软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(2):544−563 [doi: 10.13328/j.cnki.jos.005979]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 微架构瞬态执行攻击与防御方法*

吴晓慧[1,2], 贺也平[1,2,3], 马恒太[1], 周启明[1], 林少锋[1,2]

[1](基础软件国家工程研究中心(中国科学院 软件研究所),北京 100190)

[2](中国科学院大学,北京 100049)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 吴晓慧, E-mail:wuxiaohui@iscas.ac.cn; 贺也平, E-mail:yeping@iscas.ac.cn

**摘 要**: 现代处理器的优化技术,包括乱序执行和推测机制等,对性能至关重要.以Meltdown和Spectre为代表的侧信道攻击表明:由于异常延迟处理和推测错误而执行的指令结果虽然在架构级别上未显示,但仍可能在处理器微架构状态中留下痕迹.通过隐蔽信道可将微架构状态的变化传输到架构层,进而恢复出秘密数据,这种攻击方式称为瞬态执行攻击.该攻击有别于传统的缓存侧信道攻击,影响面更广,缓解难度更大.深入分析了瞬态执行攻击的机理和实现方式,对目前的研究现状与防御方法进行了总结.首先,介绍了处理器微架构采用的优化技术,并分析了其导致瞬态执行攻击的功能特征;然后,基于触发瞬态执行的原语对瞬态执行攻击进行系统化分析,揭示攻击面上的明显差异;最后,有侧重点地针对攻击模型中的关键步骤和关键组件总结了已有的防御方法,并展望了未来的研究方向.

**关键词**: 处理器;微架构;优化技术;瞬态执行攻击;防御方法

**中图法分类号**: TP303

中文引用格式: 吴晓慧,贺也平,马恒太,周启明,林少锋.微架构瞬态执行攻击与防御方法.软件学报,2020,31(2):544−563. http://www.jos.org.cn/1000-9825/5979.htm

英文引用格式: Wu XH, He YP, Ma HT, Zhou QM, Lin SF. Microarchitectural transient execution attacks and defense methods. Ruan Jian Xue Bao/Journal of Software, 2020,31(2):544−563 (in Chinese). http://www.jos.org.cn/1000-9825/5979.htm

## Microarchitectural Transient Execution Attacks and Defense Methods

WU Xiao-Hui[1,2], HE Ye-Ping[1,2,3], MA Heng-Tai[1], ZHOU Qi-Ming[1], LIN Shao-Feng[1,2]

[1](National Engineering Research Center of Fundamental Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

**Abstract**: Modern processor optimizations including out-of-order execution and speculative mechanism are critical for performance. Recent research such as Meltdown and Spectre exploit delayed exception handling and misprediction events to trigger transient execution, and leak otherwise inaccessible information via the CPU's microarchitectural state from instructions which are never committed. This type of attacks is called transient execution attack, which is different from the traditional cache side channel attack. It is more difficult to mitigate. This work deeply studied the mechanism and implementation of transient execution attacks, and summarized its research status and defenses. First, the optimizations adopted by the processor are introduced, and the essential causes of the transient execution attack are analyzed. Then, the transient execution attacks are generalized systematically based on the primitives that trigger the transient

* 基金项目: 核高基国家科技重大专项(2014ZX01029101-002); 中国科学院战略性先导科技专项(XDA-Y01-01)

Foundation item: CHB National Science and Technology Major Project of China (2014ZX01029101-002); Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01)

收稿时间: 2019-04-12; 修改时间: 2019-08-19; 采用时间: 2019-11-06; jos 在线出版时间: 2019-12-05

CNKI 网络优先出版: 2019-12-05 14:55:19, http://kns.cnki.net/kcms/detail/11.2560.TP.20191205.1454.009.html

execution, in order to reveal the attack surface. Finally, the defenses are summarized in the view of the key steps and components in the attack model, and the future research directions of this field are presented.

**Key words**: processor; microarchitecture; optimization technique; transient execution attack; defense method

现代计算机架构基本都是基于冯·诺依曼体系结构,其程序指令代码(指令)和数据都放在内存中,运行时中央处理器(CPU)从内存中逐条取指令并执行,指令在必要时访问内存数据.内存指令被取入CPU后,在CPU内部还要经过复杂的指令译码等过程,才能驱动CPU硬件电路相应部件,按照指令具体需求进行相应动作,最终完成指令执行.不同的CPU差异很大,即使指令集相同的CPU也有不同的电路实现,这种CPU内部硬件电路的结构称为CPU的微架构.程序开发者只要按照CPU指令集,用相应的指令编写程序(如果是高级语言,则由编译器或者解释器将高级语言语句最终转换为CPU指令),并将指令程序和相应数据加载到内存(现代通常由操作系统完成)即可执行程序.程序开发者只访问到计算机体系架构这一级,至于指令在CPU内部是如何进一步解析并最终执行的,则是由CPU微架构决定的.

CPU微架构是设计CPU的核心.通过收缩加工技术和增加时钟频率,过去几十年的处理器性能不断提高,但已到物理极限.为了提高性能,供应商将重点转移到增加内核数量和优化指令流水线.现代处理器有深度流水线,允许在不同流水线级或执行级的不同单元中并行操作.乱序执行和推测机制是现代CPU为了提高性能通常采用的优化方式.在乱序执行中,CPU不是严格按照程序指令顺序执行,而是对微指令进行重排序,即如果按顺序在后面的微指令操作数已经准备好(与前面微指令没有数据依赖关系或者数据已经计算好),则可以先于次序在前的微指令执行.推测执行[1]是指处理器推测性地执行指令以提高性能.无论是乱序还是推测执行,如果处理器稍后确定指令不应该被执行,则丢弃计算结果,“回滚”相应指令产生的计算结果(例如撤销已经修改的寄存器值).推测执行配合流水线和乱序执行,能够极大地提高CPU的运行速度.分支推测并不能保证100%的推测成功,乱序执行也可能会提前执行了没有权限的指令,这些已经完成的工作必须“撤销”,否则就会得到错误的结果.从理论上来说,无论指令执行“提交”还是“回滚”,都只与CPU微架构相关,程序员只能看到宏观指令按照程序流程执行.最近的研究表明,攻击者可以利用这些优化技术来进行强大的侧信道攻击.处理器在流水线的后期才执行内存管理单元(MMU)保护检查,Meltdown[2]就是利用乱序执行和该延迟处理机制来绕过硬件强制的内存隔离.Spectre[3]欺骗处理器推测地执行指令,将受害者的秘密数据加载到寄存器中,然后通过缓存侧信道泄漏数据.为了缓解这些攻击,操作系统必须对其设计进行重大更改,例如重新设计内核地址空间与用户空间的隔离[4],RISC-V指令集正在积极探讨从指令集层面直接支持内核页表分离.硬件厂商引入微码更新以添加新的指令来控制上述CPU优化技术[5].本文分析了瞬态执行攻击的机理和实现方式,对目前的研究现状与防御方法进行了梳理.通过对各种瞬态执行攻击的异同点进行总结抽象,提出了瞬态执行攻击模型.该模型对瞬态攻击进行了全面刻画,清楚地揭示出瞬态执行攻击的攻击面.在此基础上,针对瞬态执行的机理,总结了触发瞬态执行的原语.从攻击的角度来说,通过对攻击流程中某个(或某些)关键环节的不同实现方式进行尝试,有助于发现新的攻击变种.从防御的角度来说,在该攻击模型的任一阶段进行阻断都可以,如果是对不同攻击变种的共同攻击面进行的阻断,则可以达到更全面的防御效果.

本文第1节讨论处理器微架构采用的优化技术带来的信息泄露问题.第2节重点分析微架构瞬态执行攻击的基本原理及关键组件.第3节根据触发瞬态执行原语的不同,分别对基于乱序执行的Meltdown类型攻击和基于推测机制的Spectre类型攻击进行描述.第4节基于攻击模型中的关键步骤及组件,分别从4个方面总结防护方法.第5节进行总结,讨论现有研究中存在的问题,并对未来可能的研究方向进行展望.

## 1 处理器体系结构

当前,CPU的两大架构是CISC(复杂指令集)和RISC(精简指令集),Intel x86是CISC的代表架构,占领了95%以上的桌面计算机和服务器市场.ARM作为RISC的一种,在智能手机、可穿戴设备等移动处理器市场占有主要地位.RISC-V采用者瞄准了AI、机器学习和物联网等新兴市场.每种架构都有自己的主流应用.本文重

点关注处理器优化技术带来的安全问题,下面以高性能的 Intel 处理器体系架构为代表进行说明.

在 Intel 处理器体系结构中,流水线是由前端、执行引擎(后端)和内存子系统组成[6].前端模块将 x86 指令从主内存中读取出来并解码成微操作(micro-operation,简称μOP),μOP 随后会经过重排序缓冲区,进而被调度器发送给相关执行单元,如图 1 所示.重排序缓冲区负责寄存器分配、寄存器重命名和将结果提交到软件可见的寄存器(即提交单元).

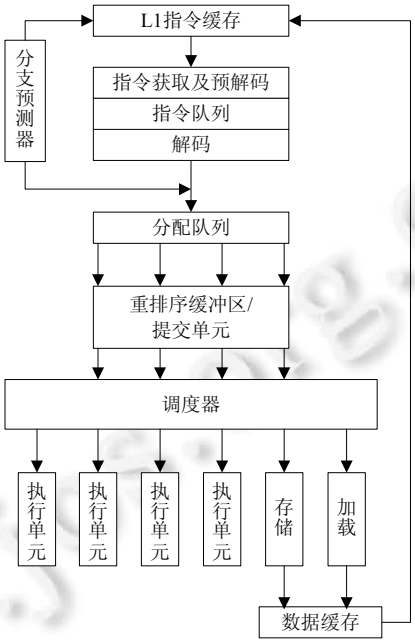


Fig.1 Illustration of a single core of Intel processor's microarchitecture

图 1 Intel 处理器的单核微架构示意图

- 指令流水线

对于复杂的微架构指令集,例如 Intel x86[7,8],在解码阶段,首先将各个指令分成较小的μOP.μOP 解码简化了处理器设计:只需要在硬件中实现实际的μOP,而不是整个复杂的指令集.该设计方案可用作超标量优化,并通过微码更新来扩展或修改某些指令的实现.指令流水线通过并行化 3 个主要阶段来提高吞吐量.首先,"指令获取-解码"单元从主内存加载指令并将其转换为相应的μOP 序列.为了最小化程序分支带来的流水线停顿,处理器的分支预测器将尝试在获取程序控制流中的下一条指令时,推测条件跳转的结果.其次,将各个μOP 调度到可用的执行单元,这可以被复制以进一步增加并行性.为了最大限度地利用可用的执行单元,同时多线程技术可以进一步交织在同一个 CPU 物理核上不同逻辑处理器的多个独立指令流的执行.最后,在指令退出阶段,μOP 的执行结果被提交到架构可见状态(例如寄存器和内存).

- 乱序执行

为了提高性能,CPU 通常会实现乱序执行.不同的乱序执行能力,带来的攻击面也是不同的.从整体上来说,ARM 处理器的乱序执行能力不如 Intel x86,而 RISC-V 开源乱序执行核心 BOOM(berkeley out-of-order machine)的性能相当于 ARM 中高端处理器水平.但由于 BOOM 使用 Rocket-Chip 的数据缓存设计,不会执行推测性内存访问,因此不受 Meltdown 攻击的影响.

乱序执行技术允许处理器不仅可以以指令流提供的顺序执行μOP,还可以并行地调度它们,尽可能地利用 CPU 执行单元,从而提高整体性能.如果μOP 所需的操作数可用,并且其对应的执行单元空闲,则即使指令流中较早的μOP 尚未完成,处理器也会开始执行.如果所有先前的μOP 尚未完成,则执行结果仅在微架构级别上可见,CPU 在重排序缓冲区中跟踪μOP 的状态.当μOP 完成执行时,它们按顺序退出,并将其结果提交到架构状态.

此外,在执行μOP 期间发生的异常均在退出期间处理.因此,CPU 执行的某些指令结果可能从未提交到架构状态.

- 推测执行

复杂软件通常不是线性的,而是包含条件分支或指令之间的数据依赖性.从理论上来说,处理器必须等到分支结果或依赖关系得到解决才能继续执行.由于停顿会显著降低性能,因此处理器会部署各种机制来推测条件分支或数据依赖性的结果.处理器继续沿着推测路径执行,在重排序缓冲区中再次缓冲结果.当推测正确时,处理器可以从重排序缓冲区中提交预先计算的结果,从而提高整体性能.但是如果推测不正确,处理器需要压缩预先计算的结果,并通过刷新管道和重排序缓冲区以回滚到最后的正确状态.因此,处理器也会执行一些指令,这些指令结果从未提交到架构状态.

- 多级高速缓存

为了加速重复的代码和数据内存访问,现代处理器为每个 CPU 物理核提供专用的 L1 和 L2 高速缓存,以及在所有物理核之间共享的最后一级高速缓存(L3 缓存).缓存组织的单位称为缓存行,大小为 64 字节.物理地址的低 6 位用于确定一行内的偏移量,其余的位用于确定存储高速缓存行的集合.当处理器访问内存地址时,会发生高速缓存命中或未命中.如果在所有高速缓存均未命中,则必须从 DRAM 获取内存行.此外,可使用 CLFLUSH 指令将指定的缓存行从整个缓存中驱逐,该指令遵循与其他内存操作相同的内存访问检查.

## 2 瞬态执行攻击

### 2.1 隐蔽信道

隐蔽信道最初是由 Lampson 在 1973 年提出的[9],定义为:不是被设计或本意不是用来传输信息的通信信道.随着后续研究的深入,隐蔽信道的含义是指允许进程以危害系统安全策略的方式传输信息的通信信道.在瞬态执行攻击过程中,瞬时指令读取特权数据之后,需要通过隐蔽信道的方式将数据发送出来,主要使用的是缓存隐蔽信道.

#### 2.1.1 缓存侧信道

侧信道攻击是由 Kocher 等人于 1996 年首先提出的,主要是指通过非直接途径泄露物理状态信息.攻击者通过测量采集,继而恢复出敏感数据.现代 CPU 使用缓存来隐藏内存延迟,然而这些延迟差异可以作为侧信道被利用[10−14].在访问内存的时候,已经被缓存的数据访问会非常快,而没有被缓存的数据访问比较慢,缓存侧信道攻击就是利用缓存引入的时间差异而进行攻击的方法.缓存侧信道技术已经被证明有效,包括 Evict+Time[12], Prime+Probe[12,15],Flush+Reload[16].Flush+Reload 方法是在单个缓存行的粒度上工作,利用共享缓存(包含最后一级缓存)进行攻击.攻击者经常使用 CLFLUSH 指令将目标内存位置的缓存刷掉,然后在一定的时间间隔后读取目标内存的数据,并测量目标内存中数据加载所需的时间.通过这个时间信息,攻击者可以猜测另一个进程是否已经将数据加载到缓存中.Flush+Reload 攻击已被用于攻击各种算法,例如密码算法[16−18]、Web 服务器函数调用[19]、用户输入[13,20,21]以及内核寻址信息[22].

缓存侧信道攻击的一个特殊使用场景是构建隐蔽信道,在这个场景中,攻击者控制隐蔽信道的发送端和接收端,也就是说:攻击者会触发程序对缓存产生影响,然后去测量数据访问的时间差.这使得攻击者可以绕过架构级别上存在的所有限制来泄漏信息.Flush+Reload 和 Prime+Probe 这两种方法都已被用于构建高性能隐蔽信道[14,23].

#### 2.1.2 其他隐蔽信道

现代 DRAM 是由通道、DIMM、行列和存储体组成的分层结构.在每个存储体内部,DRAM 单元按行和列排成二维阵列.每个存储体都有一个缓存当前活动行的行缓冲区.对于所有的内存访问,DRAM 首先激活其在存储体中的行,然后访问该行内的单元.如果该行当前处于活动状态,则直接从行缓冲区响应请求(即行命中);否则,DRAM 关闭打开的行并将所选行提取到行缓冲区以进行访问(即行未命中).Pessl 等人[24]演示了如何使用行缓冲区冲突引起的时间差异来对 x86 和 ARM 平台上的 DRAM 寻址方案进行逆向分析.基于这些知识,他们实现

了基于行缓冲区冲突的隐蔽信道.

此外,在同时多线程(SMT)架构中,每个物理核内的微架构组件比跨核攻击暴露更多.Aldaya 等人[25]探索 SMT 执行引擎共享作为侧信道泄漏源.将端口作为执行单元堆栈的目标,以便由端口争用而创建高分辨率时序侧信道.因为它不像其他缓存或基于 TLB 的攻击那样依赖于内存子系统,因此本质上是隐蔽的.

除了缓存、DRAM 及端口竞争以外,还可以从 BTB[18]或 RSB[13]等元素中提取微架构状态.通常,每个微架构隐蔽信道[22]的接收端可用于将微架构状态转换为架构状态.

### 2.2 瞬态执行攻击

#### 2.2.1 基本概念

现代处理器在引入乱序执行和推测机制的同时,设有一个重排序缓冲区,用于跟踪所有指令并按顺序提交,即与常规的有序执行没有功能差异.直观地说,如果在流水线上执行的指令依赖于前面尚未执行的指令,则无法执行.因此,为了始终保持管道满负荷,必须推测控制流或数据依赖性,甚至可能推测实际数据.被乱序执行和推测机制错误执行的指令,由于回滚而结果未在架构级别上显示的指令,称为瞬态指令[2,3,26].在现代处理器上,几乎任何指令都可能引发故障(例如页表错误或者一般的保护错误),由于异常延迟处理机制,处理器会乱序执行瞬态指令.对于推测机制,每次推测错误时都会执行瞬态指令.因此,这些瞬态指令的确执行过,然而它们的执行结果从未提交到架构级别,这种情况称为瞬态执行.

虽然瞬态指令的执行不会影响架构状态,因此无法在架构上观察到,但微架构状态可能会发生变化.瞬态执行攻击就是通过将微架构状态的变化转换为架构状态来提取敏感信息,属于侧信道攻击的范畴.Meltdown 和 Spectre 是两个典型的攻击实例,它们的共同特点是利用了瞬态执行[2,3].

通常,这些攻击利用 CPU 缓存,因为可以很容易地观察到它的变化.但是瞬态执行攻击不限于缓存:可以使用任何被改变和可观察的微架构状态(例如使用 DRAM 缓冲器[24]或执行单元拥塞[2,25]).因此,瞬态执行攻击有别于传统的缓存侧信道攻击,影响面更广,缓解难度更大.本文主要关注使用缓存隐蔽信道的瞬态执行攻击和防护方法.

#### 2.2.2 攻击模型

瞬态执行攻击包括两部分:瞬态指令执行和隐蔽信道传输,如图 2 所示.

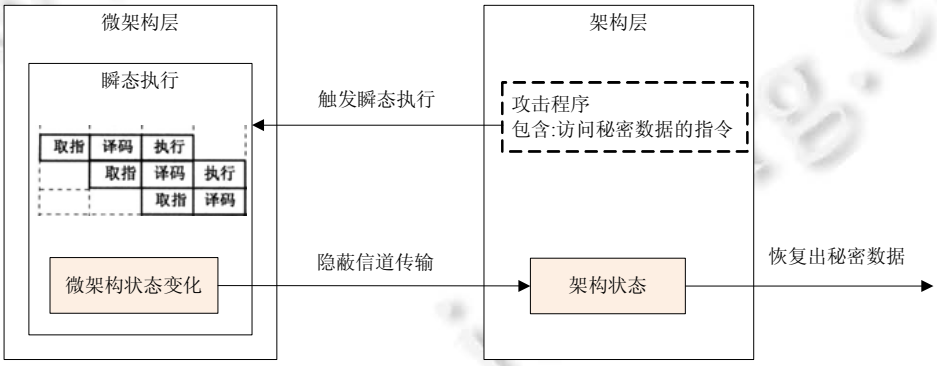


Fig.2 Transient execution attack model

图 2 瞬态执行攻击模型

其中,瞬态指令执行包括两个关键组件:a) 触发瞬态执行的原语,用来到微架构路径下进入瞬态执行;b) 时间窗口,进行瞬态执行的时间,以便通过隐蔽信道传递信息.基本的攻击流程如下:① 选择合适的原语触发瞬态指令的执行,即为瞬态指令创造足够的时间窗口来完成下一步的所有操作;② 为了实现攻击,瞬态指令序列必须访问攻击者想要获取的秘密数据并对其进行计算编码,与此同时,微架构状态受到影响而发生变化;③ 使用隐蔽信道负责将数据从微架构状态传输到持久架构状态,进而恢复出秘密数据.

- 触发瞬态执行的原语

微架构瞬态执行攻击基本使用两种不同的触发原语.第 1 种是由异常之后的乱序执行触发瞬态执行,主要包括虚拟内存异常、异常读取等.第 2 种是推测机制:分支推测、推测性存储到加载转发、地址推测等.分支推测利用了程序控制流中的时间和空间局部性,在推测是否采用条件分支或者在推测间接分支(包括调用、跳转和返回)的目标时可能发生推测触发瞬态执行.推测性存储到加载转发是一种优化,存储地址和存储数据分别作为单独的微指令无序地执行,在无序的物理核上,推测执行的存储直到提交阶段才被写回.只要存储地址和数据都可用,就允许使用先前的存储数据推测性地执行加载.推测加载会命中 L1 缓存,立即依赖指令可能会观察到值为 0 的推测.使用虚拟地址[27]或部分物理标签的地址推测技术将受到别名攻击.这些原语(见表 1)中的每一个都可以用于沿着微架构路径进入推测性执行,攻击者可以尝试以不同方式触发推测.

**Table 1** Attack classifications and corresponding triggering primitives

**表 1** 攻击分类及对应的触发原语

| 分类 | 攻击名 | 触发瞬态执行的原语 |
|---|---|---|
| 基于乱序执行的瞬态执行攻击 | Meltdown<br>Foreshadow<br>L1 Terminal Fault-OS/SMM<br>L1 Terminal Fault-VMM | 虚拟内存异常 |
| | Spectre-NG LazyFP<br>Spectre-NG Variant 3a<br>Meltdown-PK | 异常读取 |
| 基于推测机制的瞬态执行攻击 | Spectre v1<br>Spectre v1.2 | 控制流错误推测 |
| | Spectre v2<br>BranchScope<br>Spectre v1.1<br>Spoiler<br>Spectre-NG V4<br>ret2spec<br>Spectre-RSB | 分支目标注入<br>定向分支推测<br>推测性的缓冲区溢出<br>推测性加载<br>推测存储绕过<br>运行时优化返回地址推测<br>返回堆栈缓冲区推测 |

- 瞬态执行的时间窗口

假设触发瞬态执行的起始时间为 $t_1$,瞬态执行的结束时间(即 CPU 正常处理异常或验证推测结果的时间)为 $t_2$.$t_1$~$t_2$ 即为瞬态执行的时间窗口.瞬态执行的时间窗口有两个主要限制因素:流水线中瞬态指令的最大数量,以及 CPU 正常处理异常或验证推测结果的最大延迟(从周期和指令两个维度去考虑).因此,时间窗口越长,可以执行的指令数量越多,攻击的成功率越大;相反,如果时间窗口很短,瞬态执行无法完成访问秘密数据并对其编码的操作,导致竞争失败.

## 3 瞬态执行攻击实例

在瞬态执行攻击中,最关键的是触发瞬态执行.下面根据瞬态触发原语的不同,分析各种瞬态执行攻击实例.

### 3.1 基于乱序执行的瞬态执行攻击

以 Meltdown 为代表的基于乱序执行的瞬态执行攻击利用了异常延迟处理机制,即在故障指令退出时才处理异常.攻击流程如图 3 所示:没有乱序执行时,指令处理未授权的数据会发生错误,后续指令不会执行,即程序不能访问未授权的数据;而有乱序执行时,异常延迟处理机制使得流水线中后续指令提前执行,直到指令退出为止,这段时间就是瞬时执行窗口.在该时间窗口内执行的瞬态指令,可能对即将发生故障的指令的未授权结果进行计算.虽然在指令退出时处理器会丢弃此类计算结果(例如已经修改的寄存器值),但秘密信息可能通过微架构隐蔽信道泄漏.接下来根据触发瞬态执行的异常类型对攻击实例进行描述对比.

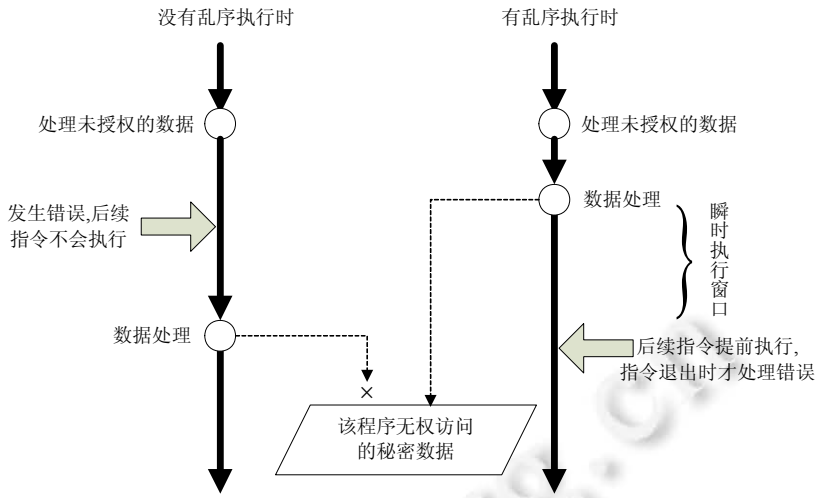


Fig.3 Flow of transient execution attack based on out-of-order execution

图 3 基于乱序执行的瞬态执行攻击流程

3.1.1 虚拟内存异常

- Meltdown

现代处理器通常为页表标记“用户/管理员”属性,表示哪些虚拟内存页属于操作系统内核.最初的 Meltdown 攻击[2]展示了如何从用户空间读取内核内存,在乱序执行的情况下,处理器标记的“用户/管理员”属性在瞬态执行时是不起效的.攻击包括 3 个步骤.第 1 步,恶意指令加载无权访问的数据到寄存器.第 2 步,在架构级处理异常之前,由于乱序执行,攻击者在微架构层执行瞬态指令序列,基于特权数据访问内存,加载到缓存.该过程构建出缓存隐蔽信道的发送端数据.第 3 步,在接收端,攻击者使用 Flush+Reload 根据访问时长来确定访问了哪个缓存行,恢复特权数据.

Meltdown 攻击者可以选择处理或抑制因未授权访问而导致的页错误.Lipp 等人[2]展示了如何通过事务内存处理器功能(如 Intel TSX[28])抑制异常来提高攻击带宽.通过在整个地址空间上迭代 Meltdown 攻击,可以转储整个地址空间.通常会将整个物理内存直接映射到内核地址,因此可以转储整个物理内存.当内核数据驻留在 CPU 缓存中时,提取率显著提高,Meltdown 甚至已经证明可以从内存中成功提取未缓存的数据[2].

程越强等人提出了 Meltdown 的变种 V3r.V3r 攻击分为两个步骤:首先,应用程序通过内核接口(比如系统调用)触发特殊推测执行事件,把目标地址上的数据加载到 L1D 缓存;然后,应用程序发起传统 Meltdown 攻击,可以可靠地读取被放入 L1D 缓存里的数据.目前,针对 Meltdown 的主要软件缓解方案是 KPTI,即分离页表防护,该方案对 V3r 同样有效.程越强等人针对 KPTI 实现中不可避免的内核内存映射提出了新的攻击变种 V3z.V3z 攻击分为两步:第 1 步,利用 V3r 复制出特权内存位置的数据(例如系统调用、中断/异常处理、eBPF 等),将其作为探测目标数据的桥梁;第 2 步,使用如下示例代码来读取目标地址上的秘密数据.

```
if (x < array1_size)
{
  temp &= array2[array1[x] * 512 + offset];
}
```

- Foreshadow

Van Bulck[26]提出了 Foreshadow,一种针对 Intel SGX 技术的 Meltdown 类型攻击.未经授权访问飞地的内存通常不会引发#PF 异常,而是以中止页面虚拟值静默替换.在不触发故障的情况下,没有瞬态执行的时间窗口来进行未授权的瞬态计算,因此无法在 SGX 上实现原始的 Meltdown 攻击.为了克服这一限制,Foreshadow 攻击者清除了映射飞地秘密的页表条目中的“当前”位,以便随后对未映射的飞地页面进行未授权访问时引发页错误.

类似于最初的 Meltdown 攻击,攻击者继续执行瞬态指令序列以编码和恢复秘密数据(例如,通过 Flush+Reload 隐蔽信道).

Intel 将该攻击命名为 L1 终端故障(L1TF),即当访问“当前”位被清除或“保留”位的页表条目时会发生终端故障.然而,由于在现代微架构中 L1 高速缓存和地址转换连接紧密,发生终端故障的页表条目的物理地址位(即帧号)仍然可以传递到 L1 高速缓存.因此无论访问权限如何,对命中 L1 高速缓存的物理地址的任何访问都将传递给瞬态执行.最初的 Foreshadow[26]攻击还展示了如何通过滥用安全页交换来将任意飞地的页预取到 L1 缓存中来恢复未缓存的飞地秘密.

Foreshadow-NG[29]从对 SGX 飞地攻击到绕过管理程序隔离的操作系统等不同攻击场景概括了 Foreshadow.该攻击的观察基础是页表条目中的物理帧号有时受到对手直接或间接的控制.例如,当将页交换到磁盘时,内核可以自由地使用除“当前”位之外的所有位来存储元数据(例如交换分区上的偏移).但是如果此偏移量是有效的物理地址,则该位置上任何被缓存的内存都会泄漏给无特权的 Foreshadow-OS 攻击者.

更糟糕的是 Foreshadow-VMM 变体,它允许不可信的虚拟机提取主机的整个 L1 数据缓存(包括属于管理程序或其他虚拟机的数据).问题的根本是客户页表中的终端故障早于地址转换过程,因此客户物理地址被错误地传递到 L1 数据缓存,而不是首先被转换为适当的主机物理地址.

3.1.2 异常读取

- Spectre-NG LazyFP

在上下文切换时,操作系统必须保存所有寄存器,包括浮点单元(FPU)和 SIMD 寄存器.这些寄存器很大,保存它们会减慢上下文切换的速度.因此,处理器采取延迟状态切换机制,即不保存寄存器,而是简单地将 FPU 标记为“不可用”.在 FPU 被标记为“不可用”之后发出的第 1 条 FPU 指令会引发设备不可用异常(#NM),以便让操作系统保存先前执行上下文的 FPU 状态,并将 FPU 再次标记为可用.

Stecklina 等人[30]提出针对上述延迟状态切换机制的攻击,攻击步骤如下:首先,等待受害者将数据加载到 FPU 寄存器;然后,处理器将上下文切换到攻击者,并将 FPU 标记为“不可用”.此时,攻击者发出使用 FPU 的指令,该指令会引发设备不可用的错误.然而在故障指令退出之前,处理器已经使用来自受害者上下文的数据瞬态执行后续指令.这样,类似于先前的 Meltdown 攻击,故障指令之后的恶意瞬态指令序列可以通过隐蔽信道(例如 Flush+Reload)对未授权的 FPU 寄存器内容进行编码.

- Spectre-NG Variant 3a

Variant 3a 允许攻击者读取特权系统寄存器,它首先由 ARM 发现并公布,随后,Intel[31]确定其处理器也容易受到该攻击.未经授权访问特权系统寄存器(例如通过 rdmsr)会引发一般性保护错误(#GP).类似于先前的 Meltdown 攻击,该攻击利用了故障指令之后的瞬态执行仍然可以计算未授权的数据,并且通过微架构隐蔽信道泄漏系统寄存器的内容.

- Meltdown-PK

Intel Skylake-SP 服务器 CPU 支持对用户空间的内存保护密钥(PKU).此功能允许进程直接从用户空间更改页面的访问权限,即不需要系统调用/超级调用.因此,使用 PKU,用户空间的应用程序可以实现对可信部分的有效硬件增强隔离[32,33].Meltdown-PK 攻击[34]可以绕过通过内存保护密钥实现的读写隔离.如果攻击者在包含的进程中执行代码,即使攻击者无法执行 wrpkru 指令(wrpkru 是用于修改保护密钥的指令),也可以绕过 PKU 隔离.此外,与跨特权的 Meltdown 攻击变体相比,该攻击没有软件解决方法.Intel 只能在新硬件或者可能通过微码更新来修复 Meltdown-PK.

### 3.2 基于推测机制的瞬态执行攻击

基于推测机制的瞬态执行攻击利用错误推测事件来触发瞬态执行,基本流程如图 4 所示:没有推测机制时,安全检查的结果出来,才能判断后续指令是否执行.因此,越权指令无法访问秘密数据.而有推测机制时,由于安全检查耗时较长,处理器会根据相应推测机制的预测结果提前执行后续指令,直到安全检查结果出来为止.这段时间就是瞬时执行窗口.在该时间窗口内执行的瞬态指令可能对预测分支上的未授权结果进行计算,进而通过

微架构隐蔽信道泄漏.本节对该类型攻击实例进行梳理.

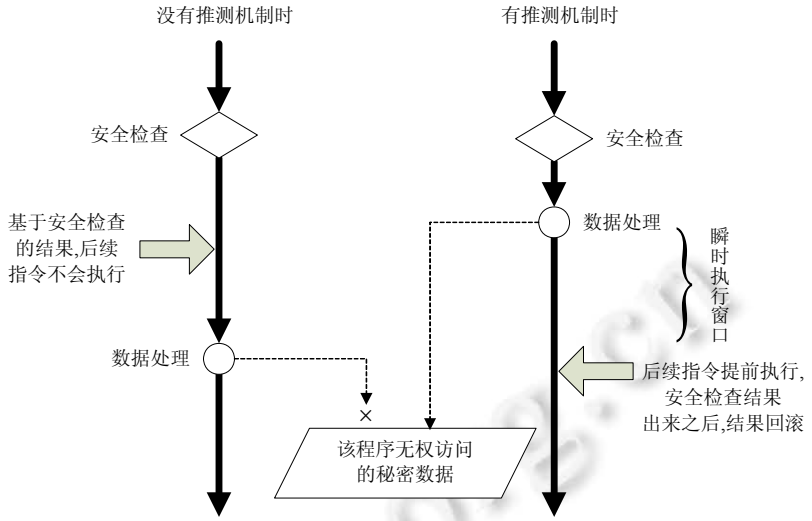


Fig.4 Flow of transient execution attack based on speculative mechanism

图 4 基于推测机制的瞬态执行攻击流程

现代处理器对于不同类型的分支具有许多不同的机制.Intel[6]为所有这些提供了推测机制,分布在不同的处理器组件中,例如分支历史缓冲区(BHB)[1]、分支目标缓冲区(BTB)[35,36]、模式历史表(PHT)[37]以及返回堆栈缓冲区(RSB)[37].相应的微架构元素在物理核之间不共享[38],但有些在逻辑核之间共享[39].针对不同的处理器组件,有不同的攻击利用方式,见表 2.

**Table 2** Components utilized by transient execution attack based on speculative mechanism

**表 2** 基于推测机制的瞬态执行攻击利用的组件

| 攻击名 | 攻击描述 | 利用的处理器组件 |
|---|---|---|
| Spectre v1<br>Spectre v1.1<br>Spectre v1.2 | 边界检查绕过 | 模式历史表 |
| BranchScope | 定向分支推测 | |
| Spectre v2<br>Spoiler<br>Spectre-NG V4 | 分支目标注入<br>推测加载<br>推测存储缓冲区绕过 | 分支目标缓冲区<br>Intel 内存子系统<br>内存消歧预测器 |
| ret2spec<br>Spectre-RSB | 函数返回的推测 | 返回堆栈缓冲区 |

此外,还有 NetSpectre[40]、SGXSpectre 和 SGXPectre[41]等是在特定的攻击场景中利用这些变体.

3.2.1 分支推测

- Spectre v1

Kocher 等人[3]首次提出了 Spectre v1.该攻击利用条件分支错误推测来瞬态加载秘密数据.

在 Intel 处理器中,主要利用的微架构元素是模式历史表(PHT).根据预测器模式,将分支指令虚拟地址中的几位和移位寄存器[37,42]进行组合来访问,其中,移位寄存器包含关于最后 $N$ 个分支的信息,$N$ 取决于体系结构[37].PHT 包含偏置位,该偏置位指示是否大概率采纳一个分支.以如下代码为示例:

$$f(x<len(array1))\ \{y=array2[array1[x]*4096];\}.$$

可以“错误地训练”,提供 $x$ 的有效值,以便分支的取值为 True.随后,攻击者可以提供一个超出范围的 $x$ 值.在程序执行过程中,PHT 推测分支取值为真,并沿着错误推测的路径进行瞬态执行.紧接着,使用隐蔽信道传输

越界读取到的值.SGXSpectre 是对 SGX 飞地发动这个攻击.

- Spectre v1.1 和 Spectre v1.2

Kiriansky 和 Waldspurger[43]表明:按照相同的原理,瞬态写入也是有可能的,即瞬态执行不遵循“读/写”页表属性.在当前权限级别内,瞬态重写只读数据的能力可以绕过依赖于只读存储器的硬件执行沙箱.攻击者在瞬态执行期间破坏了内存安全性.从示例代码“*if*(*x*<*len*(*array*)) {*array*[*x*]=*value*;}”可以看到,在攻击者控制输入 *x* 有效的情况下,某些值会在偏移 *x* 处写入数组.遵循相同的方法,攻击者会瞬态越界写.这会产生瞬态缓冲区溢出,允许攻击者从受害者域已存在的代码中来执行任意指令序列,类似于 return-to-libc 和面向返回编程(ROP)[44]攻击.

- BranchScope

BranchScope[42]攻击通过操纵定向分支预测器来推断受害程序中的任意条件分支指令的方向.通过强制分支预测器选择本地 1-级预测器来确保能可靠地创建冲突.攻击者通过执行一对分支(该分支有预先定义好的输出),测量这些分支的推测时延,并将这个信息和预测器的状态关联起来,进而获得受害程序分支的方向.具体过程为:首先,攻击者通过对分支指令执行一个精心设计的随机块,将目标 PHT 项初始化为指定状态;然后,攻击者发起一个分支的执行,意图在受害进程内部监控该分支的执行,并且等待 PHT 状态被受害程序改变;最后,攻击者执行以同一个 PHT 项为目标的多个分支指令,通过计时来观察它们的推测结果.攻击者通过将推测结果和 PHT 的状态联系起来以识别受害者分支的方向.

- Spectre v2

在 Spectre v2[3]中,攻击者污染分支目标预测器,可以通过回退机制[3]在间接分支预测器或通用分支目标缓冲区(BTB)中进行.间接分支预测器仅使用源指令地址的最低 12 位与分支历史缓冲区(BHB)组合起来确定跳转目标.BTB 包含所有最近跳转目标的位置信息,并且仅使用进一步折叠在一起[3]的最低 31 位[36].由于目标预测器结构中的冲突,攻击者可以利用被污染推测的瞬态执行来读取另一个上下文(例如另一个进程)中的任意内存.在 Spectre v1 中,攻击者通过错误训练有效值来绕过边界检查;与其相反,分支目标注入攻击训练分支预测器,使得推测跳转到恶意目的地址.然后,攻击者可以选择执行瞬态指令的任意代码位置并泄露秘密信息.这个任意代码位置称为 Spectre 小工具,负责将秘密数据传送到攻击者控制的隐蔽信道.

为了错误训练目标预测器,攻击者必须知道受害者地址空间中 Spectre 小工具的虚拟地址.知道了这个地址,攻击者在自己的地址空间内训练 BTB,使其推测到该地址(或者等价的地址),最终使得受害者推测跳转到这个小工具上执行.由于 BTB 是在逻辑核之间共享的,因此攻击者必须使这两个进程位于同一物理核上.

分支目标注入攻击的基本思想是执行攻击者选择的小工具(参见 ROP[44]).与 ROP 的区别在于:攻击者不依赖于受害者代码中的漏洞,而是对 BTB 进行错误训练来瞬态执行生成的代码路径.在 SGXPectre 中,Chen 等人[41]表明:通过使用分支目标注入,攻击者可以从 SGX 飞地中提取关键信息.

- NetSpectre

Schwarz 等人[40]提出了一种新方法.它不要求攻击者在受害者机器上执行自己的代码.NetSpectre 攻击将两个不同的小工具(即泄漏和传输小工具)合并到 NetSpectre 中.泄漏小工具负责访问由攻击者控制的索引处的一串比特流,并据此来更新微架构状态.然后,传输小工具执行一些操作,通过运行时的改变(例如,由于高速缓存命中而导致的响应时间较短)来暴露微架构状态.与其他基于分支预测的瞬态执行攻击一样,攻击者需要不断地错误训练处理器.该攻击会向目标发送多个网络数据包,包中含有总是处于条件语句边界内的值,将分支预测器训练到推测该条件判断语句为真.由于不是直接测量内存访问时间,而是测量使用相应内存位置的网络请求响应时间,因此响应时间将受到网络延迟的影响,可通过对大量网络包进行平均以降低噪声.因此,攻击者可以在测量值上使用分类器,或者首先测量两个极端情况(缓存和未缓存)以获得实际测量的阈值.

因为测量破坏缓存状态,即在第 1 次测量之后总是缓存变量,攻击者需要一种从缓存中驱逐(或刷新)变量的方法.由于受害者不太可能提供直接刷新或驱逐变量的接口,因此攻击者不能使用 Flush+Reload 缓存攻击,而采用了原始的 Thrash+Reload 方法.通过破坏缓存来驱逐整个最后一级缓存,而不是像 Evict+Reload 那样有针对性的驱逐.为了在没有代码执行的情况下破坏整个缓存,不得不使用网络可访问的接口.最简单的形式是向受害者

请求文件的方式,例如文件下载,就有机会从缓存中驱逐变量.

3.2.2 推测性存储到加载转发

- Spoiler

现代微架构包含优化技术,如推测性存储到加载转发,以改善内存瓶颈.处理器使用内存顺序缓冲区(MOB)管理内存操作.MOB 与数据缓存紧密耦合.MOB 通过遵循 Intel 内存排序规则[45]确保内存操作有效执行,其中,内存存储按顺序执行,内存加载可以无序执行.存储将被解码为两个微操作,以分别将地址和数据存储到存储缓冲区.存储缓冲区使处理器能够在存储提交之前继续执行其他指令.因此,流水线不必等存储完成.这进一步使 MOB 能够支持加载的无序执行.存储转发是一种优化机制,如果加载地址与存储缓冲区条目匹配,则会将存储数据发送到加载.这是一个推测过程,因为 MOB 无法基于存储缓冲区真正确定加载对存储的依赖性.Intel 对存储缓冲区的实现没有记录,但潜在的设计表明,它只会保存虚拟地址,并且可能包含部分物理地址[46−48].因此,虽然物理地址不匹配,但处理器可能错误地转发数据.完整的解决方案将被延迟到加载提交阶段,因为 MOB 需要向 TLB 询问完整的物理地址信息,这是很耗时的.

Spoiler[49]利用服务于推测性加载的依赖性解析逻辑来获取有关物理页面映射信息,诸如 Rowhammer 和缓存攻击之类的微架构侧信道攻击都需要对虚拟和物理地址之间的映射进行逆向分析.借助于 Spoiler 泄露的信息可将这种逆向分析加速 256 倍,对 Prime+Probe 攻击中的驱逐集搜索可提高 4 096 倍,几乎可以 100%确定 DRAM 行冲突,以普通用户权限实现双面 Rowhammer 攻击.

- Spectre-NG V4

现代处理器通过使用存储缓冲区而不是将更新的值直接写回内存来提高性能,这使得处理器可以继续执行指令.存储缓冲区中的所有数据最终都会写回主内存.带来的一个复杂因素是:内存加载会读取不新鲜的内存,因为存储缓冲区尚未将更新的值写回主内存.为了避免这种情况,处理器会检查存储缓冲区执行的每次加载.由于处理器允许未对齐访问,即当地址可能不精确匹配时允许重叠,因此对存储缓冲区搜索是复杂且耗时的.该过程称为“内存消歧”.为了进一步提高性能,处理器采用了称为“推测存储缓冲区绕过”的优化.如果通过内存消歧推测:待加载的地址与存储缓冲区中的任何地址都不冲突,才能在推测执行中绕过存储缓冲区.Horn 利用推测存储缓冲区绕过的优化技术,结合 Flush+Reload 隐蔽信道,泄漏内存位置上较早的值(例如未经过处理的值).攻击者只能泄漏相同权限级别的内存信息.

3.2.3 地址推测

Maisuradze [39]及 Koruyeh 等人[50]提出了一种利用返回堆栈缓冲区(RSB)的新变种.RSB 是硬件堆栈,通常具有 16 个条目,用于跟踪先前调用指令的返回地址.遇到 ret 指令时,RSB 的顶部被用于推测返回地址.在超线程系统上,RSB 专用于逻辑核.如果 RSB 为空,Skylake 处理器使用分支目标缓冲区(BTB)作为后备[37]. RSB 推测错误就是 RSB 提供的返回地址预测值和软件栈中真正的返回地址不一致的情况.Koruyeh 等人[50]给出了发生错误推测的 4 种场景.

- 第 1 种场景是由于 RSB 结构大小限制导致 RSB 溢出或未充满而发生错误推测.RSB 中能够保存下来用于推测的地址数量是有限的,一般来说,低端设备可以存储 4 个地址,桌面设备有 16 个,服务器则更大(例如 24 个).由于太多返回地址被压入到小 RSB 上,会发生溢出从而覆盖旧条目.之后,ret 指令会依次弹出对应地址,但是最开始的那些 ret 指令对应的返回地址被覆盖,这时 RSB 未充满,无法提供返回地址进行预测.不同的 CPU,处理未充满的方式也不同.例如,Intel Skylake 和更新的处理器使用 BTB 作为备用,从而可以利用 BTB 进行攻击.
- 在场景 2 中,攻击者可以通过覆盖软件栈的返回地址或删除它来直接污染 RSB.一方面,可以通过直接修改 call 指令为 push 和 jmp 指令,这时没有 call 隐式地将地址压入 RSB,但在软件栈中有返回地址,从而造成两者不一致,在返回时产生错误预测;另一方面,也可以修改返回指令为 pop 和 jmp,那么 call 压入软件栈的地址被取出,但是压入 RSB 的没有,导致下一次返回时预测错误.
- 场景 3 是推测性地污染 RSB.在推测执行路径(瞬态指令)中,执行 call,会将地址压入栈中.当发现预测错

误时,架构上会放弃这次修改,软件栈上是没有相应地址,但对 RSB 产生的影响没有消去.这样就使得RSB 和软件栈产生不一致,导致返回时预测错误.

- 最后一个场景是基于共享RSB导致错误推测.当上下文切换时,RSB是保留的.因此,从一个线程切换到另一个线程,在新线程中会使用旧线程填充的 RSB 内容,这时调用 ret,会使用旧线程提供的地址来预测,与新线程的软件栈中地址不一致,产生错误预测.同样的情况也适用于切换到内核线程或者 SGX 上下文.

后 3 种预测错误的场景都是使软件栈中真正的返回地址和 RSB 预测提供的返回地址不一致,从而产生错误预测,让攻击者可以利用.而第一种是使 RSB 无法进行预测,转而使用其他预测方法,进而使用其相应的攻击方法.

## 4 防御方法

根据第 2 节瞬态执行攻击模型所描述的关键步骤及组件,对瞬态执行攻击的防御方法主要围绕以下 4 个方面进行.

a) 限制瞬态指令的执行或减小瞬态执行的时间窗口;
b) 限制瞬态指令越权访问数据;
c) 使微架构状态不受瞬态执行的影响,使隐蔽信道的发送端无效;
d) 降低隐蔽信道的精度,相当于降低隐蔽信道接收端的能力.

触发瞬态执行的原语分别利用了CPU微架构的不同属性,因此,前两方面的防御方法侧重点各有不同.但是瞬态执行攻击都使用了隐蔽信道(主要是缓存隐蔽信道)将秘密数据转移到架构域,因此,最后两个方面的防御方法对使用相同隐蔽信道的瞬态执行攻击来说是通用的.

### 4.1 限制瞬态指令的执行

针对乱序执行,一般是在发生异常时,由于异常延迟处理会给攻击者留下瞬态执行的可能.直接的缓解方法是在可控范围内避免出现故障.为了防御延迟状态切换机制攻击,Linux kernel 4.6 以后的系统通过使用紧急切换代替延迟状态切换[30],使 FPU 始终可用,这样对 FPU 的访问就不会发生故障.

针对推测执行,一般是有推测错误时,会导致瞬态指令执行.因此,Intel 和 AMD 建议在分支结果上使用像lfence 这样的序列化指令[31,51].ARM 引入了完整的数据同步屏障(DSB SY)和指令同步屏障(ISB),可用于防止推测[52].但是,序列化每个分支将相当于完全禁用分支推测,严重降低性能[31].因此,Intel 进一步提出使用静态分析来最小化引入序列化指令的数量.Microsoft 使用 C Compiler MSVC 的静态分析器来检测已知错误的代码模式,并自动插入 lfence 指令.新加坡国立大学提出了 oo7,对二进制程序进行静态分析的方法[53],使用污点分析、地址分析和对推测执行进行建模来检查潜在的易受攻击的代码模式.并通过插入少量的 fence 指令来缓解推测执行攻击.该方法可以准确地识别 15 种攻击代码类型,而微软编译器只能识别 2 种.同时,性能开销较低,可应用于500 多个二进制程序.开源安全公司采用类似的方法使用静态分析.Kocher 表明,这种方法使许多可以被利用的小工具失效.

序列化指令还可以减少间接分支污染的影响.通过在分支之前插入 lfence 指令,在发生错误推测的情况下,可减小瞬态执行的时间窗口.虽然 lfence 指令阻止了推测执行,但 Schwarz 等人[40]表明,他们不能阻止在执行前发生的推测性代码提取和其他微架构行为.这包括启动 AVX 功能单元、指令缓存填充和可能泄漏数据的 iTLB 填充.

Evtyushkin 等人[42]提出了一种类似于序列化指令的方法.他们建议软件开发人员可以指出可能泄露敏感信息的分支.如果需要,处理器不推测这些分支的结果.除串行化指令外,ARM 还引入了一个新的屏障(CSDB),它与条件选择或移动相结合,来控制推测执行[52].

推测性加载固化(SLH)是 LLVM 使用的一种方法,由 Carruth 提出.其主要思路是:使用无分支的代码来检查加载,确保它们沿着有效的控制流路径执行.为此,他们在编译器级别转换代码并在条件上引入数据依赖.在推

测错误的情况下,指针被清零,防止它通过推测执行泄漏数据.该方法需要硬件支持,以实现无分支,和对寄存器的值进行不可推测的条件更新.截至目前,该功能仅适用于 x86 的 LLVM,因为 ARM 的补丁仍在审核中.GCC 采用了 SLH 的实现,支持 x86 和 ARM.它们提供内置函数,以便在确定指令是瞬态时发出推测障碍或返回安全值.

Oleksenko 等人[54]提出了一种与 Carruth 类似的方法,叫做“绕不过”.该方法利用处理器具有的检测指令之间数据依赖的机制,并对比较参数引入这种依赖.这确保只有在寄存器或在 L1 高速缓存中比较时才开始加载,从而将时间窗口减少到不可利用的大小.

Retpoline 是 Google 提出的方法,用返回指令替换间接分支的代码序列来防止分支污染.该方法确保返回指令通过 RSB 推测到一个良性无限循环以捕获推测.实际目标地址被推入栈并使用 ret 指令返回.而 Intel 公司[55]指出:在未来具有控制流增强技术[56](CET)来防御 ROP 攻击的处理器中,retpoline 可能会在 CET 的防御中触发误报.为了缓解这种可能,未来的处理器还为防御分支目标注入攻击实现了增强型 IBRS[55]的硬件支持,同时消除了对 retpoline 的需求.

为了防止 Spectre v2 攻击,可通过 CR4 的第 20 位开启 SMEP.操作系统(OS)将不能直接执行应用程序代码,甚至是推测性的.攻击者只能在 OS 源码中找小工具,使得对 OS 实施 Spectre v2 攻击变得更加困难.应用程序更难以训练 OS 代码跳转到 OS 小工具.默认情况下,所有主流操作系统都启用 SMEP.同样,在 RISC-V 架构中,RISC-V 指令集在 SSTATUS 控制寄存器中定义了 SUM 标志位.一般情况下,*SUM*=0,即内核程序不能读写用户态数据,只有必要时才临时开启 SUM 直接操纵用户数据.但是,无论如何都不能执行用户页的指令.

为了防止处理器对存储缓冲区检查做推测,Intel 提供了一个微码更新,以禁用“推测存储旁路禁用(SSBD)”机制.AMD 也支持 SSBD[57].对于即将推出的 CPU,ARM 推出了推测存储绕过安全(SSBS)的配置控制寄存器,用于防止加载和存储的重排[52].在 Skylake 和更新的架构上,Intel[55]提出了 RSB 填充以防止 RSB 未填满以及随后的 BTB 回退.因此,在每个上下文切换到内核时,RSB 都填充了良性小工具的地址.这种行为类似于 retpoline.对于 Broadwell 和较旧的架构,Intel[55]提供了微码更新,使 ret 指令可推测,使得 retpoline 成为防御分支目标注入攻击的强有力方法.

Spectre 某些变体的构建块之一是分支污染,攻击者会误导推测机制.为了应对错误训练,Intel 和 AMD 都扩展了指令集架构(ISA),采用了一种控制间接分支的机制[58].对 ISA 的扩展包括 3 个控件.

a) 间接分支受限推测(IBRS):可防止在特权代码中执行的间接分支受到较低权限代码中分支的影响.为了强制执行此操作,处理器进入 IBRS 模式,该模式不受其外任何操作的影响.

b) 单线程间接分支推测(STIBP):限制了跨超线程执行的代码之间的分支推测机制的共享.

c) 间接分支预测器屏障(IBPB):通过刷新 BTB 来防止在它之前执行的代码影响其后面的代码推测.

对于现有的 ARM 实现,没有一般的缓解技术可用.但是某些处理器实现了特定的控制,这些控制使得在上下文切换期间应该使用的分支预测器无效[52].在 Linux 上,这些机制默认启用.在 ARMv8.5-A 指令集[5]中,ARM 引入了一个新的屏障(sb)来限制后续指令的推测性执行.此外,新的系统寄存器可以限制推测性执行,还有新的推测控制指令阻止控制流推测(cfp)、数据值推测(dvp)或高速缓存预取推测(cpp).

为了防止 Spectre v1 攻击,Kiriansky 和 Waldspurger[43]提出了 SLoth,包含 3 个微架构防御,以限制“存储到加载的转发”:第 1 个是 SLoth Bear,通过微码更新,防止从瞬态存储或者到瞬态加载的“存储到加载转发”;第 2 个是 Sloth,依赖于编译器标记指令作为转发的候选者;第 3 个是 Arctic SLoth,它对加载和存储对进行动态检测来确定转发的候选者.

### 4.2 限制瞬态指令越权访问数据

对于 Meltdown,瞬态指令使用了应用程序没有权限访问的秘密数据.在典型的控制流中,应用程序永远无法访问该数据.Meltdown 利用了发生故障后的瞬态指令可以访问故障指令数据的特性.

相反地,对于推测执行,瞬态指令仅适用于应用程序可以在架构上访问的数据.瞬态执行的指令也可以在应用程序的正常控制流中执行,例如当分支条件不同时.虽然分支目标注入攻击可以跳转到当前地址空间中的任意位置,但它仍然适用于应用程序可以在架构上访问的数据.

4.2.1 限制瞬态指令越权访问微架构数据

基于乱序执行的瞬态执行攻击的根本问题是:处理器允许瞬态指令对在架构上无法访问的值进行计算,从而泄漏它们.为确保在发生故障时不继续执行原本不可访问的值,未来的硅硬件设计可减轻这种攻击.但是,必须通过微码更新或操作系统级解决方案来缓解现有微架构问题.所有这些方法的目标都是确保在架构上无法访问的数据在微架构层面也无法访问.

Gruss 等人最初提出 KAISER[4,22]来减轻打破 KASLR 的旁路攻击,但是它也可以通过防止内核数据映射到用户空间来抵御 Meltdown 攻击.表 3 对 5 个主要性能影响因素作了评估.除了性能影响外,KAISER 还有一个实际限制[2,4],即由于 x86 的设计,一些特权内存位置必须始终保留映射在用户空间中.Linux 内核已经实现 KAISER,命名为 KPTI.Microsoft 提供了类似 Windows 10 Build 17035 的补丁,Mac OS X 和 iOS 也有类似的功能.

**Table 3** Performance impact analysis of KPTI

**表 3** 对 KPTI 的性能影响分析

| 影响因素 | 评估结果 |
|---|---|
| 系统调用频率 | 在每秒(每个 CPU 核)50K 次系统调用的情形下,开销大约为 2%,并且随系统调用频率的升高而增大 |
| 上下文切换 | 增加的开销与系统调用频率类似 |
| 缺页率 | 当缺页率较高时,开销略微增加 |
| 工作集大小 | 由于 TLB 刷新,大于 10MB 的工作集将花费额外的开销,从而导致性能开销从 1%(仅有系统调用开销)升至 7% |
| 缓存访问模式 | 特定的访存模式将严重影响缓存命中状态.最坏情况下,额外增加 10%的开销 |

对 Foreshadow 和 Meltdown 利用的访问控制竞争条件进行直接解决,可能不适用于微代码更新[26].因此,Intel 提出了一种多阶段的方法来减轻 Foreshadow(L1TF)攻击对当前处理器的影响[29].首先,为了保持传统的进程隔离,操作系统内核应该注意清理未映射页表条目的物理地址字段.内核可以清除物理地址字段,也可以让它指向不存在的物理内存.在前者的情况下,Intel 声明 4KB 的虚拟数据应放在物理地址 0x0 处;另外,应在页目录和页目录指针级别清除 PS 位,以防止攻击者利用巨大的 2MB 或 1GB 页.

SGX 飞地或管理程序对不可信操作系统执行的地址转换不相信,Intel 建议将秘密数据存储在不可缓存的内存中(如 PAT 或 MTRR 中所指定),或在切换保护域时刷新 L1 数据高速缓存.随着最近微码更新,L1 在安全区退出时自动刷新,并且管理程序可以在将控制权移交给不可信虚拟机之前额外刷新 L1.在退出系统管理模式(SMM)时也会刷新缓存,以减轻对 Foreshadow-NG 攻击对 SMM 的影响.

为了缓解跨逻辑核的攻击,Intel 提供了一个微码更新,以确保在启用或禁用超线程时派生出不同的 SGX 证明密钥.为了确保当属于 SMM 的数据位于 L1 数据高速缓存中时不运行非 SMM 软件,SMM 软件必须在进入和退出时对所有逻辑核进行会合.为了在启用超线程时防止 Foreshadow-NG 攻击,管理程序必须确保在运行有不可信 VM 的兄弟核上不运行管理程序线程.

对于 Spectre-NG Variant 3a,攻击者泄漏系统寄存器的内容,这些系统寄存器在架构上是特权级别,普通用户无法访问,因此 Intel 发布了微代码更新[59].虽然 AMD 不易受到影响[60],但 ARM 在未来的 CPU 设计中采用了缓解措施,并建议在未得到缓解的 CPU 的上下文切换中用虚拟值替换寄存器值[52].

4.2.2 限制瞬态指令越权访问数据

不同的项目使用不同的技术来缓解瞬态执行攻击.WebKit 采用了两种技术来限制对秘密数据的访问:WebKit 首先用索引掩码代替数组边界检查,通过采用位掩码,WebKit 无法确保访问始终处于边界内,但是限定了能超出多大范围进行访问;在第 2 种策略中,WebKit 使用伪随机污染值来保护指针不被滥用.攻击者在使用前必须先学习污染值.更重要的影响是,对用于类型检查的分支的错误推测,会导致错误的类型被用于指针.

Google 为 Chrome 提出了另一种防御措施,称为站点隔离,把不同的网站隔离在不同的进程中,并且阻止一个进程获得其他网站的敏感信息.即使攻击者可以读取任意内存,它也只能从自己的进程中读取数据.站点隔离主要由两部分组成:进程模型的修改和跨域读取屏蔽.在 Chrome 67 桌面版上,Google 默认启用了站点隔离功能.

Kiriansky 和 Waldspurger[43]建议通过使用 Intel 内存保护密钥(MPK)[28]等技术保护密钥来限制对敏感数据的访问,但是攻击者利用 Spectre v1 可以在读取数据之前禁用保护.为了防止这种情况,可在 wrpkru 中包含

lfence 指令.

### 4.3 使微架构状态不受瞬态执行的影响

- 缓存隔离

瞬态执行攻击的根本是瞬态指令序列使得微架构状态发生改变,可以由隐蔽信道的接收端观察到.为了使微架构状态不受瞬态执行的影响,SafeSpec[61]对在执行瞬态指令期间使用的硬件结构作了影子.因此,如果处理器的推测不正确,则可以恢复微架构状态的任何改变.虽然它们的原型实现仅保护缓存,可扩展来保护 TLB,但其他信道仍待解决.Yan 等人[62]提出了 InvisiSpec,一种旨在使瞬态加载在缓存层次结构中不可见的方法.通过使用推测缓冲区,所有瞬态执行的加载都存储在此缓冲区中而不是缓存.与 SafeSpec 类似,如果推测不正确,缓冲区将失效.但是如果推测正确,则将缓冲区的内容加载到实际缓存中.为了保持内存一致性,InvisiSpec 需要将在此过程中加载的值与缓存中最近最新的值进行比较.如果发生不匹配,则瞬态加载和所有后续指令都会被恢复.由于 SafeSpec 和 InvisiSpec 只保护 CPU 的缓存层次结构,因此攻击者仍可利用其他隐蔽信道.

- 缓存分区

Kiriansky 等人[63]提出了一种对缓存分区的方法——DAWG.通过引入隔离缓存命中、缓存未命中和元数据级别的保护域,可以缓解基于缓存的隐蔽信道.这不仅需要更改缓存和适应一致性协议,还要在软件中正确地管理这些域.InvisiSpec,SafeSpec 和 DAWG 在解决问题的方式上类似,但都只考虑基于缓存的隐蔽信道,攻击者可以轻松地替换成别的隐蔽信道泄漏数据.

Percival 建议在线程之间分割 L1 缓存以消除缓存争用.虽然包括 Page[64]在内的一些人提出了硬件设计,但没有商用硬件提供此选项.Wang 和 Lee[65]提出了分区锁定缓存(PLcache),旨在为每个缓存行分配锁定属性,允许敏感数据(如 AES 表)有选择地临时锁定到缓存中.ARM 系列处理器通过 L2 缓存控制器支持缓存锁定,例如 ARM Coretex A9 平台上的 L2C-310 缓存控制器[66].Domnister 等人[67]建议:在 L1 高速缓存的每个高速缓存集中,为每个超线程保留若干高速缓存行.

Intel 的高速缓存分配技术(CAT)提供了类似技术的实现,可以防止进程替换 LLC 中的某些内容.为了防御 Prime+Probe 和 Evict+Time 等技术实施的 LLC 缓存攻击,CATalyst[68]将 LLC 划分为硬件和软件混合管理的缓存.它使用 CAT 在 LLC 中创建两个分区:一个安全区和一个非安全区.安全区仅存储缓存固定的安全页,而非安全区则作为由硬件替换算法管理的普通缓存区.为了存储安全数据,用户程序申请分配安全页,并将数据预加载到安全缓存分区中.由于安全数据被锁定在 LLC 中,因此缓解了由高速缓存行冲突引起的 LLC 定时攻击.Liu 等人[68]证明:CATalyst 在 GnuPG 1.4.13 中有效地保护了平方和乘算法,而且对系统性能的影响很小,SPEC 平均放缓 0.7%,PARSEC 平均放缓 0.5%.

SHARP[69]改变共享缓存的替换策略,以防止攻击者通过缓存驱逐来了解受害者的内存访问模式.它优先考虑驱逐不在任何私有 L1 缓存中的 LLC 缓存行和当前进程的 LLC 缓存行.然而,不可信操作系统仍然可以驱逐受害者的缓存行,因为它可以代表受害者运行.随机填充缓存架构[70]打破了内存访问和 L1 缓存填充之间的相关性,以防止基于重用的侧信道攻击.Wang 和 Lee[71]提出了对 L1 高速缓存中的内存到缓存的映射进行动态随机化的方案.这两种方法都集中在 L1 缓存上,并且可能会在更大的 LLC 上产生高性能开销.

- 禁止页共享

缓存隐蔽信道依赖于共享内存页,VMware 公司建议禁用透明页面共享功能以防止跨虚拟机的 Flush+Reload 攻击.CacheBar[72]通过自动检测此类访问并创建多个副本来阻止对共享页的并发访问.该方案称为访问备份,即当另一个安全域试图访问页时,创建该页的副本.

- 污点跟踪

Kocher 等人[3]建议通过对变量的污点跟踪来限制数据进入隐蔽信道.具体方法是:处理器跟踪在瞬态执行期间加载的数据,并防止在后续可能泄漏数据的操作中使用.从理论上来说,污点跟踪[3]减轻了所有形式的 Spectre 类型攻击,因为已经被污染的数据不能用于瞬态执行.因此,数据不会进入隐蔽信道,进而不会泄漏.

### 4.4 降低隐蔽信道的精度

瞬态执行攻击使用隐蔽信道来传递由瞬态指令序列引起的微架构状态变化,以便在架构级别上观察它.减轻瞬态执行攻击的一种方法是降低所述隐蔽信道的精度或阻止它.

- 降低定时器的精度

许多隐蔽信道需要精确的定时器以便区分微架构状态,例如测量内存访问时间以区分高速缓存命中和高速缓存未命中.如果定时器的准确性降低使得攻击者不能区分微架构状态,隐蔽信道的接收端则不能恢复出发送端发送的信息.为了减轻基于浏览器的攻击,许多 Web 浏览器通过添加抖动以降低 JavaScript 中定时器的准确性,甚至移除部分计时器.但是 Schwarz 等人[73]证明:可以用许多不同的方式构建定时器,移除所有的计时器是不切实际的,因此需要更全面的缓解措施[74].此外,它只是使攻击者更难获取信息,但可以通过进行多次测量来规避这一点.虽然 Chrome 最初禁用了 SharedArrayBuffers 来缓解 Meltdown 和 Spectre,但是在引入站点隔离之后,这个计时器源已经重新启用.

为了减轻 NetSpectre 这种远程攻击,Schwarz 等人[40]建议使用 DDoS 检测机制,因为攻击者必须向受害者发送数千个相同的数据包.第 2 种方法是将人为噪声添加到网络延迟中,增加攻击者提取单个数据位所需执行的测量次数.在某些时候,攻击在实践中变得不可行.

### 4.5 小 结

针对瞬态执行攻击的软件防御措施是通过修改程序源码以阻止受攻击的机制起作用.例如,Retpoline,IBPB 和改进的 lfence 指令等着重于防止攻击者操纵受害者代码的执行.这些防御措施并不能立即适用于现有的二进制文件.有些缓解方案还涉及修改 OS、系统管理程序和 SMM 代码等.另一方面,硬件防御有可能避免修改现有软件,例如,InvisiSpec,SafeSpec 和 Dawg 等技术防止执行错误路径以避免将秘密信息残留在缓存中进而被恢复.尽管这些技术是有效的,但是仅关闭缓存隐蔽信道不足以阻止瞬态执行攻击,因为缓存只是许多潜在隐蔽信道之一.

## 5 总结与展望

支撑软件安全技术的基本假设是处理器忠实地执行程序指令,包括安全检查.微架构瞬态执行攻击(Meltdown、Spectre 以及它们的变种)主要利用 CPU 微架构优化技术带来的安全漏洞,通过使用精心设计的宏观指令触发瞬态执行可以绕过控制流或安全检查.微架构瞬态执行攻击技术和防御方法是计算机系统安全领域中一个新的研究热点.针对这一问题,本文首先讨论了处理器微架构采用的优化技术带来的信息泄露问题,这是深入理解瞬态执行攻击机理与方法的基础;通过对各种瞬态执行攻击的异同点进行总结抽象,给出了微架构瞬态执行攻击的基本模型并分析了关键组件;根据触发瞬态执行原语的不同,分别对基于异常和基于推测机制的瞬态执行攻击进行了描述对比;最后,基于攻击模型,分别从 4 个方面总结了现有防御方案的设计思路.以 KPTI 为典型防御方案做了全面的性能评估,并对软硬件防御方法的优点和不足做了比较.由于现代处理器微架构的细节不公开,这类攻击尚未被完全理解,新的攻击变种仍在层出不穷,存在许多亟待解决的问题.

(1) 从攻击的角度来看,瞬态执行的时间窗口,是攻击能否成功的关键.在计算机系统中,瞬态执行时间窗口受哪些因素影响,如何进行量化评估、如何有效地扩大时间窗口以及如何判定瞬态执行是否成功等,是后续研究的重要问题.

(2) 不同的瞬态执行攻击利用了 CPU 的不同属性.一个 CPU 可能容易受到 Spectre 的影响,但不会受到 Meltdown(例如 AMD)的攻击,反之亦然.虽然研究人员已经提出许多防御措施和软件补丁以减轻微架构瞬态执行攻击,但它们通常只解决攻击的一方面,仍给攻击者留下其他可能的攻击面.

(3) 现有的防御措施大多对瞬态指令的执行进行了相对简单粗暴的限制,通常会导致很高的性能开销:平均比例为 10%~30%,有些甚至更高.如何对瞬态指令进行特征刻画,以及对瞬态指令不同行为的判别,是这类问题后续研究的关键问题.例如,研究是否能够对瞬态指令越权进行有效的判定,研究如何对

瞬态指令的执行范围进行合理的限制等问题.

(4) 瞬态执行攻击对传统 CPU 和操作系统安全机制带来根本性冲击.通过调整 CPU 微架构状态,CPU 设计者可以优化硬件的性能,由此引入的安全问题相对处于未验证的状态.更广泛地说,在安全性和性能之间如何进行权衡.

(5) 从系统层面上来说,处理器、编译器、设备驱动程序、操作系统和许多其他关键组件已经形成了复杂优化的复合层,不可避免的共享资源的存在及共享资源受控使用机制的失效,这也是微架构瞬态执行漏洞存在的原因.底层软硬件是上层系统构建的基础,以底层措施为基础的安全机制设计能够更彻底、更大范围地发挥作用,不容易被绕过.同时,上层的安全机制是底层系统安全机制的有效补充,主要原因如下:第一,底层系统的使用逻辑与用户的实际使用逻辑相差较大,纯底层的安全机制很难与用户的实际安全需求直接对应;第二,底层系统资源受限,将安全机制全部实现在底层,对系统开销影响太大;第三,在系统中,上层软件功能一般被看作是底层工作的封装及扩展,与之相对应,CPU 层次的安全问题可能会被上层软件屏蔽或放大,纯粹硬件的安全机制很难达到预期的访问效果.只有采用软硬件协同的手段,才能在 CPU 漏洞存在的情况下,形成兼顾功能、性能、安全性的有效系统防护体系.

**References**:

[1] Bhattacharya S, Maurice C, Bhasin S, Mukhopadhyay D. Template attack on blinded scalar multiplication with asynchronous perf-ioctl calls. Report, 2017/968, Cryptology ePrint Archive, 2017. https://eprint.iacr.org/2017/968.pdf

[2] Lipp M, Schwarz M, Gruss D, Prescher T, Haas W, Fogh A, Horn J, Mangard S, Kocher P, Genkin D, Yarom Y, Hamburg M. Meltdown: Reading kernel memory from user space. In: Proc. of the 27th USENIX Security Symp. 2018. 973–990.

[3] Kocher P, Genkin D, Gruss D, Haas W, Hamburg M, Lipp M, Mangard S, Prescher T, Schwarz M, YaromY. Spectre attacks: Exploiting speculative execution. In: Proc. of the 2019 IEEE Symp. on Security and Privacy. 2019. 1–19.

[4] Gruss D, Lipp M, Schwarz M, Fellner R, Maurice C, Mangard S. KASLR is dead: Long live KASLR. In: Proc. of the Int'l Symp. on Engineering Secure Software and Systems. Cham: Springer-Verlag, 2017. 161–176. [doi: 10.1007/978-3-319-62105-0_11]

[5] ARM. ARM A64 instruction set architecture (Beta). 2018. https://static.docs.arm.com/ddi0596/a/DDI_0596_ARM_a64_instruction_set_architecture.pdf

[6] Intel. Intel 64 and IA-32 architectures optimization reference manual. 2016. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[7] Costan V, Devadas S. Intel SGX explained. IACR Cryptology ePrint Archive, 2016. 1–118. https://eprint.iacr.org/2016/086.pdf

[8] Intel. Intel 64 and IA-32 architectures software developer's manual—Combined volumes. 2016. https://www.intel.cn/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.pdf

[9] Lampson BW. A note on the confinement problem. Communications of the ACM, 1973,16(10):613–615.

[10] Kocher PC. Timing attacks on implementations of diffehellman, RSA, DSS, and other systems. In: Proc. of the Annual Int'l Cryptology Conf. Berlin, Heidelberg: Springer-Verlag, 1996. 104–113. [doi: 10.1007/3-540-68697-5_9]

[11] Osvik DA, Shamir A, Tromer E. Cache attacks and countermeasures: The case of AES. In: Proc. of the Cryptographers' Track at the RSA Conf. Berlin, Heidelberg: Springer-Verlag, 2006. 1–20. [doi: 10.1007/11605805_1]

[12] Yarom Y, Falkner K. Flush+Reload: A high resolution, low noise, L3 cache side-channel attack. In: Proc. of the 23rd USENIX Conf. on Security Symp. 2014. 719–732.

[13] Gruss D, Spreitzer R, Mangard S. Cache template attacks: Automating attacks on inclusive last-level caches. In: Proc. of the 24th USENIX Conf. on Security Symp. 2015. 897–912.

[14] Maurice C, Weber M, Schwarz M, Giner L, Gruss D, Boano CA, Römer K, Mangard S. Hello from the other side: SSH over robust cache covert channels in the cloud. In: Proc. of the 24th Annual Network and Distributed System Security Symp. 2017. 8–11. [doi: 10.14722/ndss.2017.23294]

[15] Percival C. Cache missing for fun and profit. In: Proc. of the BSDCan. 2005. 1–13.

[16] Irazoqui G, Inci MS, Eisenbarth T, Sunar B. Wait a minute! A fast, cross-VM attack on AES. In: Proc. of the Int'l Workshop on Recent Advances in Intrusion Detection. Cham: Springer-Verlag, 2014. 299–319. [doi: 10.1007/978-3-319-11379-1_15]

[17] Gülmezoğlu B, Inci MS, Irazoqui G, Eisenbarth T, Sunar B. A faster and more realistic Flush+Reload attack on AES. In: Proc. of the Int'l Workshop on Constructive Side-channel Analysis and Secure Design. Cham: Springer-Verlag, 2015. 111−126. [doi: 10.1007/978-3-319-21476-4_8]

[18] Benger N, Van De Pol J, Smart NP, Yarom Y. "Ooh Aah... Just a Little Bit": A small amount of side channel can go a long way. In: Proc. of the Int'l Workshop on Cryptographic Hardware and Embedded Systems. Berlin, Heidelberg: Springer-Verlag, 2014. 75−92. [doi: 10.1007/978-3-662-44709-3_5]

[19] Zhang Y, Juels A, Reiter MK, Ristenpart T. Cross-tenant side-channel attacks in PaaS clouds. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2014. 990−1003. [doi: 10.1145/2660267.2660356]

[20] Lipp M, Gruss D, Spreitzer R, Maurice C, Mangard S. ARMageddon: Cache attacks on mobile devices. In: Proc. of the 25th USENIX Security Symp. (USENIX Security 2016). 2016. 549−564.

[21] Intel. An introduction to the Intel quickpath interconnect. 2009. https://www.intel.cn/content/www/cn/zh/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[22] Gruss D, Maurice C, Fogh A, Lipp M, Mangard S. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2016. 368−379. [doi: 10.1145/2976749.2978356]

[23] Liu FF, Yarom Y, Ge Q, Heiser G, Lee RB. Last-level cache side-channel attacks are practical. In: Proc. of the 2015 IEEE Symp. on Security and Privacy. IEEE, 2015. 605−622. [doi: 10.1109/SP.2015.43]

[24] Pessl P, Gruss D, Maurice C, Schwarz M, Mangard S. DRAMA: Exploiting DRAM addressing for cross-CPU attacks. In: Proc. of the 25th USENIX Security Symp. (USENIX Security 2016). 2016. 565−581.

[25] Aldaya AC, Brumley BB, Hassan SU, Garćia CP, Tuveri N. Port contention for fun and profit. In: Proc. of the 2019 IEEE Symp. on Security and Privacy (SP). IEEE, 2019. 870−887.

[26] Van Bulck J, Minkin M, Weisse O, Genkin D, Kasikci B, Piessens F, Silberstein M, Wenisch TF, Yarom Y, Strackx R. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In: Proc. of the 27th USENIX Security Symp. (USENIX Security 2018). 2018. 991−1008.

[27] Boggs D, Baktha A, Hawkins J, Marr DT, Miller JA, Roussel P, Singhal R, Toll B, Venkatraman KS. The microarchitecture of the Intel® Pentium® 4 processor on 90nm technology. Intel Technology Journal, 2004,8(1). https://pdfs.semanticscholar.org/1431/b0615b0537947b492c24af0b70bd02f14317.pdf?_ga=2.235403906.22586668.1578898525-459389303.1527239744

[28] Intel. Intel 64 and IA-32 architectures software developer's manual, Vol.3 (3A, 3B & 3C): System programming guide. 2016.

[29] Weisse O, Van Bulck J, Minkin M, Genkin D, Kasikci B, Piessens F, Silberstein M, Strackx R, Wenisch TF, Yarom Y. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. Technical Report, University of Michigan, 2018.

[30] Stecklina J, Prescher T. LazyFP: Leaking FPU register state using microarchitectural side-channels. arXiv:1806.07480, 2018.

[31] Intel. Intel analysis of speculative execution side channels. Rev 4.0. 2018. https://software.intel.com/security-software-guidance/api-app/sites/default/files/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf

[32] Vahldiek-Oberwagner A, Elnikety E, Duarte NO, Sammler M, Druschel P, Garg D. ERIM: Secure and efficient in-process isolation with memory protection keys. arXiv:1801.06822, 2018.

[33] Hedayati M, Gravani S, Johnson E, Criswell J, Scott M, Shen K, Marty M. Hodor: Intra-process isolation for high-throughput data plane libraries. In: Proc. of the 2019 USENIX Annual Technical Conf. (USENIX ATC 2019). 2019. 489−503.

[34] Canella C, Van Bulck J, Schwarz M, Lipp M, von Berg B, Ortner P, Piessens F, Evtyushkin D, Gruss D. A systematic evaluation of transient execution attacks and defenses. In: Proc. of the 28th USENIX Security Symp. (USENIX Security 2019). 2019. 249−266.

[35] Lee S, Shih M, Gera P, Kim T, Kim H, Peinado M. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). 2017. 557−574.

[36] Evtyushkin D, Ponomarev D, Abu-ghazaleh N. Jump over aslr: Attacking branch predictors to bypass aslr. In: Proc. of the 49th Annual IEEE/ACM Int'l Symp. on Microarchitecture. IEEE, 2016. 40. [doi: 10.1109/MICRO.2016.7783743]

[37] Fog A. The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers. 2012. https://www.agner.org/optimize/microarchitecture.pdf

[38] Ge Q, Yarom Y, Cock D, Heiser G. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. Journal of Cryptographic Engineering, 2018,8(1):1−27. [doi: 10.1007/s13389-016-0141-6]

[39] Maisuradze G, Rossow C. ret2spec: Speculative execution using return stack buffers. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2018. 2109–2122. [doi: 10.1145/3243734.3243761]

[40] Schwarz M, Schwarzl M, Lipp M, Gruss D. Netspectre: Read arbitrary memory over network. In: Proc. of the European Symp. on Research in Computer Security. Cham: Springer-Verlag, 2019. 279–299.

[41] Chen GX, Chen SC, Xiao Y, Zhang YQ, Lin ZQ, Lai TH. SgxPectre attacks: Stealing Intel secrets from SGX enclaves via speculative execution. In: Proc. of the 2019 IEEE European Symp. on Security and Privacy (EuroS&P). IEEE, 2019. 142–157.

[42] Evtyushkin D, Riley R, Abu-Ghazaleh N, Ponomarev D. Branchscope: A new side-channel attack on directional branch predictor. In: Proc. of the ACM SIGPLAN Notices. ACM, 2018. 693–707. [doi: 10.1145/3173162.3173204]

[43] Kiriansky V, Waldspurger C. Speculative buffer overflows: Attacks and defenses. arXiv:1807.03757, 2018.

[44] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proc. of the ACM Conf. on Computer and Communications Security. 2007. 552–561. [doi: 10.1145/1315245.1315313]

[45] Intel. Intel 64 architecture memory ordering white paper. 2008. http://www.cs.cmu.edu/~410-f10/doc/Intel_Reordering_318147.pdf

[46] Abramson JM, Akkary H, Glew AF, Hinton GJ, Konigsfeld KG, Madland PD. Method and apparatus for performing a store operation. U.S. Patent, No.6,378,062, 2002.

[47] Abramson JM, Akkary H, Glew AF, Hinton GJ, Konigsfeld KG, Madland PD, Papworth DB, Fetterman MA. Method and apparatus for dispatching and executing a load operation to memory. U.S. Patent, No.5,717,882, 1998.

[48] Kosinski S, Latorre F, Cooray N, Shwartsman S, Kalifon E, Mohandru V, Lopez P, Aviram-Rosenfeld T, Topp J, Zei LG. Store forwarding for data caches. U.S. Patent No.9,507,725, 2016.

[49] Islam S, Moghimi A, Bruhns I, Krebbel M, Gulmezoglu B, Eisenbarth T, Sunar B. SPOILER: Speculative load hazards boost rowhammer and cache attacks. arXiv:1903.00446, 2019.

[50] Koruyeh EM, Khasawneh KN, Song CY, Abu-Ghazaleh N. Spectre returns! Speculation attacks using the return stack buffer. In: Proc. of the 12th USENIX Workshop on Offensive Technologies (WOOT 2018). 2018.

[51] AMD. Software techniques for managing speculation on AMD processors. 2018. https://developer.amd.com/wp-content/resources/Managing-Speculation-on-AMD-Processors.pdf

[52] ARM. Cache speculation side-channels. 2018. https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability/download-the-whitepaper

[53] Wang GH, Chattopadhyay S, Gotovchits I, Mitra T, Roychoudhury A. oo7: Low-overhead defense against spectre attacks via binary analysis. arXiv: 1807.05843, 2018.

[54] Oleksenko O, Trach B, Reiher T, Silberstein M, Fetzer C. You shall not bypass: Employing data dependencies to prevent bounds check bypass. arXiv:1805.08506, 2018.

[55] Intel. Retpoline: A branch target injection mitigation. 2018. https://software.intel.com/security-software-guidance/api-app/sites/default/files/Retpoline-A-Branch-Target-Injection-Mitigation.pdf?source=techstories.org

[56] Intel. Control-flow enforcement technology preview. 2017. https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

[57] AMD. AMD64 technology: Speculative store bypass disable. 2018. https://developer.amd.com/wp-content/resources/124441_AMD64_SpeculativeStoreBypassDisable_Whitepaper_final.pdf

[58] Intel. Speculative execution side channel mitigations. 2018. https://software.intel.com/en-us/download/speculative-execution-side-channel-mitigations

[59] Intel. Intel analysis of speculative execution side channels. 2018. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf

[60] AMD. Spectre mitigation update. 2018. https://developer.amd.com/wp-content/resources/Architecture_Guidelines_Update_Indirect_Branch_Control.pdf

[61] Khasawneh KN, Koruyeh EM, Song CY, Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. In: Proc. of the 2019 56th ACM/IEEE Design Automation Conf. (DAC). IEEE, 2019. [doi: 10.1145/3316781.3317903]

[62] Yan MJ, Choi J, Skarlatos D, Morrison A, Fletcher CW, Torrellas J. InvisiSpec: Making speculative execution invisible in the cache hierarchy. In: Proc. of the 2018 51st Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). IEEE, 2018. 428–441. [doi: 10.1109/MICRO.2018.00042]

[63] Kiriansky V, Lebedev I, Amarasinghe S, Devadas S, Emer J. DAWG: A defense against cache timing attacks in speculative execution processors. In: Proc. of the 2018 51st Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). IEEE, 2018. 974−987. [doi: 10.1109/MICRO.2018.00083]

[64] Page D. Partitioned cache architecture as a side-channel defence mechanism. IACR Cryptology ePrint Archive, 2005. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.9926&rep=rep1&type=pdf

[65] Wang ZH, Lee RB. Covert and side channels due to processor architecture. In: Proc. of the 2006 22nd Annual Computer Security Applications Conf. (ACSAC 2006). IEEE, 2006. 473−482. [doi: 10.1109/ACSAC.2006.20]

[66] ARM. Corelink level 2 cache controller L2C-310 technical reference manual. 2010. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0246f/DDI0246F_l2c310_r3p2_trm.pdf

[67] Domnitser L, Jaleel A, Loew J, Abu-Ghazaleh N, Ponomarev D. Nonmonopolizable caches: Low-complexity mitigation of cache side channel attacks. ACM Trans. on Architecture and Code Optimization, 2012,8(4):Article No.35. [doi: 10.1145/0000000.0000000]

[68] Liu FF, Ge Q, Yarom Y, Mckeen F, Rozas C, Heiser G, Lee Ruby B. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In: Proc. of the 2016 IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). IEEE, 2016. 406−418. [doi: 10.1109/HPCA.2016.7446082]

[69] Yan MJ, Gopireddy B, Shull T, Torrellas J. Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel atacks. In: Proc. of the 2017 ACM/IEEE 44th Annual Int'l Symp. on Computer Architecture (ISCA). IEEE, 2017. 347−360. [doi: 10.1145/3079856.3080222]

[70] Liu FF, Lee RB. Random fill cache architecture. In: Proc. of the 47th Annual IEEE/ACM Int'l Symp. on Microarchitecture. IEEE Computer Society, 2014. 203−215. [doi: 10.1109/MICRO.2014.28]

[71] Wang ZH, Lee RB. New cache designs for thwarting software cache-based side channel attacks. ACM SIGARCH Computer Architecture News, 2007,35(2):494−505. [doi: 10.1145/1250662.1250723]

[72] Zhou ZQ, Reiter MK, Zhang YQ. A software approach to defeating side channels in lastlevel caches. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2016. 871−882. [doi: 10.1145/2976749.2978324]

[73] Schwarz M, Maurice C, Gruss D, Mangard S. Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript. In: Proc. of the Int'l Conf. on Financial Cryptography and Data Security. Cham: Springer-Verlag, 2017. 247−267. [doi: 10.1007/978-3-319-70972-7_13]

[74] Schwarz M, Lipp M, Gruss D. JavaScript zero: Real JavaScript and zero side-channel attacks. In: Proc. of the Network and Distributed Systems Security (NDSS) Symp. 2018. Article No.12. [doi: 10.14722/ndss.2018.23094]

**吴晓慧**(1988－),女,安徽亳州人,博士生,工程师,CCF 学生会员,主要研究领域为系统安全,操作系统安全.

**贺也平**(1962－),男,博士,研究员,博士生导师,主要研究领域为系统安全,隐私保护.

**马恒太**(1970－),男,博士,副研究员,主要研究领域为软件安全分析,操作系统安全.

**周启明**(1973－),女,高级工程师,主要研究领域为操作系统安全,操作系统兼容,操作系统分析.

**林少锋**(1995－),男,博士生,主要研究领域为操作系统,操作系统安全.