计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(8): 2058−2097, 2025

瞬态执行攻击综述

李 扬[1,5] 马自强[1,5] 林璟锵[2] 孟令佳[4] 李冰雨[3] 苗 莉[1,5] 高 菲[1,5]

[1](宁夏大学信息工程学院 银川 750021)

[2](中国科学技术大学网络空间安全学院 合肥 230026)

[3](北京航空航天大学网络空间安全学院 北京 100191)

[4](网络空间安全防御重点实验室(中国科学院信息工程研究所) 北京 100085)

[5](宁夏"东数西算"人工智能与信息安全重点实验室(宁夏大学) 银川 750021)

(leeyang1109@163.com)

Survey of Transient Execution Attacks

Li Yang[1,5], Ma Ziqiang[1,5], Lin Jingqiang[2], Meng Lingjia[4], Li Bingyu[3], Miao Li[1,5], and Gao Fei[1,5]

[1](*School of Information Engineering*, *Ningxia University*, *Yinchuan* 750021)

[2](*School of Cyber Science and Technology*, *University of Science and Technology of China*, *Hefei* 230026)

[3](*School of Cyber Science and Technology*, *Beihang University*, *Beijing* 100191)

[4](*Key Laboratory of Cyberspace Security Defense* (*Institute of Information Engineering*, *Chinese Academy of Sciences*), *Beijing* 100085)

[5](*Ningxia Key Laboratory of Artificial Intelligence and Information Security for Channeling Computing Resources from the East to the West* (*Ningxia University*), *Yinchuan* 750021)

Abstract Transient execution attacks (TEAs) exploit processor optimizations to bypass security checks and exfiltrate sensitive information through covert channels. Among them, Meltdown and Spectre attacks have become prominent, affecting mainstream commercial processors such as Intel, ARM, and AMD. Despite the defensive measures implemented by processor manufacturers, variants of these attacks continue to be discovered and disclosed by researchers. To improve the understanding of TEAs and deploy robust defenses, this paper comprehensively analyzes TEAs under various covert channels. Initially, the common characteristics of TEAs are extracted, and a novel model for TEAs is systematically constructed. Subsequently, we summarize the various types of covert channels involved in existing research, classify the TEAs into three types: Meltdown type attacks driven by out-of-order execution (OoOE), Spectre type attacks driven by branch misprediction, and microarchitecture data sampling (MDS) type attacks driven by data misprediction, and delineate the key aspects and relationships of each type of attack. Notably, this paper systematically compiles and categorizes MDS type attacks for the first time. Then, the capabilities of each attack variant are meticulously analyzed and evaluated from three dimensions: covert channel, attack applicable scenarios, and microarchitecture immunity status, which aids security researchers in developing new and more destructive attack types based on the deficiencies of the existing attack-related research. Finally, combined with the above-mentioned comprehensive and in-depth analysis, and the summary of processor microarchitecture and covert channels, this paper anticipates the future trajectory of TEAs research, hoping to provide strong support for subsequent research work.

收稿日期: 2024-03-12; **修回日期**: 2024-09-19

基金项目: 宁夏回族自治区自然科学基金项目(2021AAC03078); 宁夏回族自治区重点研发计划项目(2021BEB04004, 2021BEB04047, 2022BDE03008); 宁夏大学研究生科技创新基金项目(CXXM2023-20)

This work was supported by the Natural Science Foundation of Ningxia Hui Autonomous Region of China (2021AAC03078), the Key Research and Development Program of Ningxia Hui Autonomous Region (2021BEB04004, 2021BEB04047, 2022BDE03008), and the Graduate Science and Technology Innovation Foundation of Ningxia University (CXXM2023-20).

通信作者: 马自强(maziqiang@nxu.edu.cn)

Key words convert channel; transient execution attack; side-channel attacks; microarchitecture data sampling attack; cache side channel; system security

摘 要 瞬态执行攻击利用处理器优化措施绕过安全检查，进而通过隐蔽信道传输并窃取敏感信息. 其中，Meltdown 和 Spectre 攻击尤为知名，波及包括 Intel, ARM, AMD 在内的主流商用处理器. 尽管处理器制造商已采取相应防御措施，但相关变种攻击仍不断被研究人员发现并公之于众. 为深化对瞬态执行攻击的理解并实施有效防御，对各种隐蔽信道下的瞬态执行攻击进行了剖析. 首先，提炼出了瞬态执行攻击的共同特征，并系统性构建了全新的瞬态执行攻击模型. 其次，总结了现有研究中涉及的各类隐蔽信道，将瞬态执行攻击归纳总结为 3 类：乱序执行驱动的熔断型攻击、错误分支预测驱动的幽灵型攻击以及错误数据预测驱动的数据采样型攻击，并梳理了各类型攻击的核心要点及关联性. 其中，对数据采样型攻击进行了系统性归纳和整理. 接着，从隐蔽信道利用、攻击适用场景和微架构通用性 3 个维度分析和评估了各攻击变种的能力. 最后，结合上述针对处理器微架构和隐蔽信道的深入分析与总结，展望了瞬态执行攻击研究的未来研究方向，以期为后续研究工作提供有力支撑.

关键词 隐蔽信道；瞬态执行攻击；侧信道攻击；微架构数据采样攻击；缓存侧信道；系统安全

中图法分类号 TP309

DOI: 10.7544/issn1000-1239.202440167 **CSTR:** 32373.14.issn1000-1239.202440167

自 2018 年以来，以 Meltdown[1] 和 Spectre[2] 为代表的瞬态执行攻击（transient execution attack, TEA）发展迅速，引起了学术界和工业界的极大关注. TEA 利用现代处理器集成的乱序执行和预测执行等性能优化措施窃取数据，并将其通过隐蔽信道传输，影响了包括 Intel 和 ARM 在内的几乎所有架构、所有型号的 CPU. TEA 已成为计算机系统安全领域的重要研究方向.

隐蔽信道（convert channel）是决定 TEA 成功与否的重要因素. 它是一种以违背系统安全策略的形式传送信息的通信通道[3]. 理论上，攻击者可以使用任何隐蔽信道完成攻击. 已有研究表明，TEA 可利用端口争用[4]、除法器[5] 和性能监控单元[6] 等 CPU 微架构组件作为隐蔽信道，传输敏感信息. 除此之外，还有通过监控 CPU 运行时频率等外部物理信息，构造隐蔽信道完成数据窃取的研究[7]. 然而，这些信道实现条件苛刻、易受噪音干扰，实际攻击成功率较低. 相比之下，使用高速缓存（cache）作为隐蔽信道的理论和实践研究[8-16] 较为成熟，其实现简单、抗噪音能力强、攻击成功率高[8]. 因此，获得了安全研究人员的重点关注.

TEA 可突破常规基于隔离的安全保障. 例如：Meltdown[1] 攻击可突破操作系统地址空间隔离来窃取内核数据；Foreshadow[17] 攻击可从英特尔软件保护扩展[18]（Intel software guard extensions, Intel SGX）建立的可信执行环境（trusted execution environment, TEE）中窃取飞地数据；Crosstalk[19] 攻击可从运行在另一物理核心的进程中窃取机密数据. 并且，TEA 还在不断发展演变，衍生出信息提取能力更强、适用场景更丰富的变种[20-21]. 例如：Foreshadow 的变种 Foreshadow-NG[20] 可跨操作系统和虚拟机窃取敏感信息；Spectre 的变种攻击 NetSpectre[21] 可通过网络数据包远程窃取密钥. 同时，各攻击利用的微架构漏洞来源不同，其适用场景、攻击能力也存在差异[22-25]. 例如：利用浮点运算单元（float point unit, FPU）延迟状态切换的特性，跨进程窃取数据[22]；利用单指令多数据（single instruction multiple data, SIMD）流向量寄存器读操作并行时错误延迟处理的特性，跨虚拟机窃取数据[23]；利用缓存一致性协议和异常时的错误数据转发，跨物理核心窃取数据[24-25]. 如何系统性地总结 TEA 相关研究是一项具有挑战性的工作.

目前，已有研究[26-33] 对缓存隐蔽信道以及 TEA 进行了总结. 2020 年苗新亮等人[26] 着重介绍了访问驱动的缓存隐蔽信道. 张伟娟等人[27] 从范围上拓展了文献 [26] 的工作，并从场景变化和实现层次等角度对缓存隐蔽信道发展状况进行了总结. 吴晓慧等人[28] 和李晔等人[29] 则建立了 TEA 模型，但所建模型均缺乏对 TEA 攻击窗口及代码利用方式的讨论. 一些研究还评估了各攻击变种的能力. 例如，2019 年 Canella 等人[30] 通过统计防御措施覆盖的变种范围，间接评估各攻击和防御方案的能力. 然而，该研究中部分被选为评价指标的防御方案是针对某一变种研

发的,以其能否覆盖其他变种来评估变种的攻击能力并不适合. 2021 年 Ragab 等人[31]重点总结并分类了由机器清除(machine clear)[34]造成的 TEA,并以 TEA 窗口大小和信息泄露率作为评价指标,评估各变种的攻击能力. 同年, Xiong 等人[32]基于对攻击场景进行建模,以各变种的攻击场景限制评估其数据泄露能力. 然而,该研究统计的攻击场景仅限于内存隔离视角,不能全面概括实际攻击所涉场景. 2023 年 Fiolhais 等人[33]立足于计算机架构视角,概述了易受 TEA 影响的微架构组件设计原理,尤其是缓存的软硬件设计. 然而,该研究缺乏对攻击的抽象建模,使读者把握攻击的共同特性更为困难,且其信道分析范围仅限于缓存.

此外,目前综述性工作还存在 3 个共性问题:一是弱化了隐蔽信道与 TEA 之间的密切联系,侧重对 TEA 攻击过程的阐述,缺乏对安全问题来源的系统性分析与建模. 二是未能全面总结微架构数据采样(microarchitecture data sampling, MDS)攻击[35-37]及其相关变种[38-44]的研究成果,同时缺少对使用非缓存型隐蔽信道[4-7, 21,45]发动 TEA 的研究总结. 三是缺少对 TEA 的隐蔽信道选用特性、攻击场景适用性的分析,以及对攻击的微架构通用性总结. 作为 TEA 的重要内容,上述 3 方面的工作不可忽视.

鉴于当前综述性工作存在的不足,本文重新整理了系统安全领域高水平期刊与学术会议中最新研究成果及处理器制造商公布的技术文档,对 TEA 进行了更为合理和详尽的分类综述. 具体而言,本文的贡献包括 3 个方面:

1)从处理器微架构视角和隐蔽信道视角出发,分析了 TEA 的安全问题来源. 全面系统地建立了 TEA 模型,有利于安全研究人员对 TEA 建立宏观性认识.

2)从基本原理出发,综述了幽灵型和熔断型攻击的现有研究,并系统地总结了数据采样型攻击的相关工作. 有利于安全研究人员对 TEA 建立具像性认识.

3)从隐蔽信道选用特性、攻击适用场景和微架构通用性 3 个角度对 TEA 进行了总结、对比和分析,并就其发展趋势及其面临的挑战提出了若干可行的研究方向.

1 TEA 的硬件根源

本节从处理器的硬件设计入手,分析 TEA 的安全问题来源. 首先介绍处理器结构和指令执行的基本流程;然后结合 CPU 优化措施的实现,分析其引入的信息泄露风险;最后结合缓存的设计和实现,分析其引入的隐蔽信道利用风险.

1.1 处理器结构

1.1.1 处理器架构与微架构

处理器架构是处理器逻辑层面的结构,通常指的是指令集架构(instruction set architecture, ISA). ISA 定义了 CPU 支持的指令集合、内存模型和执行模型,是 CPU、软件和程序员之间的接口,因此,架构层被视为处理器设计的核心[46]. 当指令的执行结果写入寄存器或内存时,应用程序或程序员才能对数据进行操作,此时称该数据为“架构层可见”.

处理器微架构是物理实现层面的结构,通常指 CPU 具体的物理实现,特别是 CPU 内部的硬件设计和组织方式. 应用程序或程序员无法获取正在 CPU 组件间处理及尚未写入寄存器的数据,此时称该数据为“架构层不可见”或“微架构层可见”.

微架构与架构之间的数据隔离,保证了指令的计算结果能在到达架构层之前,经过权限检查、内存越界访问检查等安全措施的筛查,从而避免不安全数据读取行为. 然而, TEA 可利用微架构实现特性,在安全措施实施前,抢先得到计算结果.

1.1.2 微架构的组成结构

TEA 强依赖于处理器微架构实现,然而,不同型号的微架构之间在具体实现细节上存在差异且结构复杂. 为便于读者理解,本文建立如图 1 所示的微架构概念图,并以此介绍指令的执行过程.

指令的执行过程可概括为取指、译指、执行、存储 4 个阶段.

1)取指和译指阶段

微架构前端的工作包括取指和译指,即负责将指令解析为对应的微操作(micro-operation, μOP). 首先,指令预取器从首级指令缓存(L1 instruction cache, L1I Cache)中获取待执行指令,并对复杂指令进行预解码. 预解码后的指令被存放在指令队列中,等待多路解码器进一步解码. 分配队列负责对解码后的 μOP 进行优化,例如,将可融合的多个 μOP 组合成一个 μOP 以节省计算资源. 最后, μOP 被分配队列传递给执行引擎执行.

2)执行阶段

执行引擎负责执行前端传送的 μOP,并将执行结果传递给存储子系统写入内存. 调度器根据程序顺序,将 μOP 分派给对应执行单元(execution unit, EU)串行执行,这种方式被称为“顺序调度”. 顺序调度的

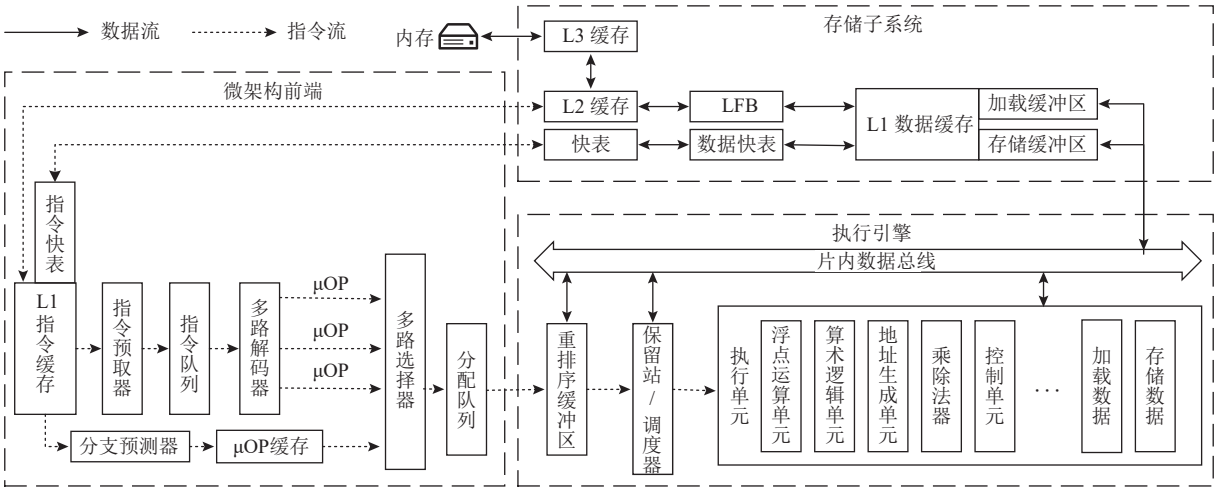


Fig. 1 Concept diagram of microarchitecture

图 1 微架构概念图

缺点十分明显：若前序 μOP 因数据读取等操作暂时停滞，后序 μOP 也会停滞，即使它不依赖于前序 μOP 的执行结果，这将严重降低 CPU 的性能.

3）存储阶段

存储子系统从内存中读取数据以供执行阶段的计算使用，并将执行引擎传递的结果写回内存. 早期的存储系统存储和加载数据（合称“存取”）时间长，增加了 CPU 的等待时间，降低了 CPU 的计算效率. 为减少存取时延，CPU 在存储子系统中引入了缓存、行填充缓冲区（line fill buffer, LFB）、存储缓冲区（store buffer, SB）和加载缓冲区（load buffer, LB）等.

1.2 处理器优化措施

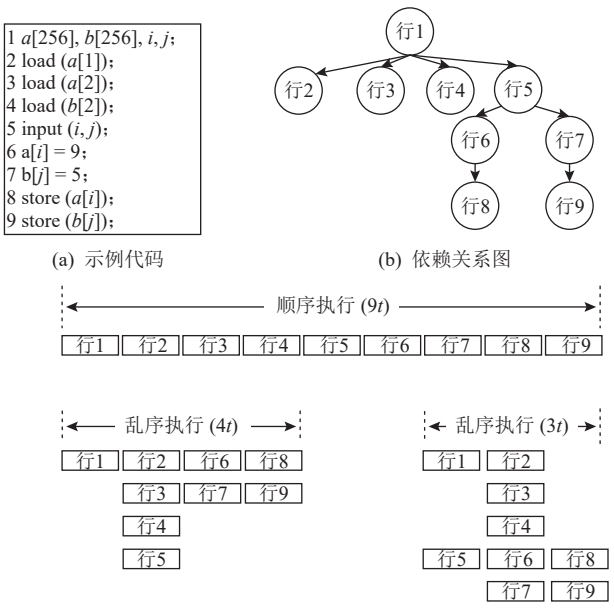
从 1.1 节的介绍可知，指令的顺序执行造成了计算资源的严重浪费. 为加速指令执行，工业界引入了一系列方法来提高指令执行效率. 然而，这也为 TEA 的爆发埋下了隐患. 本节从它们的基本实现原理出发，分析其带来的安全风险.

1.2.1 乱序执行

乱序执行[47]（out of order execution, OoOE）允许 CPU 根据输入数据的可用性重新确定指令的执行顺序，以最大限度利用执行资源，提高指令级并行性和执行效率. 为使无数据依赖指令能并行，微架构中引入了重排序缓冲区（reorder buffer, ROB）和保留站（reservation station, RS），并将指令的执行阶段重新设计为发射、执行、清退和提交 4 个阶段[48].

ROB 在 μOP 进入 RS 之前记录其原始顺序，RS 替代调度器对 ROB 传递的 μOP 进行乱序调度. μOP 在发射后被插入到 RS 的一个表项中，并通过监听片内数据总线获取所需的操作数状态，一旦所需的所有操作数均已就绪，RS 便立即调度该 μOP 至 EU 执行，并暂存其执行结果. 清退阶段会对执行完毕的 μOP 进行权限检查，通过，则进入提交阶段，提交阶段将计算结果按照 ROB 中记录的原始顺序写回至寄存器和内存，此时数据才在架构层可见；否则，将清空指令流水线，以清除错误执行的影响.

OoOE 解除了对指令执行顺序的限制，极大地提升了 CPU 性能. 本文以图 2（a）所示的指令为例，简要展示 OoOE 的性能提升效果. 假设每条指令执行时间均为 1 时钟周期，记为 1t. 如图 2（c）所示，顺序执行时，指令需要 9t 才能完成，此时 CPU 大部分时钟周期用于数据存取，计算资源闲置. 图 2（b）刻画了图 2（a）



(a) 示例代码 (b) 依赖关系图

(c) 顺序执行与乱序执行时间提升对比

Fig. 2 Out of order execution

图 2 乱序执行

中各行指令间的依赖关系, 可见, 行 2~4 对不同地址数据的加载指令和行 8~9 对不同地址数据的存储指令之间, 并不存在数据依赖或者控制依赖, 实际情况下可并行. 此时, OoOE 执行时间缩短为 4t. 在更理想的情况下, 若行 5 的输入与行 1 同时完成, 此时行 6~7 指令的执行前置条件已经满足, 执行时间仅为 3t.

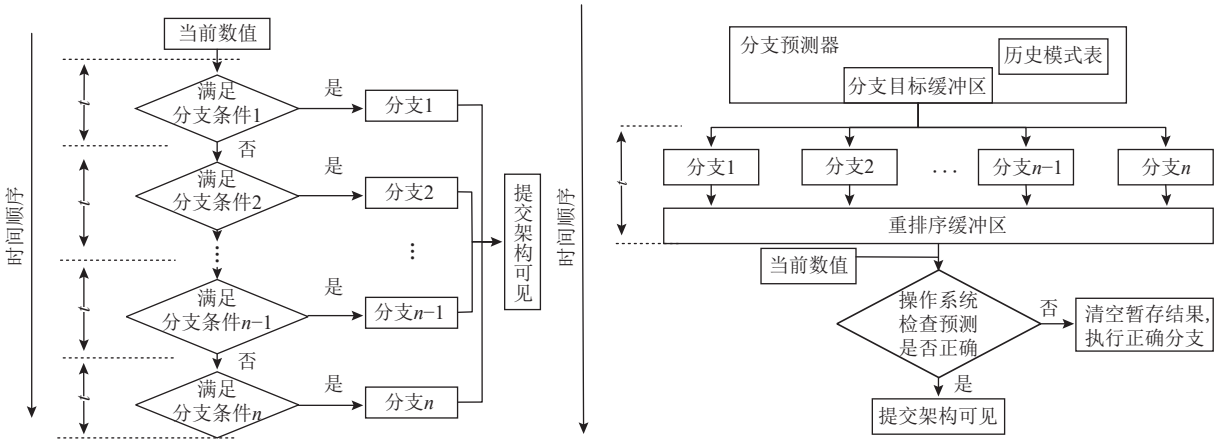
1.2.2 预测执行

预测执行[49](speculative execution, SE)可根据指令历史行为提前预测数据或控制依赖结果, 并基于预测结果不阻塞地继续执行. 这样能避免因等待未执行指令结果而导致的阻塞, 从而提高 CPU 性能. 预测执行通常包括分支预测和数据预测.

分支预测的核心在于预测指令执行流. 为了使指令能预测执行, 微架构中引入了分支预测器(branch predictor, BP), 使分支条件解析和分支执行并行.

数据预测的核心在于预测指令数据流, 并在数据真正可用前进行计算. 为使指令能提前预测数据的依赖关系和可用性, 微架构中引入了包括内存消歧器在内的一系列组件, 使数据地址解析和指令执行并行.

分支预测和数据预测可以极大地提高 CPU 的数据处理速度. 本文以分支预测为例, 简要说明分支预测的性能提升效果. 如图 3(a)所示, 当拥有 n 个分支的程序的每条分支指令都依赖于高延迟分支条件时, 其平均执行时间为 $(n+1)t/2$. 而基于历史跳转记录预测分支方向的分支预测器可以保证超过 95% 的预测准确度, 如图 3(b)所示, 其平均执行时间将缩短至 $0.95t + 0.025(n+1)t$.



(a) 顺序执行 (b) 预测执行

Fig. 3 Speculative execution

图 3 预测执行

1.2.3 数据存取转发

为提升 CPU 存储效率, 存储子系统在存取缓冲区中引入了存储到加载转发(store to load forwarding, STL)预测来提升存取指令的执行性能. STL 依靠 LFB, LB, SB 实现, 如图 1 中存储子系统所示.

1)LFB. LFB 是 L1 和 L2 缓存之间的桥梁, 负责跟踪未完成的存储请求、组合写和管理不可缓存的内存访问等[35-36]. CPU 在 L2 中查找数据时, 若数据存在, 则将其加载至 LFB, 并更新至 L1D Cache 和 LB. 若未在 L2 中找到, 则向 LLC 发出核外请求[39].

2)LB 和 SB. 加载(存储)指令解码后, 会在 LB 中创建一个与目标地址的虚拟地址和物理地址相关的条目. 当 CPU 需要从内存中加载(存储)数据时, 这些加载(存储)请求会被加入 LB(SB)中的加载(存储)队列. LB(SB)会处理剩余操作, 这样 CPU 可以继续执行后续指令而无需等待加载(存储)指令全部完成. 同时, 加载(存储)的数据会暂存在 LB(SB)中, 这样能有效减少内存访问次数, 提高指令的执行效率.

3)STL. 存储与加载指令指向同一物理地址, 且全地址匹配时, SB 将对应地址的数据转发至 LB. 而在微架构的实现中, 与架构层要求的虚拟地址或物理地址全匹配不同, 仅低位匹配就预测性完成转发操作. 这导致对内存的存储值可能会被错误转发给加载指令.

1.2.4 处理器优化措施带来的信息泄露风险

1.2.1~1.2.3 节介绍了与 TEA 相关的 CPU 优化措施, 本节将总结其设计时的正确性设想以及被 TEA 利用的微架构安全漏洞.

1)架构设计时的指令正确性设想

若执行顺序错误或预测错误, 则刷新流水线、丢

弃错误执行结果，以清除错误执行的影响，然后重新发射指令以获取正确结果. 在此期间任何错误的结果均不会被架构层直接获取，以此来保证指令执行的正确性以及数据安全.

2）TEA 利用的微架构安全漏洞

①异常延迟处理. 异常延迟处理是熔断型和数据采样型攻击的漏洞来源. 为简化硬件机构，OoOE 具有原子性. 即使指令执行过程中发现了错误或异常，执行也不会中断，而是延迟至清退阶段处理. 这意味着恶意指令在执行阶段时，可突破严格的内存隔离与权限管理，越界读取其他地址空间数据.

②错误预测. 错误预测是幽灵型和数据采样型攻击的漏洞来源. 正确的预测执行可避免等待结果解析而带来的时延损失，提升处理器性能. 但错误的预测执行会使处理器转向恶意代码执行，在结果解析之前，这种错误执行并不会被纠正. 这意味着在错误预测执行时，恶意指令可越权访问其他进程的数据.

③可基于瞬态执行读取的值做计算. 可基于瞬态执行读取的值做计算是秘密传输的关键. 计算操作能将秘密信息转换为微架构组件的状态变化，而这些状态变化不会因错误执行而撤销. 攻击者通过观测组件状态变化推断秘密，从而达到信息传递的目的.

1.3 处理器高速缓存

1.2 节中介绍了 TEA 利用的微架构层面的安全漏洞. 然而，瞬态执行过程中非法访问的数据会被撤销，不能被架构层观测. 因此，需通过隐蔽信道将其转换为架构层可见. 缓存隐蔽信道是 TEA 研究的重点，缓存的特性是其实现的基础，本节从它的设计和实现出发，分析 Cache 特性带来的安全风险.

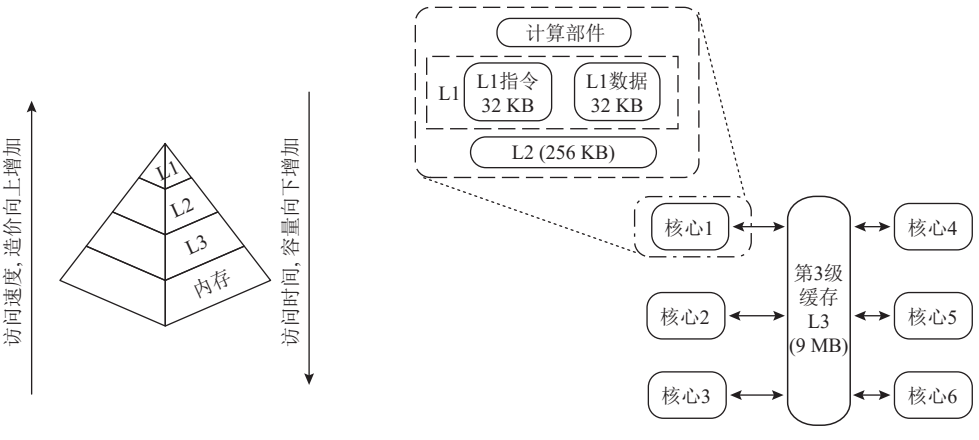
1.3.1 缓存的理论基础

CPU 对数据的访问存在着时间局部性和空间局部性[50]. 时间局部性是指最近访问的数据在短时间内极大可能再次被 CPU 访问. 而空间局部性则表明后续指令访问的数据与近期访问数据在物理存储空间上连续或相近. 缓存利用了这些时间和空间的局部性，通过优化硬件结构，提高 CPU 对最近访问数据及空间上相邻数据的访问速度. 这种优化能弥补 CPU 与内存在读取速度上的巨大差异，减少 CPU 等待时间，进而提升效率.

1.3.2 缓存的硬件架构

内存由读写速度较慢但价格低廉的动态随机存取存储器（dynamic random access memory, DRAM）构成，而缓存则采用读写速度较快但造价昂贵的静态随机存取存储器（static random access memory, SRAM）. 由于经济成本的限制，缓存的大小不能无限制地增加.

为了在成本和性能之间取得平衡，设计上的解决方案是采用类似如图 4（a）所示的 3 级塔型结构. 这种结构中，靠近 CPU 的缓存速度更快、成本更高，但相应的容量会减少. 图 4（b）为 Intel Core i7-8850H 的缓存架构，它有 6 个物理核心，其中 L1，L2 为各物理核心独占，最接近内存的最后一级缓存（last level cache, LLC）在物理核心间共享[8].



(a) 缓存的3级塔型架构　　(b) 缓存架构示例

Fig. 4 Cache architecture

图 4 Cache 架构

1.3.3 缓存特性带来的隐蔽信道利用风险

1.3.1~1.3.2 节介绍了缓存的理论基础和硬件架构. 本节将总结它们在设计时的安全设想以及被 TEA 利用的安全漏洞.

1）缓存设计时的安全设想. 缓存仅作为数据存储区域，不能直接用来传递信息. 操作系统（operating

system, OS)提供严格的内存地址隔离,确保存储在缓存的进程数据不能被其他进程直接访问.

2)TEA 利用的缓存特性. 缓存的共享性和状态可测量性是缓存隐蔽信道的基础. 本节以下述代码为例,简述如何利用共享性和状态可测量性推断数据.

① if (x){
② $y = 1$;
③ }else{
④ $z = 2$;
⑤ }

假设进程 P_1 和 P_2 分别运行于物理核心 1 和物理核心 2,布尔值 x 为 P_1 独占,代码、变量 y 和 z 为 P_1 与 P_2 共享,且 y 和 z 分别保存在内存块 B_1 和 B_2 中. 此时,因内存隔离,P_2 无法获取 x 值.

1)空间共享性

LLC 在物理核心间共享. 若 $x = 1$, B_1 被加载至缓存;反之,加载 B_2. 若 P_2 在 LLC 中检索到 B_1,可推断 x 值为 1. 这种共享性还可进一步推广,若攻击者与受害者运行在同一物理核心,攻击者可在 L1, L2 缓存检索.

2)数据共享性

缓存的数据共享方式主要分为 2 种:①高级缓存中数据是低级缓存中数据的真子集. ②高级缓存中数据与低级缓存中数据的交集为空.

第 1 种方式称为包含式(inclusive)缓存,P_1 对 L1 中 B_y 的数据修改会基于缓存一致性协议[51-52]同步更新 L3 中的 B_1 副本. 因此 P_2 可通过 L3 探测 B_1 的状态变化. 因其硬件实现简单且高效,包括 Intel 在内的大部分 CPU 都使用包含式缓存[2]. 因此,缓存隐蔽信道研究[8-16]也大量集中于包含式缓存.

第 2 种方式称为非包含式(non-inclusive)缓存[53-54],数据从 L3 读取到 L1 或 L2 后,L3 数据副本不再保存,此时常规 L3 缓存隐蔽信道[8,13]失效. 因此,使用非包含式缓存的 CPU 不受原始 Spectre 影响[2].

3)缓存状态可测量

缓存状态可测量是空间共享性和数据共享性中检索 B_1 和 B_2 的基础. 缓存状态可通过数据访问时间测量,相较于从内存中读取数据需花费几百个时钟周期,从缓存读取仅需几个时钟周期. P_1 可基于测得的数据访问时间长短来推断本身是否位于缓存中.

2 TEA 模型

本节介绍 TEA 模型. TEA 在具体实施上各有差异. 通过掌握它们之间的共同特征,有助于全面地理解这类攻击. 首先,建立全新的抽象模型介绍 TEA 的基本思路和步骤;然后,对 TEA 执行过程中的要点进行建模介绍.

2.1 抽象模型

本文在理解各种攻击执行过程的基础上,建立 TEA 的抽象模型. 该模型能描述所有类型攻击的基本过程. 为便于描述,我们先给出 4 个定义.

定义 1. 载体组. 攻击者掌握的一系列可存取、测量的内存区域.

定义 2. 载体项. 载体组的基本单位,所有载体项合称一个载体组.

定义 3. 秘密载体项. 攻击者通过编码操作建立起的值与载体项的双射(bijection). 秘密对应的载体项为秘密载体项.

定义 4. 驱逐集. 驱逐集是映射到同一缓存集的内存块地址的集合,其基于缓存的映射策略和替换策略生成,访问驱逐集会导致对应的缓存集发生冲突,从而产生缓存行替换.

如图 5 所示,抽象模型分为 5 个阶段,包括:1)预处理;2)瞬态触发;3)瞬态访问;4)秘密编码传输;5)编码信息接收与秘密还原.

1)阶段 T1. 预处理. 该阶段是 TEA 的准备阶段. 为确保可瞬态访问秘密数据,攻击者需根据微架构的实现进行准备. 例如,找到可利用的代码片段(称为“Gadget”)、准备隐蔽信道并构建攻击窗口.

2)阶段 T2. 瞬态触发. 攻击者执行一些事先选定的指令来触发瞬态执行,该指令的后续操作会被清除并回滚,我们将其称为瞬态触发指令. 例如, Spectre V1 的瞬态触发指令是执行与实际相悖的错误分支;Meltdown 的瞬态触发指令则是对内核地址的访问.

3)阶段 T3. 瞬态访问. 攻击者在乱序执行或错误预测中访问正常程序流程中无法接触的敏感数据. 例如,因进程间地址隔离,进程 1 无法访问存储在缓存中的进程 2 的密钥,但瞬态访问时可实现这一点.

4)阶段 T4. 秘密编码传输. 阶段 T2 和阶段 T3 获取的秘密最终无法在架构层观察到结果,需将其转变为秘密载体项来间接保存. 攻击者基于阶段 T3 获取的秘密做运算的行为称为编码,编码会改变缓存状态,秘密经过编码后转化为秘密载体项.

5)阶段 T5. 编码信息接收与秘密还原. 该阶段是攻击的最终目标. 攻击者遍历载体组,并基于 1.3.3 节中总结的 3 个特性探测缓存,以确定秘密载体项. 然后对秘密载体项进行解码. 解码能将秘密从微架构

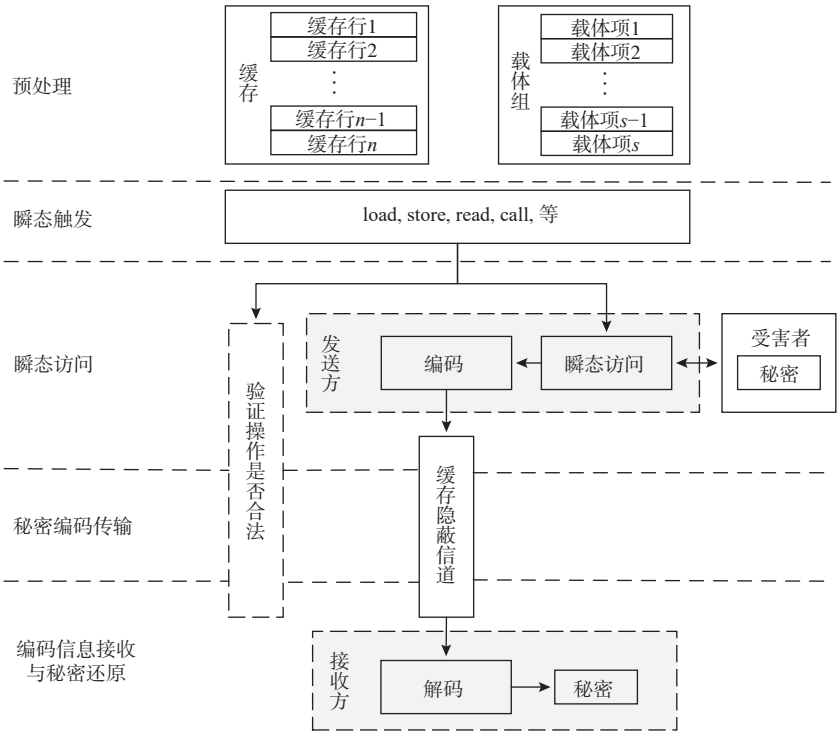


Fig. 5 Abstract model of transient execution attack

图 5 瞬态执行攻击抽象模型

提取到架构层面.

2.2 代码利用模型

寻找 Gadget 是阶段 T1 的一项主要工作，早期由人工挑选获得，随着相关防御研究深入，目前还可以通过使用瞬态漏洞检测工具[55-57]取得. TEA 可利用的 Gadget 形式杂、种类多，但根据本文的总结，Gadget 可以分为 2 类：显式 Gadget 和隐式 Gadget. 它们的模型分别如图 6(a)和图 6(b)所示.

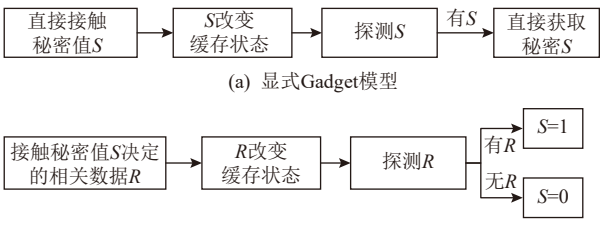


Fig. 6 Two types of Gadget models

图 6 2 类 Gadget 模型

显式 Gadget 与信息流中的显式流(explicit information flow, EIF)相关，是一种将秘密直接传递到隐蔽信道的代码片段. 这类 Gadget 依赖于操作数的硬件资源使用，并且会直接接触数据. 例如，数据加载指令对敏感数据的直接读取.

隐式 Gadget 与信息流中的隐式流(implicit information flow, IIF)相关，是一种能将秘密间接传递到隐蔽信道的代码片段. 这类 Gadget 依赖于资源使用的变化揭示秘密信息. 例如，秘密是条件分支中的条件，那么基于秘密的分支结果将决定后续分支的执行，进而间接决定是否使用特定于某个分支的功能单元. 通过观察特定功能单元的使用情况，可以推断出秘密.

目前 TEA 利用代码多选用显式 Gadget，而隐式 Gadget 使用较少. 本文认为，这与瞬态访问时可无视处理器安全检查机制直接访问数据的特性密切相关，选用隐式 Gadget 反而会增加阶段 T4 和 T5 的分析开销. 随着检测显式 Gadget 并对其进行消除性重构的防御研究出现[55-57]，使用隐式 Gadget 进行更隐蔽的信息窃取或许会成为攻击研究的未来趋势.

2.3 竞争模型

2.2 节介绍了阶段 T1 的 2 种 Gadget 模型. 然而，并不代表 TEA 一定能找到并执行 Gadget. 阶段 T2 瞬态触发后，攻击者还需处理 2 类竞争：一是阶段 T3 瞬态访问和 CPU 异常处理之间的竞争，二是阶段 T4 秘密编码传输和指令清退之间的竞争. 攻击者在任何一个竞争中落后均会导致 TEA 失败.

如图 7(a)所示，设 T_0, T_1, T_2 分别为指令开始执行、CPU 捕捉异常、指令清退开始时刻，t_1 和 t_2 分别为数据读取完成和隐蔽信道传输完成时刻. 期间可能出现 3 种情况：

1) $t_1<T_1<t_2<T_2$

在异常处理之前数据已经可用, 并且在指令清退前已经完成了秘密编码, 这种情况下, 隐蔽信道可以完成秘密传递.

2) $T_1<t_1<t_2<T_2$

如图 7(b)中①所示, 此时瞬态触发指令造成的系统不正常状态(例如异常等)已被 CPU 捕捉并开始处理. 此时, 攻击者瞬态读取行为被停止, 导致攻击失败.

3) $t_2>T_2$

如图 7(b)中②所示, 此时指令已进入清退阶段, 处理系统异常状态带来的影响以及流水线中包括编码在内的所有指令被清除. 此时, 无论秘密是否已被攻击者成功瞬态读取, 均无法将秘密编码为秘密载体项, 导致攻击失败.

为使 TEA 顺利完成, 攻击者可采用以下 2 种方式拓展瞬态执行窗口(speculative execution windows, SEW), 如图 7(c)所示.

1)增大Ⅰ区间, 延后处理器异常捕获. 大多数攻击方式会从缓存中清除与分支解析相关的操作数, 迫使 CPU 重新从内存中获取这些操作数. 此时 T_1 会沿箭头方向后移. 这种方式能为攻击提供 100~200 个 μOP 的窗口时间[58]. 例如, Spectre V1 将条件分支中的边界条件 n 从缓存清空, 迫使其从内存中重新读取, 为阶段 T3 和阶段 T4 提供宝贵的窗口时间.

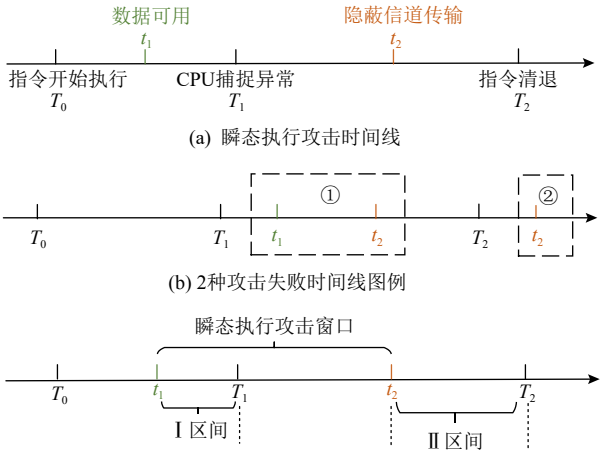
2)减小Ⅱ区间, 减少编码操作所需时间. 使用简短高效的瞬态访问及编码指令, 会使 t_2 沿箭头方向 T_2 靠近. 然而, 选择合适的瞬态访问及编码指令是一个难点: 指令过短可能无法达到攻击目的, 而指令过长会导致 $t_2>T_2$ 描述的情况.

2.4 缓存隐蔽信道模型

2.2 节和 2.3 节中介绍了代码利用模型和竞争模型. 然而, 即使攻击者找到了 Gadget 并在 2 种竞争中胜出, 依然无法直接获取数据, 还需分析缓存状态以还原秘密. 但缓存隐蔽信道种类较多, 其执行细节存在差异. 因此本节首先建立缓存隐蔽信道的抽象模型, 并介绍基本步骤, 接着介绍时序模型, 最后总结 2 类模型缓存隐蔽信道的基本原理.

2.4.1 缓存隐蔽信道的抽象模型

缓存隐蔽信道涉及发送方和接收方. 本文在深入理解各种缓存隐蔽信道传递信息的基础上, 对缓存隐蔽信道的信息传递过程进行建模, 该模型能描述所有类型的缓存隐蔽信道. 如图 8 所示, 该模型可分为 4 个阶段, 包括: 1)预处理阶段; 2)发送方编码阶段; 3)监测目标执行阶段; 4)接收方探测缓存状态与解码.



注: ①$T_1<t_1<t_2<T_2$, ②$t_2>T_2$

Fig. 7 Competition model

图 7 竞争模型

1)阶段 C1. 预处理. 该阶段是缓存隐蔽信道的准备阶段, 发送方需要消除无关缓存事件的干扰. 例如, 驱逐接收方已在缓存中的载体项 a.

2)阶段 C2. 发送方编码. 发送方通过执行简单算术指令, 将瞬态读取的秘密 i 编码为秘密载体项 $a[i\times64]$.

3)阶段 C3. 监测目标执行. 在这一阶段, 接收方需要等待瞬态执行编码完成, 并跟踪缓存变化状态, 选择在适当的时候执行阶段 C4.

4)阶段 C4. 接收方缓存状态探测与解码. 接收者通过监视和分析目标程序对缓存状态改变, 推断发送方编码在缓存隐蔽信道中的秘密载体项, 并对其进行逆运算以获取秘密 i.

2.4.2 隐蔽信道视角的时序模型

隐蔽信道各个阶段并不完全集中在阶段 T4 和 T5, 而是分散在 TEA 各阶段. 并且 TEA 各个阶段也并非完全是线性的, 它们在时间上存在重叠. 本节建立时序模型, 描述它们在执行时间上的重叠关系, 如图 9 所示.

1)阶段 C1 是阶段 T1 的子集. 这是由于阶段 T1 除了初始化隐蔽信道, 还需进行其他准备工作, 例如, 检测程序中可能存在的 Gadget、将检查条件逐出缓存以拓展 SEW[1,21-22,59] 以及循环调用某一函数毒害间接分支预测器[2,60-66] 或返回跳转预测器[67-68];

2)阶段 T1 与阶段 T2 以及阶段 C1 与阶段 C2 之间存在间隔. 这是由于 Gadget 夹杂在正常程序中, 预处理完成后通常需执行一段正常代码, 然后再跳转至 Gadget 执行.

3)阶段 T2, T3, T4 之间存在重叠, 阶段 C2 可能贯穿上述这 3 个阶段. 这是由于 Gadget 种类丰富导

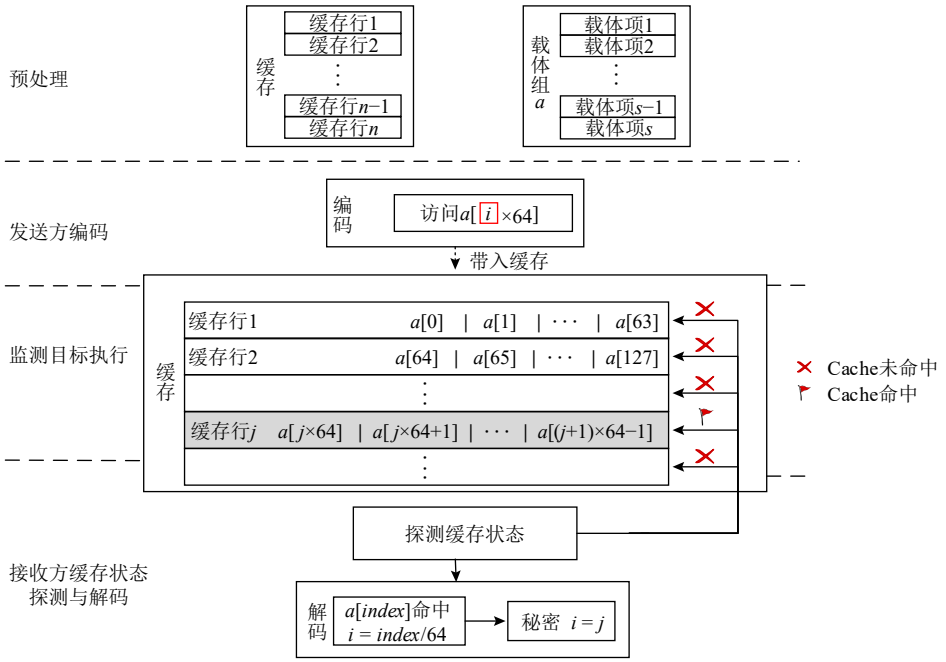


Fig. 8 Abstract model of cache covert channel

图 8 缓存隐蔽信道抽象模型

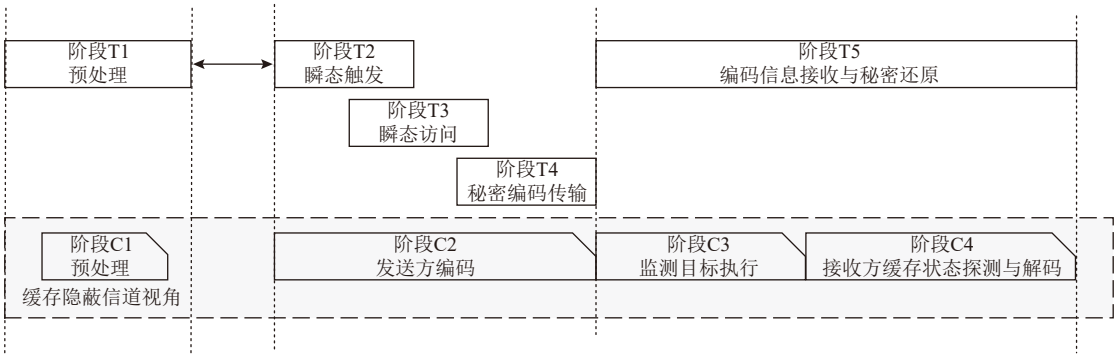


Fig. 9 Temporal sequence model

图 9 时序模型

致的，例如 $y=a[i]$，若 i 为内核空间地址，a 为载体项，执行该指令的同时完成了瞬态触发（访问内核地址 i）、瞬态访问（读取 $a[i]$ 值）、秘密编码传输（$a[i]$ 被暂存在缓存）. 若不考虑分支内指令的 OoOE，4.2 节中 Spectre V1 的概念验证（proof of concept, PoC）代码则是 3 个阶段不存在重叠的一个 Gadget 示例.

2.4.3 缓存争用模型与数据重用模型

1.3 节中介绍了缓存的共享性分为数据共享性和空间共享性. 根据对缓存共享性的不同利用，缓存隐蔽信道也分为 2 种：缓存争用型信道和数据重用型信道，本节将对这 2 类信道进行建模.

1）缓存争用模型

设同一物理核上运行 2 个进程：攻击进程 A 和受害进程 V，且它们的部分数据映射在同一缓存集. 缓存争用模型如图 10（a）所示，其中每个小方块代表一个缓存行，A 首先使用自身数据占用缓存行，然后触发 V 运行. 若短时间内 A 再访问步骤①中的数据，发生多次未命中，则可以推断 V 也在同时使用该缓存集，造成 A 的数据被驱逐出缓存行. A 通过监测这些冲突的缓存集，可以推断 V 访问缓存的模式，进而分析敏感信息.

2）数据重用模型

设同一物理核上运行 2 个进程：攻击进程 A 和受害进程 V，且它们需要使用同一文件 F（如加密库等）. 为优化存储空间，缓存中只保留了 F 的 1 份副本 F_0，换言之，当 A 或 V 调用文件 F 时，它们都将读取 F_0. 数据重用模型如图 10（b）所示，首先，A 清空缓存中的共享数据，然后等待 V 运行. A 调用 F 时，本需要从内存读取数据，若 A 通过测量数据加载时间 t，探测到 F_0 已位于缓存，则可以推断 V 曾调用 F，此时，

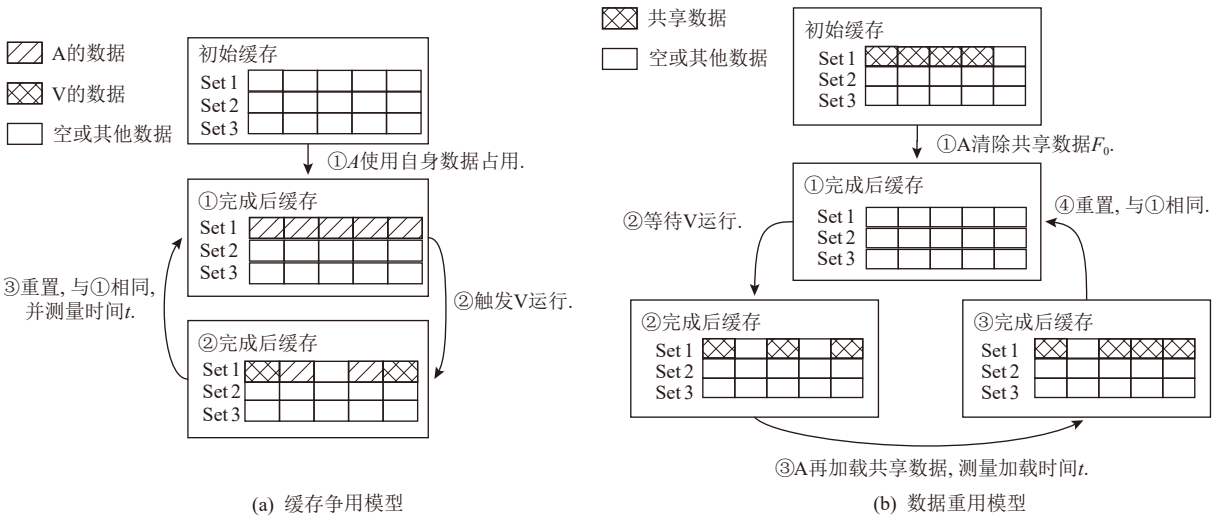


Fig. 10 Cache contention model and data reuse model

图 10 缓存争用模型与数据重用模型

A 可基于缓存命中情况获取 V 对 F 的历史访问情况, 进而通过监测这些共享的缓存行分析 V 的访问踪迹, 推断敏感信息.

3 TEA 中的隐蔽信道

隐蔽信道作为 TEA 的信息传递通道, 决定着攻击的成败. 本节总结了现有研究中涉及的 5 类缓存型隐蔽信道和 5 类非缓存型隐蔽信道.

3.1 缓存争用型信道

根据对缓存利用方式的不同, 缓存型信道可分为缓存争用型信道和数据重用型信道. 本节介绍缓存争用型隐蔽信道.

3.1.1 Evict+Time

2005 年, Bernstein[69] 尝试使用缓存引起的读取时间变化对 AES 算法进行攻击, 但这需要在相同配置下测量已知密钥下的加密操作作为参考, 且对加密的定时依赖较高、信噪比低, 在实际系统中较难应用. 2006 年, Osvik 等人[10] 提出的 Evict+Time 信道可有效解决上述问题. 该方法利用 AES 算法查表优化加解密运算的特征, 在单机单核心的系统中, 通过测量加密操作的执行时间, 成功恢复了全部的 AES 密钥, 从理论和实践上验证了缓存隐蔽信道传输信息的可行性. 如图 11 所示, Evict+Time 信道包含 Wait, Evict, Time 三个阶段.

1) Wait 阶段. 发送方触发瞬态执行, 这会将秘密值编码进入缓存隐蔽信道. 同时, 记录该次执行所需时间 t_1.

2) Evict 阶段. 接收者访问 E_i 中的 n 个内存地址, 将秘密载体项逐出缓存.

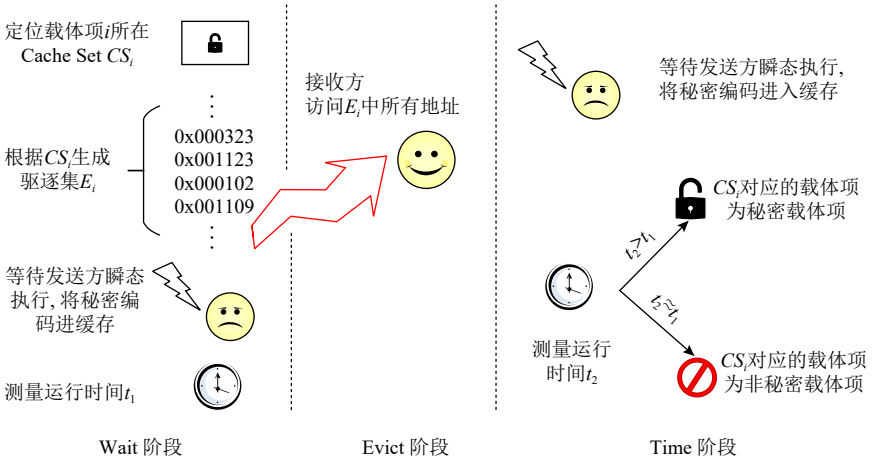


Fig. 11 Basic steps of Evict+Time covert channel

图 11 Evict+Time 隐蔽信道基本步骤

3）Time 阶段. 接收者再次触发瞬态执行，并测量此次执行所需时间 t_2.

根据秘密载体项判断，若 $t_2>t_1$ 则说明 Evict 阶段中驱逐的缓存集对应的载体项为秘密载体项.

3.1.2 Prime+Probe

文献 [69] 还提出了适用于 L1 缓存的 Prime+Probe 信道. Liu 等人[13] 将 Prime+Probe 拓展到了各物理核心共享的 L3 缓存. 与 Evict+Time 通过测量 AES 查找表的读取时间不同，他们提出通过接收方测量对自身数据的访问时间推断目标数据. 如图 12 所示，Prime+Probe 信道包含 Prime, Wait, Probe 三个阶段.

1）Prime 阶段. 发送方使用自身数组填充缓存，将载体组逐出缓存.

2）Wait 阶段. 等待瞬态指令将秘密编码进入缓存隐蔽信道. 编码操作会将秘密载体项加载至缓存，使发送方填充的缓存集被秘密载体项替换.

3）Probe 阶段. 接收者再次访问准备阶段的自身填充数据，并测量每个载体项的访问时间 t_i.

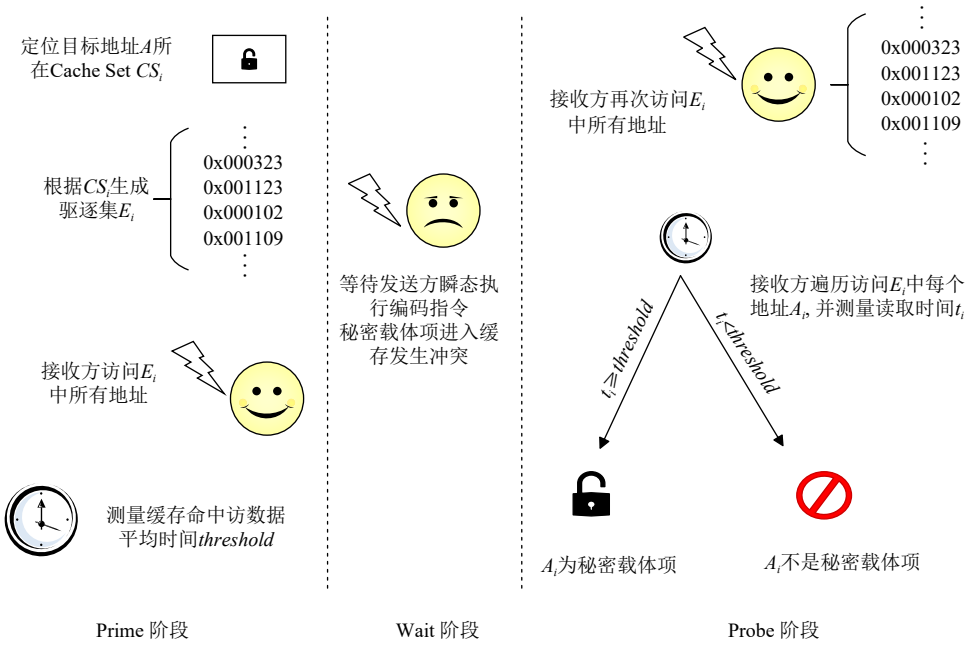


Fig. 12 Basic steps of Prime+Probe covert channel

图 12 Prime+Probe 隐蔽信道基本步骤

根据秘密载体项判断，若 t_i 明显大于平均访问时间 *threshold*，说明该缓存集 CS_i 对应的载体项 i 为秘密载体项.

3.1.3 Evict+Reload

Lipp 等人[15] 对 Flush+Reload 信道[11] 的前期步骤做出改进，将 Flush 替换为 Evict 操作，使用驱逐集逐出目标地址，这一改进使缓存信道在指令 clflush 禁用的系统中也可以使用. 如图 13 所示，该信道包括 Evict, Wait, Reload 三个阶段.

1）Evict 阶段. 发送者使用预先选定的驱逐集将载体组从缓存中驱逐.

2）Wait 阶段. 等待瞬态指令执行将秘密编码进缓存隐蔽信道.

3）Reload 阶段. 接收者访问目标载体项并记录访问的时间 t_i.

根据秘密载体项判断，观测 Reload 阶段载体项 i 的访问时间 t_i，若 t_i 小于平均访问时间 *threshold*，则载体项 i 为秘密载体项.

3.2 数据重用型信道

缓存争用型信道的一个重要特点就是依赖于驱逐集，驱逐集生成依赖于对被攻击系统缓存替换策略及硬件结构的了解，不够轻量级. 数据重用型信道可解决这一问题. 本节介绍 2 种数据重用型隐蔽信道.

3.2.1 Flush+Reload

Gullasch 等人[11] 提出了 Flush+Reload 信道的基本思想，随后 Yarom 等人[8] 将其引入到各个核心共享的 L3 缓存中，并将信道粒度精确到缓存行. 如图 14 所示，该信道包含 Flush, Wait, Reload 三个阶段.

1）Flush 阶段. 发送方使用 clflush 指令清空缓存集中各缓存行，确保载体组不在缓存中.

2）Wait 阶段. 等待瞬态指令将秘密编码进缓存隐蔽信道.

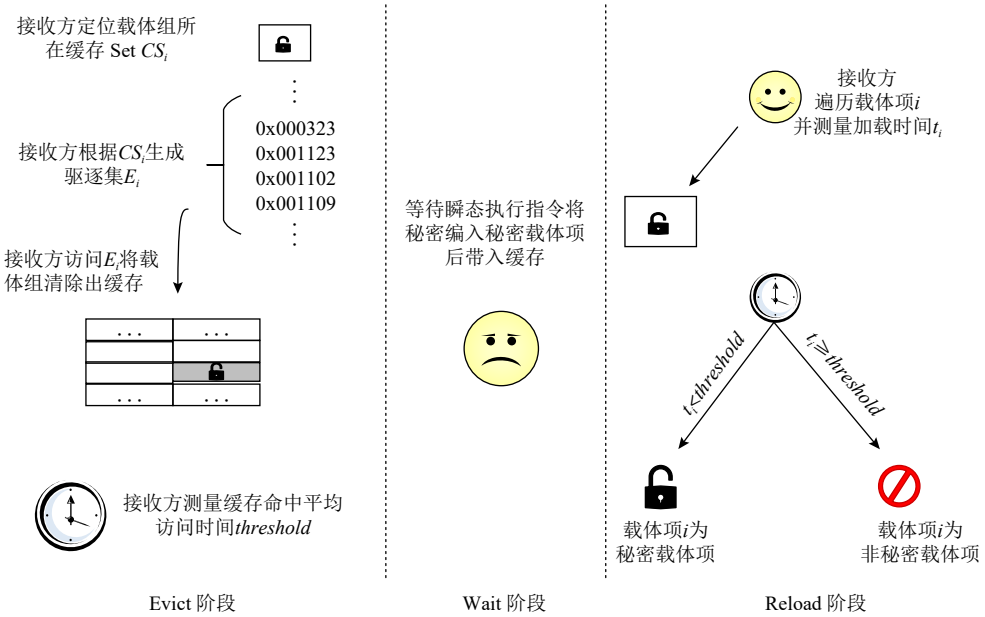


Fig. 13 Basic steps of Evict+Reload covert channel

图 13 Evict+Reload 隐蔽信道基本步骤

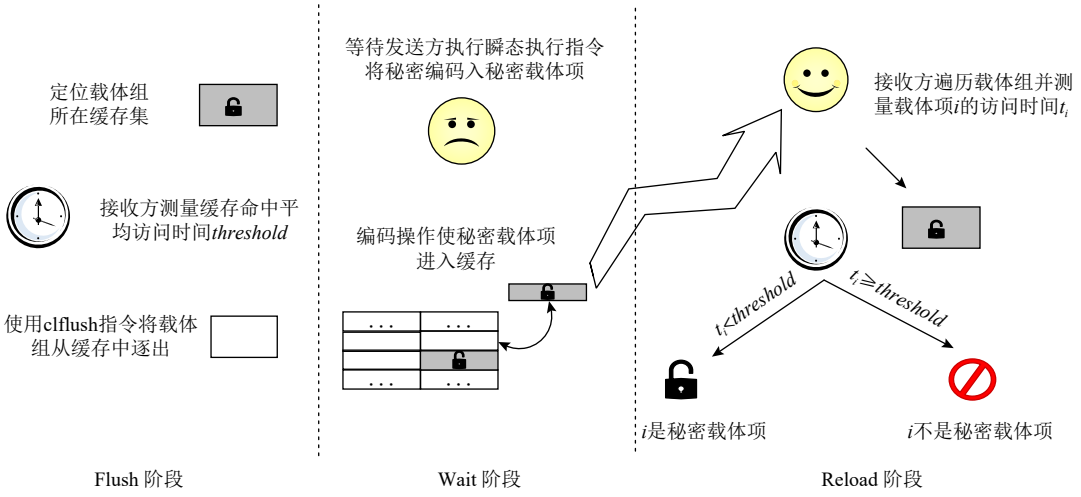


Fig. 14 Basic steps of Flush+Reload covert channel

图 14 Flush+Reload 隐蔽信道基本步骤

3)Reload 阶段. 接收者遍历载体组, 记录载体组中每个载体项 i 的访问时间 t_i.

根据秘密载体项判断, 访问时间 t_i 明显小于平均访问时间 *threshold* 的载体项 i 为秘密载体项.

3.2.2 Flush+Flush

Gruss 等人[14] 观察到, 缓存行存在数据时, clflush 指令的执行时间较长; 而当缓存行为空时, clflush 指令执行时间更短. 他们利用 clflush 指令的这种时间特性, 将 Flush+Reload 的后阶段 Reload 操作改为 Flush 操作, 提出了 Flush+Flush 信道. 如图 15 所示, 该信道包含 Flush, Wait, Flush 三个阶段.

1)Flush 阶段. 攻击者首先清空目标缓存行, 确保载体组不在缓存中.

2)Wait 阶段. 等待瞬态指令将秘密编码进入缓存隐蔽信道.

3)Flush 阶段. 再次执行 clflush 指令遍历刷新载体组, 并测量刷新指令执行时间 t_i. 记录每个载体组的刷新时间, 推断目标数据是否在缓存中.

根据秘密载体项判断, 若 t_i 明显高于其他载体项刷新时间 t, 说明载体项 i 为秘密载体项.

3.3 非缓存型隐蔽信道

3.1 节和 3.2 节总结了缓存型隐蔽信道, 随着从中断缓存隐蔽信道角度出发的防御研究出现, 也有少量使用非缓存型隐蔽信道的 TEA 研究, 本节将总

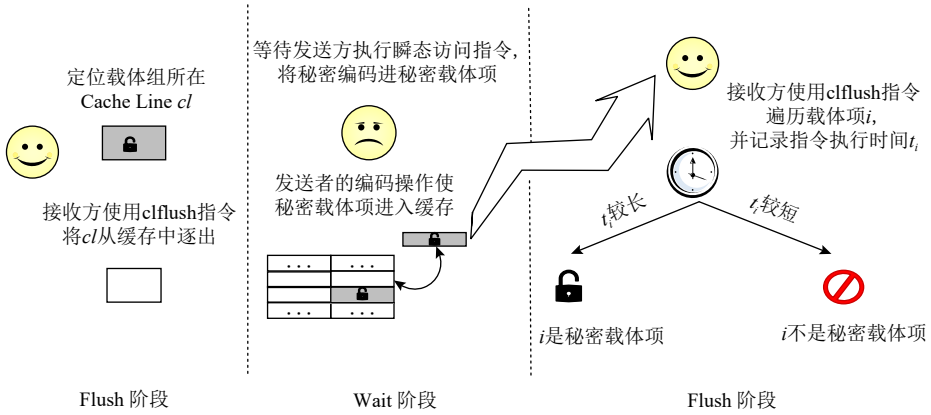


Fig. 15 Basic steps of Flush+Flush covert channel

图 15 Flush+Flush 隐蔽信道基本步骤

结这部分工作.

3.3.1 端口争用隐蔽信道

CPU 的 1 组执行单元共享同一端口(port), 对同一端口 EU 的争用会导致 μOP 的执行序列发生变化, 从而使 μOP 的整体执行时间产生差异. 2019 年 Bhattacharyya 等人[4] 基于端口争用造成的时间差异, 提出端口争用(port contention)信道, 通过记录受害者程序运行时间的差异, 推断指令执行序列, 进而推断敏感值. 该信道可分为预处理和测量 2 个阶段.

1)预处理阶段. 找到会发生端口争用的秘密相关代码序列 *S*.

2)测量阶段. 等待程序执行, 记录 *S* 执行时间 *t*.

根据秘密载体项判断, 若执行时间 *t* 更长则发生端口争用, 秘密为 1, 否则秘密为 0, 反之也成立.

3.3.2 除法器隐蔽信道

除法器是一种非流水线组件, 即该组件不能同时处理多条除法指令. 这使得攻击者可以根据除法指令执行时间推断敏感值. 2020 年 Fustos 等人[5] 基于除法器的这一特性成功窃取密钥. 该信道包含测量、等待、再测量 3 个阶段.

1)测量阶段. 记录除法指令执行前时间戳.

2)等待阶段. 触发瞬态执行访问秘密, 并将秘密编码为具有一系列除法操作分支的分支条件.

3)再测量阶段. 测量所有指令执行完毕时的时间戳.

根据秘密载体项判断, 较短的执行时间 *t* 说明秘密为 0, 否则秘密为 1.

3.3.3 AVX 隐蔽信道

SIMD 是一种并行架构类型, 能并行处理同一指令流的不同数据流. 高级向量拓展[70](advanced vector extensions, AVX)是英特尔公司提出的 SIMD 拓展指令集. 为节约能耗, AVX 单元在不使用时会断电关闭, 执行 AVX 指令前, AVX 单元首先要上电, 此时指令会停滞执行, 该延迟在通电后会大大降低, 因此执行前几条 AVX 指令时会引入时间延迟. 基于 AVX 单元的特点, Schwarz 等人[21] 成功完成对密钥的逐比特推断. 该信道包含监测和推断 2 个阶段.

1)监测阶段. 记录指令的执行时间.

2)推断阶段. 具有较长执行时间的指令为秘密相关指令.

根据秘密载体项判断, 较长的执行时间 *t* 说明秘密为 0, 否则秘密为 1.

3.3.4 EFLAGS 隐蔽信道

EFLAGS 寄存器是一种保存与处理器状态相关的 CPU 寄存器, JCC 指令允许根据 EFLAGS 寄存器的内容进行条件跳转. 2023 年, Jin 等人[45] 根据条件语句的执行时间推断 EFLAGS 寄存器中值, 例如, 中值为 0 时比较时间更短, 中值为 1 时比较时间更长. 该信道包含触发和测量 2 个阶段.

1)触发阶段. 触发瞬态执行并通过 EFLAGS 寄存器编码秘密数据.

2)测量阶段. 测量 JCC 指令的执行时间来推断数据.

根据秘密载体项判断, 较短的执行时间 *t* 说明秘密为 0, 否则秘密为 1, 反之也成立.

3.3.5 PMU 隐蔽信道

性能监控单元(performance monitor unit, PMU)是 CPU 中的重要硬件模块. PMU 会记录成功提交的指令触发的处理器事件, 理论上瞬态指令结果不会被 PMU 记录. 2023 年, Qiu 等人[6] 发现, 一些由瞬态指令触发的事件会被 PMU 记录, 且记录的值与密钥相关, 由此可通过记录 PMU 中某些事件值的变化推

断秘密. 该信道包含预处理、读初值、触发、读现值4个阶段.

1)预处理阶段. 构造依赖数值 v_0 并会触发 PMU 事件 w 的 Gadget, 即该 Gadget 确保当值等于 v_0 时会触发事件 w.

1)读初值阶段. 记录 PMU 中事件 w 的初始值 v_1.

2)触发阶段. 触发瞬态执行访问秘密.

3)读现值阶段. 记录 PMU 中事件 w 的当前值 v_2.

根据秘密载体项判断, 若 v_2 更新了且在设定范围内, 则秘密为 v_0, 否则秘密为其他值.

3.3.6 电磁隐蔽信道

不同于前面利用 CPU 微架构组件的非缓存型隐蔽信道, De Meulemeester 等人[7]在 USENIX Security 2023 会议上提出将电磁侧信道(electromagnetic side channel)应用于 TEA. 他们使用外部设备记录程序执行过程中泄露的频率迹, 并与采样的已知数值的频率迹做匹配, 通过比对推断秘密的具体数值. 该信道可分为预处理和记录2个阶段.

1)预处理阶段. 遍历值 i 并记录对应的频率迹 f_i, 所有 i 与 f_i 构成集合 S.

2)记录阶段. 测量执行过程中的频率迹 f_{exe}.

根据秘密载体项判断, 遍历 S, 若 $f_{exe}=f_i$ 则秘密为 i.

信道特征决定了当前 TEA 研究中隐蔽信道的选用及分布特性, 本文将在第5节对其进行分析与讨论.

4 TEA 研究进展

本节介绍 TEA 的相关进展. 按照实现原理, 本文将 TEA 分为3类: 一是由乱序执行驱动的熔断型攻击; 二是由错误分支预测驱动的幽灵型攻击; 三是由错误数据预测驱动的数据采样攻击. 首先, 对这3类攻击进行综合梳理. 然后, 结合抽象模型, 对比它们在各阶段的异同.

4.1 乱序执行驱动的熔断型攻击

本节将介绍熔断型攻击在内核空间数据读取、CPU 寄存器数据读取和可信执行环境数据读取3个方面的应用.

4.1.1 内核空间数据读取

OS 将虚拟地址空间分为用户空间和内核空间, 用户进程不具有对内核空间的访问权限. 本节将介绍可突破地址空间隔离并窃取内核数据的熔断型攻击.

1)Meltdown

Lipp 等人[1]于 2018 年揭露的 Meltdown 攻击是熔断型攻击的开创性研究工作, 后续的熔断型攻击多由其演变而来. 本文将结合下述 PoC, 详细介绍其基本思路.

① i = 0x80197701; //内核空间地址

② *check*(i); //越界检查

③ *exception* ( ); //检查不通过时执行

④ $y = a[(*i)\times j + k]$; //基于 i 计算

其中, i 为指针变量, $(*i)$ 为地址 i 中的数值, 变量 j 为缓存行大小, 变量 k 为小于 j 的任意非负值, a 为攻击者可访问的数组, 此处作为载体组.

顺序执行时, 该段代码执行顺序为 [1, 2, 3, 4]. 行②的越界检查保证了该程序不能访问 OS 为其分配内存空间之外的地址空间, 此时该程序并不会造成数据泄露.

乱序执行时, 上述各行代码被同时发射至 RS. 行①赋值完毕后, 行④指令通过片内数据总线监听到 i 已准备就绪, RS 立即将其调度至 EU 执行, 即计算 $(*i)*j + k$ 结果 r, 并将数组元素 $a[r]$ 从内存读取至缓存. 行④指令执行后, 结果 y 被暂存, 等待行②越界检查完毕后再按原序提交. 由于 i 是内核空间地址, 用户进程无法直接访问, 因此会引发页面错误(page fault, PF)异常, 记为#PF, 微架构刷新流水线并清空暂存的结果 y 和 r.

然而, 秘密已转化为秘密载体项 $a[(*i)\times j + k]$ 存储在缓存中. 最后使用第3节介绍的信道分析方法推断值 $*i$, 通过将内核空间地址遍历赋值给 i, Meltdown 可以完成对已在缓存中的内核数据窃取.

2)Meltdown V3c/V3r/V3z

Meltdown 攻击在执行过程中会产生#PF. 因此易被内核感知和检测. 实际攻击时, 可以使用英特尔事务同步扩展[71](Intel transactional synchronization extensions, Intel TSX)来拦截异常, 但并非所有处理器都支持 Intel TSX, 使 Meltdown 缺乏普适性.

作为改进, 百度安全实验室的程越强等人[72]利用错误分支预测不会产生异常的特性, 将 Meltdown Gadget 置于 Spectre V1 或 V2 的分支路径上以绕过#PF. 这种“Meltdown+V1”或“Meltdown+V2”的组合方式被命名为 Meltdown V3c. 然而, 无论是 Meltdown 还是 V3c 攻击都需满足2个条件:

①保证目标地址已在页表中建立映射.

②内容已经被加载到 L1D 缓存.

这意味着上述2种攻击无法窃取不位于 L1D 缓存的内核数据, 数据窃取范围受限. 作为改进, 程越强等人[72]又提出 Meltdown V3r 攻击, V3r 通过系统调

用触发特殊预测执行事件，把目标地址上的数据加载到 L1D 缓存，然后发起 Meltdown 或 V3c 攻击读取 L1D 中的数据.

但是 V3/V3c/V3r 均可被强制内核页表隔离[73] (kernel page table isolation, KPTI)防御，它们的改进型攻击 Meltdown V3z[74] 可以绕过 KPTI. V3z 攻击的思路可描述为“V3r+Meltdown”，即首先发动 V3r 攻击获取目标地址 *a*，然后在瞬态访问阶段读取**a*，并将其编码为秘密载体项.

3)SWAPGS

全局段寄存器(global segment, GS)存储着各个段的物理基址. GS 在用户进程和内核进程间共享，且进程切换时不会被清除[75]，因此，GS 同时拥有用户和内核数据，Lutas 等人[76] 发现，利用切换 GS 值的特权指令 SWAPGS，可以窃取内核空间数据. 本文结合下述 PoC，介绍 SWAPGS 攻击基本思路.

① test

② je malicious

③ SWAPGS;

④ mov r10, qword ptr gs:[1 000 h]

⑤ mov rbx,[r10]

⑥ …

⑦ malicious:

⑧ mov rcx, qword ptr gs:[2 000 h]

⑨ mov rbx, [rcx]

若 GS 的条目中保存用户空间值，并且预测为采用跳转，则预测执行路径将会跳转到行⑦执行，此时的内存读取将以 GS 中的用户空间值为基址，若该基址为恶意程序指定的地址，即可以完成对内核空间数据的读取. 另一种情况下，若 GS 保存着内核值，并预测行①不采用，此时会执行 SWAPGS 指令，切换到用户空间值 GS，也可能导致用户提供的恶意地址加载内核数据.

4.1.2 寄存器数据读取

4.1.1 节介绍了突破地址空间隔离的熔断型攻击. 但它们主要窃取缓存中的数据，CPU 从缓存中读取的数据被暂存至寄存器以待计算，本节总结从寄存器窃取数据的熔断型攻击.

1)LazyFP

进程切换时，OS 需要保存当前进程存储在寄存器中的数据，包括浮点运算单元(floating point unit, FPU). 但 FPU 拥有较多位数，相比于 16 b 数据寄存器，保存 FPU 需耗费更多的时钟周期，影响进程切换速度. 考虑到并非每个进程都会使用 FPU，OS 在进程切换时会简单地将 FPU 标记为“不可用”(device not available, NA). 当其他进程需要使用 FPU 时，会触发设备不可用异常，记为#NA. 在#NA 异常处理程序中，OS 会保存 FPU 中的数据，然后再将其交付给请求使用 FPU 的进程.

LazyFP[22] 利用 FPU 延迟保存数据的特性来完成攻击. LazyFP 等待受害者将数据加载至 FPU，然后迅速切换至攻击者进程. 尽管此时 FPU 已被标记为不可用，但数据仍存储在其中. 攻击者利用 OoOE，在#NA 异常处理前，越权读取 FPU 中的秘密数据，将其编码至缓存隐蔽信道后还原.

2)Spectre V3a

Spectre V3a[77] 是 Meltdown 的变种(Meltdown 别名 Spectre V3). V3a 攻击者可窃取特权寄存器中的数据. 用户进程访问特权寄存器会引发一般性保护(general protection, GP)错误，记为#GP. 同时唤起#GP 异常处理程序，V3a 攻击者可使用 Intel TSX 等异常抑制措施延缓异常捕获. 在#GP 异常处理完毕前读取秘密，然后使用缓存隐蔽信道传递还原.

4.1.3 可信执行环境数据读取

4.1.1~4.1.2 节介绍的熔断型攻击暴露了 OS 运行时安全问题. 为了保护程序运行时数据的机密性和完整性，安全研究人员设计了独立于 OS 的可信执行环境，但瞬态攻击研究[17-20] 表明可信执行环境的安全性也是有限的. 本节介绍能从可信执行环境窃取数据的熔断型攻击.

1)Foreshadow

Intel SGX 是 Intel 公司推出的可信执行环境技术[78]. 它提供一个名为“飞地”(enclave)的安全隔离区来保障系统关键代码和隐私数据的安全. 无论软件有何种权限均无法直接访问飞地，存储在内存中的飞地数据也会被加密保护. 原始 Meltdown 攻击越界读取飞地数据时，会触发 SGX 的“页面终止语义”，即在越界检查完成前，任何对飞地内的数据读取操作均视为非法，且只能读到值为−1 的假值(0xFF).

作为改进，van Bulck 等人[17] 提出了一种针对 SGX 的熔断型攻击 Foreshadow. 该攻击使用系统调用 mprotect 清除飞地页表项的“present”位，引发#PF 异常，异常时页面中止语义不会被启用. 由于地址转换机构与缓存之间存在紧密的联系，飞地页表项中标记的物理页面帧号可快速传递到缓存. 攻击者可基于硬件传递的帧号快速计算出目标物理地址，然后使用 Meltdown 读取其值，该攻击可以成功提取飞地中所有数据. 同时，还实现了 SGX 的加密密钥、本地认证

密钥和远程认证密钥三大密钥提取, 这意味着加密存储在飞地外的数据以及远程飞地中的数据也可被窃取, 击破了 SGX 声称的安全保障.

2) Foreshadow-NG

Foreshadow 的攻击范围限制在单 OS 环境中, 作为改进, Weisse 等人[20]将 Foreshadow 的攻击能力拓展到 OS 和虚拟机数据窃取, 进一步拓展了 Foreshadow 的威胁性.

页表从内存移至磁盘时, OS 会清除页表项的“present”位. 当页表项被标记为已清除后, OS 可以自由设置除“present”位之外的其他页表项数据. 此时 Foreshadow-OS 攻击者可以使用瞬态指令读取物理地址对应的页表项中的已缓存的内容, 从而窃取 OS 数据.

Foreshadow-VMM 攻击者控制恶意客户虚拟机的首个地址映射, 直接触发终端错误并绕过主机地址转换步骤, 将客户物理地址直接传递给 L1 缓存. 这使得 Foreshadow-VMM 能够读取整个缓存的内容, 包括其他虚拟机、宿主机以及虚拟机管理程序的数据.

4.1.4 小 结

本文结合第 2 节提出的 TEA 抽象模型, 总结并对比了熔断型攻击的差异和共同之处, 如图 16 所示. 可见, 差异主要体现在预处理和瞬态访问阶段.

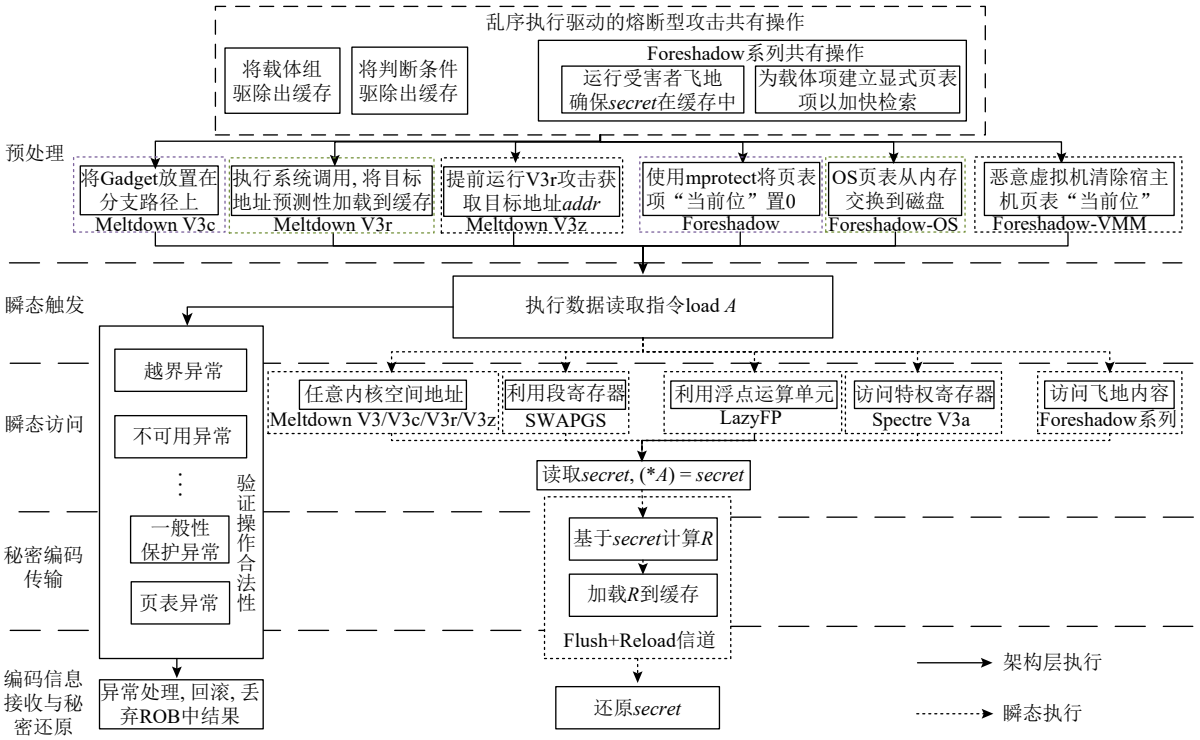


Fig. 16 Phased comparison of meltdown type attacks driven by out-of-order execution

图 16 乱序执行驱动的熔断型攻击分阶段比较

Meltdown[1], V3c/V3r[72]/V3z[74], Foreshadow[17]/Foreshadow-NG[20] 的差异主要集中在预处理阶段. Meltdown 会引发#PF, 而 V3c 将 Gadget 放置在分支路径上以避免引发#PF, V3r 和 V3z 主动执行系统调用, 将目标地址数据加载到缓存, 弥补了 Meltdown 和 V3c 只能泄露存在于缓存中的内核数据的缺陷. 而 Foreshadow-OS 和 Foreshadow-VMM 变种分别清空 OS 页表和宿主机页表的“present”位, 利用 TLB 和缓存的紧密联系计算物理地址, 隐蔽地窃取 OS 和同一物理机上其余虚拟机的隐私数据.

LazyFP[22], SWAPGS[75], Spectre V3a[77] 攻击间的差异主要集中在瞬态访问阶段. LazyFP 观察到 FPU 的延迟加载, SWAPGS 利用段寄存器在用户与内核之间共享, Spectre V3a 利用在#GP 异常处理完毕前的瞬态执行窗口读取特权寄存器值.

4.2 错误分支预测驱动的幽灵型攻击

分支预测器结构复杂, 包含条件分支预测、间接分支预测和返回跳转预测等多类预测组件. 图 17(a) 展示了分支预测器的基本架构, 其主要由 4 种部件组成: 分支历史缓冲区(branch history buffer, BHB)、模式历史表(pattern history table, PHT)、分支目标缓冲区(branch target buffer, BTB), 返回堆栈缓冲区(return

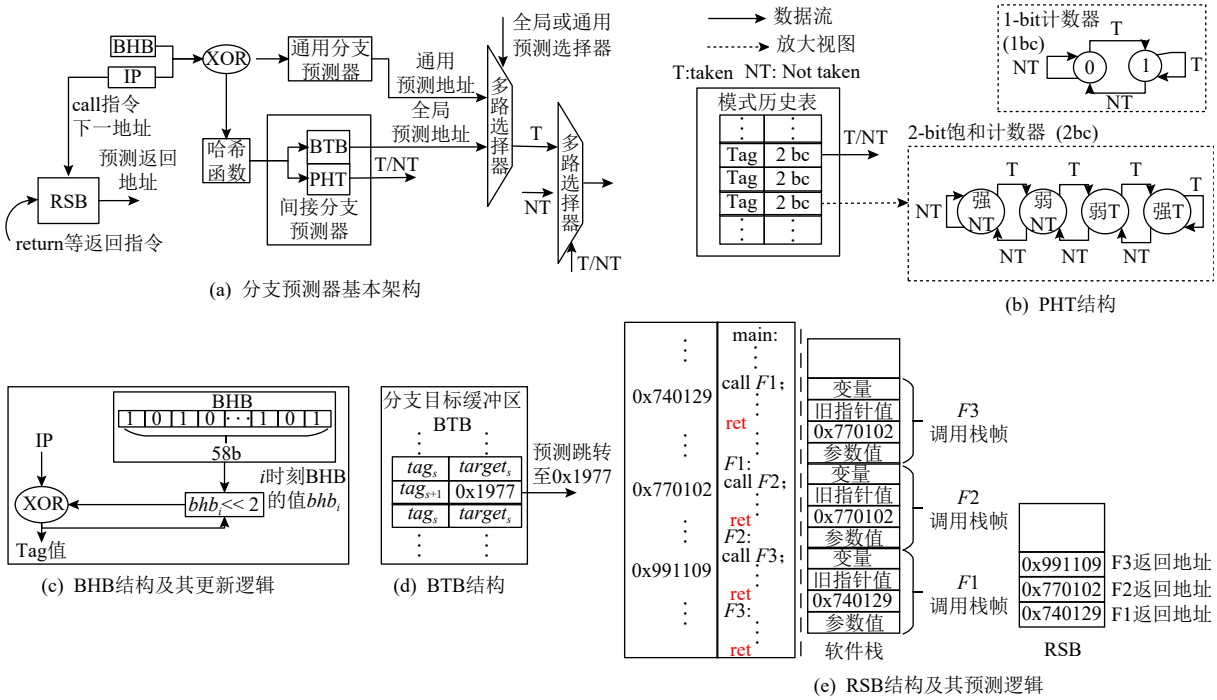


Fig. 17 Branch prediction mechanism hardware composition legend

图 17 分支预测机制硬件组成图例

stack buffer, RSB), 这四大部件也是幽灵型攻击的主要来源. 本节将结合分支预测器的硬件设计, 介绍 3 类幽灵型(spectre-type)攻击.

4.2.1 基于条件分支预测

条件分支预测依赖于 PHT. 最简单的条件分支预测就是随机猜测当前分支跳转或不跳转, 但此时仅有 50% 的准确率, 无法满足现代处理器的要求.

因此, 工程师开始设计更优秀的预测机制. 图 17(b)右上方是一个 1 位计数器(1-bit counter, 1bc)的逻辑图, 1bc 可作为简单的预测单元, “1”记为跳转, “0”记为不跳转, 根据上次该分支跳转情况更新 1bc, 即本次预测是否跳转与上次实际跳转情况保持一致. 这种设计初步体现了“过去决定未来”的思想. 但在分支切换频繁的情况下, 1bc 的预测效果较差.

作为改进, 引入了 2 位饱和计数器(2-bit saturation counter, 2 bc). 如图 17(b)右下方所示, 2 bc 有 4 种状态: 强跳转、弱跳转、强不跳转、弱不跳转. 增设状态位使 2 bc 能更好地应对分支频繁转换[79], 因此被用于简单分支预测. 如图 17(b)左侧所示, PHT 的每个表项由 Tag 和 2 bc 组成, 其中 Tag 是指令地址的哈希值. PHT 是基于条件分支预测的幽灵型攻击来源, 本节将总结相关工作.

1) Spectre V1

Kocher 等人[2] 提出的 Spectre 攻击是幽灵型攻击的开创性研究工作. 文献 [2] 初步描述了 2 类 Spectre 攻击, 其中 Spectre V1 利用 PHT 发动攻击. 本文结合下述 PoC, 描述其基本实现思路.

① *mis_pht*( );

② *flush* (*a*[]);

③ *flush* (*n*);

④ if (*x*<*n*){//B1

⑤ *y*=[*x*]×*j* + *k*;

⑥ *temp* &= *a*[*y*];

⑦ }//B2

函数 *flush*(*i*) 可逐出参数 *i* 所在的缓存行, *n* 为载体组 *a* 的载体项数量, 其他变量与 Meltdown 的 PoC 描述相同. 行①函数 *mis_pht*() 循环使用小于 *n* 的 *x* 值, 反复执行行④~⑦的 if 语句. 执行次数达到设定阈值后, PHT 相应表项标记跳转分支 B1. 行②③将 *a* 和 *n* 逐出缓存, 使它们从内存重载, 以增大 SEW.

发起攻击时, 攻击者赋给 *x* 一个大于 *n* 的值, 按照程序语义, 此时 if 语句应转向分支 B2 执行. 然而, 在行④分支条件解析之前, CPU 根据 PHT 的标记, 预测性执行分支 B1, 导致攻击者越界读取数据 [*x*]. 通过行⑤的计算, 秘密 [*x*] 被行⑥代码编码在秘密载体项 *a*[*y*] 中. 即使清退阶段会撤销⑤⑥行 *y* 和 *temp* 的赋值操作, 但并不会改变 *a*[*y*] 对缓存状态的影响. 最后攻击者可以使用 Flush+Reload 信道将 [*x*] 从秘密载

体项中还原. SgxSpectre[80] 是 V1 在 SGX 场景的应用.

Spectre V1 和 SgxSpectre 的限制是为了读取指定地址的数据, x 需处于攻击者控制之下. 程序中满足这种模式的 Gadget 较少. SpecHammer[59] 攻击提出将 V1 与比特翻转攻击[81] 相结合, 在存储器中修改 x 的比特位, 消除了 x 必须处于攻击者掌控下的限制.

2) Spectre V1.1 和 Spectre V1.2

Spectre V1.1 和 Spectre V1.2[82-83] 是 Spectre V1 的 2 类变种攻击, Spectre V1.1 又被称为存储时边界绕过 Spectre-BCBS(Spectre bounds check bypass store), 可向受害者地址空间中写入数据. Spectre V1.2 又被称为只读保护绕过 Spectre-ROPB(Spectre read only protection bypass), 它能够绕过只读存储器的强制写保护修改数据. 本节结合下述 PoC, 介绍 Spectre V1.1 和 Spectre V1.2 攻击的基本思路.

① *mis_pht*( );
② *flush* (*n*);
③ if (*x*<*n*){
④ *b*[*x*] = *z*; //V1.1
⑤ …
⑥ *c*[*x*] = *z*; //V1.2
⑦ }

其中, b 为受害者地址空间基址, c 为只读存储区地址基址, z 为攻击者期望写入的值, 其他变量或函数与 V1 的 PoC 描述相同. Spectre V1.1 和 Spectre V1.2 攻击的基本步骤与 Spectre V1 相同, 不同之处在于条件分支内代码.

①Spectre V1.1. OS 为了确保不同进程间的数据安全性, 地址空间被严格隔离, 防止进程修改彼此数据. 然而, Spectre V1.1 攻击掌握受害者进程基址 b 和偏移量 x, 利用瞬态执行过程中#PF 异常延后处理的特性绕过隔离, 将地址 $b+x$ 的数据修改为 z.

②Spectre V1.2. 只读存储区存储着关键数据结构, 例如程序的静态代码和代码指针等. 若能篡改代码指针, 就可以操控程序执行路径转向恶意代码. 为避免这种情况, OS 对只读存储器实施了强制写保护, 禁止对该区域的写操作, 否则将触发写保护(write protection, WP)异常, 记为#WP. Spectre V1.2 攻击者掌握只读存储区基址 c 和偏移量 x, 利用瞬态执行中#WP 异常延后处理, 绕过对只读存储区的强制保护, 通过修改 $c+x$ 处的值, 攻击者可以控制程序执行流.

3) NetSpectre

Spectre V1 系列攻击只能窃取存储在本地机器上的秘密信息. 作为改进, Schwarz 等人[21] 提出了一种名为 NetSpectre 的攻击变种, 其能通过网络发起远程攻击. NetSpectre 首先向受害者发送多个读取条件分支语句边界内合法数据的网络请求以毒害 PHT. 然后, 发送含有读取非法地址值的网络请求, 非法地址中的数据在错误预测中被编码至缓存. 需要指出的是, 由于远程攻击者不能在受害者机器上直接执行 clflush 指令, 因此 NetSpectre 并不适用 Flush+Reload 信道, 同时 Flush+Reload 信道粒度过细, 也不适用远程攻击.

NetSpectre 是使用 Evict+Reload 信道[15] 的一种粗粒度变体, 通过破坏缓存, 驱逐整个 LLC 构造明显的时间差异, 这种差异将间接反馈在网络请求响应时间中. 因此, 攻击者通过最后一步测量相应内存位置的网络请求响应时间判定秘密载体项. 但它的限制也是明显的: 对网络时延较为敏感. 因此, 攻击者需要通过大量的网络请求来排除非攻击网络时延带来的噪声干扰. NetSpectre 还能通过非缓存型的 AVX 信道发起攻击, 以规避针对缓存的防御措施.

4.2.2 基于间接分支预测

PHT 在条件分支预测中表现出较高的准确率, 但在预测间接分支跳转时表现不好. 因此, CPU 使用由图 17(c)(d)组成的间接分支预测器来预测间接分支跳转, 其中, BHB 是一个 58 b 寄存器, 能够存储 29 次间接跳转的函数调用模式. 每次更新时, BHB 从 IP 中按照某种模式取 2 b 存入, 例如, 取 IP 的第 2 位与倒数第 1 位. BTB 组织方式如图 17(d)所示, 其表项存储了已执行指令地址的标签(Tag)以及上一次分支跳转的目标地址(Target). 其中, Tag 是经过 IP 与 BHB 复杂异或运算后得到的哈希值, 预测间接分支的跳转方向时, 根据指令 IP 快速检索 BTB 中相应的 Tag 字段, 若匹配, 则跳转至 Target 字段的地址继续执行. 这种设计综合了全局历史跳转情况, 使 BTB 在间接分支预测方面表现更为优异[84].

1) Spectre V2

除了基于 PHT 的 Spectre V1 攻击, Kocher 等人[2] 还提出了针对 BTB 的攻击变种 Spectre V2. 他们对 BTB 的逆向工程发现, 哈希函数仅使用 IP 的低位地址生成 Tag 字段. 这意味着, 若攻击者 A 提供与受害者 V 函数低位相同的 IP, 可能会生成相同的 Tag, 进而导致 BTB 表项冲突, 使 Target 被替换为 A 指定的地址.

本文结合图 18, 介绍 Spectre V2 攻击的基本思路. 其中, 函数 N 为受害者 V 独占, 函数 *mis_btb* 处于攻击者 A 控制之下, M 为预处理阶段找到的 Gadget.

V 执行前, A 通过执行 *mis_btb*, 反复调用起始地

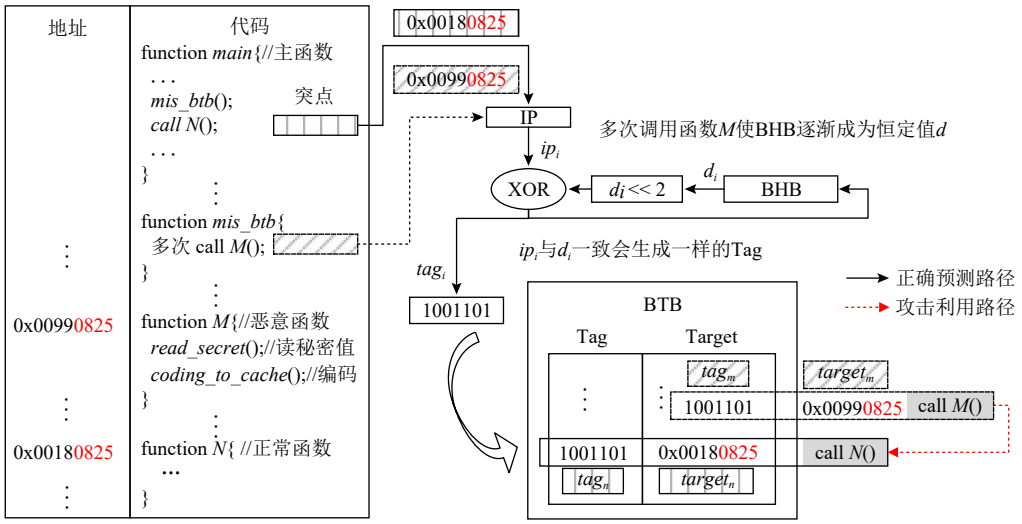


Fig. 18 Illustration of Specter V2 attacking PoC

图 18 Spectre V2 攻击 PoC 示意图

址为 0x990825 的函数 *M*, 这将逐步使 BHB 趋向于某一固定数值 *d*. 正常情况下, *V* 执行 call *N* 会跳转至地址 0x180825 执行, 但由于函数 *N* 的指令指针(instruction pointer, IP)(记为 IP_n)与函数 *M* 的 IP(记为 IP_m)的低位相同, 并且经过攻击者的恶意训练, BHB 的中值已为 *d*, 因此, 可能会造成 tag_n=tag_m. 此时, *V* 调用函数 *N* 时可能会错误地预测性跳转至 *A* 指定的恶意地址 0x990825 执行. 在正确跳转地址解析之前, *V* 会执行 *M* 中的恶意代码, 并通过数据访问将 *A* 期望的数据加载到缓存中, 最终, *A* 使用 Flush+Reload 信道解码并还原秘密值.

2)Spectre BHI 和 Spectre BHI-native

除了直接对 BTB 进行毒害训练, 2022 年 Barberis 等人[84]还提出了一种通过 BHB 间接控制 BTB 的攻击方法. 根据实现特点, 该攻击被命名为“分支历史记录注入攻击”Spectre BHI(branch history injection). 该研究表明, BHB 在不同权限级别之间并不隔离, 攻击者可以在权限切换时的空窗期操纵分支历史记录. 由于 BHB 参与 BTB 表中 Tag 的生成, 所以攻击者可以间接控制 BTB, 从而使受害者程序在间接分支预测时选用特定的 BTB 表项.

然而, BHI 依赖于能在用户权限下修改内核代码的拓展伯克利滤包器(extended Berkeley packet filter, eBPF), 以确保注入的目标地址包含 Gadget. 当禁止用户调用 eBPF 时, BHI 攻击将失效. 作为改进, 2024 年 Wiebing 等人[85]提出了一种不依赖于 eBPF 的攻击变种 BHI-native, 其通过搜索受害者代码中隐藏的 Gadget 完成分支历史注入.

3)SgxPectre

幽灵型攻击除了能在常规场景中进行, 同样也能突破可信执行环境的安全保障. Chen 等人[60]在 IEEE S&P 会议提出的 SgxPectre 是 Spectre V2 在 SGX 场景的变种. 除需与受害者运行在同一物理核心, 还要求攻击者具有提前分析飞地内运行代码和启动飞地的权限, 其 PoC 示例如图 19 所示.

首先, SgxPectre 使用飞地外的分支指令训练 BTB, 将其跳转目标地址设置为 Gadget 所在起始地址 0x002209. 接着, 攻击者驱逐分支条件的目标地址, 并清空 RSB, 使其回退使用 BTB 预测. 然后将寄存器中的值设置为攻击目标地址, 并通过 *EENTER* 调用执行飞地中的代码. 当飞地内 0x001109 地址处的 ret 执行时, 由于攻击者进程训练了 BTB 的跳转地址, 导致错误地跳转到 0x002209 处执行, 此时, 飞地内的秘密值 0x1113 将被 Gadget 编码至缓存, 攻击者可通过隐蔽信道还原秘密.

4)TTE-BTB

前述的攻击均在预处理阶段污染 BTB. 这种毒害方式易被基于模式检测的防御方式捕捉. 作为改进, 2023 年 Trujillo 等人[61]提出 TTE-BTB(training BTB in transient execution)攻击, 利用瞬态执行训练 BTB, 从而将对 BTB 的毒害切换到了瞬态触发与瞬态访问阶段, TTE-BTB 的 PoC 代码如下所示:

① function *TEE_btb* ( ){
② if (*a*){//B1
③ *mis_btb* ( );
④ }//B2
⑤ …
⑥ }

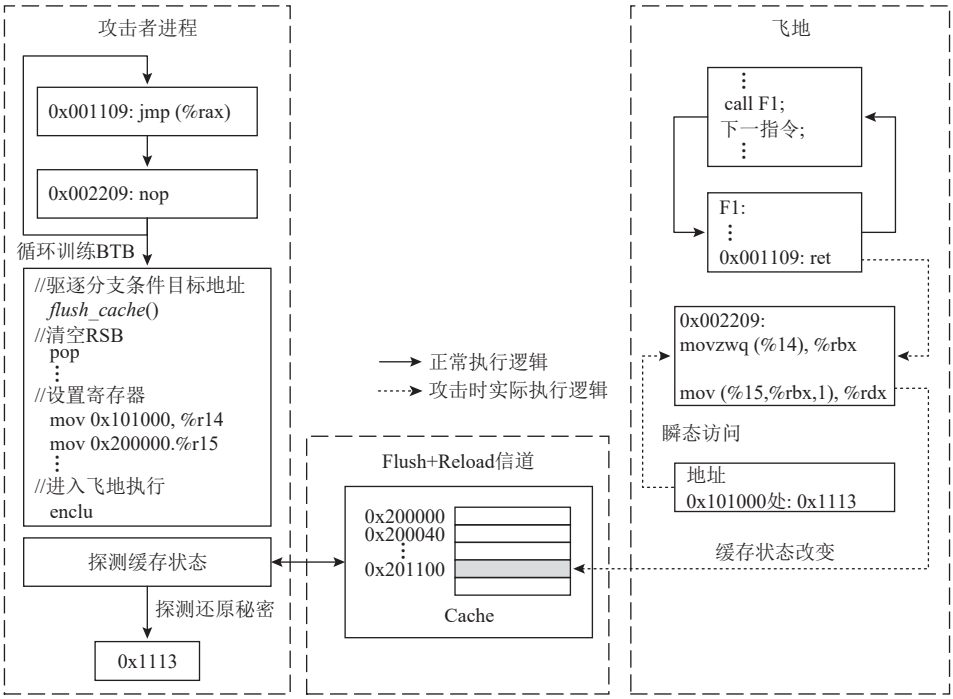


Fig. 19 Illustration of SgxPectre attacking PoC

图 19 SgxPectre 攻击 PoC 示意图

函数 *mis_btb* 的作用与 Spectre V2 PoC 代码介绍的相同, 不同的是它被一个 if 语句包裹. 攻击者首先训练 PHT 使其错误地将执行流转向 B1. 在分支内使用函数 *mis_btb* 来污染 BTB, 将 Target 字段覆写为含有非法读取敏感数据和缓存隐蔽信道编码操作的 Gadget 地址. CPU 检测到 if 分支的错误预测时, 行③指令被回滚, 但函数 *mis_btb* 对 BTB 状态的改变不会撤销.

TTE-BTB 在瞬态触发与访问阶段训练 BTB, 瞬态执行时的状态改变并不会被提交到架构层, 因此 TTE-BTB 可更为隐蔽地污染 BTB, 以绕过对预处理阶段污染 BTB 行为进行检测的防御措施.

4.2.3 基于返回分支预测

4.2.2 节介绍了利用 BTB 的幽灵型攻击. BTB 用于预测间接跳转指令的跳转目的地址, 与此相反, RSB 用于预测间接跳转执行完毕后的返回地址, 本节介绍利用 RSB 的幽灵型攻击.

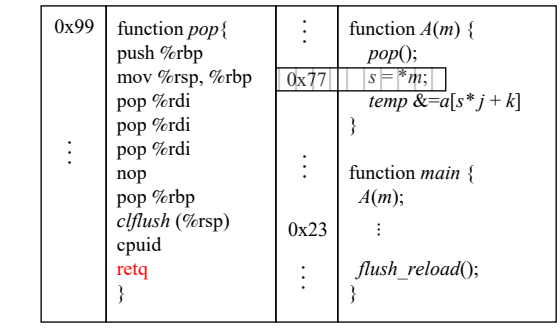
如图 17(e)所示. 左侧代码中调用的每个函数, 都会将调用处的下一条指令地址压入 RSB, 并同时在软件栈(software stack, SS)中保存当前函数的参数值等上下文数据. 函数 *F*3 执行返回指令 ret 时, 若 SS 栈顶栈帧中返回地址与 RSB 栈顶地址不匹配, CPU 需从内存重新获取正确返回地址. 在获取正确地址之前, 处理器会继续使用 RSB 中的返回地址预测性执行, 以避免流水线停滞, 因此程序跳转到 0x991109 继续执行.

1)Spectre RSB 和 ret2spec

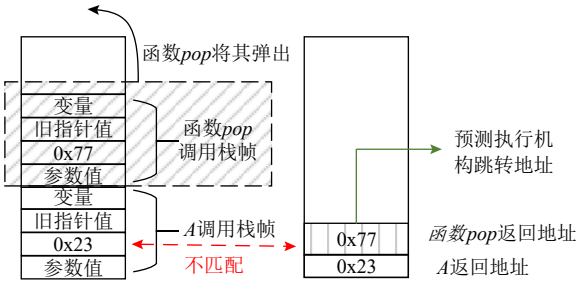
Koruyeh 等人[67] 和 Maisuradze 等人[62] 几乎同时发现了 RSB 的漏洞, 并分别提出了 Spectre RSB 和 ret2spec 攻击. 这 2 种攻击对 RSB 的利用原理基本相同, 本节以 Spectre RSB 为例介绍该类型攻击的基本原理和思路. 图 20(a)为 Spectre RSB 的 PoC 代码, 图 20(b)展示了攻击时 SS 和 RSB 的变化情况.

主函数 *main* 首先调用函数 *A*, 此时调用处的下一条指令地址 0x23 被同时保存到 RSB 和 SS. 随后, 程序切换到函数 *A* 执行, 而 *A* 中第 1 条语句是调用函数 *pop*. 与上一次函数调用类似, 首先, 下一条指令地址 0x77 会被保存至 RSB 和 SS, 然后切换至函数 *pop* 执行, 而函数 *pop* 的作用是移除 SS 栈顶栈帧, 如图 20(b)灰色区域所示. 函数 *pop* 执行完毕后, SS 栈顶栈帧指示函数返回地址为 0x23, 而 RSB 中指示的返回地址为 0x77. 由于返回地址不一致, CPU 需从内存读取正确返回地址. 为避免 CPU 停顿, 预测性跳转到 RSB 指示的地址 0x77 执行, 即执行语句 *s*=**m*, 若 *m* 指向内核地址, 则可完成对内核数据的窃取.

即时(just-in-time, JIT)编译机制允许将 JavaScript 代码在运行时编译为本机代码, 这种特性使得它可被用于嵌入恶意代码以窃取浏览器数据[63]. 为防御 JIT 可能存在的漏洞, 浏览器生产商提出了名为沙盒(sandbox)的软件隔离措施, 沙盒内运行的代码无法



(a) Spectre RSB 示例代码



(b) 攻击时软件栈和RSB变化示例

Fig. 20 Illustration of Specter RSB attacking PoC

图 20 Spectre RSB 攻击 PoC 示意图

访问外部数据，ret2spec 通过恶意训练 RSB 使其转向具有恶意行为的 Gadget 执行，在预测执行时绕过浏览器沙盒隔离，瞬态读取沙盒外浏览器内存数据.

2）SpecROP

Spectre RSB 与 ret2spec 要求在受害者代码中需要找到一段同时具有数据读取与缓存隐蔽信道编码功能的 Gadget，但在实际程序很少能找到满足这种要求的 Gadget. SpecROP[68] 攻击将 Spectre RSB 与面向返回的代码重用攻击[86-87] 相结合，使用通过毒害多个返回流指令而链接起来的小 Gadget 集合与单 Gadget 效果相同，Chowdhuryy 等人[66] 对 Intel Sky Lake 微架构处理器的逆向工程发现，经过 12 次返回跳转训练后，攻击者就可以控制下次返回跳转地址，进一步证明了 SpecROP 的可行性.

3）RetBleed

当程序执行的返回指令数量多于 RSB 表项时会发生 RSB 下溢（underflow），Wikner 等人[64] 发现某些 Intel 处理器中出现 RSB 下溢时，返回指令的返回地址预测更倾向于使用 BTB；而在 AMD 处理器中，无论 RSB 状态如何返回预测均类似于间接分支. 他们基于上述特性提出了 RetBleed 攻击.

不同于 Spectre V2 中循环调用某一函数来毒害 BTB，RetBleed 使用一系列返回指令来替换 BTB 的 Target 字段. 因此，RetBleed 首先扫描内核程序代码，并从中找到可利用的一系列由返回指令组成的代码片段，并完成 BTB 污染. 此时攻击者可以向 BTB 中注入内核地址空间内的分支目标地址. 随后，当 BTB 发生错误预测时，受害者进程会错误地转向攻击者注入的 Gadget 地址，以泄露内核数据.

RetBleed 使用返回流指令污染 BTB，因此能够绕过针对 V2 的某些防御措施. 尽管 Intel 和 AMD 为应对 RetBleed 发布了相应的防御补丁[88-89]，但 RetBleed 研究团队对补丁的实验评估表明，补丁会导致 14%~39% 不等的性能开销. 另外，在内核转换时刷新 BTB 的防御措施，会使得性能开销高达 209%[65].

4）TTE-RSB

4.2.2 中介绍了 TTE-BTB 攻击，Trujillo 等人[61] 还将这种攻击思路拓展到 RSB，即 TTE-RSB. 与 Spectre RSB 等在预处理阶段毒害 RSB 的攻击不同，TTE-RSB 在瞬态执行过程中训练 RSB 并注入攻击者期望的跳转地址，挟持程序执行流转向攻击者期望的 Gadget，TTE-RSB 的 PoC 代码如下所示：

① function *TEE_RSB* ( ){
② if (*a*){//B1
③ *mis_RSB* ( );
④ }
⑤ return;
⑥ }
⑦ …
⑧ function *M* ( );

设函数 *M* 含有非法读取敏感数据和缓存隐蔽信道编码指令，函数 *mis_RSB* 能将 RSB 栈顶替换为 *M* 地址，但它被一个 if 语句包裹.

攻击者首先训练 PHT 使其错误转向 B1 分支. 当处理器检测到分支跳转预测错误时，行③指令被回滚，但不会撤销其对 RSB 的表项的更改. 因此，行⑤函数返回时，并不会跳转到调用函数 *TEE_RSB* 的下一指令位置，而是跳转到行⑧函数 *M* 执行.

TTE-RSB 在瞬态触发与访问阶段训练 RSB，瞬态执行时的状态改变并不会被提交到架构层，因此 TTE-RSB 可更为隐蔽地污染 RSB，以绕过当前对预处理阶段污染 RSB 行为进行检测的防御措施.

4.2.4 小 结

本文在图 21 中总结了错误分支预测驱动的幽灵型攻击的共同之处，并对比了它们在执行过程不同阶段中的差异. 其中，这些攻击主要在预处理和瞬态触发存在差异.

在预处理阶段，这些攻击差异主要体现在对分

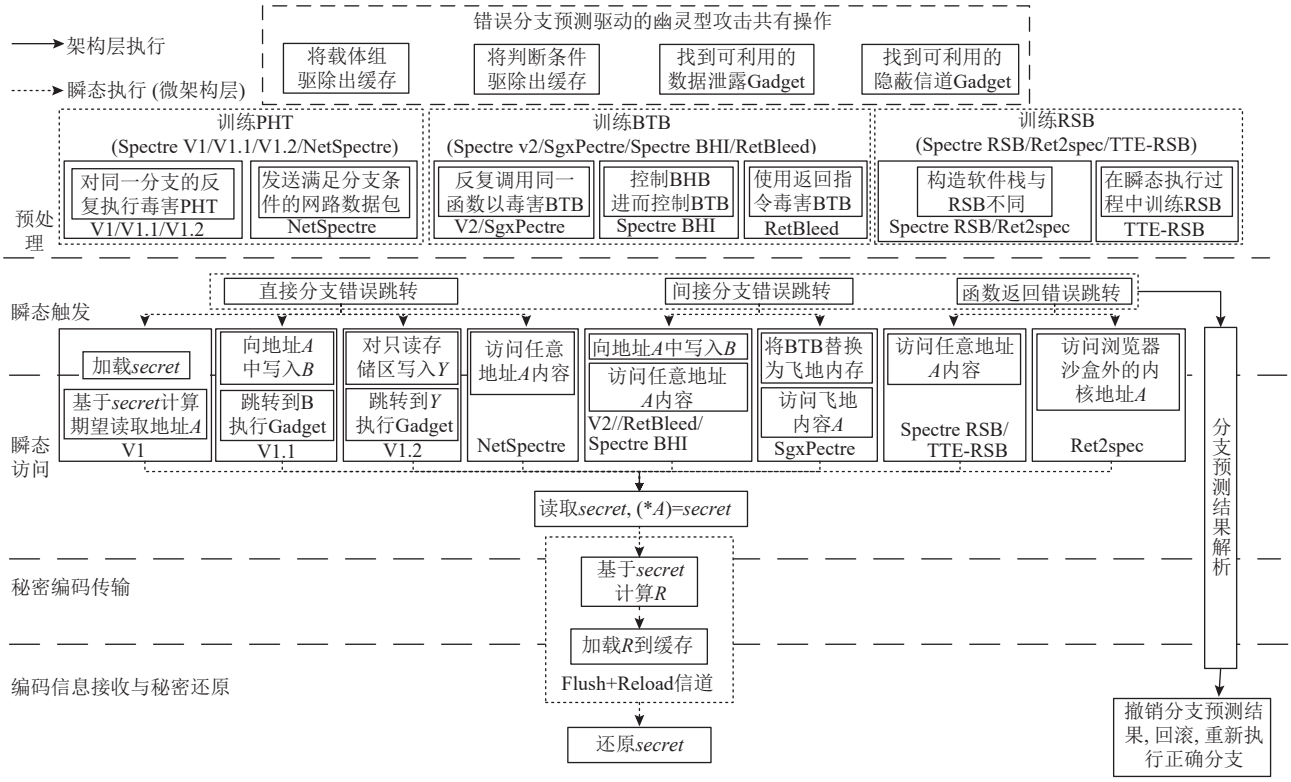


Fig. 21 Phased comparison of spectre-type attacks driven by incorrect branch prediction execution

图 21 错误分支预测执行驱动的幽灵型攻击分阶段比较

支预测机构的训练方式上. 例如, V1 系列攻击通过反复执行同一分支语句毒害 PHT, 而 NetSpectre 则通过重复发送满足分支条件内请求的数据包远程毒害 PHT. 相比之下, BTB 和 RSB 的毒害方式则更加多元化, 如 V2 和 SgxPectre 使用传统的循环调用方式来毒害 BTB; 而 Spectre BHI 和 RetBleed 则分别通过污染 BHB 和 RSB 来间接毒害 BTB; Spectre RSB 和 Ret2spec 通过构造 SS 和 RSB 不同来误导返回跳转; TTE-RSB 攻击则更为隐蔽, 嵌套一个条件分支的瞬态执行污染 RSB.

瞬态触发阶段的差异表现为不同的瞬态触发指令. 例如, 直接分支错误跳转的触发指令为条件分支语句, 而间接分支错误跳转的触发指令则为 call 等间接跳转指令.

4.3 错误数据预测驱动的数据采样攻击

现有研究对利用存储子系统中的存取缓冲区漏洞的 TEA 分类较为模糊. 因其实现类似于 OoOE, 部分研究[36]将其归类为熔断型攻击, 然而其利用错误数据预测和无法精确泄露所需数据的特点明显与熔断型攻击相异. 基于以上情况, 本文将结合了 OoOE 和 SE 中数据预测特点的 MDS 攻击分离出来进行独立总结. 首先, 介绍基于内存消歧(memory disambiguation)预测和基于存储到加载转发预测的 MDS 攻击. 然后, 梳理可控的 MDS 攻击.

4.3.1 基于内存消歧器

加载操作通常只有在所有存储操作完成之后才执行, 这样可以保证读取到更新后的数据. 处理器引入基于内存消歧器[90]的对加载和存储操作地址的别名预测, 允许在存储地址确定之前预测性执行加载. 此时会发生 2 种情况:

1)预测正确. 加载指令不依赖前面的存储指令(无别名), 加载操作无需等待存储指令执行完毕再执行, CPU 性能得到了提升.

2)预测错误. 加载指令对前面的某条存储指令有依赖关系(有别名), 加载指令读取的是错误的数据. CPU 需在刷新流水线后重新发射该指令.

Spectre V4[91]攻击利用内存消歧错误预测完成秘密窃取. 研究表明, 使用非别名存储执行相同的加载 64+15 次足以确保下一个预测是“无别名”[31]. 此时可以使用别名存储来执行加载, 以触发可读取旧值的瞬态路径, Spectre V4 的 PoC 代码如下:

① $\&a$ = 0x1109, $\&b$ = 0x1109;

② $x = 0$, $y = 0$, $a = 0$;

③ $a = a + 1$;

④ $y = x + b$;

其中, a 与 b 指向同一内存地址 0x1109, 此时 a, b 互

为“别名”. 若执行顺序为 1, 2, 4, 3, 则 y 会加载旧值 0, 但其应读值为 1. 基于旧值, 攻击者可推断程序的执行状态.

4.3.2 基于存储到加载转发预测

为提高数据存取指令的执行效率, 存储子系统中引入了集成的 OoOE 和 SE 的多种存取缓冲区. 然而, STL 的实现特性却被利用, 爆发了以 ZombieLoad 为代表的微架构数据采样型(MDS-type)攻击.

1)RIDL

RIDL[35] 利用 LFB 和 LB 会暂存加载操作的数据以及 STL 错误转发的特性完成对数据的窃取. 当攻击者与受害者进程同时执行加载操作, 且两者读取的数据低位地址相同时, 由于 STL, 攻击者可能获取受害者最新加载的数据.

LFB 等存取缓冲区在物理核心上共享, 通过利用错误转发, RIDL 攻击可以绕过隔离, 窃取同一物理核心上其他普通进程、内核进程、虚拟机甚至可信执行环境中的敏感信息. 例如, 内核不会执行未经验证的用户指针, 但在 Linux 内核中, 系统调用 *copy_from_user* 允许内核预先访问用户指针指向的数据. 攻击者可以将指针指向内核地址, 然后调用 *copy_from_user*, 这会导致内核数据被加载至 LFB. 此时 RIDL 攻击者可利用 STL 窃取敏感数据. 通过多次迭代, 攻击者甚至能够获取任意内核地址的数据.

RIDL 攻击还可以通过 LFB 发起跨虚拟机攻击. 例如, Linux 系统的影子文件存储了所有用户的账号与密码信息, 正常情况下, 这些信息仅对内核权限用户可见. 但具有用户权限的攻击者虚拟机若重复请求 SSH 授权认证, 会导致内核加载影子文件进行授权验证. 加载操作势必会将影子文件带入 LFB, RIDL 通过监测 LFB 可将整个影子文件转储, 进而获取所有用户的账号密码信息.

LFB 和 LB 并不会进行权限检查, 因此 RIDL 攻击可以窃取 LFB 中出现的数据. 但是, RIDL 只能泄露不在 L1D 缓存的数据, 因为当数据在 L1D 缓存时, 不会执行内存读操作, 所以数据并不会经过 LFB.

2)ZombieLoad

某些错误或异常处理非常复杂, 此时需要微码辅助(microcode assist)处理. 当数据加载过程中发生需要微码辅助处理的异常时, 加载指令会在提交阶段之前读取一个旧值. 基于这一观察, Schwarz 等人[36] 提出 ZombieLoad 攻击. 该攻击在异常处理完成前将读取的旧值编码到缓存隐蔽信道完成数据的窃取.

相比于 RIDL 攻击只能泄露不在 L1 缓存中的数据, ZombieLoad 攻击的泄露范围有所扩大, 它能泄露所有加载指令的数据. 但其控制泄露的能力也受到与 RIDL 同样的限制, 即不能控制泄露数据.

3)Fallout

RIDL 和 ZombieLoad 攻击利用 LFB 和 LB 的实现特性完成攻击, Fallout[37] 则提出 SB 同样能利用 STL 的错误转发预测完成数据窃取.

如下 PoC 代码所示, 指针变量 a 与 v 分别指向不同的内存空间, 行④对 $v[i]$ 写入数据 46 时, 会在 SB 中创建对应 i 的表项; 行⑤编码操作访问 $a[i]$ 时, 由于 i 相同, STL 会将行④对 $v[i]$ 的存储操作值, 直接赋值给 $a[i]$, 编码后通过缓存隐蔽信道可分析其值.

① v = *mmap* ( );

② a = 0x808000;

③ int i = 7;

④ $v[i]$ = 46;

⑤ *access* ($a[i]$ * j + k);

Fallout 还可以探测某虚拟地址是否有对应物理页. 当虚拟地址没有映射到明确的物理地址时 STL 不能读取到虚拟值, 当虚拟地址映射到实际物理页面才会触发存储到加载转发预测. 如下 PoC 代码所示:

① mov (0)→$dummy;

② mov $$x$→ ($p$);

③ mov (p)→$$y$;

④ mov ($mem + $$y$ * 4 096)→$dummy;

行①触发异常, 行②将 x 存储到地址 p 处. 行③重加载地址 p, 行④是编码操作. 若 (p)=x, 则 p 对应的物理页面存在, 否则 p 不受物理页支持. 通过将 p 指为虚拟内核地址并逐个测试, 可以得到某些虚拟地址是否被实际的物理页支持, 进而破解 KPTI[73].

4)Medusa

Medusa[38] 是 ZombieLoad 的攻击变种, 其利用 rep mov 和 rep stos 操作期间 LFB 中的组合写来完成信息窃取. 与 1)~3)中的 MDS 攻击不同, Medusa 使用组合写来避免内存操作, 能在更小噪声情况下获取 LFB 泄露信号. 然而, 这也将其窃取行为限定在组合写期间. 与 ZombieLoad 相同, Medusa 同样无法控制确切的偏移量, 只能对泄露的数据进行部分采样. Medusa 对内核进行攻击的数据泄露率为 12 Bps, 并且需要利用字节平均技术降噪[38]. 对于非结构化数据(如 RSA 密钥), 需要 400 个 CPU 小时的格攻击[92-93] 才能从泄露中恢复 1 024 位 RSA 密钥.

5)VRS

向量寄存器中的部分数据可能会传播到 SB 的

未使用部分. 基于此, VRS[42](victor register sampling, VRS)攻击利用向量寄存器采样, 窃取先前运行在同一物理核心的进程使用的向量数据. 相较于前 4 类 MDS 攻击能采样整个 LB, SB, LFB 中的内容, 而 VRS 仅能推断向量寄存器的高位数据.

6)Snoopy 和 Crosstalk

缓存一致性协议中, 当一个核尝试读取或写入某个内存地址时, 其他核会“嗅探”(snoop)该总线事务, 以检查本核心的缓存中是否也包含这个地址的数据, 并更新各级缓存中的数据副本. Snoopy 攻击[25]利用 snoop 机制, 可完成跨物理核心的数据窃取.

设受害进程 V_1 和 V_2 分别运行在物理核 C_1 和 C_2, 攻击进程 A 运行在 C_2. 若 V_1 更改其 L1 缓存中秘密值 k, snoop 机制将运行, 以避免 V_2 加载到 k 的旧值. 但当 snoop 到达 C_2 时, 若 A 也在执行加载操作, 则可能会读取到 snoop 中包含的 V_2 数据, A 将该数据编码到缓存隐蔽信道即可完成信息窃取. 然而, 即使 Snoopy 攻击可完成形式上的跨物理核数据窃取, 但实际上仍要求攻击者与受害者运行在同一物理核.

Crosstalk[19]利用随机数生成器(random number generator, RNG)输出结果跨物理核心共享, 进行跨物理核攻击. 当加载操作的目标值位于其他物理核时, 需通过“特殊寄存器读取”操作从核外读取数据, 读取完毕的数据首先暂存在物理核心间共享的缓冲区中, 再传输到请求该数据的物理核 LFB. RDRAND 指令会返回从随机数生成器派生的随机数, 并且可在所有权限级别(包括用户空间和 SGX 飞地)使用. 当共享的暂存缓冲区(staging buffer)在加载值更新时, 仅修改该加载操作所需的暂存缓冲区部分, 并不修改其他部分, 未修改的部分可能包含来自其他寄存器的旧数据. 当共享的暂存缓冲区整个被复制到核心的 LFB 时, 就可以读取来自其他物理核心的数据.

7)Downfall-GDS

SIMD 常用来实现数据并行计算, gather 指令可将内存中非连续的数据加载到各指令共享的 SIMD 向量寄存器中. Moghimi 等人[23]发现, 为加快数据读取, 处理器会预测性地并行发射多个读操作, 即使发生错误读取, 也会将 gather 的执行状态保留以避免阻断其他并行执行的读取操作. Downfall-GDS[23]基于 gather 指令的上述特性从其他安全域窃取数据.

① lea $addr_1$, %rdi;

② *clflush* (%rdi);

③ mov (%rdi), %rax;

④ lea $addr_2$, %rsi;

⑤ mov $0b1, %rdi;

⑥ kmovq %rdi, %k1;

⑦ vpxord %zmm1, %zmm1, %zmm1;

⑧ vpgatherdd 0 (%rsi, %zmm1, 1), %zmm5{%k1};

⑨ movq %xmm5, %rax;

⑩ encode_eax;

⑪ call *cache_channel*.

如上 PoC 代码所示, 行①~③将地址 $addr_1$ 中数据逐出缓存, 以拓展 SEW; 行④~⑧使用 gather 指令访问一个不可缓存的目标地址 $addr_2$; 行⑨~⑩将 512 位向量寄存器中的双字编码为缓存行; 行⑪从缓存信道中还原秘密.

4.3.3 可控的 MDS 攻击

4.3.1~4.3.2 节介绍的 MDS 攻击有 2 个明显特征:

1)攻击者只能窃取当前正在访问的数据, 并要求目标数据在指定缓冲区中, 否则无法读取.

2)攻击者无法具体控制泄露所需的数据. 因此, 这些攻击通常需要结合对目标数据已知结构的了解, 例如固定前缀和后缀, 以便分离信息[35-38]. 使用上述攻击恢复密钥需要依靠对大量数据的采样对比作为支撑, 以从微小的泄露中重建或者恢复受害者的敏感数据. 因此它们被命名为“数据采样攻击”. 使 MDS 攻击的目标数据泄露可控成为一项热点工作, 目前已经取得了一些成果, 本节将总结可控 MDS 攻击的相关研究.

1)CacheOut

van Schaik 等人[43]发现, L1 缓存驱逐的数据暂存在 LFB 中等待写回内存. 这为选择 MDS 攻击的内容提出了新思路: 使用驱逐集主动驱逐缓存中的目标数据, 然后再使用 MDS 攻击从 LFB 读取.

CacheOut 攻击基于这一思路实现, 它包含 3 个步骤: 首先, 确定目标数据在缓存中的位置, 精确地生成驱逐集, 以控制泄露内容. 然后, 将指定数据从 L1D 缓存中驱逐, 逐出的信息暂存在 LFB 中. 最后, 从 LFB 中选取任意字节数据进行读取. 与 MDS 攻击相比, CacheOut 能够选择访问的数据内容, 不再限于通过存取操作带进 LFB 的数据. CacheOut 甚至能够窃取整个 L1D 缓存中的内容.

其实, Schwarz 等人[36]最初提到了驱逐 L1 缓存数据到 LFB 以进行可控的数据泄露的设想. 但在实验中仅检测到 0.1 Bps 的泄露, 所以他们粗略的认为这种泄露可以忽略不计. 相较于 ZombieLoad 主动驱逐只有 0.1 Bps 的尝试, CacheOut 的泄露速率达到了 2.85 KBps, 可满足实际攻击要求, 这与 LFB 容量远大

于 LB 相关，CacheOut 可一次采集更多数据.

2）LVI

加载值注入（load value injection, LVI）攻击是 van Bulck 等人[44]提出的一种 MDS 攻击变种. 当加载指令前发生#PF 时，LFB 会不经过地址匹配直接将刚存储的值转发给加载指令，如下代码所示：

① int *p = 0x110900; //攻击者进程内执行；

② a[*(*v)* j + k]; //受害者进程内执行.

行①指针变量 p 指向攻击者设置的受害者进程地址 0x110900，执行该行代码会将值*p 带入 LFB. 若此时系统发生了#PF；行②*v 解析时，可能会将刚加载的*p 直接赋给*v，此时*(*v)就是以值*p 作为地址的数据值，并且行②执行结束后，*(*v)被编码至缓存，通过隐蔽信道可还原其值.

行①的操作类似于 Spectre V2 的注入行为. 但与其不同的是，Spectre V2 的目标注入是在分支目标地址知晓之前，而 LVI 的控制流重定向是在受害者尝试获取分支目标地址之后. 前面介绍的 MDS 攻击对 LFB 的利用需要低位地址匹配，而 LVI 可以不经过地址匹配，直接读取攻击者指定地址的值，SGAxe[40]是 LVI 在 SGX 场景的应用.

3）Downfall-GVI

2023 年，Deniel 等人[23]将 GDS 攻击与 LVI 相结合提出 GVI 攻击，它可以将收集的数据转化为微架构数据注入. 首先，攻击者在受害者代码中找到 gather 指令，然后执行与数据相关的操作，类似于 Gadget. 如下 PoC 代码所示.

① i = *gather* (*base*, j);

② *value* = a[i];

③ *cache_channel* (*value*);

其中，行①使用 gather 指令从另一地址空间中收集的双字或者 4 字结果用作内存索引. 执行行②代码会加载秘密载体项到缓存. 程序的正常执行不会使用越界值访问内存，但瞬态执行时攻击者可赋给 j 任意值，然后从 SIMD 中加载并完成对索引 i 的瞬时注入，这样 i 就会指向受害者地址空间内的指定地址，造成数据泄露.

4.3.4 小 结

结合第 2 节提出的 TEA 抽象模型，本文总结并对比了错误数据预测驱动的数据采样攻击的差异和共同之处，如图 22 所示.

熔断型攻击多样性体现在绕过地址隔离方式上，幽灵型攻击多样性体现在对分支预测组件污染方式上，数据采样攻击多样性体现在将数据组织到漏洞组件方式上. 因此，MDS 攻击差异也集中体现在预处理阶段. 它的研究难点是方式松散，如何利用一系列

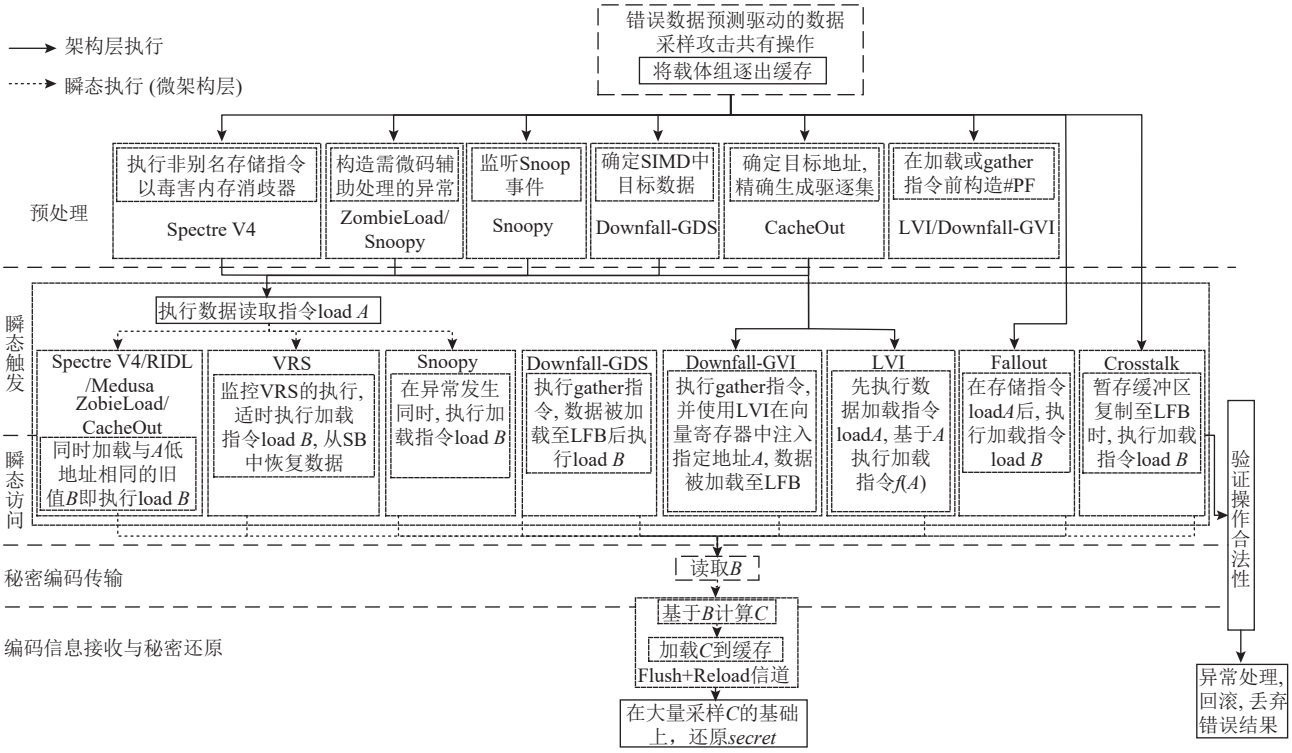


Fig. 22 Phased comparison of data sampling attacks driven by incorrect data prediction

图 22 错误数据预测驱动的数据采样型攻击分阶段比较

相关性较弱的组件, 并将数据组织至 LFB, LB, SB. 例如, Snoopy 利用异常时缓存一致性协议数据的错误转发, Crosstalk 则利用 LFB 从核外共享缓冲区加载数据会带入其他核心数据.

5 TEA 分析

本节首先分析 TEA 中隐蔽信道选用特点及其形成原因. 然后, 归纳 TEA 的权限场景和执行场景, 并基于此, 总结和分析其场景适用性. 最后, 根据 Intel 微架构对 TEA 的免疫情况, 分析其通用性.

5.1 隐蔽信道分析

5.1.1 信道特征分析

TEA 可根据目标系统和信道特点, 选用合适的隐蔽信道传递秘密信息. 本文在表 1 和表 2 中分别总结和对比了现有隐蔽信道的应用研究情况及隐蔽信道特点.

Table 1 Covert Channel Application Statistics

表 1 隐蔽信道应用情况统计

相关工作	信道名称	利用组件	攻击目标	相关工作	信道名称	利用组件	攻击目标
Osvik 等人[10]	E+T	L1D	OpenSSL AES	Yarom 等人[8]	F+R	LLC	GnuPG RSA
Wampler 等人[58]	E+T	LLC	内核隔离	Van de Pol 等人[92]	F+R	LLC	ECDSA
Hund 等人[94]	E+T	LLC	内核隔离	Benger 等人[93]	F+R	LLC	ECDSA
Lipp 等人[1]	P+P	L1D	OpenSSL AES	Allan 等人[95]	F+R	LLC	OpenSSL AES
Percival[96]	P+P	L1D	OpenSSL RSA	Zhang 等人[97]	F+R	LLC	用户活动
Liu 等人[13]	P+P	LLC	GnuPG ElGamal	Irazoqui 等人[98]	F+R	LLC	TLS/DTLS
Neve 等人[99]	P+P	L1I	AES	Irazoqui 等人[100]	F+R	LLC	GnuPG ElGamal/OpenSSL AES
Aciiçmez[101]	P+P	L1I	OpenSSL RSA	Gruss 等人[102]	F+R	LLC	键盘输入数据
Kayaalp 等人[103]	P+P	LLC	AES	Lipp 等人[15]	E+R	LLC	AES
Brasser 等人[104]	P+P	LLC	RSA	Yan 等人[54]	E+R	LLC	GnuPG RSA
Schwarz 等人[105]	P+P	LLC	RSA	Gruss 等人[14]	F+F	LLC	AES
Gullasch 等人[11]	F+R	LLC	OpenSSL AES	Disselkoen 等人[16]	P+A▲	LLC	OpenSSL AES
Irazoqui 等人[106]	F+R	LLC	GnuPG RSA	Schwarz 等人[21]	AVX 信道	AVX 单元	
Briongos 等人[12]	R+R▲	LLC	OpenSSL AES/RSA	Fustos 等人[5]	除法器信道	浮点运算单元	浏览器数据
Jin 等人[45]	EFLAGS	寄存器		Qiu 等人[6]	PMU 信道	性能监视单元	SGX
Bhattacharyya 等人[4]	端口争用	执行端口	OpenSSH	De Meulemeester 等人[7]	电磁信道	外部设备	

注: *"▲"表示已有相关研究, 但尚未应用在 TEA 中; "\"表示该研究尚停留在 PoC 阶段, 暂未提及在实际中的应用. E+T, P+P, F+R, R+R, P+A, E+R 分别表示 Evict+Time, Prime+Probe, Flush+Reload, Refresh+Reload, Prime+Abort, Evict+Reload.

1)缓存争用型信道特征

缓存争用型信道通过测量攻击进程读取自身数据的执行时间, 能更隐蔽地实现信息传递, 但需依据被攻击系统的缓存结构定制化的构造驱逐集, 可迁移能力较弱. 其次, 该类信道通过访问驱逐集的方式来初始化攻击、测量秘密载体项及重置攻击. 因此, 通常伴随短时间内对同一缓存集的大量换出及缓存未命中事件. 目前已有基于监测缓存争用特征事件来检测隐蔽信道活动进而防御 TEA 的研究[107-108].

2)数据重用型信道特征

数据重用型信道分辨率高, 但它依赖共享数据. 缓存访问时间差异是判断数据是否在缓存中的重要方式, 现有数据重用型信道滥用高精度计时器探测秘密载体项, 因此, 禁用或粗粒度化高精度计时器[109-110]的方法在一定程度上防御利用数据重用型隐蔽信道的攻击. 但同时需要关注不基于计时器的缓存隐蔽信道利用研究, 例如 Prime+Abort[16] 可将信道测量方式从时间阈值转变为 Intel TSX 的事务中断阈值.

3)非缓存型信道特征

非缓存信道同样利用了处理器组件暂存数据的特性, 相较于缓存, 其数据携带能力更弱, 大多数非缓存信道只能逐比特推断秘密, 并且抗噪音干扰能力较弱, 秘密的推断需要进行大量的重复实验以排除噪音干扰, 但该类新兴的隐蔽信道可以绕过当前主流的针对缓存型信道 TEA 的防御方法, 危险性与隐蔽性都高于缓存型信道.

5.1.2 信道选用分析

TEA 中隐蔽信道的选用特性与信道特征密不可

Table 2 Summary and Comparison of Covert Channel Characteristics

表 2 隐蔽信道特点总结与对比

信道类型	信道名称	粒度	信道特性				
			①	②	③	④	⑤
缓存争用型	E+T	S		√	√		√
	P+P	S		√	√		√
	E+R	S		√	√		√
数据重用型	F+R	L	√	√			√
	F+F	L	√	√			√
非缓存型	端口	B		√			√
	除法器	B		√		√	√
	AVX	B		√			√
	寄存器	B		√			√
	PMU	V/B					
	电磁	V	需要外接测量设备				

注：E+T，P+P，E+R，F+R，F+F 分别表示 Evict+Time，Prime+Probe，Evict+Reload，Flush+Reload，Flush+Flush. 缓存集粒度记为“S”，缓存行粒度记为“L”，比特粒度记为“B”，直接可得值记为“V”. ①②③④⑤分别表示滥用 clflush 指令、滥用计时器、短时大量缓存换出、借助除法器、时间测量.

分. 本文在表 3 中总结了 TEA 研究中隐蔽信道的选用情况，并分析其形成原因. 由表 3 可得当前研究的 2 个特点.

特点 1. TEA 研究偏向选用缓存隐蔽信道.

本文认为，这与缓存信道的以下 3 个特性相关：

1）缓存信道利用 CPU 内部组件，实用性强

电磁信道需要外部设备监测芯片的频率迹变化，能在实验室环境中实现，但在现实攻击场景中实用性较低. 除此之外，外部设备的测量易受环境噪声的影响，因此，对秘密的还原需要建立在大量数据分析上，有时还需借助机器学习技术辅助区分[7]. 相对而言，缓存信道利用的是 CPU 自身组件，受计算机外部因素影响较小，噪音来源于计算机内部.

2）缓存信道数据存储稳定，鲁棒性强

端口争用信道[4]、除法器信道[5]、AVX 信道[21] 和 EFLAGS 信道[45] 同样利用 CPU 微架构组件的指令执行时间差异推断秘密，但这些组件容量小、暂存数据能力差，例如 EFLAGS 信道仅能使用单比特标志位 ZF 指示秘密，且必须马上分析目标值，否则 CPU 内部激烈的竞争在平均 6~9 时钟周期就会重置 ZF 位，无法还原秘密[45]. 而缓存容量为 KB, MB 量级，竞争相对小，只要秘密载体项不被替换，秘密可一直保存在缓存中.

3）缓存信道的时间差异显著，区分度强

Table 3 Summary of Related Research on TEAs Combined with Attack Models

表 3 结合攻击模型的瞬态执行攻击相关研究总结

相关工作	Gadget 形式	SEW 拓展方式*	信道名称
Meltdown[1]	显式	Ⅰ	F+R
Spectre V1[2]	显式	Ⅰ/Ⅱ	F+R
Spectre V2[2]	显式	Ⅰ/Ⅱ	F+R
Foreshadow[17]	显式	Ⅰ	F+R
Crosstalk[19]	显式	Ⅰ/Ⅱ	F+R
Foreshadow-NG[20]	显式	Ⅰ	F+R
NetSpectre-Cache[21]	显式	Ⅰ	F+R
LazyFP[22]	显式	Ⅰ	F+R
Downfall[23]	显式	Ⅱ	F+R
RIDL[35]	显式	Ⅰ	F+R
ZombieLoad[36]	显式	Ⅰ	F+R
Fallout[37]	显式	Ⅰ	F+R
Medusa[38]	显式	Ⅰ	F+R
SGAxe[40]	显式	Ⅰ/Ⅱ	F+R
CacheOut[43]	显式	Ⅰ	F+R
LVI[44]	显式	Ⅰ	F+R
SgxPectre[60]	显式	Ⅰ/Ⅱ	F+R
ret2spec[62]	显式	Ⅰ/Ⅱ	F+R
RetBleed[64]	显式	Ⅰ/Ⅱ	F+R/P+P
Spectre RSB[67]	显式	Ⅰ	F+R
Spectre V3r/z[72]	显式	Ⅰ	F+R
Spectre V3a[77]	显式	Ⅰ	F+R
Spectre BHI[84]	显式	Ⅰ/Ⅱ	F+R/E+R
BHI-native[85]	显式	Ⅰ/Ⅱ	F+R/E+R
Spectre V4[91]	显式	Ⅰ	F+R
SMoTherSpectre[4]	隐式		执行端口
SpectreRewind[5]	隐式		除法器
NetSpectre-AVX[21]	隐式		AVX
EFLAGS attack[45]	隐式		寄存器
PMU-Spill[6]	显/隐式		PMU
SpectrEM[7]	显/隐式		电磁信息

注：*表示使用非缓存型隐蔽信道的 TEAs 在瞬态执行时即刻还原密钥，因此本文不讨论其窗口拓展方式. Ⅰ，Ⅱ表示图 7 所示的 2 个拓展区间，F+R，P+P，E+R 分别表示 Flush+Reload，Prime+Probe，Evict+Reload.

提升缓存命中时数据读速能提升 CPU 效率，优化查找算法和硬件设计使从缓存获取的数据与从内存获取的数据之间的时间差异显著，缓存命中时数据访问时间为 3 个时钟周期，而内存访问约 300 个时钟周期. 这种时间尺度的量级差异使攻击者更易区分秘密载体项. 而非缓存型信道时间差异则较小，例如，除法器信道值为 0 时程序执行时间约为 990 个时

钟周期, 值为 1 时为 1 100 个时钟周期[5], 同量级区分度较弱.

特点 2. 基于缓存隐蔽信道的 TEA 研究偏向选用 F+R 信道.

本文认为, 这与 F+R 信道的以下 4 个特性相关:

1)F+R 隐蔽信道运行在 L3 缓存

L1 信道和 L3 信道在信道范围、实施复杂性和效果等方面存在显著差异. L1 信道专注于单个核心, 信道范围较小, 且 L1 缓存为物理核心独占, 这要求攻击者与受害者双方需运行在同一物理核心. 相比之下, L3 缓存在核心间共享, L3 信道可跨核心传递信息, 具有更广泛的信道范围. 同时, L3 存储容量更大, 一次信道传输中能传递的信息更多, 而 L1 信道需要更频繁地操作才能获取相同数量的信息.

2)F+R 隐蔽信道准确率和粒度较高

相较于缓存争用型信道的缓存集粒度, F+R, F+F, R+R 信道粒度可精确至缓存行. 更精确的信道粒度意味着缓存分析范围更小, 需要处理系统中因其他进程争用缓存, 引入噪音也更少. 另一方面, 与 F+R 信道具有相同粒度的 F+F 信道通过分辨 clflush 指令执行时仅具有的几个时钟周期微小差异来判断秘密载体项, 这使得 F+F 信道在实际使用中更易受干扰, 误报率较高; R+R 信道[12]需基于对缓存替换策略的详细了解, 以确保待测量缓存行为优先级最高的驱逐候选; 而 F+R 信道具有更为明显的时间差异, 无需驱逐集参与, 利于攻击者从载体组分辨秘密载体项.

3)F+R 信道更为轻量级

缓存争用型隐蔽信道在 T1 阶段需要基于目标系统的缓存组织特性和数据替换策略生成驱逐集, 在 T4 阶段需要重新填充缓存集重置信道以传递下一数据. 这会带来额外的时空开销, 浪费宝贵的瞬态执行窗口时间. 而 F+R 信道并不依靠驱逐集设定信道, 执行 clflush 指令即可完成信道准备与重置.

4)TEA 攻击者兼任信道的发送方与接收方

缓存隐蔽信道的常规应用中具有攻击者与受害者 2 个进程, 进程之间具有独立的内存空间, 这使得常规应用中信道需处理攻击者与受害者间的协同. 而 TEA 中信道由攻击者支配, 极大地便利了配置以及秘密载体组探测.

然而, TEA 中的信道选用偏向会带来一些问题, 本文将在第 6 节详细探讨.

5.2 场景适用性分析

同样原理的攻击能在多个攻击场景下使用, 对不同攻击场景的适用性间接反映了攻击能力的强弱. 本节首先从攻击者的权限场景和执行场景 2 个方面分析各类场景, 然后总结各类隐蔽信道和 TEA 在不同攻击场景下的适用性.

5.2.1 攻击者的权限场景

攻击者权限通常分为用户权限和内核权限. 用户权限的攻击者可以启动进程并执行系统调用, 例如, 使用系统调用 rdtsc 获取处理器时间戳, 以测量数据读取时间, 从而推算缓存命中情况. 绝大多数从用户权限发起的 TEA 都依赖于时间测量, 用于分析缓存命中状态或触发瞬态指令执行等操作. 因此, 针对用户权限攻击者的防御比较简单, 内核权限的防御者可以通过干扰用户权限攻击者的时间测量来干扰攻击, 这使得只拥有用户权限的攻击适用性较弱.

相比之下, 内核权限的攻击者拥有更强大的攻击能力. 它可以直接操作底层资源, 使得特定组件容易被设置为攻击者所期待的状态. 因此, 具有内核权限的攻击者发动的攻击往往具有更大的破坏性. 内核攻击者通常具备以下 6 种能力:

1)利用性能计数器或其他硬件监测组件, 可更精确地感知攻击目标在隐蔽信道和瞬态执行中的状态. 例如, 使用缓存隐蔽信道时, 内核攻击者有能力指定受害者进程的运行核心, 以便更有效地监测受害者的活动; 而在瞬态攻击中, 内核攻击者可以借助性能计数器来辅助攻击, 例如 Intel Skylake 微架构为内核设置了专门缓冲区, 以记录最近几条分支的执行情况[111], 这些硬件功能单元相较于用户权限下的事件测量更为精确, 从而能获得更高的攻击成功率.

2)可以更轻松地调整进程的调度. 例如, SGX-Step[112]框架允许对飞地进行单步执行, 这种更为精确的指令同步控制意味着相较于只具有用户权限的攻击者, 内核攻击者可以更细粒度地获取飞地内代码执行状态, 从而提升攻击的准确性和成功率.

3)操纵受害者运行位置. 能否成功控制受害者地址空间的详细位置关乎攻击成败. 例如, 内核地址空间布局随机(kernel address space layout randomization, KASLR)向内核地址空间的起始地址添加随机偏移值, 防御恶意程序对内存的直接访问. 但拥有内核权限的攻击者可以轻易将受害者程序指定在某个攻击者已知或已掌握的位置上, 从而避免了对偏移值的探索, 节省了攻击时延.

4)可关闭处理器的安全配置. 可关闭处理器当前的安全配置是内核攻击者的有力手段之一. 例如, 内核攻击者可以关闭处理器对 Spectre 攻击的安全补丁[113-115], 从而基于原始的 Spectre 攻击探索新型幽灵

型漏洞. 另一个直观例子就是内核攻击者可直接关闭KASLR, 使受害者的地址空间暴露在攻击者视野中.

5)进行定制攻击. 具有内核权限的攻击者可以通过执行系统内核函数或者修改内核状态为某些攻击创造攻击条件, 而这是只具有用户权限的攻击者无法达到的.

6)降低噪声提升分辨率. 在实际攻击场景中, 处理器通常同时运行多个进程, 这导致对缓存的争用更加激烈, 同时这也意味着攻击者期望数据被逐出缓存的可能性将大大提高. 拥有内核权限的攻击者可以限制不必要的上下文切换来达到避免或减弱攻击噪声的目的.

5.2.2 攻击者的执行场景

Xiong 等人[32]从内存隔离视角对攻击场景进行了初步总结. 本文基于他们提供的攻击场景模型做了进一步拓展, 以更全面地概括适用于实际情况的攻击场景. 本文从处理器物理实现、操作系统以及内存隔离 3 个视角, 总结了 8 个执行场景.

1)处理器物理实现视角

①同核心攻击. 同核心攻击指攻击者与受害者运行在同一个物理核心上. 幽灵攻击中的基本要求就是攻击者与受害者驻在同一物理核心上, 这是由 CPU 硬件设计决定的, 即 CPU 分支预测机构只在同一物理核心内共享.

在包含式缓存中, LLC 中包含了各核心 L1, L2 缓存的数据副本, 因此实现跨核心数据窃取较为简单. 相反, 在非包含式缓存中, LLC 并不包含 L1, L2 缓存的数据副本, 这就要求将攻击者挂载到同一物理核心以探测 L1 中的受害者数据, 否则需要进行额外设置来实现此目的.

②跨核心攻击. 物理实现视角下的跨核心攻击是指攻击者与受害者运行在不同物理核心上. 目前针对跨物理核心的 TEA 的应用研究相对较少. 这是因为分支预测攻击和加载到存储转发攻击都依赖于缓冲区硬件共享, 在大多数桌面处理器中, 它们仅在物理核心中共享. 能否实现跨物理核心的 TEA 是一个值得探索的方向.

2)操作系统视角

①同进程攻击. 攻击者对受害者的信息窃取在同一地址空间内完成, 这种场景下, 攻击者无需切换进程, 也无需处理进程切换引入的噪音. 但是, 这种攻击对攻击者的要求较高, 限制较多. 例如, Spectre V1 的 PoC 程序就是一个同进程攻击的应用, 攻击者使用同一个分支对 PHT 进行恶意训练和触发攻击, 同时还要求攻击者掌握 x 的值.

②跨进程攻击. 跨进程攻击是指在同一物理核上交替执行攻击者和受害者进程, 达到攻击目的. 物理核独有 PHT, RSB 等硬件, 对于基于硬件共享完成的攻击同核心是必要的. 但是跨进程攻击对用户权限的攻击者而言难以进行, 因为进程切换带来的噪声是不可控的; 对内核权限的攻击者而言, 需要进行额外的处理来降低噪声.

③同步多线程攻击. 同步多线程(simultaneous multi-threading, SMT)是指 OS 在同一物理核的基础上将其划分为 2 个逻辑核, PHT 等预测部件在逻辑核上共享, 通常情况下 OS 将跨逻辑核的进程视为跨核运行. 所以 SMT 可以在满足跨进程的前提下不进行上下文切换. 对攻击者而言, SMT 既可以降低噪声又可以对共享部件执行操作. 例如, 缓存隐蔽信道中对受害者缓存集的探测以及 TEA 中对分支预测器恶意训练.

3)内存隔离视角

①用户到内核. 用户空间是 OS 分配给应用程序的内存空间, 其访问权限受限, 只能访问有限的系统和硬件资源. 相比之下, 内核空间具有对系统和硬件资源的完全访问权限. 攻击者可以通过复制整个内核空间数据来了解系统的运行方式, 获取加密密钥或者其他敏感信息.

②跨虚拟机攻击. 攻击者程序与受害者程序分别在不同的虚拟机中运行, 在各虚拟机之间可以运行不同的 OS, 2)中场景主要针对同一 OS 情况. 跨虚拟机攻击者需要单独处理.

③可信执行环境攻击. 该场景下要求攻击者对可信执行环境的具体实现细节有详细的了解. 相对于普通环境, 可信执行环境对数据的机密性和完整性要求更高, 对它的攻击往往伴随着对加密密钥的精确获取, 对攻击者的噪声控制要求较高.

基于本文总结的 8 个执行场景, 在表 4 中整理了现有文献中明确展现的各类隐蔽信道的适用场景, 在表 5 中梳理了 Flush+Reload 信道(缓存型)和其他非缓存型隐蔽信道下 TEA 的适用场景.

由表 4 可见, 隐蔽信道涉及场景多, 从开始时仅限于同一进程信道, 发展至跨进程信道[105]、跨核心信道[8,13-14]以及跨虚拟机信道[101]. 信道传输范围不断扩大, 信息传递能力不断增强. 其中, L3 缓存型信道的硬件设计决定了它可以完成跨物理核心攻击的特殊性, 而 L1, L2 缓存型信道和其他基于核内组件共享的非缓存型信道在实现跨核心信息窃取时受到了物

Table 4 Summary of Research on Covert Channel Applicable Scenarios

表 4 隐蔽信道适用场景研究总结

信道名称	物理核		操作系统			内存隔离		
	同核心	跨核心	同进程	跨进程	SMT	内核	跨 VM	TEE
Evict+Time[10]	√	-	√	√	-	-	-	-
Prime+Probe[13]	√	√	√	√	-	-	-	-
Evict+Reload[15]	√	√	√	√	-	-	-	-
Flush+Reload[8]	√	√	√	√	√	-	-	-
Flush+Flush[14]	√	√	√	√	-	-	-	-
端口争用[4]	√	-	√	-	√	√	-	-
除法器[5]	√	-	√	√	√	√	-	-
AVX[21]	√	-	√		-	√	-	-
EFALGS[45]	√	-	√	√	-	√	-	-
PMU[6]	√	-	√	√	-	√	-	√
电磁信息[7]	√	√	√	√	√	√	-	-

注: "-"表示原文献中未明确指出该攻击场景.

理隔离的限制. 以电磁信道为代表的外部信道则可以突破这种限制.

由表 5 可见, 相较于 Meltdown 和 Spectre 等原始攻击, 近年出现的 TEA 新攻击变种如 Crosstalk[19], Downfall[23], SGAxe[40] 等, 展现出打破安全隔离机制的能力更强, 适用于攻击场景更多样化的趋势. 具体而言, 立足于物理视角, 当前研究集中于同物理核心, 这是由预测机构只在核内共享造成的, 对跨物理核心场景的研究有待进一步深入. 立足于 OS 视角, 各攻击基本都能实现同进程和跨进程攻击, 而 SMT 能在与其他进程共享硬件的基础上尽量避免进程切换带来的噪声, 因此近期的研究也多涉及对该场景下的应用研究. 立足于内存隔离视角, 相较于熔断型和幽灵型攻击, 数据采样型攻击能更为便捷地绕过 OS 或虚拟机管理系统提供的地址空间隔离进行信息窃取.

5.3 微架构通用性分析

当前的综述研究忽视了对微架构部分的总结. 本文结合微架构发展历程, 总结了微架构对 TEA 的免疫情况, 如图 23 所示. 这种免疫情况间接反映了 TEA 的通用性.

由图 23 可见, 微架构正朝着芯片制程更小且安全机制更强的方向发展. TEA 的发展与微架构的更新形成了一种相互对抗的关系. 在 TEA 爆发前发布的几乎所有 Intel 处理器微架构都存在被攻击风险. 随着防御技术的发展以及底层硬件的重设计, 更新的微架构已经能够天然免疫有限类型的攻击. 例如, 2018 年之后发布的微架构宣称可以免疫原始 Meltdown 攻击, 2020 年之后发布的微架构宣称可以免疫 MDS 攻击, 近 2 年发布的微架构宣称可以免疫 LVI 攻击.

Table 5 Summary of Applicable Scenarios for TEAs

表 5 瞬态执行攻击适用场景总结

攻击名称	物理核		操作系统			内存隔离			攻击名称	物理核		操作系统			内存隔离		
	同核心	跨核心	同进程	跨进程	SMT	内核	跨 VM	TEE		同核心	跨核心	同进程	跨进程	SMT	内核	跨 VM	TEE
Meltdown[1]	√	-	√	√	-	√	-	-	VRS[42]	√	-	√	√	√	√	√	-
LazyFP[22]	√	-	√	√	-	√	-	-	Foreshadow[17]	√	-	-	√	√	-	-	√
Spectre V1[2]	√	-	√	√	-	√	-	-	Foreshadow-OS[20]	√	-	-	√	√	-	-	√
Spectre V1.1[82]	√	-	√	√	-	√	-	-	Foreshadow-VMM[20]	√	-	-	√	√	-	√	√
Spectre V1.2[83]	√	-	√	√	-	√	-	-	RIDL[35]	√	-	√	√	√	√	√	√
Spectre V2[2]	√	-	√	√	-	√	-	-	ZombieLoad[36]	√	-	√	√	√	√	√	√
Spectre V3a[77]	√	-	√	√	-	√	-	-	Fallout[37]	√	-	√	√	√	√	√	√
Spectre V4[91]	√	-	√	√	-	√	-	-	Medusa[38]	√	-	√	√	√	√	√	√
Spectre RSB[67]	√	-	√	√	-	√	-	√	LVI[44]	√	-	√	√	√	√	√	√
ret2spec[62]	√	-	√	√	-	√	-		CacheOut[43]	√	-	√	√	√	√	√	√
Snoopy[25]	√	√	√	√	√	√	√	√	SGAxe[40]	√	-	√	√	√	√	√	√
RIDL-TAA[41]	√	-	√	√	√	-	√	√	Crosstalk[19]	√	√	√	√	√	√	√	√
SMoTherSpectre[4]	√	-	√	√	√	√	-	-	Downfall[23]	√	-	√	√	√	√	√	√
EFLAGS attack[45]	√	-	√	√	-	√	-	-	NetSpectre-cache[21]	√	-	-	√	-	√	-	-
SpectreRewind[5]	√	-	√	√	√	√	-	-	NetSpectre-AVX[21]	√	-	-	√	-	√	-	-
PMU-Spill[6]	√	-	√	√	-	√	-	√	SpectrEM[7]	√	√	√	√	-	√	-	-

注: "-"表示原文献中未明确指出该攻击场景.

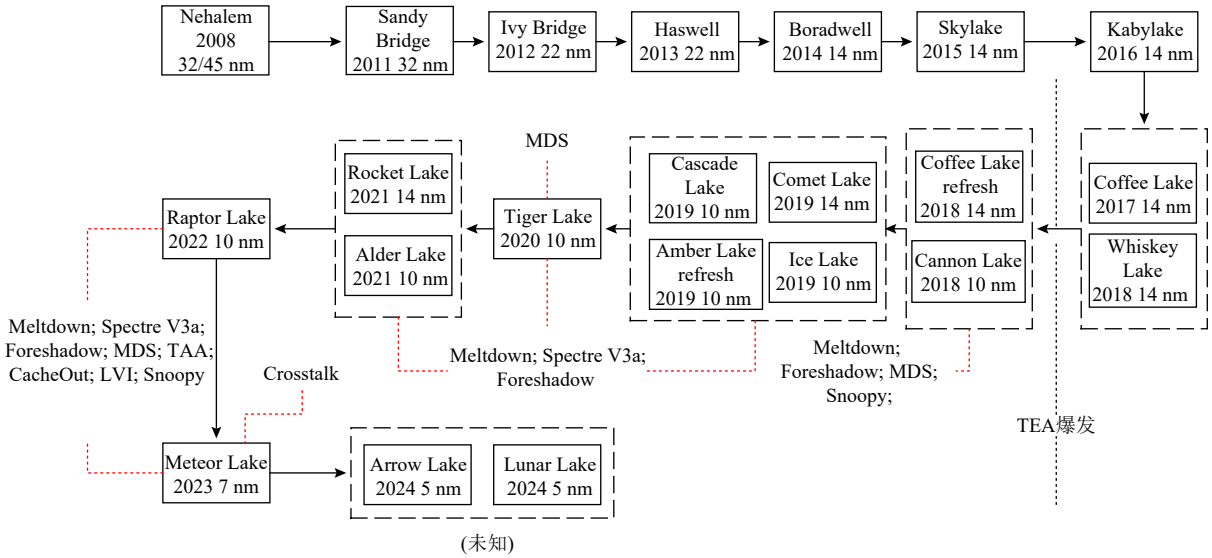


Fig. 23 Summary of Intel processor immune attack

图 23 Intel 处理器免疫攻击总结

然而，微架构的更新并不能完全抵制针对某一型号微架构的专门攻击变种. 例如，SpecROP[68] 攻击对 Coffee Lake 与 Comet Lake 微架构的 BTB 的嵌套预测机制进行了逆向工程，BranchSpectre[66] 对 Cascade Lake 微架构的 PHT 进行了逆向工程，发现通过 3 次错误预测 PHT 就会转为与 BHB 结合的全局历史预测模型，然后仅使用 6 次正确与错误分支交替转换的执行序列就可以完成对 PHT 表模式的转换控制.

微架构当前的防御措施存在一定局限性，有时甚至会成为未知攻击的帮凶. 例如，Cascade Lake 微架构引入了 verw 指令，在进程切换时清空 LFB 缓冲区以抵抗 RIDL 攻击，但基于微码辅助的 LFB 攻击，甚至能借助 verw 加速数据的泄露速率[36].

截至 2023 年，Intel 发布的微架构仍无法在微架构设计层面彻底免疫 Spectre V1，Spectre V2，Spectre V4 和 RSB 攻击，只能依靠增添额外性能开销的软硬件防御措施进行防御. 因此，本文认为，绕过现有软件防御，发展 Spectre 攻击的变种以及对新微架构的 TEA 漏洞挖掘研究都是具有广阔前景的研究方向.

6 未来研究方向

TEA 变种不断演变，仍有许多问题需要在研究中发现并进一步解决. 本节结合当前防御策略和攻击研究的发展趋势，从处理器微架构视角和隐蔽信道视角出发，探讨 TEA 研究的未来方向.

6.1 处理器微架构角度

1）现有微架构防御措施的破解. 即研究如何规避现有软硬件防御策略. 在过去 6 年中，针对 TEA 的防御措施不断涌现，例如，将用户与内核页表分离设计[115-118]以及将用户与内核代码划分至不同物理核心运行[119]，以避免 Meltdown 对内核敏感数据造成的威胁. 或是对预测组件添加额外限制防止其被 Spectre 利用[113-115]，例如，防止在内核程序中执行低权限代码[113]、限制分支预测器在跨线程执行的代码间共享[114] 以及在进程切换时刷新 BTB[115] 等方式防止恶意程序对微组件的恶意注入，消除对其他进程数据构成的安全威胁.

然而，近年的一些攻击研究以规避这些防御策略为目的，通过对攻击进行定制化改进，产生可突破现有防御策略的新攻击变种. 例如，ExSpectre[58] 采用控制流不可达代码填充分支预测器进行攻击，可规避基于 Gadget 检测的防御手段[55-57]；BHI[84] 则通过控制 BHB 来间接毒害 BTB，使其能在部署了 IBRS[113] 的处理器上进行数据窃取；LeakIDT[116] 甚至可通过观察 Meltdown 窃取非机密数据的成功率来间接推断秘密，从而绕过 KPTI[73]. 在可预见的未来，破解现有微架构防御措施以及拓展瞬态攻击的攻击能力和适用场景，仍是攻击研究的热点方向之一.

2）现有微架构中组件的未知漏洞挖掘. 即研究如何利用微架构组件其他机制来实现或强化瞬态攻击. 挖掘微架构中尚未被发现的漏洞或组件的不安全行为是扩展攻击变种的重要方式，逆向工程和程序分析是挖掘未知漏洞的常用手段.

逆向工程以攻击为导向，通过设计一段测试代码和监视 PMU 记录的微架构事件，推断硬件尚未公开的执行机制. 当前，逆向工程在瞬态攻击中的应用

已逐渐成熟,是发现新攻击的主要手段.如 Meltdown[1], Spectre[2], ZombieLoad[36] 等大部分瞬态攻击的研究工作均建立在逆向工程之上.

程序分析以防御为导向,通过对模糊测试[120] 等传统软件测试方法进行改进来搜索隐藏在正常程序中的瞬态漏洞.一些传统测试方法带有漏洞挖掘属性,使以防御为初衷的研究可能会发现 TEA 新变种.例如,Hur 等人[121] 使用模糊测试检测 RISC-V 处理器中的瞬态漏洞时,意外发现了 2 类新攻击变种 Birgus 和 Boombard.这类方法也成为发现新攻击的辅助手段.然而,现有研究仍然存在诸多问题;如何缩减符号执行的分析路径[55,122]、如何提升模糊测试用例生成效率[56,123] 以及如何更加精确地自动化判定瞬态漏洞[57,124-125],都还没被充分研究.本文认为,如何继续利用上述 2 种方式,探索处理器中隐藏的瞬态漏洞,并基于此发展新型攻击,将是瞬态攻击中的一个具有前景的研究方向.

3)现有微架构的 SGX 等安全增强机制.即研究如何突破现有安全增强机制.例如,Intel SGX 依赖硬件隔离确保安全,即使拥有内核权限的攻击者也无法获取飞地中的数据,但能通过对 PHT[80]、BTB[60] 或其他组件[126] 的污染发动瞬态攻击窃取.在 ARM 架构中,TrustZone[127] 与 SGX 安全边界的设定类似,但两者在实现上有许多差别.TrustZone 依赖于安全内核,CPU 隔离程度更高,但在 ARM 架构上无法像 Intel x86 架构那样通过 SMT 共享分支预测器等微架构组件.因此,目前尚未发现在 ARM 架构 CPU 上的跨 TrustZone 的 TEA.本文认为,SGX 已有相对丰富的瞬态攻击研究,但 TrustZone 及其他安全增强机制的瞬态攻击研究有待深入.例如,提供虚拟机加密保护的 ADM SEV[128] 和基于硬件寄存器配置内存访问权限的 RISC-V PMP[129] 等.

4)新型微架构的安全新特性破解.即研究如何破解新架构的安全特性.处理器制造商推出的新型微架构通过重新设计或强化微架构组件来防御 TEA.例如,Cascade Lake 引入 verw 特性使处理器在进程切换前清空 LFB,以防御 RIDL 攻击.但研究发现[36],在需微码辅助处理的异常情况下,verw 甚至还会加快数据泄露.新型微架构如 Raptor Lake 和 Meteor Lake 分别于 2022 年和 2023 年发布,目前暂无研究表明这些新型微架构容易受到先前已知的 TEA 的影响,其安全新特性也不透明.如何了解它们的安全新特性,并在此之上评估微架构的脆弱性,不仅能指导未来的微架构设计和安全性改进,也对微架构安全的发展至关重要,是需要长期关注的研究方向.

5)针对其他架构的攻击.即研究针对除 x86 架构之外的其他架构的攻击.尽管目前已有针对 ARM 或 RISC-V 等其他处理器架构的瞬态攻击研究[121],但相对 x86 架构而言,其研究体量仍相对较小.一旦产生基于其他架构的新攻击变种,其危害不可估量,例如,Hetterich 等人[130] 将 Spectre 拓展到 ARM 架构下的 Apple M1 芯片,暴露了苹果设备存在的瞬态安全隐患,直接威胁 iPhone 和 Mac 设备的数据安全.

将已在 x86 架构上验证的 TEA 移植到 ARM, RISC-V 等架构,或针对这些架构的实现特性展开针对性攻击研究,始终是拓展新攻击变种的重要研究方向,对其他架构的处理器安全具有重要意义.

6.2 隐蔽信道角度

1)与其他缓存隐蔽信道结合.即研究使用其他形式的缓存隐蔽信道.F+R 信道因其实现思路简单,限制条件少和高分辨率被 TEA 攻击研究广泛选用.然而,这种信道选用偏向会带来一些问题,例如,F+R 信道强依赖于 clflush 指令,这意味着通过禁用 clflush 指令即可抵御大部分基于 F+R 信道实现的攻击.消除 F+R 信道可中断 TEA 的数据传递从而挫败瞬态攻击,因此针对 F+R 信道的防御技术大量涌现并逐渐发展完善,例如,通过性能计数器监控[131] 或缓存分区[132] 中断 F+R 信道.探寻与其他形式的缓存隐蔽信道结合的攻击方式可绕过这些防御手段并拓展新型攻击,因此也将成为 TEA 研究的主要研究内容.例如,Prime+Abort[16] 信道不依赖 clflush 指令,Reload+Refresh[12] 信道可以基于对缓存替换算法的逆向工程规避基于性能计数器的检测方式[131,133-134].

2)对非包含式缓存攻击研究.即研究在非包含式缓存隐蔽信道下发动瞬态攻击.目前研究主要集中于包含式缓存,而非包含式缓存的设计打破了缓存隐蔽信道的共享特性,天然具有防御多数基于包含式缓存信道 TEA 的优势.因此,处理器制造商在芯片设计中也逐步考虑使用非包含式缓存,如 Intel Skylake-SP[54] 和 AMD Zen3[135].一些攻击研究[15,54,136] 也开始向非包含式缓存发展,但使用非包含式缓存作为隐蔽信道时需要处理更加复杂的缓存情况,例如处理更为复杂的缓存一致性协议、不同级别缓存之间的协作,以及缓存替换策略的多样性.这些复杂性都增加了攻击者在非包含式缓存中建立稳定隐蔽信道的难度.本文认为,将当前包含式缓存攻击改进以适用于非包含式缓存,使攻击更加通用,在未来可预见的一段时间内将是一个热点.

3)结合 AI 新技术的信道降噪研究. 即研究如何使用机器学习(machine learning, ML)和 AI 新技术优化隐蔽信道的秘密传递. ML 模型在大规模数据处理中表现出的独特优势，能助力攻击者从超标量处理器的缓存冲突中提取特征和规律，降低缓存活动噪声以提高隐蔽信道传播的准确率. 目前，已有部分使用 ML 模型进行信道降噪的攻击研究，例如，SpectrEM[7]借助多层感知机[137](multilayer perceptron, MLP)排除噪声，协助辨别频率迹与敏感值的对应关系. 然而，现有研究[106,138-140]的整体规模仍相对有限，如何与其他已发展成熟的模型结合，取得更优异的降噪效果，有待研究的进一步深入. 同时，本文也注意到，近年来兴起的 AI 新技术(如大模型等)，已在传统漏洞修复等安全领域取得成果[141]. 但在瞬态攻防领域暂无相关研究，如何使用大模型等新型 AI 技术辅助信道降噪，提高信道分辨率或许也是未来的一个潜在研究方向.

4)开辟非缓存型隐蔽信道. 即研究开辟缓存以外的其他隐蔽信道. 阻断隐蔽信道传输的防御策略利用 TEA 高度依赖缓存进行数据传递和还原的特性，从软硬件角度对缓存进行限制. 例如，通过划分缓存或禁止含有敏感数据的缓存行被替换等方式削弱缓存共享性[142-143]，或通过设计新型缓存结构为缓存增加更多的随机性以及破坏常规缓存信道的共享性与测量基础[132,144-145].

本文注意到，基于这样的防御态势，TEA 近期研究呈现出“去缓存化”发展的趋势. 例如，使用除法器信道的 SpectreRewind[5]和 PMU 信道的 PMU-Spill[6]攻击. 其中，除法器信道仍然依赖于计时器，这表明基于破坏计时的防御策略[109-110,146]仍适用于 SpectreRewind. 而 PMU-Spill 则摆脱了对计时器的依赖，使前述防御手段失效，目前仅能通过 PMU 重设计或软件禁用进行防御[147]. 使用非缓存型隐蔽信道的 TEA 是防御研究的短板，一旦发现 TEA 能通过未知的新型信道发动攻击，短时间内难以研发出安全与性能折中的防御手段. 本文认为，在未来相当长的一段时间内，这类研究方向需获得重点关注，对 TEA 和防御研究均具有重要意义.

7 总　　结

以 Meltdown 和 Spectre 为代表的 TEA 严重威胁计算机系统中的数据安全. 本文从硬件设计原理出发，深入分析了导致 TEA 漏洞发生的安全问题的根源，然后通过抽取 TEA 的关键特性，系统性地建立了全新的 TEA 模型. 同时，我们总结了现有攻击研究中使用的隐蔽信道，并对其秘密传输步骤进行了介绍. 之后，本文以 TEA 的硬件来源作为分类依据，整理归纳了学术界主要系统安全会议和期刊中有关 TEA 的最新进展，并结合 PoC 代码或图示描述了每个攻击的基本思想和攻击过程. 最后，本文从隐蔽信道、攻击场景和微架构通用性 3 个角度对 TEA 进行了系统性的深入分析. 未来，该领域仍然有许多值得探索的未知领域，结合防御研究和对文献的总结，本文提出了 9 个潜在的研究方向，供安全研究人员参考，以期为 TEA 和防御研究发展做出一份应有的贡献.

作者贡献声明：李扬负责论文的总体规划、主要内容的调研以及论文的撰写；马自强和林璟锵负责论文整体结构的梳理及写作指导；孟令佳和李冰雨提出指导意见并在论文撰写过程中提供支持；苗莉负责论文内容的梳理；高菲负责图表内容校对.

参　考　文　献

[1] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading kernel memory from user space[C]//Proc of the 27th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2018: 18−33

[2] Kocher P, Horn J, Fogh A, et al. Spectre attacks: Exploiting speculative execution[C]//Proc of the 2019 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 1−19

[3] Szefer J. Survey of microarchitectural side and covert channels, attacks, and defenses[J]. Journal of Hardware and Systems Security, 2019, 3(3): 219−234

[4] Bhattacharyya A, Sandulescu A, Neugschwandtner M, et al. SMoTherSpectre: Exploiting speculative execution through port contention[C]//Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 785−800

[5] Fustos J, Bechtel M, Yun H. SpectreRewind: Leaking secrets to past instructions[C]//Proc of the 4th ACM Workshop on Attacks and Solutions in Hardware Security. New York: ACM, 2020: 117−126

[6] Qiu Pengfei, Gao Qiang, Liu Chang, et al. PMU-Spill: A new side channel for transient execution attacks[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(12): 5048−5059

[7] De Meulemeester J, Purnal A, Wouters L, et al. SpectrEM: Exploiting electromagnetic emanations during transient execution[C]//Proc of the 32nd USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2023: 6293−6310

[8] Yarom Y, Falkner K. Flush+ Reload: A high resolution, low noise, L3 cache side-channel attack[C]//Proc of the 23rd USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2014: 719−732

[9] Ristenpart T, Tromer E, Shacham H, et al. Hey, you, get off of my

cloud: Exploring information leakage in third party compute clouds[C]//Proc of the 16th ACM Conf on Computer and Communications Security. New York: ACM, 2009: 199−212

[10] Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: The case of AES[C]//Proc of the 2006 the Cryptographers' Track at the RSA Conf on Topics in Cryptology. Berlin: Springer, 2006: 1−20

[11] Gullasch D, Bangerter E, Krenn S. Cache games bringing access based cache attacks on AES to practice[C]//Proc of the 2011 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2011: 490−505

[12] Briongos S, Malagón P, Moya J M, et al. Reload+Refresh: Abusing cache replacement policies to perform stealthy cache attacks[C]//Proc of the 29th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2020: 1967−1984

[13] Liu Fangfei, Yarom Y, Ge Qian, et al. Last-level cache side-channel attacks are practical[C]//Proc of the 2015 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 605−622

[14] Gruss D, Maurice C, Wagner K, et al. Flush+ Flush: A fast and stealthy cache attack[C]//Proc of the 13th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2016: 279−299

[15] Lipp M, Gruss D, Spreitzer R, et al. ARMageddon: Cache attacks on mobile devices[C]//Proc of the 25th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2016: 549−564

[16] Disselkoen C, Kohlbrenner D, Porter L, et al. Prime + Abort: A timer-free high-precision L3 cache attack using Intel TSX[C]//Proc of the 26th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2017: 51−67

[17] 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]//Proc of the 27th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2018: 991−1008

[18] Mukhtar M A, Bhatti M K, Gogniat G. Architectures for security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone[C]//Proc of the 2nd Int Conf on Communication, Computing and Digital Systems. Piscataway, NJ: IEEE, 2019: 299−304

[19] Ragab H, Milburn A, Razavi K, et al. Crosstalk: Speculative data leaks across cores are real[C]//Proc of the 2021 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2021: 1852−1867

[20] Weisse O, Van Bulck J, Minkin M, et al. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution [EB/OL]. 2018[2023-11-29]. https://lirias.kuleuven.be/2089352

[21] Schwarz M, Schwarzl M, Lipp M, et al. NetSpectre: Read arbitrary memory over network[C]//Proc of the 24th European Symp on Research in Computer Security. Berlin: Springer, 2019: 279−299

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

[23] Moghimi D. Downfall: Exploiting speculative data gathering[C]// Proc of the 32nd USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2023: 7179−7193

[24] Intel. Snoop assisted L1D sampling advisory[EB/OL]. (2021-05-11)[2023-11-29]. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00330.html

[25] Intel. Snoop-assisted L1 data sampling[EB/OL]. (2020-03-10)[2023-11-29]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/snoop-assisted-l1-data-sampling.html

[26] Miao Xinliang, Jiang Liehui, Chang Rui. Survey of access-driven cache-based side channel attack[J]. Journal of Computer Research and Development, 2020, 57(4): 824−835 (in Chinese)
(苗新亮, 蒋烈辉, 常瑞. 访问驱动下的 Cache 侧信道攻击研究综述[J]. 计算机研究与发展, 2020, 57(4): 824−835)

[27] Zhang Weijuan, Bai Lu, Ling Yuqing, et al. Cache side-channel attacks and defenses[J]. Journal of Computer Research and Development, 2023, 60(1): 206−222 (in Chinese)
(张伟娟, 白璐, 凌雨卿, 等. 缓存侧信道攻击与防御[J]. 计算机研究与发展, 2023, 60(1): 206−222)

[28] Wu Xiaohui, He Yeping, Ma Hengtai, et al. Microarchitectural transient execution attacks and defense methods[J]. Journal of Software, 2020, 31(2): 544−563 (in Chinese)
(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)

[29] Li Ye, Li Peinan, Zhao Lutan, et al. Overview of transient execution vulnerability attacks and defenses[J]. High Technology Communications, 2020, 30(8): 774−782 (in Chinese)
(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)

[30] Canella C, Van Bulck J, Schwarz M, et al. A systematic evaluation of transient execution attacks and defenses[C]//Proc of the 28th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2019: 249−266

[31] Ragab H, Barberis E, Bos H, et al. Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks[C]//Proc of the 30th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2021: 1451−1468

[32] Xiong Wenjie, Szefer J. Survey of transient execution attacks and their mitigations[J]. ACM Computing Surveys, 2021, 54(3): 1−36

[33] Fiolhais L, Sousa L. Transient execution attacks: A computer architect perspective[J]. ACM Computing Surveys, 2023, 56(3): 1−38

[34] Intel Cor. Intel 64 and IA−32 architectures optimization reference manual volume 1[EB/OL]. (2023-09-05)[2023-11-09]. https://www.intel.com/content/www/us/en/content-details/814198/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html

[35] van Schaik S, Milburn A, Österlund S, et al. RIDL: Rogue in-flight data load[C]//Proc of the 2019 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 88−105

[36] Schwarz M, Lipp M, Moghimi D, et al. ZombieLoad: Cross-privilege-boundary data sampling[C]//Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 753−768

[37] Canella C, Genkin D, Giner L, et al. Fallout: Leaking data on

meltdown-resistant CPUs[C]//Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 769−784

[38] Moghimi D, Lipp M, Sunar B, et al. Medusa: Microarchitectural data leakage via automated attack synthesis[C]//Proc of the 29th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2020: 1427−1444

[39] Borrello P, Kogler A, Schwarzl M, et al. ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture[C]//Proc of the 31st USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2022: 3917−3934

[40] Van Schaik S, Kwong A, Genkin D, et al. SGAxe: How SGX fails in practice[EB/OL]. 2020[2024-01-01]. https://sgaxe.com/files/SGAxe.pdf

[41] VUSec Group. Addendum 1 to RIDL: Rogue in-flight data load, RIDL variant as the TSX asynchronous abort[EB/OL]. (2019-11-12)[2024-01-01]. https://mdsattacks.com/files/ridl-addendum.pdf

[42] VUSec Group. Addendum 2 to RIDL: Rogue in-flight data load, vector register sampling[EB/OL]. (2020-01-27)[2023-12-28]. https://mdsattacks.com/files/ridl-addendum2.pdf

[43] van Schaik S, Minkin M, Kwong A, et al. CacheOut: Leaking data on Intel CPUs via cache evictions[C]//Proc of the 2021 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2021: 339−354

[44] van Bulck J, Moghimi D, Schwarz M, et al. LVI: Hijacking transient execution through microarchitectural load value injection[C]//Proc of the 2020 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2020: 54−72

[45] Jin Yu, Qiu Pengfei, Wang Chunlu, et al. Timing the transient execution: A new side-channel attack on Intel CPUs[J]. arXiv preprint, arXiv: 2304.10877, 2023

[46] Šilc J, Robic B, Ungerer T. Processor Architecture: From Dataflow to Superscalar and Beyond; with 34 Tables[M]. Berlin: Springer, 1999

[47] Shen J P, Lipasti M H. Modern Processor Design: Fundamentals of Superscalar Processors[M]. Long Grove, Illinois: Waveland Press, 2013

[48] Tomasulo R M. An efficient algorithm for exploiting multiple arithmetic units[J]. IBM Journal of Research and Development, 1967, 11(1): 25−33

[49] González J, González A. Speculative execution via address prediction and data prefetching[C]//Proc of the 11th Int Conf on Supercomputing. New York: ACM, 1997: 196−203

[50] Denning P J. The working set model for program behavior[J]. Communications of the ACM, 1968, 11(5): 323−333

[51] Nagarajan V. A Primer on Memory Consistency and Cache Coherence[M]. Berlin: Springer, 2022

[52] Papamarcos M S, Patel J H. A low-overhead coherence solution for multiprocessors with private cache memories[C]//Proc of the 11th Annual Int Symp on Computer Architecture. New York: ACM, 1984: 348−354

[53] Zahran M, Albayraktaroglu K, Franklin M. Non-inclusion property in multi-level caches revisited[J]. International Journal of Computers and Their Applications, 2007, 14(2): 99−108

[54] Yan Mengjia, Sprabery R, Gopireddy B, et al. Attack directories, not caches: Side channel attacks in a non-inclusive world[C]//Proc of the 2019 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 888−904

[55] Guo Shengjia, Chen Yueqi, Li Peng, et al. SpecuSym: Speculative symbolic execution for cache timing leak detection[C]//Proc of the 42nd ACM/IEEE Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2020: 1235−1247

[56] Oleksenko O, Trach B, Silberstein M, et al. SpecFuzz: Bringing spectre-type vulnerabilities to the surface[C]//Proc of the 29th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2020: 1481−1498

[57] Qi Zhenxiao, Feng Qian, Cheng Yueqiang, et al. SpecTaint: Speculative Taint analysis for discovering Spectre gadgets[C]//Proc of the 2021 Symp on Network and Distributed System Security. Piscataway, NJ: IEEE, 2021: 841−855

[58] Wampler J, Martiny I, Wustrow E. ExSpectre: Hiding malware in speculative execution[C]//Proc of the 2019 Symp on Network and Distributed System Security. Piscataway, NJ: IEEE, 2019: 316−330

[59] Tobah Y, Kwong A, Kang I, et al. SpecHammer: Combining spectre and Rowhammer for new speculative attacks[C]//Proc of the 2022 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2022: 681−698

[60] Chen Guoxing, Chen Sanchuan, Xiao Yuan, et al. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution[C]//Proc of the 2019 IEEE European Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 142−157

[61] Trujillo D, Wikner J, Razavi K. INCEPTION: Exposing new attack surfaces with training in transient execution[C]//Proc of the 32nd USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2023: 7303−7320

[62] Maisuradze G, Rossow C. ret2spec: Speculative execution using return stack buffers[C]//Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2018: 2109−2122

[63] Maisuradze G, Backes M, Rossow C. Dachshund: Digging for and securing against (non-) blinded constants in JIT code[C]//Proc of the 2017 Symp on Network and Distributed System Security. Piscataway, NJ: IEEE, 2017: 200−215

[64] Wikner J, Razavi K. RetBleed: Arbitrary speculative code execution with return instructions[C]//Proc of the 31st USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2022: 3825−3842

[65] COMSEC Computer Security Group. Retbleed: Arbitrary speculative code execution with return instructions[EB/OL]. 2022[2023-12-30]. https://comsec.ethz.ch/research/microarch/retbleed

[66] Chowdhuryy M H I, Yao Fan. Leaking secrets through modern branch predictors in the speculative world[J]. IEEE Transactions on Computers, 2021, 71(9): 2059−2072

[67] Koruyeh E M, Khasawneh K N, Song C, et al. Spectre returns! Speculation attacks using the return stack buffer[C/OL]//Proc of the 12th USENIX Conf on Offensive Technologies. Berkeley, CA: USENIX Association, 2018[2024-02-13]. https://www.usenix.org/

system/files/conference/woot18/woot18-paper-koruyeh.pdf

[68] Bhattacharyya A, Sánchez A, Koruyeh E M, et al. SpecROP: Speculative exploitation of ROP chains[C]//Proc of the 23rd Int Symp on Research in Attacks, Intrusions and Defenses. Berkeley, CA: USENIX Association, 2020: 1−16

[69] Bernstein D J. Cache-timing attacks on AES[EB/OL]. 2005[2023-12-29]. https://mimoza.marmara.edu.tr/~msakalli/cse466_09/cachetiming-20050414.pdf

[70] Intel Cor. Intel architecture instruction set extensions programming reference[EB/OL]. 2023[2023-11-25]. https://cdrdv2.intel.com/v1/dl/getContent/671368

[71] Hammarlund P, Martinez A J, Bajwa A A, et al. Haswell: The fourth-generation Intel core processor[J]. IEEE Micro, 2014, 34(2): 6−20

[72] Baidu Security. Meltdown V3c &V3r[EB/OL]. (2018-03-07)[2023-11-25]. https://anquan.baidu.com/article/143

[73] Müller L. Kpti a mitigation method against meltdown[C/OL]//Proc of the 4th Wiesbaden Workshop on Advanced Microkernel Operating Systems. 2018[2024-02-13]. https://www.cs.hs-rm.de/~kaiser/events/wamos2018/wamos18-proceedings.pdf#page=43

[74] State Key Laboratory of Computer Architecture. Meltdown V3z and it mitigation[EB/OL]. (2019-03-26)[2023-11-25]. http://www.carch.ac.cn/hzjl/xshd/201906/t20190628_497154.html

[75] Intel. Speculative behavior of swapgs and segment registers[EB/OL]. (2019-08-06)[2023-12-21]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-behavior-swapgs-and-segment-registers.html

[76] Lutas A, Lutas D. Bypassing KPTI using the speculative behavior of the SWAPGS instruction[EB/OL]. 2019[2023-12-25]. https://i.blackhat.com/eu-19/Thursday/eu-19-Lutas-Bypassing-KPTI-Using-The-Speculative-Behavior-Of-The-SWAPGS-Instruction-wp.pdf

[77] The MITRE Corporation. CVE−2018−3640: Spectre variant V3a[EB/OL]. (2020-06-05)[2023-12-25]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-3640

[78] Zheng Wei, Wu Ying, Wu Xiaoxue, et al. A survey of Intel SGX and its applications[J]. Frontiers of Computer Science, 2021, 15(1): 1−15

[79] Smith J E. A study of branch prediction strategies[C]//Proc of the 25th Annual Int Symp on Computer architecture. New York: ACM, 1998: 202−215

[80] O'Keeffe D, Muthukumaran D, Aublin P L, et al. Spectre attack against SGX enclave[EB/OL]. 2018[2023-12-25]. https://github.com/lsds/spectre-attack-sgx

[81] Kim Y, Daly R, Kim J, et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors[J]. ACM SIGARCH Computer Architecture News, 2014, 42(3): 361−372

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

[83] Sternberger M. Spectre-NG: An avalanche of attacks[C/OL]//Proc of the 4th Wiesbaden Workshop on Advanced Microkernel Operating Systems. 2018[2024-02-23]. https://www.cs.hs-rm.de/~kaiser/events/wamos2018/wamos18-proceedings.pdf#page=23

[84] Barberis E, Frigo P, Muench M, et al. Branch history injection: On the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks[C]//Proc of the 31st USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2022: 971−988

[85] Wiebing S, de Faveri Tron A, Bos H, et al. InSpectre Gadget: Inspecting the residual attack surface of cross-privilege Spectre v2[C/OL]//Proc of the 33rd USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2024[2024-07-25]. https://download.vusec.net/papers/inspectre_sec24.pdf

[86] Bletsch T, Jiang X, Freeh V W, et al. Jump-oriented programming: A new class of code-reuse attack[C]//Proc of the 6th ACM Symp on Information, Computer and Communications Security. New York: ACM, 2011: 30−40

[87] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls on the x86[C]//Proc of the 14th ACM Conf on Computer and Communications Security. New York: ACM, 2007: 552−561

[88] AMD Cor. Software techniques for managing speculation on AMD processors, revision 5.09. 23[EB/OL]. (2023-05-09)[2024-01-16]. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/tuning-guides/software-techniques-for-managing-speculation.pdf

[89] INTEL. Speculative execution side channel mitigations[EB/OL]. (2021-05-26)[2024-01-16]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-execution-side-channel-mitigations.html

[90] Huang A S, Slavenburg G, Shen J P. Speculative disambiguation: A compilation technique for dynamic memory disambiguation[J]. ACM SIGARCH Computer Architecture News, 1994, 22(2): 200−210

[91] Intel. Speculative store bypass[EB/OL]. (2018-05-21)[2023-12-30]. https://www.intel.cn/content/www/cn/zh/developer/articles/technical/software-security-guidance/advisory-guidance/speculative-store-bypass.html

[92] Van de Pol J, Smart N P, Yarom Y. Just a little bit more[C]//Proc of the 2015 Cryptographers' Track at the RSA Conf on Topics in Cryptology. Berlin: Springer, 2015: 3−21

[93] Benger N, Van de Pol J, Smart N P, et al. "Ooh Aah.. Just a Little Bit": A small amount of side channel can go a long way[C]//Proc of the 16th Int Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2014: 75−92

[94] Hund R, Willems C, Holz T. Practical timing side channel attacks against kernel space ASLR[C]//Proc of the 2013 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2013: 191−205

[95] Allan T, Brumley B B, Falkner K, et al. Amplifying side channels through performance degradation[C]//Proc of the 32nd Annual Conf on Computer Security Applications. New York: ACM, 2016: 422−435

[96] Percival C. Cache missing for fun and profit[EB/OL]. 2005[2023-12-29]. https://css.csail.mit.edu/6.858/2014/readings/ht-cache.pdf

[97] Zhang Yinqian, Juels A, Reiter M K, et al. Cross-tenant side-channel attacks in PaaS clouds[C]//Proc of the 2014 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2014:

990−1003

[98] Irazoqui G, Inci M S, Eisenbarth T, et al. Wait a minute! A fast, Cross-VM attack on AES[C]//Proc of the 17th Int Symp on Research in Attacks, Intrusions and Defenses. Berkeley, CA: USENIX Association, 2014: 299−319

[99] Neve M, Seifert J P. Advances on access-driven cache attacks on AES[C]//Proc of the 13th Int Workshop on Selected Areas in Cryptography. Berlin: Springer, 2007: 147−162

[100] Irazoqui G, Eisenbarth T, Sunar B. Cross processor cache attacks[C]// Proc of the 11th ACM on Asia Conf on Computer and Communications Security. New York: ACM, 2016: 353−364

[101] Aciiçmez O. Yet another microarchitectural attack: Exploiting I-Cache[C]//Proc of the 2007 ACM Workshop on Computer Security Architecture. New York: ACM, 2007: 11−18

[102] Gruss D, Spreitzer R, Mangard S. Cache template attacks: Automating attacks on inclusive last-level caches[C]//Proc of the 24th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2015: 897−912

[103] Kayaalp M, Abu-Ghazaleh N, Ponomarev D, et al. A high-resolution side-channel attack on last-level cache[C]//Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016[2024-02-16]. https://doi.org/10.1145/2897937.2897962

[104] Brasser F, Müller U, Dmitrienko A, et al. Software grand exposure: SGX cache attacks are practical[C]//Proc of the 11th USENIX Conf on Offensive Technologies. Berkeley, CA: USENIX Association, 2017: 11−11

[105] Schwarz M, Weiser S, Gruss D, et al. Malware guard extension: Using SGX to conceal cache attacks[C]//Proc of the 14th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2017: 3−24

[106] Irazoqui G, Inci M S, Eisenbarth T, et al. Lucky 13 strikes back[C]//Proc of the 10th ACM Symp on Information, Computer and Communications Security. New York: ACM, 2015: 85−96

[107] Depoix J, Altmeyer P. Detecting spectre attacks by identifying cache side-channel attacks using machine learning[C/OL]//Proc of the 4th Wiesbaden Workshop on Advanced Microkernel Operating Systems. 2018[2024-01-23]. https://www.betriebssysteme.org/wp-content/uploads/2018/10/WAMOS_2018_paper_12.pdf

[108] Ahmad B A. Real time detection of Spectre and Meltdown attacks using machine learning[J]. arXiv preprint, arXiv: 2006.01442, 2020

[109] Panda B. Fooling the sense of cross-core last-level cache eviction based attacker by prefetching common sense[C]//Proc of the 28th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2019: 138−150

[110] Almeida J B, Barbosa M, Barthe G, et al. Verifying constant-time implementations[C]//Proc of the 25th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2016, 16: 53−70

[111] Lee S, Shih M W, Gera P, et al. Inferring fine-grained control flow inside SGX enclaves with branch shadowing[C]//Proc of the 26th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2017: 557−574

[112] Van Bulck J, Piessens F, Strackx R. SGX-Step: A practical attack framework for precise enclave execution control[C/OL]//Proc of the 2nd Workshop on System Software for Trusted Execution. New York: ACM, 2017[2024-02-14]. https://doi.org/10.1145/3152701.3152706

[113] Intel Cor. Intel Feature documentation: Indirect branch restricted speculation[EB/OL]. (2018-01-03)[2024-01-15]. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/indirect-branch-restricted-speculation.html

[114] Intel Cor. Intel Feature documentation: Single thread indirect branch predictors[EB/OL]. (2018-01-03)[2024-01-15]. https://www.intel.cn/content/www/cn/zh/developer/articles/technical/software-security-guidance/technical-documentation/single-thread-indirect-branch-predictors.html

[115] Intel Cor. Intel Feature documentation: Indirect branch predictor barrier[EB/OL]. (2018-01-03)[2024-01-15]. https://www.intel.cn/content/www/cn/zh/developer/articles/technical/software-security-guidance/technical-documentation/indirect-branch-predictor-barrier.html

[116] Weber D, Thomas F, Gerlach L, et al. Indirect meltdown: Building novel side-channel attacks from transient-execution attacks[C]//Proc of the 28th European Symp on Research in Computer Security. Berlin: Springer Nature, 2023: 22−42

[117] Gruss D, Lipp M, Schwarz M, et al. Kaslr is dead: Long live kaslr[C]//Proc of the 9th Int Symp on Engineering Secure Software and Systems. Berlin: Springer, 2017: 161−176

[118] LWN Net. Jonathan corbet: A page-table isolation update[EB/OL]. (2018-04-25)[2024-02-21]. https://lwn.net/Articles/752621/

[119] Hertogh M, Wiesinger M, Osterlund S, et al. Quarantine: Mitigating transient execution attacks with physical domain isolation[C]//Proc of the 26th Int Symp on Research in Attacks, Intrusions and Defenses. New York: ACM, 2023: 207−221

[120] Zeller A, Gopinath R, Böhme M, et al. The fuzzing book[EB/OL]. 2019[2024-02-15]. https://publications.cispa.saarland/3120/1/index.html

[121] Hur J, Song S, Kim S, et al. SpecDoctor: Differential fuzz testing to find transient execution vulnerabilities[C]//Proc of the 2022 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2022: 1473−1487

[122] Guarnieri M, Köpf B, Morales J F, et al. Spectector: Principled detection of speculative information flows[C]//Proc of the 2020 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2020: 1−19

[123] Oleksenko O, Guarnieri M, Köpf B, et al. Hide and seek with spectres: Efficient discovery of speculative information leaks with random testing[C]//Proc of the 2023 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2023: 1737−1752

[124] Wang Guanhua, Chattopadhyay S, Gotovchits I, et al. oo7: Low-overhead defense against spectre attacks via program analysis[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2504−2519

[125] Johannesmeyer B, Koschel J, Razavi K, et al. Kasper: Scanning for generalized transient execution gadgets in the Linux kernel[C/OL]//Proc of the 2022 Network and Distributed System

Security Symp. Piscataway, NJ: IEEE, 2022[2024-02-16]. https://www.ndss-symposium.org/ndss-paper/auto-draft-247

[126] Chen Yun, Hajiabadi A, Carlson T E. GADGETSPINNER: A new transient execution primitive using the loop stream detector[C]//Proc of the 2024 IEEE Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2024: 15−30

[127] Pinto S, Santos N. Demystifying arm trustzone: A comprehensive survey[J]. ACM Computing Surveys, 2019, 51(6): 1−36

[128] AMD Cor. AMD secure encrypted virtualization (SEV)[EB/OL]. 2023[2024-01-25]. https://www.amd.com/zh-cn/developer/sev.html

[129] RISC-V International. The RISC-V instruction set manual[EB/OL]. (2017-05-07)[2024-01-25]. https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

[130] Hetterich L, Schwarz M. Branch different-spectre attacks on Apple silicon[C]//Proc of the 2022 Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2022: 116−135

[131] Wu Minjun, McCamant S, Yew P C, et al. PREDATOR: A cache side-channel attack detector based on precise event monitoring[C]//Proc of the 2022 IEEE Int Symp on Secure and Private Execution Environment Design. Piscataway, NJ: IEEE, 2022: 25−36

[132] Townley D, Arıkan K, Liu Y D, et al. Composable cachelets: Protecting enclaves from cache side-channel attacks[C]//Proc of the 31st USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2022: 2839−2856

[133] Yan Hui, Cui Chaoyuan. CacheHawkeye: Detecting cache side channel attacks based on memory events[J]. Future Internet, 2022, 14(1): 24−36

[134] Luo Mulong, Xiong Wenjie, Lee G, et al. Autocat: Reinforcement learning for automated exploration of cache-timing attacks[C]//Proc of the 2023 IEEE Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2023: 317−332

[135] Evers M, Barnes L, Clark M. The AMD next-generation “Zen 3” core[J]. IEEE Micro, 2022, 42(3): 7−12

[136] Purnal A, Turan F, Verbauwhede I. Double trouble: Combined heterogeneous attacks on non-inclusive cache hierarchies[C]//Proc of the 31st USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2022: 3647−3664

[137] Popescu M C, Balas V E, Perescu-Popescu L, et al. Multilayer perceptron and neural networks[J]. WSEAS Transactions on Circuits and Systems, 2009, 8(7): 579−588

[138] Sönmez B, Sarıkaya A A, Bahtiyar Ş. Machine learning based side channel selection for time-driven cache attacks on AES[C/OL]//Proc of the 4th Int Conf on Computer Science and Engineering. New York: ACM, 2019[2024-02-17]. https://doi.org/10.1109/UBMK.2019.8907211

[139] Ding Ruyi, Zhang Ziyue, Zhang Xiang, et al. A cross-platform cache timing attack framework via deep learning[C]//Proc of the 2022 Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2022: 676−681

[140] Tol M C, Gulmezoglu B, Yurtseven K, et al. FastSpec: Scalable generation and detection of spectre gadgets using neural embeddings[C]//Proc of the 2021 IEEE European Symp on Security and Privacy. Piscataway, NJ: IEEE, 2021: 616−632

[141] Zhang Quanjun, Fang Chunrong, Yu Bowen, et al. Pre-trained model-based automated software vulnerability repair: How far are we?[J]. IEEE Transactions on Dependable and Secure Computing, 2024, 21(4): 2507−2525

[142] Fustos J, Farshchi F, Yun H. Spectreguard: An efficient data-centric defense mechanism against spectre attacks[C/OL]//Proc of the 56th Annual Design Automation Conf. New York: ACM, 2019[2024-02-11]. https://dl.acm.org/doi/pdf/10.1145/3316781.3317914

[143] Loughlin K, Neal I, Ma Jiacheng, et al. DOLMA: Securing speculation with the principle of transient non-observability[C]//Proc of the 30th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2021: 1397−1414

[144] Werner M, Unterluggauer T, Giner L, et al. ScatterCache: Thwarting cache attacks via cache set randomization[C]//Proc of the 28th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2019: 675−692

[145] Bandara S, Kinsy M A. Adaptive caches as a defense mechanism against cache side-channel attacks[C]//Proc of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop. New York: ACM, 2019: 55−64

[146] García C P, Brumley B B. Constant-time callees with variable-time callers[C]//Proc of the 26th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2017: 83−98

[147] Qiu Pengfei, Gao Qiang, Wang Dongsheng, et al. PMU-Leaker: Performance monitor unit-based realization of cache side-channel attacks[C]//Proc of the 28th Asia and South Pacific Design Automation Conf. Piscataway NJ: IEEE 2023: 664−669

Li Yang, born in 1999. Master candidate. Student member of CCF. His main research interests include computer system security and information security.

李　扬, 1999 年生. 硕士研究生. CCF 学生会员. 主要研究方向为计算机系统安全、信息安全.

Ma Ziqiang, born in 1990. PhD, associate professor. Member of CCF. His main research interests include computer system security, blockchain application security, and network traffic identification.

马自强, 1990 年生. 博士，副教授. CCF 会员. 主要研究方向为计算机系统安全、区块链应用安全、网络流量识别.

Lin Jingqiang, born in 1978. PhD, professor. Member of CCF. His main research interests include applied cryptography, network security, and system security.(linjq@ustc.edu.cn).

林璟锵, 1978 年生. 博士，教授. CCF 会员. 主要研究方向为应用密码学、网络安全、系统安全.

Meng Lingjia, born in 1996. PhD candidate. Student member of CCF. His main research interests include system security and applied cryptography.(menglingjia@iie.ac.cn)

孟令佳, 1996 年生. 博士研究生. CCF 学生会员. 主要研究方向为系统安全、应用密码学.

Li Bingyu, born in 1990. PhD, associate professor. Member of CCF. His main research interests include cryptographic application security and network security.(libingyu@buaa.edu.cn)

李冰雨, 1990 年生. 博士, 副教授. CCF 会员. 主要研究方向为密码应用安全、网络安全.

Miao Li, born in 1986. PhD, associate professor. Member of CCF. Her main research interests include cyberspace security and information security.(limiao_smile@nxu.edu.cn)

苗　莉, 1986 年生. 博士, 副教授. CCF 会员. 主要研究方向为网络空间安全、信息安全.

Gao Fei, born in 1995. Master candidate. Student member of CCF. Her main research interests include system security and information security.(gaofei@stu.nxu.edu.cn)

高　菲, 1995 年生. 硕士研究生. CCF 学生会员. 主要研究方向为系统安全、信息安全.