“分支目标注入(branch target injection,BTI)”。Spectre-RSB 漏洞利用的限制较大,因为 RSB 是栈结构,每次返回指令只能访问栈顶元素,并且栈的内容都是由配对的调用指令自动压入的,无法直接修改。想要利用 Spectre-RSB,攻击者需要诱导受害者修改软件栈上保存的返回地址,使 RSB 和软件栈指向的地址不一致;或是诱导受害者的调用链深度超出 RSB 上限,从而使 RSB 底部压入的地址被覆盖,在相应的返回指令执行时会转而使用 BTB 进行预测,该攻击又被称为 Retbleed[36]。

对于 Spectre-PHT 和 Spectre-BTB,攻击者在利用前都需要训练相关部件。具体的训练方式可以从以下 2 种维度审视:1) 是否跨域(cross domain),即攻击者是否在相同的进程空间内完成的训练;2) 训练时的分支 PC 是否和瞬态执行时的分支 PC 相同(in place),即攻击者是否是通过部件索引 hash 碰撞完成的训练。通常来说,攻击者可以在用户态通过碰撞式的训练而控制内核态代码的分支,也可以在另一个现场中(开启超线程)通过跨域式的训练而控制另一个现场中的分支。另外,现代处理器中 BTB 通常是由全局 PHT 表的结果和 PC 共同索引的,所以攻击者通过训练全局 PHT 表也可以间接控制 BTB,这就是所谓的“分支历史注入(branch history injection,BHI)”[37]。

(3)乱序投机

攻击者通过控制投机时指令的执行顺序,绕过某些具备防御功能的代码,访问敏感数据。比如攻击者通过训练 MOB,使其预测某条读指令和其之前的写指令之间不存在依赖,则读操作就可先于写指令执行,从而读到了写指令要清零的数据。因为出现问题的是写读数据前递机制,这种漏洞又被称为 Spectre-STL[34]。同样,在 X86 的写全序(total store order,TSO)模型下,读指令间的错误乱序投机也可能被攻击者利用[38]。另外,除了访存指令的乱序投机外,其他一些指令的执行顺序也可能关系到安全,

一旦被错误乱序,可能会产生安全隐患,比如 SWAPGS 指令是某些操作系统内核在上下文切换时用来切换保存栈指针的全局段寄存器的指令,如果其被绕过,可能导致内核栈上数据被泄露[39]。

4.2 基于 Meltdown 类漏洞的攻击

Meltdown 类漏洞主要出现在 Intel 处理器中。其致错原因是:处理器在执行某些指令时,并没有等待权限检查完成,就将指令要读取的数据传递给后续的指令,使其能完成隐蔽信道传输,而权限检查异常直到提交阶段才处理。根据异常类型,将 Meltdown 类漏洞分为以下 2 类。

(1)内存页访问异常

攻击者非法访问敏感数据所在的内存页,最终会触发页异常。在经典的 Meltdown 漏洞中,攻击者处于用户态,直接访问内核态的数据,处理器在地址转换过程中,会触发页表 U/S 位检查异常。该攻击还有个前提条件:被泄露的内核态数据必须在 L1 D-Cache 中,所以也被称为“恶意数据缓存读攻击(rogue data cache load,RDCL)”[40]。除了 U/S 异常外,Intel 的内存页保护机制(memory protection key,MPK)可以自由切换当前代码可访问内存页权限,该机制也存在被绕过的漏洞[4]。

如果增加一些条件,攻击者还可以绕过更多权限检查。比如在 Intel 硬件辅助虚拟化特性中,客户机(Guest)和管理系统(Host)间又引入了一层扩展页表(extension page table,EPT),用于记录 Guest 物理地址到 Host 物理地址的转化关系。在“伏笔(Foreshadow)”攻击中[41],攻击者可以先重置敏感数据所在页表中的 Present 位(表示该页已被换出),而后在访问该数据过程中,处理器可以无视 EPT 表,直接以 Guest 物理地址访问内存中的数据。因为 Present 位被清除后产生的异常又被称为终止异常(terminal fault,TF),而且目标数据还是要在 L1 D-Cache 中,所以 Foreshadow 攻击也被称为 L1TF。

(2)寄存器访问异常

攻击者非法访问敏感数据所在的寄存器,最终会触发不可访问异常。比如攻击者在用户态直接访问内核态下才能访问的 CR3、MSR、DR 等特权寄存器,该攻击又被称为“恶意系统寄存器读取(rogue system register read,RSRR)”[42]。另一种特殊的寄存器异常是针对向量和浮点寄存器的。现代操作系统在进程上下文切换时会将这些寄存器封存,而后其他进程使用到这些寄存器时会触发异常,此时操作系统再对这些寄存器进行上下文切换。这种懒化(lazy)的切换方式可以降低上下文切换的开销。然而,这种异常也存在被绕过的漏洞,所以又被称为 Lazy-FP[43]。

4.3 基于 MDS 类漏洞的攻击

MDS 类漏洞只存在于 Intel 处理器中,其致错原因是:一些指令集层面不可见的缓冲区在某些条件下存在错误的数据复用,即访问这些缓冲区未命中时,会把其中残留的数据先递给访问的指令,以保持后续数据流的畅通,直到真正的数据到来后,处理器发现这种复用是非法的,才会触发软件不可见的异常,并重新执行相应的指令。攻击者正是利用这种错误的数据复用,完成了敏感数据的泄露。根据利用方法的不同,本文将其分为以下 2 类。

(1)直接泄露类

所谓直接泄露类,就是错误的数据复用直接将缓冲区中的敏感数据传递给后续的隐蔽信道指令。触发错误数据复用的条件包括以下 3 类:1)获得敏感数据的那条访存指令必须存在某些异常,比如访问一个非正则的地址(non-cannical address)、访问一个被换出的内存页;2)获得数据的那条访存指令隐含硬件辅助操作(assist load),比如访问一个页表项 Access 位被清空的地址,此时硬件会自动置位页表项上的 Access 位;3)获得敏感数据的那条访存指令能触发 Intel 事务内存机制 TSX(transaction synchronization extension)终止。目前,已公开的存在 MDS 漏洞的缓冲区包括 SB、LFB、读指令对应的发射端口缓冲区、内存映射 I/O 寄存器[44]。相比于 Meltdown、Spectre 类漏洞,MDS 类漏洞的利用条件更宽松,因为被泄露的数据和泄露指令之间不需要任何关联关系,只要是缓冲区中残留的数据,就可以被后续执行的线程或者同核心上其他超线程所获得。

(2)间接泄露类

在间接泄漏漏洞中,攻击者不是直接利用错误

的数据复用获得敏感数据,而是反过来先利用其将攻击者自身的数据注入到某个寄存器中,如图 6 中第 2、4 行所示的操作,使其作为后续第 6 行访存指令的访存地址,从而控制该访存的目标,读取敏感数据,并最终通过隐蔽信道传输出去。所以,这种攻击又被称为“读值注入(load value injection,LVI)”[45]。除了数据指针外,攻击者还可以通过注入控制代码指针,在代码指针的调用过程中完成敏感数据泄露。相比于直接泄露,LVI 的攻击场景受限,通常是在可信计算或者云环境下,恶意的操作系统或 Host 通过修改页表,使受害者的访存指令具备触发 MDS 漏洞的能力,再构造 LVI 攻击。

```
1: // Inject 'mal_a' in vulnerable buffer
2: *dummy = mal_a;
3: // Load faults, forwarded 'mal_a' to 'ptr'
4: ptr = *pptr;
5: // Load secret from address 'mal_a'
6: val = *ptr;
7: // Transmit secret over covert channel
8: ret = oracle[val * 4096];
```

图 6 LVI 漏洞 PoC 核心代码

5 针对瞬态执行攻击的防御

Meltdown 类和 MDS 类漏洞都属于设计缺陷,厂商可以在后续型号的处理器中将其修复。Intel 在第 9 代 Coffee Lake 架构后就修复了 Meltdown 类漏洞,在第 10 代 Ice Lake 架构后也修复了已公开的 MDS 类漏洞。而针对 Spectre 类漏洞,以及早期型号处理器上的 Meltdown 漏洞和 MDS 漏洞,研究人员提出了多种防御机制。本文根据这些机制的作用点,将其分为图 4 所示的 5 类。其中,作用点在第 3 阶段的防御机制就是针对单一时间侧信道攻击的防御机制,上文已经介绍过。表 1 列出了这些防御机制针对各类瞬态执行攻击的防御能力,以及性能开销。

5.1 避免投机部件被训练

这类机制可以缓解直接/间接分支类 Spectre 漏洞,其核心思想是:避免不同域间的代码共用投机部件,使攻击者无法对该部件实施跨域训练。比如 Intel 处理器在 Coffee Lake 架构后引入了 IBRS/STIBP/IBPB 机制[46]。其中,IBRS 机制在开启后,可以避免低权级代码影响高权级代码的投机执行过程;STIPB 机制在开启后,可以避免不同超线程间的投机执行相互影响;IBPB 机制在开启后,可以避免后续代码的投机执行被之前代码执行所影响。ARM 处理器也引入了和 Intel IBRS 机制类似的 CSV2 机制[47]。但是,需要注意的是这类机制都无法阻止域内的训练。另外,IBRS 和 CSV2 机制也被证明无法防御 4.2 节中提到的 BHI 攻击。

表 1 瞬态执行攻击防御机制的防御能力及性能开销

	Spectre 类漏洞				Meltdown 类漏洞				MDS 类漏洞		性能开销(区间)
	Spectre-PHT	Spectre-BTB/RSB	Spectre-STL	SWAPGS	RDCL	L1TF	RSRR	LazyFP	直接泄露 MDS	LVI	
IBRS/STIBP/IBPB	√	√									20% ~50%
SSBD			√								2% ~8%
lfence 插桩	√		√	√						√	68% ~224%
SLH/Index Mask	√										29% ~36%
Retpoline		√									6% ~40%
PTI/Site Isolation					√						10% ~20%
L1-D Cache Flush						√					-3% ~31%
VERW Flush									√		8% ~10%
RDMSR 微码补丁							√				——
eagerfpu switching								√			——
限制投机	√	√	√	√	√	√	√	√	√	√	5% ~15%
投机不可见	√	√	√	√	√	√	√	√	√	√	-3% ~22%

5.2 阻止恶意投机发生

这类机制的基本原理是:对风险代码进行加固,

使其投机过程无法被攻击者利用。根据防御目标的不同,可分为以下几类。

(1)执行屏障

对于直接分支投机类和乱序投机类 Spectre 漏洞,一种简单有效的防御措施是在风险访存指令前插入“屏障(barrier)”指令,比如 X86 指令集的 lfence 指令,以及 ARM 指令集的 isb/dsb 指令,确保之前指令的执行结果已生效[46]。另一种替代机制是把分支指令和后续访存指令合并为条件指令,比如 X86 指令集下的 cmov 指令、ARM 指令集下的带条件后缀的 ld 指令,从而绕过分支预测,这种方法又被称为“投机读增强(speculative load hardening, SLH)”或“索引掩码(index mask)”[48]。这类机制最大的问题是:正常程序中存在大量需要插入执行屏障指令的位置,而这会频繁破坏处理器的投机执行,产生较大的性能开销。

(2)返回跳板

对于间接分支类 Spectre 漏洞,在所有可能的跳转目标前插入“屏障”指令是不实际的,因为任何 PC 的指令都可能成为攻击者训练的跳转目标。而在间接跳转指令前插入“屏障”指令虽然能缩小间接跳转指令的投机窗口,但是研究表明,攻击者依然可以通过构造超线程间的资源竞争,使间接跳转指令的执行被阻塞,从而让流水线中还能执行一些预测的目标位置的指令,完成信息泄露[49]。一种更安全的防御方法是返回跳板技术(retpoline)[50]。它将间接跳转/调用指令都转化为返回指令,并利用 RSB 的特性实现保护。具体过程如图 7 所示,先将间接跳转/调用指令转化为直接调用指令,调用的目标是

```
1: call load_label;
2: capture_label:
3:     lfence;
4:     jmp capture_label;
5: load_label:
6:     mov %rax, (%rsp);
7:     ret;
```

(a) jmp (*) retpoline

```
1: jmp call_label;
2: link_label:
3:     call load_label;
4: capture_label:
5:     lfence;
6:     jmp capture_label;
7: load_label:
8:     mov %rax, (%rsp);
9:     ret;
10: call_label:
11:     call link_label;
12: ...;continue execution
```

(b) call (*) retpoline

图 7　Retpoline 变换后代码

一段序列 load _ label,该序列将真正的跳转/调用目标存入栈顶,再通过返回指令完成转移。而返回指令在处理器执行过程中,RSB 会选择刚才调用指令的后一条指令,也就是 capture _ label 序列投机执行,而 capture _ label 序列本身是个死循环,其中又有 lfence 指令作为屏障,所以其投机范围不会跳出该序列中,直到最后返回指令取到了真正的目标,完成控制流转移。

返回跳板技术虽然能有效阻断间接分支类漏洞,但是其性能开销较高(25% ~50%)。后续研究者又提出了优化方案——通过“间接调用提升(indirection call promotion)”减少间接跳转/调用出现的频度,即通过运行时采集间接跳转/调用的目标(profiling),然后重新编译[51]或是动态修改二进制代码[52],将一部分间接调用/跳转转变为直接分支指令。

5.3　阻止特定数据被瞬态访问

这类机制可以保护特定数据免于瞬态执行攻击。其核心思想是:利用地址空间隔离技术,使针对特定数据的非法投机访问不能发生;或是在风险可能发生前,对保存特定数据的 L1 D-Cache 或其他缓冲区进行清理,使得后续的投机无法获取到该数据。

(1)页表隔离

“页表分离(page table isolation, PTI)”[53]的基本原理是:令受保护的数据和普通数据使用不同的页表。当只能访问普通数据的代码运行时,系统强制处理器使用普通页表,此时即使投机执行中要访问保护数据,也会因无法完成地址转换而失败。PTI 技术最初是保护内核免于用户态的 Meltdown 攻击,后续也被应用于浏览器中,被称为“站点隔离(site isolation)”[54]。然而,该机制最大的问题是切换页表时刷新 TLB 导致的性能开销。

(2)数据清理

对于 L1TF 漏洞来说,因为 EPT 表的地址翻译也被绕过,所以只能采取在上下文切换时清空 L1-D Cache 的方法[55]。类似的方法也被应用在直接泄露类 MDS 漏洞的阻断中。因为 MDS 漏洞绕过了所有可能的检查机制,所以地址隔离对其是无效的,唯一有效的方法就是不让数据出现在风险缓冲区中。具

体来说,风险来自 2 方面:1)超线程间可能泄露,所以需要通过线程调度避免不信任的线程同时运行;2)同一超线程在上下文切换时,残留的数据可能被泄露。为此,Intel 通过微码补丁,为废弃的 VERW 指令增加了清理微架构缓冲区的功能。该指令已被主流操作系统加入进程上下文切换的代码中。这类机制最大的问题是清理缓冲区要花费大量时间,所以在系统调用密集的应用上会产生较大开销。

(3)微码更新及软件升级

对于寄存器异常类的 Meltdown 漏洞,目前唯一可行的解决方案就是安装针对 RDMSR 指令的微码补丁[56],以及在上下文切换时关闭对向量寄存器的 lazy 切换策略。研究表明,这些修补策略并不会引入明显的性能开销[43]。

5.4 阻止瞬态数据传入隐蔽信道

以上防御机制都是在攻击早期进行阻断,需要对各种漏洞采取不同的策略;并且大部分机制都需要上层软件的修改,甚至编译器的配合,实际部署不是很便捷。一种对所有瞬态漏洞有效且对上层软件透明的防御机制是:重新设计处理器中的投机执行机制,避免其副作用传递到潜在的隐蔽信道中。具体来说,这类机制可分为以下 2 类。

(1)限制投机

这类机制可在投机执行早期阻断其副作用的传播。其核心原理是将指令的执行过程分为投机和非投机 2 个阶段,当处于投机阶段时,对该指令可能产生的影响进行追踪,即识别出后续可能依赖这条指令的其他指令,以及可能被该指令污染的功能部件,这些影响都可能构成隐蔽信道;然后推迟这些影响的发生,直到该指令投机成功[57]。然而这种推迟投机执行的方式会造成流水线阻塞,产生较大的性能开销。一种优化策略是使得后续投机执行阻塞的时间更短,比如为后续依赖投机指令的其他指令引入值预测机制,使其可以在缺少真实操作数的情况下,根据值预测器给出的值继续计算[58];还比如修改缓存一致性协议,使投机读指令装入 cache 中的数据无法被换出,从而缩短了读指令提交的等待,使其可以更快地转变为非投机状态[59]。

(2)投机不可见

这类机制允许投机操作执行,只是根据投机操作可能的影响进行消除,使其无法完成信息传递。一种实现投机不可见的方法是为投机访存增加专用 cache,使其不会影响原有 cache 系统。等指令投机结束后,如果投机失败,则直接将专用 cache 中的数据丢掉;如果成功,再将数据同步回原有 cache 系统中[60]。相比于限制投机,该机制解放了访存指令,但是又引入了令一个问题——多核下投机 cache 和原有 cache 中的数据一致性问题。针对这一问题,直接的解决方案是在投机读指令成功提交后,在原有 cache 系统中再次执行该指令,并将获取的数据与专用 cache 中的数据进行比较,如果不一样,则认为存在一致性分歧,重新执行后续指令。但这一解决方案最大的问题是大部分读操作都会投机成功,所以都需要进行重读和比较,从而产生不可忽视的开销。一种提升方案是不增加专用 cache,允许投机访存修改 cache 布局,等其被发现投机失败时,再对 cache 布局进行回滚[61];另一种是直接在原有 cache 中划分出投机域,通过标签切换实现数据同步[62]。不过,以上工作都是针对以 cache 为代表的持久型信道,目前还没有扩展到易失型信道上。

6 结 论

现代处理器复杂的微架构和激进的优化机制是时间侧信道广泛存在的根本原因。随着处理器设计的不断演进,新的攻击面不断被发现,从传统体系结构层面可见的部件,如 cache、TLB、浮点计算部件,再到微架构层面因优化而引入的部件,如 BTB、微码 cache、cache 路预测器。然而,受限于攻击能力,相关问题并未引起设计人员的足够重视。瞬态执行的出现给时间侧信道注入了新的活力。一方面它打破了系统中原有权限层级和访问控制机制的约束,使攻击范围被极大扩展;另一方面,它将攻击模式从被动等待转变为主动获取,使攻击变得更加灵活、隐蔽。目前,除了探索新的隐蔽信道外,安全人员的研究更多聚焦在挖掘新的瞬态执行漏洞上,并且逐步向自动化方向上发展。

而防御技术主要有 2 条发展路线:(1)紧跟攻

击,在已有防御机制上不断加固,构造塔式防御体系;(2)提升性能,或者在性能和安全性上作出更优的权衡,并最终反向推动处理器架构的升级,实现高效的软硬件协同防御。针对单一时间侧信道攻击的威胁,安全人员先提出了软件缓解机制,比如禁用高精度时间度量指令、关闭超线程、基于安全等级的线程调度等。但这些机制不能完全切断信息传输通路,并且限制了系统的可用性。后续安全人员又提出了更高强度的防御机制,比如对敏感代码进行常量时间化变换,或者对风险部件进行划分隔离,并最终催生了 Data-oblivious RAM、Intel CET 等相应辅助硬件。

在瞬态执行攻击出现后,安全人员首先是通过 PTI、Retpoline 等软件加固方法,确保内核、虚拟机管理系统等重要软件的运行时安全。然后是通过微码补丁、编译器分析插桩、地址隔离等手段,为通用软件提供多种缓解办法。考虑到防御机制部署的便捷性,安全人员的研究重点逐步转向重新设计处理器微架构投机机制,使其能从源头避免各种瞬态执行问题,并对上层软件透明。目前的研究热点主要聚焦于如何提升安全投机执行机制的性能,使其能更早地恢复投机执行的效率,或者对敏感数据更具特异性。这也是未来处理器设计的一个重要研究方向。

参考文献

[1] 李晔, 李沛南, 赵路坦, 等. 瞬态执行漏洞攻击及防御综述[J]. 高技术通讯, 2020,30(8):774-782.

[2] 吴晓慧, 贺也平, 马恒太, 等. 微架构瞬态执行攻击与防御方法[J]. 软件学报, 2020,31(2):544-563.

[3] 王崇, 魏帅, 张帆, 等. 缓存侧信道防御研究综述[J]. 计算机研究与发展, 2021,58(4):794-810.

[4] CANELLA C, VAN BULCK J, SCHWARZ M, et al. A systematic evaluation of transient execution attacks and defenses[C]// USENIX Security Symposium. Santa Clara, USA: USENIX Association, 2019:249-266.

[5] GE Q, YAROM Y, COCK D, et al. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware[J]. Journal of Cryptographic Engineering, 2018,8:1-27.

[6] YAROM Y, FALKNER K. FLUSH + RELOAD: a high resolution, low noise, L3 cache side-channel attack[C]// USENIX Security Symposium. San Diego, USA: USENIX Association, 2014:719-732.

[7] GRAS B, RAZAVI K, BOS H, et al. Translation leak-aside buffer: defeating cache side-channel protections with TLB attacks[C]// USENIX Security Symposium. Baltimore, USA: USENIX Association, 2018:955-972.

[8] GRAS B, RAZAVI K, BOSMAN E, et al. ASLR on the line: practical cache attacks on the MMU[C]// Network and Distributed System Security Symposium. San Diego, USA: NDSS, 2017:26.

[9] EVTYUSHKIN D, RILEY R, ABU-GHAZALEH N C S E, et al. Branchscope: a new side-channel attack on directional branch predictor[J]. ACM SIGPLAN Notices, 2018,53(2):693-707.

[10] REN X, MOODY L, TARAM M, et al. I see dead μops: leaking secrets via Intel/AMD micro-op caches[C]// International Symposium on Computer Architecture. Valencia, Spain: IEEE, 2021:361-374.

[11] YAN M, SPRABERY R, GOPIREDDY B, et al. Attack directories, not caches: side channel attacks in a non-inclusive world[C]// IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2019: 888-904.

[12] GRUSS D, MAURICE C, FOGH A, et al. Prefetch side-channel attacks: bypassing SMAP and kernel ASLR[C]// Conference on Computer and Communications Security. Vienna, Austria: Association for Computing Machinery, 2016:368-379.

[13] BRIONGOS S, MALAGÓN P, MOYA J M, et al. Reload + refresh: abusing cache replacement policies to perform stealthy cache attacks[C]// USENIX Security Symposium. Boston, USA: USENIX Association, 2020: 1967-1984.

[14] LIPP M, HADŽIĆ V, SCHWARZ M, et al. Take a way: exploring the security implications of AMD's cache way predictors[C]// Asia Conference on Computer and Communications Security. Taibei, China: Association for Computing Machinery, 2020:813-825.

[15] YAO F, DOROSLOVACKI M, VENKATARAMANI G. Are coherence protocol states vulnerable to information leakage? [C]// International Symposium on High Performance Computer Architecture. Vienna, Austria: IEEE, 2018:168-179.

[16] ALDAYA A C, BRUMLEY B B, ULHASSAN S, et al. Port contention for fun and profit[C]//2019 IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2019:870-887.

[17] ANDRYSCO M, KOHLBRENNER D, MOWERY K, et al. On subnormal floating point and abnormal timing[C]// 2015 IEEE Symposium on Security and Privacy. San Jose, USA: IEEE, 2015:623-639.

[18] TARAM M, REN X, VENKAT A, et al. SecSMT: securing SMT processors against contention-based covert channels[C]//USENIX Security Symposium. Boston, USA: USENIX Association, 2022:3165-3182.

[19] BERNSTEIN D J, CHOU T, SCHWABE P. McBits: fast constant-time code-based cryptography [C] // Cryptographic Hardware and Embedded System. Santa Barbara, USA: Association for Computing Machinery, 2013:250-272.

[20] ZAHUR S, EVANS D. Obliv-C: a language for extensible data-oblivious computation[J]. Cryptology ePrint Archive, 2015,1153:1-20.

[21] KIM T, PEINADO M, MAINAR-RUIZ G. STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud[C]//USENIX Security symposium. Bellevue, USA: USENIX Association, 2012: 189-204.

[22] KIRIANSKY V, LEBEDEV I, AMARASINGHE S, et al. DAWG: a defense against cache timing attacks in speculative execution processors[C]//International Symposium on Microarchitecture. Fukuoka, Japan: IEEE, 2018: 974-987.

[23] TOWNLEY D, PONOMAREV D. SMT-COP: defeating side-channel attacks on execution units in SMT processors [C]// International Conference on Parallel Architectures and Compilation Techniques. Seattle, USA: IEEE, 2019:43-54.

[24] WU X, HE Y, ZHOU Q, et al. Partial-SMT: core-scheduling protection against SMT contention-based attacks[C]// International Conference on Trust, Security and Privacy in Computing and Communications. Guangzhou, China: IEEE, 2020:378-385.

[25] KERAMIDAS G, ANTONOPOULOS A, SERPANOS D N, et al. Non deterministic caches: a simple and effective defense against side channel attacks[J]. Design Automation for Embedded Systems, 2008,12:221-230.

[26] LIU F, LEE R B. Random fill cache architecture[C]// International Symposium on Microarchitecture. Cambridge, UK: Association for Computing Machinery, 2014:203-215.

[27] QURESHI M K. New attacks and defense for encrypted-address cache[C]// International Symposium on Computer Architecture. Phoenix, USA: IEEE, 2019: 360-371.

[28] CHEN G, WANG W, CHEN T, et al. Racing in hyperspace: closing hyper-threading side channels on SGX with contrived data races[C]//IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2018: 178-194.

[29] HU W M. Reducing timing channels with fuzzy time[J]. Journal of Computer Security, 1992,1(3-4):233-254.

[30] SCHWARZ M, MAURICE C, GRUSS D, et al. Fantastic timers and where to find them: high-resolution microarchitectural attacks in JavaScript[C]//Financial Cryptography and Data Security. Sliema, Malta: Springer, 2017:247-267.

[31] ZHANG Y, REITER M K. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud[C]//Conference on Computer & Communications Security. Berlin, Germany: Association for Computing Machinery, 2013:827-838.

[32] MUSHTAQ M, AKRAM A, BHATTI M K, et al. Nightswatch: a cache-based side-channel intrusion detector using hardware performance counters[C]//International Workshop on Hardware and Architectural Support for Security and Privacy. Los Angeles, USA: Association for Computing Machinery, 2018:1-8.

[33] MIRBAGHER-AJORPAZ S, POKAM G, MOHAMMADIAN-KORUYEH E, et al. Perspectron: detecting invariant footprints of microarchitectural attacks with perception [C] // International Symposium on Microarchitecture. Athens, Greece: IEEE, 2020:1124-1137.

[34] KOCHER P, HORN J, FOGH A, et al. Spectre attacks: exploiting speculative execution[J]. Communications of the ACM, 2020,63(7):93-101.

[35] KIRIANSKY V, WALDSPURGER C. Speculative buffer overflows: attacks and defenses[EB/OL]. (2018-07-10) [2023-02-21]. https: // arxiv. org/pdf/1807.

03757. pdf.

[36] WIKNER J, RAZAVI K. RETBLEED: arbitrary speculative code execution with return instructions[C]// USENIX Security Symposium. Boston, USA: USENIX Association, 2022:3825-3842.

[37] BARBERIS E, FRIGO P, MUENCH M, et al. Branch history injection: on the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks[C]// USENIX Security Symposium. Boston, USA: USENIX Association, 2022:971-988.

[38] Intel. Speculative load disordering[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/speculative-load-disordering.html.

[39] Bitdefender. SWAPGS attack mitigation solutions[EB/OL]. [2023-02-21]. https://www.bitdefender.com/business/swapgs-attack.html.

[40] LIPP M, SCHWARZ M, GRUSS D, et al. Meltdown[EB/OL]. (2018-01-03)[2023-02-21]. https://arxiv.org/pdf/1801.01207.pdf.

[41] VAN BULCK J, MINKIN M, WEISSE O, et al. Foreshadow: extracting the keys to the IntelSGX kingdom with transient out-of-order execution[C]// USENIX Security Symposium. Baltimore, USA: USENIX Association, 2018:991-1008.

[42] Intel. Rogue system register read[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/rogue-system-register-read.html.

[43] STECKLINA J, PRESCHER T. LazyFP: leaking FPU register state using microarchitectural side-channels[EB/OL]. (2018-06-19)[2023-02-21]. https://arxiv.org/pdf/1806.07480.pdf.

[44] Intel. Microarchitectural data sampling[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-microarchitectural-data-sampling.html.

[45] VAN BULCK J, MOGHIMI D, SCHWARZ M, et al. LVI: hijacking transient execution through microarchitectural load value injection[C]// IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2020: 54-72.

[46] Intel. Speculative execution side channel mitigations[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-execution-side-channel-mitigations.html.

[47] ARM. Vulnerability of speculative processors[EB/OL]. [2023-02-21]. https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability.

[48] LLVM. Speculative load hardening[EB/OL]. [2023-02-21]. https://llvm.org/docs/SpeculativeLoadHardening.html.

[49] MILBURN A, SUN K, KAWAKAMI H. You cannot always win the race: analyzing the LFENCE/JMP mitigation for branch target injection[EB/OL]. (2022-03-08)[2023-02-21]. https://arxiv.org/pdf/2203.04277.pdf.

[50] Intel. Retpoline: a branch target injection mitigation[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/retpoline-branch-target-injection-mitigation.html.

[51] DUTA V, GIUFFRIDA C, BOS H, et al. PIBE: practical kernel control-flow hardening with profile-guided indirect branch elimination[C]// International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: Association for Computing Machinery, 2021:743-757.

[52] AMIT N, JACOBS F, WEI M. JumpSwitches: restoring the performance of indirect branches in the era of Spectre[C]// USENIX Annual Technical Conference. Renton, USA: USENIX Association, 2019:150.

[53] GRUSS D, LIPP M, SCHWARZ M, et al. KASLR is dead: long live KASLR[C]// Engineering Secure Software and Systems Symposium. Bonn, Germany: Tugraz, 2017:161-176.

[54] REIS C, MOSHCHUK A, OSKOV N. Site isolation: process separation for web sites within the browser[C]// USENIX Security Symposium. Santa Clara, USA: USENIX Association, 2019:1661-1678.

[55] Intel. L1 terminal fault[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-l1-terminal-fault.html.

[56] Intel. CPUID enumeration and architectural MSRs[EB/OL]. [2023-02-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/cpuid-enumeration-and-architectural-msrs.html.

[57] YU J, YAN M, KHYZHA A, et al. Speculative taint tracking (STT) a comprehensive protection for speculatively accessed data[C]// International Symposium on Microarchitecture. Columbus, USA: IEEE, 2019:954-968.

[58] YU J, MANTRI N. Speculative data-oblivious execution (SDO): mobilizing safe prediction for safe and efficient speculative execution[C]// International Symposium on Computer Architecture. Valencia, Spain: IEEE, 2020: 707-720.

[59] ZHAO Z N, JI H, MORRISON A, et al. Pinned loads: taming speculative loads in secure processors[C]// International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland: ASPLOS, 2022:314-328.

[60] YAN M, CHOI J, SKARLATOS D, et al. Invisispec: making speculative execution invisible in the cache hierarchy[C]// International Symposium on Microarchitecture. Fukuoka, Japan: IEEE, 2018:428-441.

[61] SAILESHWAR G, QURESHI M K. Cleanupspec: an 'undo' approach to safe speculation[C]// International Symposium on Microarchitecture. Columbus, USA: Association for Computing Machinery, 2019:73-86.

[62] TANG B, WU C, WANG Z, et al. SPECBOX: a label-based transparent speculation scheme against transient execution attacks[J]. IEEE Transactions on Dependable and Secure Computing, 2022,20(10):827-840.

A survey of timing-based side channel attacks and defenses

TANG Bowen, WU Chenggang, WANG Zhe

(State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(University of Chinese Academy of Sciences, Beijing 100049)

Abstact

The designers of modern processors have proposed a variety of optimizations to pursuit extreme performace, yet they often underestimate the hidden security risk behind them. Timing-based side channel attacks are the most famous type of security threats. With the emergence of transient execution attacks, the capability of timing-based side channel attacks is further extended so that the foundation of many upper defenses is shaken. To defeating these attacks, a large number of defenses have been proposed by processor vendors and developers. They have different protection scopes and performance overheads. Meantime, newer transient execution vulnerabilities and covert channels are being discovered continuously to bypass these mechanisms. The war between attacks and defenses of timing-based side channels is ignited. This work will introduce the principles of various attack and defense techniques, and review the protection scopes and performance overheads of the representative defense work. This work aims to provide a comprehensive roadmap for new hardware and software development, and also inspire the following security technology exploration.

Key words: microarchitecutre, timing-based side channel attack, covert channel, transient execution attack, speculative execution, defense mechanism