

瞬态执行漏洞攻击及防御综述^①李 晔^{②*} 李沛南^{**} 赵路坦^{**} 侯 锐^{③**} 张立新^{*} 孟 丹^{**}

(* 中国科学院计算技术研究所 北京 100190)

(** 中国科学院信息工程研究所 北京 100093)

(***) 中国科学院大学 北京 100049)

摘 要 介绍了瞬态执行漏洞的攻击及防御的研究现状,概述了利用现代处理器中由乱序执行和推测执行机制引起的超前执行窗口的熔断类漏洞和幽灵类漏洞,对观测微体系结构状态变化并窃取敏感信息的 16 个变种漏洞进行了分类,包括提前执行例外或中断后的指令漏洞和错误执行分支预测或访存消歧后的指令漏洞。讨论了缓存瞬态执行漏洞攻击的 3 个步骤、可恶意训练的微体系结构硬件和对应 3 个攻击步骤的防御方法。展望了处理器体系结构的发展方向,在设计处理器的伊始,将性能优化机制进行周密的安全性分析,兼顾性能和安全的架构是未来处理器微体系结构设计的重要趋势之一。

关键词 瞬态执行;侧信道攻击;分支预测;乱序执行;内存消歧;恶意训练分支预测器

0 引 言

2018 年初,谷歌公布了幽灵漏洞^[1]与熔断漏洞^[2],揭示了当前主流的高性能处理器架构中存在着泄露敏感数据的安全风险。这 2 种漏洞的揭露像是打开了潘多拉魔盒,一系列变种接踵而至,层出不穷^[3-5]。这些变种利用的根本原理是处理器中的乱序执行及推测执行。这 2 种技术可显著地提升性能,被作为最基本的优化技术,广泛应用于主流的商业处理器中。个人终端以及云端处理器的内核^[2]或其他进程^[2]的数据、状态寄存器^[6],运行在 SGX 中的私密数据^[3],系统管理模式 SMM 以及虚拟机管理器 VMM^[7]中的敏感信息都暴露给攻击者。幽灵和熔断漏洞在学术界与工业界引起了巨大的震动,微体系结构设计厂商 Intel、AMD 以及 ARM 陷入恐慌,云端至终端的运营商和使用者也都在惶恐不安。各大公司研究者们积极并迅速地提出各种防御机

制,避免由于此类漏洞引起重大信息泄露,以求平稳地度过安全危机。

在推测执行与乱序执行机制中,存在一些指令被提前推测执行,但最终却被废弃、无法提交,这些指令被统称为瞬态指令。虽然瞬态指令的执行指令不会影响体系结构的状态,不改变程序的运行结果,但其在微体系结构上遗留的信息可能隐含敏感信息,继而被攻击者挖掘。故此类漏洞被称为瞬态执行漏洞或推测执行侧信道^[8]。深究瞬态执行漏洞产生的根源,是传统的体系结构均以性能优先作为设计导向,对于各种性能优化技术的安全性并没有做深入、详实的分析。

本文从幽灵漏洞与熔断漏洞及相关变种入手,研究已知的瞬态执行漏洞的软硬件防御方案,剖析现有高性能处理器中面临的安全隐患,讨论引入安全性因素后,处理器体系结构的发展方向,并提出相应的策略与建议。

① 中国科学院前沿科学重点研究(QYZDB-SSW-JSC010)和国家自然科学基金优秀青年科学基金(61522212)资助项目。

② 男,1987 年生,博士;研究方向:计算机体系结构增强扩展安全;E-mail: liye01@ict.ac.cn

③ 通信作者,E-mail: hourui@iie.ac.cn

(收稿日期:2019-08-19)

1 瞬态执行漏洞

Spectre 变种 1 (bounds check bypass, BCB)^[1]

属于攻击者利用条件分支是否跳转会被错误推测的特性。通过执行特定的程序流,恶意训练分支预测器的执行,从而确保在受害者程序执行时,依据跳转历史信息,分支预测器将推断此次分支跳至本不该执行的分支。如表 1 所示,x 为访问数组的索引,第 1 行为常见的边界检查操作,第 2 行为 x 合法时,对数组的访问。

表 1 变种 1 攻击示例

行号	代码
1	if (x < array1 _ size)
2	y = array2[array1[x] × 256];

若此段代码中,数组 array1 后的空间存储敏感信息,攻击者便可以通过训练分支预测器,使得在 x 越界时,仍然推测执行第 2 行代码,从而访问到 array1 索引后的敏感数据。由于此信息属于敏感区域,对攻击者不可见,为此,通过将此信息作为间接索引,访问攻击者可见区域 array2。推测执行的指令不会改变体系结构的状态,但考虑到实现复杂度,微体系结构中发生的改变并不会被抹去。如果推测执行读取了新的缓存行(cache line),在发现推测错误后,该状态不会被消除,再次访问时将会因为命中而仅有较短的访问时间。因此,攻击者可以通过检测对共享区域访问时间的变化推出由 array1 访问到的敏感数据。

Spectre 变种 2 (branch target injection, BTI)^[1] 在体系结构中,分支预测器不仅推测是否跳转,对于间接跳转及函数调用指令,还会预测跳转方向。变种 2 中,攻击者通过恶意训练分支预测器,使得受害者程序在执行时,依据分支目标缓冲器(branch target buffer, BTB),跳转至特定的程序段,例如代码 1 中的第 2 行。与变种 1 相同,在推测执行中遗留的对于微体系结构的影响会被攻击者观测到,从而推出敏感信息。

Meltdown 变种 3 (rogue data cache load, RDCL)^[2] 在页表项中有特定的标识位来区分页面的访问权限,包括用户权限或内核权限。操作系统默认硬件可以保证用户与内核的访问是隔离的,为降低系统调用的开销,内核的空间都被映射到每个用户进程的虚拟地址空间。在程序执行时,由页表缓存(translation lookaside buffer, TLB)进行虚拟地址到物理地址的转换。若硬件监测到运行状态不满足页面访问权限,将产生例外,并告知操作系统。在 Intel 处理器中,页表访问权限标识的检查是在指令提交阶段完成。因此,攻击者在程序中恶意访问内核数据,在指令提交之前,后续的指令可以使用获取的数据影响攻击程序空间的状态,继而被攻击者感知并获取到内核数据。

Spectre-NG 变种 1.1 (bounds check bypass store, BCBS)^[4] 变种 1 中攻击者通过分析推测执行中读操作对于微体系结构的改变获取敏感数据。而变种 1.1 主要利用推测执行时写操作的影响。例如示例代码 2 表 2 所示,若 c[y] 指向存储函数返回地址的区域,z 为恶意代码片段的入口,第 2 行代码运行过程中若执行返回指令,推测时写操作的值将被前递,从而将正常的执行流转向恶意代码,从而将敏感信息泄露。

表 2 变种 1.1 攻击示例

行号	代码
1	if (y < array _ len)
2	c[y] = z;

Spectre-NG 变种 1.2 (read-only protection bypass, RPB)^[4] 在执行访存操作时,TLB 会检查页表项的读写权限标识位,从而确定对当前页面的操作是否合法。在 Intel 处理器中,对权限检查是在指令提交阶段判断。因此,攻击者在对某段只读空间进行写操作后,如与控制流相关的代码指针等,在等待提交的时间窗口内,被修改的数据会被后续指令使用,跳转至恶意代码片段,从而泄露信息。

Spectre-NG 变种 3a (rogue system register read, RSRR)^[6] 用户态的程序在读取仅内核态可

访问的寄存器时,会产生异常。但与变种3类似,该异常在 ARM 及 Intel 的处理器中是在指令提交阶段处理,在等待异常被处理前的窗口内,攻击者有机会将敏感数据窃取。

Spectre-NG 变种 4 (speculative store bypass, SSB)^[9] 处理器采用读写队列(load store queue)统一处理向存储结构发起的访存请求,既要保证访存一致性,同时为了达到足够高的性能,还支持访存推测机制。在某访存指令地址未被解析时,访存推测机制推测后续的访存指令与该指令间是否存在一致性要求。若推测后续访存不依赖于之前的指令,则可以先执行。在后续若发现存在一致性要求,则会置推测执行的访存数据无效。与分支推测相同,这种推测机制对体系结构无影响,但在微体系结构残留的敏感信息可能被攻击者获取。在表3中,array1[x]原本存储敏感数据,a[x-5]也指向array1[x],第2行将敏感数据覆盖为非敏感数据。但这段代码的执行存在这种执行场景:a[x-5]的地址计算较长,但第3行的计算速度较快,第3行超前于第2行执行。即第3行本应该访问非敏感数据,但此时由于超前执行,将敏感数据暴露给攻击者。

表3 变种4攻击示例

行号	代码
1	a = &array1[5];
2	a[x-5] = 3;
3	y = array2[array1[x] × 256];

Foreshadow 变种 5 (L1 terminal fault, L1TF)^[7,10] 页表项中的 Present 标识位被置位0或保留位被标识为1时,经 TLB 页表转换后的物理地址无效,会产生例外。但同样由于例外处理的时机较晚,仍然存在推测执行的时间窗口。当该地址所指向的敏感数据仍然在一级缓存时,便可能被读取并被后续指令使用,从而导致信息泄露。采用这种方法,攻击者成功地从 SGX、操作系统、SMM 以及 VMM 中窃取私密数据。

Spectre-NG SpectreRSB^[11,12] 返回指令的跳转目标由 RSB(return stack buffer)来预测。在 RSB 中存储的地址与实际软件栈中的地址不同时就会引

起错误推测。有多种方法可以导致返回地址错误推测,包括直接或推测地对返回地址进行重写、上下文切换时共享上一敏感进程的返回地址信息、在 RSB 为空时返回指令采用 BTB 的信息来推测跳转目标(Intel)。通过改变返回指令的预测方向,可以执行恶意代码片段,如表1的第2行,将敏感数据泄漏给攻击者。

Spectre-NG LazyFP^[13] 在上下文切换时,由于 FPU 及 SIMD 寄存器保存起来较为耗时,因此,处理器将保存操作延后处理,在此之前会将 FPU 及 SIMD 单元设置为不可用,当后续指令访问相关寄存器时将产生例外。但后续推测执行的指令依然有可能泄露敏感数据。

SgxPectre^[3] 为增强原处理器架构的安全性,Intel 提出 SGX(Intel software guard extentions)。通过增加一组指令,将程序划分为可信部分与非可信部分,并为程序的可信部分创建独立的执行环境,称为 enclave,防止不可信部分对于可信区域敏感信息的窃取。而 SgxPectre^[3] 通过利用 Intel 处理器中, enclave 与不可信区域仍然共享 RSB 与 BTB;而且,采用 SpectreRSB^[11] 的攻击方法,在进入 enclave 前,恶意训练 BTB 并强制排空 RSB 内容,使得进入 enclave 后,第1条返回指令跳转至恶意代码片段,将 enclave 内部的数据暴露给攻击者。

SpectrePrime/MeltdownPrime^[14] 多核处理器中也需要维护缓存一致性,因此一个核中对于缓存的操作可能会影响到其他核中缓存的状态。SpectrePrime/MeltdownPrime 即利用了推测执行写操作时,会将其他核中同地址的缓存行置位无效。此时,攻击者再访问该核中对应的数据,则会由于缓存失效而有较长的访问延迟。与变种1中分析敏感数据的原理相同,攻击者也可以从此类变化中获取敏感数据。

NetSpectre^[5] 采用推测执行漏洞,除了可以窃取本地计算机数据,NetSpectre 通过监测远端计算机的缓存状态或者计算单元,均成功地获取到敏感数据。

SPOILER^[15] 在访存推测错误情况下会引起较大的性能代价,实验发现 Intel 处理器中低 20 位

地址相同时便推测执行。利用此机制,可以反推虚拟地址与物理地址的映射关系,发起 Rowhammer 类型的攻击。

ZombieLoad^[16] Intel 处理器中存在多种缩短访存延迟的前递机制(forwarding mechanism),例如,将写缓冲区(store buffer)、填写缓冲区(fill buffer)和读端口(load ports)中的有效数据及时地赋值给读操作。但在具体的实现中,Intel 认为已经提交却仍然存在于缓冲区或端口中的数据依旧有效。在后续执行过程中若发现数据失效,则刷新流水线重新执行,从而保证程序的正确性。但这种机制可能会将本不该前递的敏感数据传给攻击者,并改变微体系结构,导致信息泄露。

2 处理器漏洞

在传统的高性能体系结构中,首要的设计目标是提升性能,即每个时钟周期内所执行的指令数。然而,由于程序本身存在控制依赖及数据依赖,在执行过程中还存在资源依赖,这些依赖关系将使得流水线堵塞,降低指令及数据并行处理的能力。为此,微体系结构设计者采用多种技术来消除这些依赖带来的性能损失,以提升指令级并行处理能力、数据级处理能力以及线程级处理能力^[17]。而幽灵漏洞、熔断漏洞及相关变种的产生,恰恰是利用了 2 种基本的性能优化方式,即乱序执行和推测执行。

2.1 乱序执行

现有体系结构有 3 种成熟的性能优化方法用来提升单时钟周期内指令的并行处理能力。采用多级流水线技术在同一时钟周期执行多条指令,即并行处理不同阶段的指令,在时间上实现指令集并行;通过多发射技术,减少资源依赖造成的流水线堵塞,在空间上提升指令并行处理能力;通过采用动态流水线技术,采用重命名技术消除数据依赖,允许源操作数准备好的、不存在依赖关系的指令先于当前被堵塞的指令执行,从而进一步提升指令的并行处理能力^[18]。

当一条指令被堵塞或者处理需要较长时间时,指令的并行处理能力越高,就会有越多的后续源操

作数已准备好的指令被提前执行。在处理器发现一条指令发生例外时,需要撤销提前执行的指令对体系结构造成的影响,比如寄存器映射关系、各个指令队列中的状态位等。但对于微体系结构的影响,比如将后续的访存数据加载到缓存中,将后续的页表项加载到 TLB 中等。这些微体系结构状态的改变并不影响程序的正确性。因此,考虑到硬件实现的复杂度,超前执行的指令已改变的微体系结构并不会撤销并恢复成该指令未执行前的状态。

2.2 推测执行

通常,程序并不是线性地按指令序列执行,而是通过许多分支、跳转、返回指令,来控制程序的方向,执行所需的操作。然而,在这些指令的地址未被解析时,并不能确定程序真正的跳转方向。此时,若堵塞后续指令,待分支解决后再执行会严重影响程序的执行效率。因此,处理器常采用多种机制存储程序执行的跳转历史,包括跳转方向及是否跳转。处理器依据历史信息预测可能的执行方向,提前跳转至相应的指令段,与正常的指令执行一样,完成重命名、记录于 ROB 中。与此同时,处理器会保留观察点,记录分支跳转前流水线的状态。待分支解决后,若预测正确,则可以将执行结果提交,消除了由分支指令引入的控制依赖;若预测错误,则将流水线恢复至跳转前的状态。

依据内存一致性模型,当执行流中写操作后有读操作,且写操作的地址还未被解析时,后续的读操作不能执行,以免后续的读操作依赖于该写操作。但在实际的实现中,为了降低这种依赖关系导致的流水线堵塞,处理器引入内存消歧(memory disambiguation)机制^[19],记录读取指令的执行历史,并预测该指令是否可以在写操作的地址解析出来之前提前执行。在地址解析后,若不存在依赖关系,可以继续使用提前读回的数据;若存在依赖关系,将回退并重新执行读操作及后续的指令。

与乱序执行机制相同,推测执行与内存消歧方法也存在提前执行指令却不恢复微体系结构状态的情形。

2.3 漏洞分类

基于漏洞利用的机理,已知的变种可以按表 4

分为熔断类漏洞与幽灵类漏洞。熔断类漏洞利用发生例外或中断的指令,提前乱序执行后续的指令;而幽灵类漏洞利用了分支预测或访存消歧后,错误地推测执行后续的指令。这2个漏洞都在利用超前执行的窗口内将敏感信息带入微体系结构。在此之后,攻击者识别出微体系结构状态的变化,从而获取敏感信息的值。

表4 熔断和幽灵漏洞分类

变种名称	漏洞类型
变种 3 ^[2]	熔断
变种 1.2 ^[11]	熔断
变种 3a ^[6]	熔断
变种 5 ^[7,10]	熔断
LazyFP ^[13]	熔断
MeltdownPrime ^[14]	熔断
ZombieLoad ^[16]	熔断
变种 1 ^[1]	幽灵
变种 2 ^[1]	幽灵
变种 1.1 ^[11]	幽灵
变种 4 ^[6]	幽灵
SpectreRSB ^[11,12]	幽灵
SgxPectre ^[3]	幽灵
SpectrePrime ^[14]	幽灵
NetSpectre ^[5]	幽灵
SPOILER ^[15]	幽灵

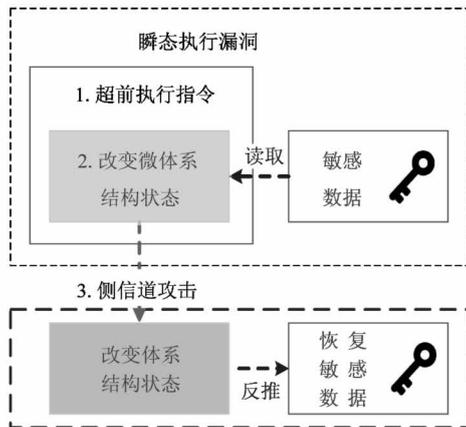


图1 瞬态执行攻击步骤

击进程的执行历史会影响推测机制对于执行流的判断。已知的可以恶意训练的模块包括:

PHT (pattern history table): 分支预测器采用 PHT 判断分支指令是否跳转。因此,攻击者可以通过训练 PHT 来构造瞬态执行攻击,如变种 1。

BTB (branch target buffer): 处理器采用 BTB 记录每条分支或间接跳转指令的 PC 值及跳转方向。变种 2 及 SgxPectre 即利用 BTB 的错误推测来执行恶意代码片段。

RSB (return stack buffer): 当 RSB 存储的信息与软件栈不同时,会存在错误推测的情况。SpectreRSB 采用直接、推测的方式改变 RSB 内容,或利用处理器对返回指令推测机制的漏洞执行恶意代码,窃取敏感信息。

STL (store to load): 为避免由于写操作地址未被解析,无法判断是否与后续读操作存在地址冲突导致的流水线堵塞,微处理器引入内存消歧。内存消歧依据之前的执行历史,预测当前的读操作是否与地址未解析的写操作存在地址冲突。变种 4 及 SPOILER 即利用此机制在错误推测时引起的微体系结构改变从而窃取信息。

此外,为进一步节省索引历史信息引入的面积,仅采用指令的部分信息作为索引值。这使得攻击者在同一进程或不同进程间都可以构造错误推测。

熔断类漏洞通过利用指令例外产生到处理的时间窗口来获取敏感数据。在微体系结构中,存在多种方式均可以被用来构造泄露信息的渠道,如越权访问内核区域、越权写只读区域、越界访问数组、采

3 攻击原理

完成瞬态执行漏洞的攻击具体可以分为 3 个步骤,如图 1 所示。首先,攻击者需要构造错误推测或指令例外的产生,创造超前执行的条件;其次,在超前执行的窗口内,恶意代码片段需要改变微体系结构的状态;最后,可以通过侧信道方法观测微体系结构的变化,提取敏感信息。

3.1 构造错误推测或例外

幽灵类漏洞通过恶意训练推测执行的历史信息,诱使受害者进程错误地推测执行特定的代码。在单核处理器中,微体系结构采用多种硬件资源记录程序的执行历史。为降低硬件实现的复杂度,所有的进程共享这些资源。在受害者进程执行时,攻

用无效的地址转换表项、访问地址不对齐等。这些例外都有可能被攻击者用于窃取数据。但需要注意,许多例外的发现及处理比较及时,难以构造有效的攻击。

3.2 改变微体系结构状态

如前文所述,由于错误推测或例外的延后处理,超前执行的指令对于体系结构上的改变在流水线发现错误后会被恢复至之前的状态。但微体系结构上的改变,由于并不影响程序的正确性,因而考虑到硬件实现的复杂度,该修改不会被撤销。例如:某内存数据是否在缓存、缓存的一致性状态,甚至是多核间的计算资源冲突。

为真正改变微体系结构的状态,在具体攻击的实施中,还需要满足 2 个条件:知悉恶意代码片段执行前的微体系结构状态,例如,通过 CLFLUSH 指令将某数据从缓存中移除,或通过用数组将缓存内容替换;为恶意代码片段构造较长的执行窗口,例如,通过将推测所依赖的分支指令或发生异常的指令源操作移出缓存,或构造较长的数据依赖链。

3.3 侧信道分析微体系结构的变化

目前,已知有多种方法识别出在被攻击指令执行前后微体系结构状态的变化,这类方法被称为侧信道。通过侧信道方法可以窃取在微体系结构中所隐含的私密信息。例如,窃取加密算法的密钥、用户键入值、内核地址映射信息等。已知的方法包括缓存侧信道、时间侧信道、功耗侧信道以及电磁侧信道等。

由于缓存侧信道实现简单,精度较高,因此广泛应用于真实的攻击中。高性能处理器中一般采用多级缓存结构,由于访问不同层级的缓存延迟不同,所以攻击者可以探测数据区域的访问时间或指令的执行时间,从而识别出受害者进程的执行流或者敏感数据。已知的瞬态执行漏洞均采用缓存侧信道方法来反推敏感信息。

已知缓存侧信道攻击分为 2 大类。当攻击者没有获取受害者进程的地址空间分配信息的权限时,可采用基于最小驱逐集策略 (eviction set based method) 的方法,如 Prime + Probe、Evict + Time、Prime-Abort;或者攻击者可以利用已知受害者进程

的地址空间,构造基于共享数据区策略的攻击 (shared memory based method),如 Flush + Reload、Flush + Flush。第 2 种方法较第 1 种方法更加灵活、危害性也更高。它不仅可以窃取单核内的敏感信息,而且可以窃取共享末级缓存的数据。表 5 以 Flush + Reload 算法为例说明其攻击原理。

表 5 Flush + Reload 攻击算法

步骤	算法
1	攻击者获取受害者进程在执行时能影响到的数据区地址,称为观测区域
2	攻击者将观测区域数据刷至高级缓存或内存中
3	执行受害者进程,读取观测区域数据,改变缓存的状态
4	攻击者检测重新读取观测区域数据的时间,若访问时间较短,说明受害者进程访问过该数据
5	通过分析受害者程序,获得敏感信息

4 漏洞防御

对应瞬态执行攻击的 3 个步骤,其防御方法有 3 类措施。对于攻击,第 1 步,防御幽灵类漏洞可以通过防止恶意训练分支预测器或内存消歧,防御幽灵类攻击可以通过及时处理避免例外的产生。第 2 步,可以防止超前执行的指令对于微体系结构上的改变。第 3 步,防止攻击者具备有效的途径通过侧信道方法将微体系结构的变化提取出来。

4.1 防止被恶意训练及隔离敏感数据

对于幽灵类漏洞变种 1、变种 2、SgxPectre 以及 SpectrePrime, Intel 提出更新微码的方法,在程序中插入特定的指令,针对性地防止不同等级的安全域之间分支预测信息的共享。如通过 SMEP 防止用户态和内核态的共享;通过 IBPB 防止不同逻辑核之间的共享;通过 IBRS 防止逻辑核内不同进程间的共享^[20]。Google 提出的 Retpoline^[21],将间接跳转和函数调用指令替换为安全的返回指令实现,预测信息不再通过分支目标缓冲区 (BTB),而是由返回地址来实现,从而防止被恶意训练。BRB^[22]通过在上下文切换时,将分支预测器历史清空,从而阻断攻

击程序与受害者之间的分支训练。同时,为 BRB 引入硬件缓存模块,负责保存较为重要的分支预测信息,从而降低由清空引入的代价。

对于熔断类漏洞变种 3 及 Meltdown,操作系统不再依赖硬件对权限进行检查产生例外,而是引入内核页表隔离(kernel page table isolation, KPTI)机制^[23],将内核空间从用户空间中隔离,用户程序在执行时将无法直接访问到内核数据。变种 5 攻击成功有 2 个要素:存在无效的虚拟地址到物理地址的映射,敏感数据在一级缓存中。因此防御变种 5 可以通过在页表项无效时将物理地址设置为无效值,或者在页无效时将敏感数据擦除或移出缓存。

4.2 防止微体系结构上的变化

对于幽灵类漏洞变种 1、变种 2、变种 3、SgxPectre 以及 SpectrePrime,防止推测执行对缓存结构的变化可以通过堵塞推测访存的执行,例如:可以在推测的访存指令前添加 LFENCE 指令,在之前的分支指令均被解决后,即确定跳转方向后再执行访存指令。针对幽灵漏洞,微软提出共 15 种危险模式,oo7 通过静态分析程序的二进制^[24],提取出潜在的危险片段,并在合适的地方插入 LFENCE 指令,从而保证推测执行的访存指令不改变微体系结构中缓存的状态。

另外,也可以采用硬件防御机制将推测执行引起的微体系结构状态的改变全部都转移至独立的硬件模块,不改变原本需要更新的结构。待其访存操作转为安全后,即推测状态被消除后,从缓冲区中移至真正的微体系结构,否则废弃相关改变。由伊利诺伊大学香槟分校和加州大学河滨分校分别开发的 InvisiSpec^[25]和 SafeSpec^[26]即利用了此防御原理。

InvisiSpec 针对数据缓冲区,采用与读写队列(load store queue, LSQ)一一对应的推测缓存模块(speculative buffer, SB)记录推测的访存信息,对于体系结构其他模块都不可见。在访存操作转为非推测状态期间,可能由于其他核对相同地址的写操作而失效。因此,需要重新发射访存指令,将数据从缓存或内存中重新读取,与缓存模块存储的值进行比较。若比较结果相同,则移至真正缓存,否则,采用新取回的值,废弃缓存中的值。保证推测缓存模块

也满足内存一致性模型,此过程称为核实(validation)。在一条访存操作在 ROB 第一条时,不会被由于其他指令被废弃。此时,采用核实方法会需要较多的时钟周期,容易导致流水线堵塞。经过分析总结出一系列不会违反内存一致性模型的访存操作,可以通过显露(exposure)直接将缓存内容移至缓存中。

InvisiSpec 提出了多种方法来保持性能与安全性的平衡:显露操作可以并行执行;推测缓存中存储的信息也可以支持前递操作;及早发现并废弃错误推测执行的指令;在末级缓存也引入推测缓存,从而降低由于核实现操作多次访问内存而引入的延迟。

SafeSpec 同样采用单独模块,存储推测执行过程中引起的微体系结构的变化。相比于 InvisiSpec,不仅对数据缓存的推测操作进行缓存,同时对指令以及地址转换项的内容进行缓存。为保证在执行过程中,不会由于容量不够引起缓存结构内部的冲突,通过在 LSU 中附加指针,一一指向为推测的数据访问添加的独立结构;通过在 ROB 中添加指针,指向为推测的指令访问和地址转换项添加的独立结构。

4.3 防止侧信道观测微体系结构变化

已知的幽灵类漏洞以及熔断类漏洞均通过时间信息观测微体系结构的变化,因此,谷歌浏览器通过降低时钟的精确度,从而降低对数据是否在缓存的识别精度,使得攻击者无法从改变的微体系结构中分辨出敏感信息。

对于依赖分支指令的幽灵类漏洞,文献[27]提出动态分配路径保护(dynamically allocated way guard, DAWG)方法,动态地将缓存依据不同的保护域进行隔离,保护域之间不共享缓存内容,因而不存在跨保护域的推测执行侧信道攻击。

5 结论

本文首先介绍瞬态执行漏洞的相关变种,并依据不同变种所利用的基本原理分为幽灵类漏洞与熔断类漏洞。通过分析 2 类漏洞基本的 3 大攻击步骤,提出 3 种防御点,并将现有的防御方式进行了分类。

幽灵类漏洞以及熔断类漏洞爆发的根本原因是当前的处理器架构始终以性能优先作为导向,却未对性能优化机制进行周密的安全性分析,从而导致漏洞频频出现。在现有的硬件基础上进行防御,仅能通过有限的方法对不同漏洞各个突破。并且,将不同的防御方案集成到同一系统中,势必会带来严重的性能损失。在计算机体系结构顶级国际会议 ISCA 2018 上,图灵奖获得者、精简指令集的创始人 David A. Patterson 以及 John L. Hennessy 做了特邀报告,提出当前体系结构设计面临着严重的安全漏洞,并提出需要推出体系结构 2.0,而其重要特征之一便是从处理器设计伊始就引入安全性分析。

参考文献

- [1] Kocher P, Genkin D, Gruss D, et al. Spectre attacks: exploiting speculative execution[J]. *arXiv*:1801.01203, 2018
- [2] Lipp M, Schwarz M, Gruss D, et al. Meltdown[J]. *arXiv*:1801.01207, 2018
- [3] Chen G, Chen S, Xiao Y, et al. Sgxspectre attacks: leaking enclave secrets via speculative execution[J]. *arXiv*:1802.09085, 2018
- [4] Kiriansky V, Waldspurger C. Speculative buffer overflows: attacks and defenses [J]. *arXiv*:1807.03757, 2018
- [5] Schwarz M, Schwarzl M, Lipp M, et al. Netspectre: read arbitrary memory over network[J]. *arXiv*:1807.10535, 2018
- [6] ARM. Vulnerability of speculative processors to cache timing side-channel mechanism [EB/OL]. <https://developer.arm.com/support/security-update>; ARM, 2018
- [7] Weisse O, Van Bulck J, Minkin M, et al. Foreshadow-NG breaking the virtual memory abstraction with transient out-of-order execution[R]. America: Technion, 2018
- [8] Canella C, Van Bulck J, Schwarz M, et al. A systematic evaluation of transient execution attacks and defenses [J]. *arXiv*:1811.05441, 2018
- [9] Horn, J. Speculative execution, variant 4: speculative store bypass [EB/OL], <https://bugs.chromium.org/p/project-zero/issues/detail?id=1528>; Monorail, 2018
- [10] Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution [C] // The 27th USENIX Security Symposium, Baltimore, USA, 2018: 991-1008
- [11] Koruyeh E M, Khasawneh K N, Song C, et al. Spectre returns! speculation attacks using the return stack buffer [C] // The 12th USENIX Workshop on Offensive Technologies, Baltimore, USA, 2018:676-687
- [12] Maisuradze G, Rossow C. ret2spec: speculative execution using return stack buffers [C] // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, 2018: 2109-2122
- [13] Stecklina J, Prescher T. Lazyfp: leaking fpu register state using microarchitectural side-channels[J]. *arXiv*:1806.07480, 2018
- [14] Trippel C, Lustig D, Martonosi M. Meltdownprime and spectreprime: automatically-synthesized attacks exploiting invalidation-based coherence protocols[J]. *arXiv*:1802.03802, 2018
- [15] Islam S, Moghimi A, Bruhns I, et al. SPOILER: speculative load hazards boost rowhammer and cache attacks [J]. *arXiv*:1903.00446, 2019
- [16] Schwarz M, Lipp M, Moghimi D, et al. Zombieload: cross-privilege-boundary data sampling[J]. *arXiv*:1905.05726, 2019
- [17] 胡伟武. 计算机体系结构[M]. 北京:清华大学出版社, 2011:32-69
- [18] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach [M]. Amsterdam: Elsevier, 2011:389-450
- [19] Gallagher D M, Chen W Y, Mahlke S A, et al. Dynamic memory disambiguation using the memory conflict buffer [J]. *ACM SIGPLAN Notices*, 1994, 29(11): 183-193
- [20] Intel. Speculative execution side channel mitigations[EB/OL], <https://software.intel.com/security-software-guidance/api-app/sites/default/files/336996-Speculative-Execution-Side-Channel-Mitigations>; Intel, 2018
- [21] Google. Retpoline: a software construct for preventing branch-target-injection [EB/OL], <https://support.google.com/faqs/answer/7625886>; Google, 2018
- [22] Vougioukas I, Sandberg A, Nikoleris N, et al. BRB: mitigating branch predictor side-channels [C] // HPCA 2019 the 25th International Symposium on High-Performance Computer Architecture, Washington, USA, 2019: 466-477

- [23] LWN. The current state of kernel page-table isolation [EB/OL], <https://lwn.net/SubscriberLink/741878/eb6c9d3913d7cb2b/Dec>; LWN, 2017
- [24] Wang G, Chattopadhyay S, Gotovchits I, et al. oo7: low-overhead defense against spectre attacks via binary analysis[J]. *arXiv*:1807.05843, 2018
- [25] Yan M, Choi J, Skarlatos D, et al. InvisiSpec: making speculative execution invisible in the cache hierarchy[C] //The 51st Annual IEEE/ACM International Symposium on Microarchitecture, Fukuoka, Japan, 2018: 428-441
- [26] Khasawneh K N, Koruyeh E M, Song C, et al. Safespec: banishing the spectre of a meltdown with leakage-free speculation[J]. *arXiv*:1806.05179, 2018
- [27] Kiriansky V, Lebedev I, Amarasinghe S, et al. DAWG: a defense against cache timing attacks in speculative execution processors[C] //The 51st Annual IEEE/ACM International Symposium on Microarchitecture, Fukuoka, Japan, 2018: 974-987

A survey of transient execution attacks and defenses

Li Ye^{****}, Li Peinan^{**}, Zhao Lutan^{**}, Hou Rui^{**}, Zhang Lixin^{*}, Meng Dan^{**}

(^{*} Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(^{**} Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

(^{***} University of Chinese Academy of Sciences, Beijing 100049)

Abstract

The current research status of attacks and defenses of transient execution vulnerabilities are introduced. The meltdown-type and spectre-type attacks which exploit the execution window caused by branch speculation and out-of-order execution in modern processors are expounded. Specific vulnerabilities which observe the micro-architectural changes to steal sensitive information are summarized, including sixteen variants exploiting the transient execution after exception or interrupted instruction, mis-predicted branch, and memory disambiguation. Three steps to construct transient execution attacks, vulnerable micro-architecture components and corresponding defense steps are discussed. The research direction of processor architecture is predicted. The methodology of introducing security analysis for performance optimization and balancing performance and security at the beginning of CPU design will be one of the important trends of computer micro-architecture design in the future.

Key words: transient execution, side-channel attacks, branch speculation, out-of-order execution, memory disambiguation, mis-trained branch prediction mechanism