

基于模型检测的处理器预测类漏洞验证方法研究

叶青瑜¹, 柯 谦², 王海霞³, 邱鹏飞², 刘 畅⁴, 汪东升^{1,4}

¹ 中关村实验室 北京 中国 100094

² 北京邮电大学可信分布式计算与服务教育部重点实验室 北京 中国 100867

³ 清华大学北京信息科学与技术国家研究中心 北京 中国 100084

⁴ 清华大学计算机科学与技术系 北京 中国 100084

摘要 长期以来, 处理器为追求高性能目标在微架构设计中引入了众多优化机制。预测执行技术是一类应用较为广泛的处理器硬件优化机制, 主要包括分支预测技术和值预测技术两种, 其在微架构中由配套设计的分支预测单元、存储缓冲区等硬件组件实现。近年来, 以 Spectre、微架构数据采样(MDS)等漏洞为代表的处理器硬件相关安全问题频频出现, 表明现代处理器的微架构预测机制存在重大安全隐患。然而处理器微架构设计复杂, 传统的测试验证方法效率低下, 难以覆盖所有漏洞, 基于形式化方法的安全验证方法成为新的研究热点。本文通过分析分支预测类漏洞和微架构数据采样类漏洞的攻击原理, 归纳了两类漏洞违反的安全属性, 提出了一种对微架构单元建模验证的方法, 该方法具有架构适应范围广、硬件单元适用种类多、验证效率高的优点。本文使用模型检测工具 NuSMV 对上述两类漏洞做了实验验证, 实验结果表明, 本文提出的预测类漏洞的验证方法能够有效验证现有的分支预测类漏洞和微架构数据采样类漏洞在特定微架构硬件单元上的存在情况。分支预测类漏洞验证实验发现的攻击序列能够覆盖现有的所有分支预测攻击类型, 并在分支预测器模型上发现了一种新的利用分支历史缓冲区泄露分支历史信息攻击方式。微架构数据采样类漏洞验证实验发现的攻击序列能够覆盖所有存在于存储缓冲区、行填充缓冲区的数据采样攻击类型, 以及一类可以绕过微架构数据采样类漏洞防御措施的攻击类型。

关键词 微架构安全; Spectre 漏洞; MDS 漏洞; 模型检测

中图法分类号 TP309.1 DOI号 10.19363/J.cnki.cn10-1380/tn.2025.07.08

Research on Verification Method of Processor Prediction Vulnerabilities Based on Model Checking

YE Qingyu¹, KE Qian², WANG Haixia³, QIU Pengfei², LIU Chang⁴, WANG Dongsheng^{1,4}

¹ Zhongguancun Laboratory, Beijing 100094, China

² Key Laboratory of Trustworthy Distributed Computing and Service, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100867, China

³ Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China

⁴ Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract For a long time, processors have introduced many optimization mechanisms in microarchitecture design to achieve high performance goals. Predictive execution technology is a widely used hardware optimization mechanism in processors, mainly including branch prediction technology and value prediction technology, which are implemented in microarchitecture through hardware components such as branch prediction unit and store buffer. In recent years, hardware-related security issues of processors, represented by Spectre and microarchitecture data sampling (MDS) vulnerabilities, have frequently occurred, indicating significant security risks in the microarchitecture prediction mechanisms of modern processors. However, processor microarchitecture design is complex, and traditional testing and verification methods are inefficient and difficult to cover all vulnerabilities, leading to the emergence of new research hotspots based on formal methods for security verification. This article analyzes the attack principles of branch prediction and microarchitecture data sampling vulnerabilities, summarizes the security properties violated by the two types of vulnerabilities, and proposes a method for modeling and verifying microarchitecture units. This method has the advantages of a wide range of architectural adaptation, applicable to various hardware units, and high verification efficiency. The article uses the model checking tool NuSMV to experimentally verify the two types of vulnerabilities, and the results show that the proposed verification method for predictive vulnerabilities can effectively verify the existence of existing branch prediction and microarchitecture data sampling vulnerabilities on specific microarchitecture hardware units. The branch prediction

通讯作者: 叶青瑜, 硕士, 助理工程师, Email: yeqy@zgclab.edu.cn.

本课题得到国家自然科学基金(No. 62372258)、国家自然科学基金区域创新发展联合基金重点项目(No. U24A20289)和北京市自然科学基金面上项目(No. 4242026)的资助。

收稿日期: 2023-08-24; 修改日期: 2024-01-08; 定稿日期: 2025-05-13

vulnerability verification experiment found attack sequences that can cover all existing branch prediction attack types and discovered a new attack method that exploits the leakage of branch history information in the branch history buffer model. The microarchitecture data sampling vulnerability verification experiment found attack sequences that can cover all data sampling attack types existing in store buffer and line fill buffer, as well as a type of attack that can bypass microarchitecture data sampling vulnerability defense measures.

Key words microarchitectural security; Spectre vulnerabilities; MDS vulnerabilities; model checking

1 引言

处理器硬件的安全是上层操作系统以及应用程序安全的基础,在硬件不可信的情况下,上层软件的安全也无法得到保证。2018年1月3日,Google公司的Project Zero等安全团队披露了重大处理器硬件安全漏洞——“幽灵^[1](Spectre)”和“熔断^[2](Meltdown)”,Intel、AMD、ARM等处理器设计厂商近20年发售的大多数处理器芯片均受到了这两个漏洞的影响。2019年5月14日,英特尔公司内部研究人员及行业伙伴公布了一组微架构数据采样类漏洞^[3-7],此类漏洞可泄露所有经过特定微架构组件的数据,其危害之大不言而喻,2011年以来发布的英特尔酷睿家族及英特尔至强家族处理器均会受到影响,尤其是云主机服务器会受到较大的冲击。此后,“幽灵”、“熔断”和微架构数据采样漏洞的各种变种层出不穷,令处理器设计厂商疲于奔命。上述漏洞表明处理器硬件设计存在缺陷,处理器硬件并不安全。针对上述安全漏洞提出一种处理器硬件单元的验证方法,是当前处理器硬件安全领域中亟待解决的重要问题之一。

长期以来,处理器设计者针对高性能的设计目标,提出了大量的硬件优化机制,并在处理器内部引入对应的硬件单元,这些优化机制与硬件单元配合大幅提升了处理器的性能。目前,预测执行技术是一类应用较为广泛的处理器硬件优化机制,主要包括两类技术:(1) 分支预测技术:当程序执行遇到分支,而分支跳转结果的提交需要多个指令周期,为避免流水线堵塞,处理器内部的分支预测器会对分支跳转结果提前做出预测,使程序跳转至预测结果处提前执行。(2) 值预测技术:由于数据的时间局部性,前一条指令产生的数据,后面的指令很可能继续使用,所以前一条指令在执行阶段结束时会将数据暂存在内部缓冲区,且不经写回阶段直接由内部缓冲区预测性地将数据转发给下一条指令,这一行为称为数据转发。当这些预测行为的执行出现错误时,处理器通过回滚操作,将流水线清空,消除相应指令对处理器架构层面的影响,但并不会回滚高速缓存的状态。上述处理器漏洞利用的就是硬件优化

机制的错误执行(分支预测错误、数据转发错误等)配合高速缓存时间侧信道攻击技术^[8-9]实现数据泄露。

幽灵、微架构数据采样类等漏洞频发,究其原因设计者对硬件安全方面的考量不足,对分支预测器、存储缓冲区、行填充缓冲区等硬件单元的安全验证工作不够充分。由于预测执行技术相关漏洞具有较为隐蔽、检测困难的特征,传统的验证方法,如手动测试和随机测试,存在不能自动化检测和不能保证覆盖所有漏洞的局限,无法从根本上解决问题。此外,处理器硬件在发行售出之后,如果发现存在漏洞,难以弥补,因而在设计时对处理器硬件单元使用形式化方法进行充分完整的验证意义重大。

本文针对分支预测漏洞和微架构数据采样类漏洞,研究了其攻击原理、安全属性、建模方法,并使用模型检验的方法对这两类漏洞进行了验证,提出了一种通用的验证方法。本文以部分技术手册、发明专利和逆向工程披露的信息为基础,对英特尔的分支预测单元和访存单元做了抽象建模。但本文的验证方法不局限于某种特定的处理器,是对处理器硬件单元抽象模型的验证方法。

在验证分支预测漏洞方面,本文结合分支预测器设计,基于信息流控制理论和条件非干涉理论,提出了分支预测漏洞违反的安全属性——预测非干涉:如果存在两个安全等级不同的使用者共享同一个分支预测器,两个使用者之间不能在预测执行的情况下产生比顺序执行更多的额外信息流。本文将分支预测攻击描述为一系列分支操作的组合,并将分支跳转结果融入分支操作属性,简化了输入,并建模了攻击者视角下的分支预测器设计模型,并实现了对分支预测漏洞的验证。

在验证微架构数据采样类漏洞方面,本文分析了攻击过程和攻击原理,研究了造成漏洞的关键成因,提出了微架构数据采样类漏洞违反的安全属性:在预测执行条件下,每个Load指令只能读取到与其地址完全匹配的数据。本文将微架构数据采样类攻击描述为一系列访存、刷新、提交等操作构成的操作序列,建模了处理器微架构设计与漏洞相关的组件行为、不同组件之间的正常数据

通路以及异常数据通路,可以对微架构数据采样类漏洞进行验证。

本文的主要贡献如下:

1. 本文分别提出了分支预测类漏洞和微架构数据采样类漏洞违反的安全属性,该安全属性可以用于对应漏洞验证和指导安全设计。
2. 本文提出了一种基于模型检测的针对处理器特定漏洞的验证框架,该框架具有架构适应范围广、硬件单元适用种类多、验证效率高的优点。
3. 本文在分支预测类漏洞的验证工作中发现了一种新的利用分支历史缓冲区泄露分支历史信息攻击方式。
4. 本文在微架构数据采样类漏洞的验证工作中,对防御措施进行了评估,发现了不同于传统微架构数据采样类漏洞的攻击路径,该攻击路径可以绕过

处理器厂商的特定防御措施。

本文将在第二章介绍上述漏洞的攻击原理、形式化方法等相关背景知识和验证工作。在第三章详细介绍验证框架的构成、两类预测执行类漏洞安全属性分析以及建模方法等内容。在第四章介绍验证结果,包括分支预测器模型中发现的泄露分支历史信息的新攻击和针对微架构数据采样类漏洞防御措施的新攻击路径。

2 背景知识及相关工作

2.1 处理器微架构的设计

为提高处理器性能,现代处理器采用了流水线技术,将指令的执行分为取值、译码、执行、访存、写回等多个阶段,并将处理器微架构分为:前端、乱序执行引擎和内存子系统三个部分,如图1所示。

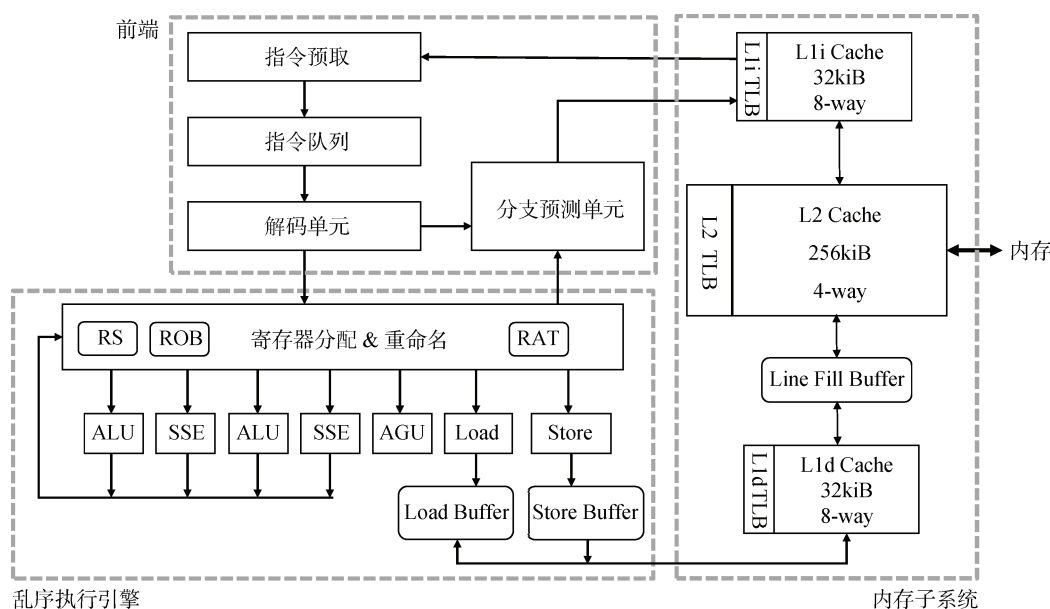


图1 典型处理器微架构

Figure 1 Typical processor microarchitecture

前端负责将架构层的指令解码为微架构层乱序执行引擎可以直接处理的微操作,并尽量保证这些微操作可以形成连续齐整的序列,因此需要前端有准确的分支预测单元和高效的解码单元。分支预测单元能够预测分支指令的跳转方向和跳转地址,如果预测正确,则极大地提升了流水线的性能,理想情况下,能够达到流水线的最大取指带宽,也即流水线的最大性能;如果预测错误,则需要清空流水线,撤销预测指令的执行,并重新从正确的地址处取指,恢复正确的执行顺序。

乱序执行引擎负责完成微操作的乱序执行和顺序提交,主要包括资源调度分配、内存访问、数据运

算等工作。由于内存访问数据的速度慢于乱序执行引擎处理数据的速度,严重影响了流水线的执行效率,因此乱序执行引擎中引入了读取缓冲区(Load Buffer)、存储缓冲区(Store Buffer)等缓冲区专用于处理访存微操作,并在存储缓冲区中设计了数据转发技术(Store-to-load Forwarding)^[10]。然而根据专利^[11]所述,某些处理器的设计中存储缓冲区的数据转发技术采用了一些更加激进的策略,在某些异常情况下,仅根据数据的部分地址匹配就会转发数据,这一策略带来了泄露信息的可能。

内存子系统负责前端和乱序引擎与内存的数据交互,为提高访问速度,内存子系统往往采用多级

高速缓存(Cache)设计, 将一级缓存分为一级指令缓存(L1i Cache)和一级数据缓存(L1d Cache), 并在一级数据缓存和二级缓存间引入了行填充缓冲区(Line Fill Buffer), 用于实现异步内存请求、写结合(Write Combine)等技术^[3]。行填充缓冲区同样存在和 Store Buffer 类似的激进转发策略。

2.2 分支预测漏洞

分支预测漏洞是一类利用分支预测单元破坏程序控制流的漏洞, 现有的分支预测漏洞主要利用分支预测单元中的模式历史表(Pattern History Buffer, PHT)、分支目标缓冲区(Branch Target Buffer, BTB)和返回栈缓冲区(Return Stack Buffer, RSB)等部件泄露信息。分支预测漏洞的本质成因在于攻击者和受害者对分支预测单元的共享使用, 导致攻击者可以控制受害者的执行流或推测受害者的执行情况, 从而泄露私密数据。分支预测类漏洞的攻击过程如图 2 所示:

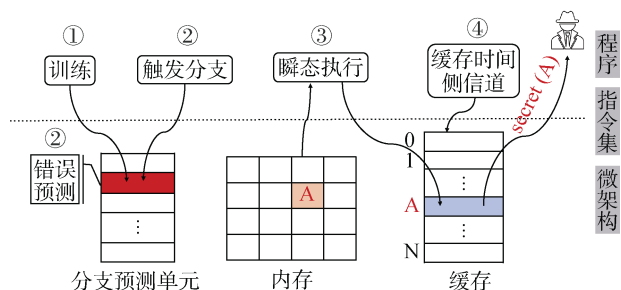


图 2 分支预测漏洞攻击流程

Figure 2 Attack flow of branch prediction vulnerability

① 训练预测器, 其目的是使分支预测器处于人为可控的状态或者使攻击者预知预测结果信息, 以便于后续攻击成功进行。PHT 的训练方法是执行大量的同方向跳转分支指令, BTB 的训练方法是执行大量同地址跳转分支指令, 而填充 RSB 时则利用其栈结构的特点, 在特定的地址处调用函数, 将指令压至栈顶。

② 触发分支的指令隐藏了一些攻击者感兴趣的信息, 例如程序的执行流或者 RSA 密钥。当触发指令执行时, 分支预测器会预测错误的跳转方向或者跳转地址, 使得触发指令后续的执行流改变执行路径, 在这种情况下, 后续指令的执行是瞬态的。

③ 攻击者会找一些可利用的恶意代码片段瞬态执行, 这些代码片段会访问位于内存的私密数据 A, 并将私密数据 A 重编码至探测数组的地址中, 同时在缓存中留下访问痕迹。

④ 攻击者程序最终通过缓存时间侧信道技术

实现对秘密数据的恢复。

2.3 微架构数据采样类漏洞

微架构数据采样(Microarchitectural Data Sampling, MDS)类漏洞是一类主要针对英特尔处理器的瞬态执行漏洞的总称, 包括 RIDL^[3]、Fallout^[4]、Zombieload^[5]漏洞等。该漏洞可以将部分微架构组件内的秘密信息泄露到缓存中, 并通过缓存侧信道实现秘密信息的窃取。微架构数据采样类漏洞的攻击流程如图 3 所示。

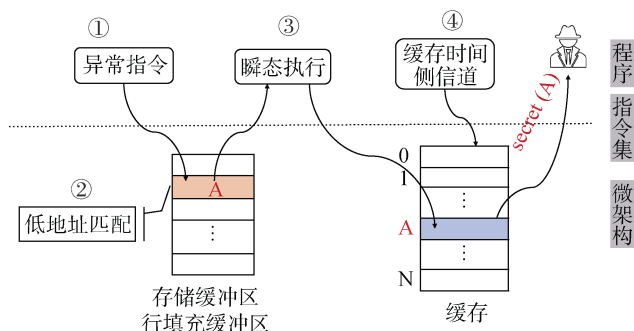


图 3 微架构数据采样类漏洞的攻击流程

Figure 3 Attack flow of microarchitecture data sampling vulnerabilities

受害者程序在执行过程中, 会将私密数据 A 遗留在存储缓冲区、行填充缓冲区等微架构数据缓冲区中。攻击者执行下述步骤, 实现数据窃取:

① 攻击者程序首先访问一些特殊的内存地址, 对这些特殊内存地址的访问会触发异常或者一些复杂的边缘情况, 例如微码辅助等, 这种情况下的访存指令称为异常访存指令(Faulting Load)或需要微码辅助的访存指令(Assisting Load)^[7]。

② 对于步骤①中的异常访存指令, 即使该异常指令访存的物理地址并未就绪, 微架构数据缓冲区也会因为低位地址的匹配进行数据转发, 此时异常指令的访存行为得到了私密数据 A。

③ 处理器并不会立即处理异常指令, 而是会等该异常指令执行至提交阶段再处理, 因此流水线中的后续指令仍能够执行, 后续指令的执行同样是瞬态执行。在瞬态执行时, 会在缓存中留下私密数据 A 相关的访问痕迹。

④ 攻击者程序最终通过缓存时间侧信道技术实现对秘密数据的恢复。

微架构数据采样类漏洞的核心原因在于攻击者通过构造异常访存或微码辅助(Microcode Assist)的访存指令在瞬态窗口中拿到未经权限检查的数据。异常指令和需要微码辅助的指令的共同点是: 都会引起处理器流水线的回退, 会触发瞬态执行窗口。

在瞬态执行窗口中, 因为异常访存指令或者微码辅助的访存指令在流水线中是无法正常完成的, 例如某些异常访存指令的物理地址是非法的, 处理器为了让该指令能够迅速完成流水线的各阶段以便提交时进行异常处理, 会根据一些部分地址匹配的结果进行数据转发。由于触发异常或者微码辅助的指令结果不会正常提交, 因此这些权限检查失误的数据只是起占位的作用, 当进行流水线回退后会被从架构中清除掉。但是这些数据在瞬态窗口中, 可以被攻击者利用, 通过重编码以及缓存时间侧信道实现私密信息泄露。

2.4 形式化方法

模型检测: 模型检验方法最初由 Clarke、Emerson、Queille 和 Sifakis 在 20 世纪 80 年代年代初各自独立提出, 它是一种算法式的验证方法, 其基本思想是: 检验一个结构是否满足一个公式要比证明公式在所有结构下均被满足容易得多, 进而面向并发系统创立了在有穷状态模型上检验公式可满足性的验证新形式^[12]。模型检验通过自动遍历系统模型的有穷状态空间来检验系统的语义模型与其性质规约之间的满足关系。模型检验中最常见的是时序模型检验或逻辑模型检验, 其系统规约大多是基于模型的规约, 使用操作语义描述系统行为, 形式模型使用自动机、标记迁移系统等; 待检验的性质是用时序逻辑描述的基于性质的规约。如果系统模型不满足性质, 模型检验算法会给出系统行为不满足性质规约的反例, 用户可以根据反例进行分析和调试; 如果模型检验未发现反例, 则系统一定满足所检验的性质。

信息流控制理论: 信息流一般指信息在系统中各个单元间的流动, 而信息流控制则是为了确保信息在系统中的流动遵循一定的控制策略。Denning^[13]提出了基于格理论的信息流模型。格是一个数学概念, 是一种特殊的偏序集。信息在系统中和一个安全标签相关联, 安全标签由安全格 $\mathcal{L}$ 确定, 并且格内的安全标签遵循偏序规约 $\subseteq$ 。基于格理论可以构建信息流控制策略, BLP 模型^[14]是最早基于格理论的保护数据私密性的信息流模型, 其核心的安全规则是“下读上写”, 即对于一个高安全级的主体和一个低安全级的客体, 高安全级的主体可以读低安全级的客体(下读), 低安全级的主体可以写高安全级的客体(上写), 信息的流动方向是自下而上。与此相对应的是 Biba^[15]模型, 是基于格理论的保护数据完整性的信息流模型, 其核心思想和 BLP 模型相反, 即“上读下写”, 低安全级的主体可以读高安全级的客体

(上读), 高安全级的客体可以写低安全级的主体(下写), 信息的流动方向为自上而下。

条件非干涉理论: Guanciale 等人^[16]在论文中提出了条件非干涉的概念, 其思想为存在两个程序的执行模型, 分别是参考模型和目标模型, 条件非干涉要求目标模型不能比参考模型泄露更多的信息。上述条件非干涉的定义允许参考模型本身泄露信息, 但是目标模型不能泄露参考模型已泄露信息之外的其他信息。

2.5 相关工作

目前, 已有的针对 Spectre 类漏洞的形式化研究主要集中在形式语义学, 即以数学为工具, 利用符号和公式, 精确地定义和严格地解释计算机程序设计语言的语义。Mcilroy^[17]建模了分支预测的语义, 其定义的指令执行的流水线阶段包括发射、执行、提交三个阶段, 主要建模了分支预测器、高速缓存的状态, 能够对 Spectre v1^[1]的攻击进行建模。Guarnieri 等人^[18]和 Cheang 等人^[19]都分别定义了预测执行的语义及与之对应的分支预测的安全属性, 并在此基础上设计了静态的分析工具, 能够用于检测程序中 Spectre v1 漏洞。Disselkoen 等人^[20]提出了基于 pomset 的语义, 能够支持描述分支预测和乱序执行, 但是也存在局限性, 无法支持对跳转目标地址的预测、对指令依赖的预测。Cauligi 等人^[21]提出的语义能够建模乱序执行、分支跳转、间接跳转的过程, 文中用语义描述了多种 Spectre 类漏洞的攻击过程, 但是依旧无法支持所有已知 Spectre 类漏洞的描述。

现有的对微架构数据采样类漏洞进行研究主要还是测试的方式, 例如通过模糊测试(Fuzz)的手段来发掘处理器微架构中存在的漏洞。Medusa^[7]即是对现有的微架构数据采样类漏洞进行分析, 通过组合漏洞的基本块生成变种样例, 从而实现潜在的微架构数据采样类漏洞变种进行挖掘。

3 处理器微架构单元的验证方法

3.1 基于模型检验的验证系统

本课题使用的验证方法的整体框架如图 4 示, 模型检验(Model Checking)^[12]是一种常见的验证方法。在模型检验的方法中, 会将待验证的系统建模成有限状态机的模型, 而安全属性则表示为有限状态机的路径规范, 模型建立后使用模型检验工具对系统的状态机上所有的路径进行搜索。如果搜索到违反安全属性的状态, 则从初始状态迁移到危险状态的路径为反例路径。

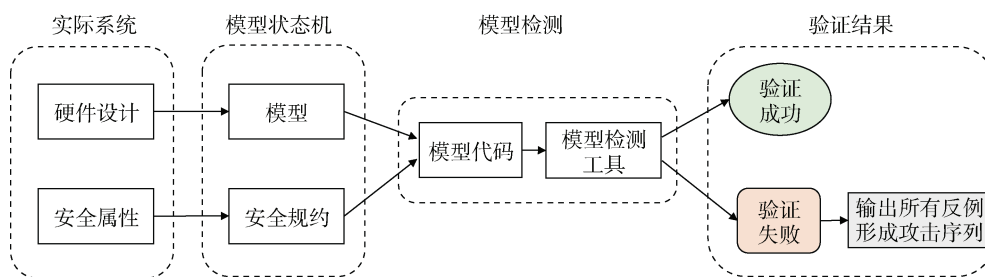


图4 基于模型检验的验证系统

Figure 4 Verification system based on model checking

具体验证流程如下:

(1) 安全属性抽取。首先我们需要针对具体漏洞或具体问题, 分析其利用过程, 找寻关键成因, 结合具体硬件设计形成安全属性。实际上, 不论是模型检测还是定理证明, 确定安全属性都是最先需要完成的工作, 也是最难解决的问题之一。

(2) 建立抽象模型。由于实际硬件设计的复杂繁琐, 注定不能直接使用实际设计用于模型检测, 必须对其抽象约简后再进行验证。因而抽象模型与实际硬件设计之间必然存在差异, 这一差异不能影响安全属性的验证。在我们的建模方法中, 我们主要考虑两方面因素: 一是仅建模关键、必需硬件单元, 省略与安全属性或漏洞攻击过程无关的硬件单元; 二是仅建模相关硬件机制, 且相关硬件机制仅实现主要功能, 忽略底层行为细节。

(3) 安全属性的形式化描述。用自然语言描述的安全属性无法被验证工具所理解, 我们使用计算树逻辑(Computation Tree Logic, CTL)将安全属性描述为逻辑公式, 即安全规约, 从而被验证工具所接纳。CTL 由路径量词和时态算子构成, 路径量词有全称量词 A 和存在量词 E 两种, 时态算子有 X (下一个状态)、 F (未来某个状态)、 G (所有状态)、 U (直到某个状态)等。在我们的验证模型下, 我们需要所有的状态均满足安全属性, 因此使用状态量词 AG 。若 P 为模型需要满足的安全属性, 则对应的安全规约为 $AG(P)$ 。安全规约是安全属性的实际实现, 在本文中表现为状态机状态迁移的路径规范, 是一种待检验模型应当遵守的状态迁移约束。3.2、3.3 节会以 PHT 的模型为例说明安全属性的形式化描述以及安全规约的形成。

(4) 模型检测与结果分析。在形成模型状态机后, 将其转换为代码描述, 输入至模型检测工具, 由模型检测工具搜索模型状态机的所有状态空间, 如果发现某个状态不满足性质规约时, 则将达到该状态的输入序列作为反例输出, 以便使用者修改设计以

满足性质规约。模型检测工具 NuSMV^[22]对于每个不满足规约的状态仅能输出一条反例, 我们修改了其内部算法, 可输出所有反例。

3.2 安全属性

3.2.1 分支预测漏洞违反的安全属性

分支预测攻击的本质是对分支方向和分支地址的攻击。现有的分支预测攻击原理表明, 主要有四种攻击方式: 攻击者控制受害者的跳转方向; 攻击者控制受害者的跳转地址; 攻击者获取受害者的跳转方向; 攻击者获取受害者的跳转地址。这四种方式在信息安全的角度下, 分别破坏了分支信息的完整性和私密性。而造成这两个安全属性被破坏的原因是不同分支预测器使用者间产生了信息的流动。

对于分支方向预测, 做出如下假设:

- 现有两个使用者分别是安全级别为高的受害者和安全级别为低的攻击者, 两者共享同一个分支预测器。
- 存在一段分支指令序列, 其在顺序执行模式下的执行情况为 $TTTTF$ (T 代表执行分支指令的跳转, F 代表分支指令的不跳转)。
- 前四条分支指令可实现填充充分支预测器的效果, 进而影响下一条指令的预测结果, 即无论分支预测器的初始状态如何, 在经过四条分支指令后必定会到达一个确定的状态。

考虑如表 1 所示的预测执行模式下的六种可能, 其中 v 和 vii 构成攻击, 两者的共性是: 两个线程的安全层级不同且共享同一个分支预测器, 两个线程之间在预测执行的情况下产生了信息流。但是区别在于 v 是攻击者通过前四条指令填充充分支预测器, 在微架构层控制篡改了受害者的跳转方向信息, 破坏的是数据完整性策略, 信息流由攻击者流向受害者, 从低安全级流向高安全级, 违反的是 Biba 模型的“下写”原则。而 vii 是受害者的信息被攻击者得知, 破坏的数据私密性策略, 信息流由受害者流向攻击者, 从高安全级流向低安全级, 违反的是 BLP

表 1 不同执行模型下的分支指令执行情况

Table 1 Execution of branch instructions under different execution models

编号	执行模型	执行情况	预测正确性	预测历史归属	被预测指令归属	构成攻击	违反的安全策略
i	顺序模型	TTTTF	-	-	-	-	-
ii	预测模型	TTTTF	✓	受害者	受害者	否	-
iii	预测模型	TTTTT	×	受害者	受害者	否	-
iv	预测模型	TTTTF	✓	攻击者	受害者	否	-
v	预测模型	TTTTT	×	攻击者	受害者	是	下写
vi	预测模型	TTTTF	✓	受害者	攻击者	否	-
vii	预测模型	TTTTT	×	受害者	攻击者	是	下读

模型的“下读”原则。分支目标预测的情况与分支方向类似, 仅是泄露的信息为分支目标地址这一区别, 但其相关的攻击所违反的安全原则一致。

由于分支预测器的共享属性, 和目前已有攻击中的数据私密性和完整性均遭到了破坏这一现实, 基于上述分析, 本文提出分支预测攻击违反的安全属性——预测非干涉, 用自然语言描述为: 存在两个安全等级不同的使用者共享同一个分支预测器, 两个使用者之间不能在预测执行的情况下产生比顺序执行更多的额外信息流。

对于任意的分支预测攻击, 本文将其定义为一组分支操作的组合。这些操作可以由拥有私密信息的受害者发起, 也可以由设法获取私密信息的攻击者发起。对于分支操作序列 Q , 由 n 个分支操作 op 组成, 如公式(3-1)所示:

$$Q = [op_1 \rightarrow op_2, \dots, \rightarrow op_n] \quad (3-1)$$

其中, 在分支预测器模型下, 每一个分支操作 op 由六个属性组成: 操作类型、操作行为类型、操作地址类型、分支跳转方向、目标地址类型和操作安全等级。

给定两个分支操作序列 Q_1 和 Q_2 , 这两个分支操作序列共享同一个分支预测器, 其中: 操作的发出者可能是低安全级的攻击者, 也可能是高安全级的受害者; Q_1 是顺序执行情况下的分支操作序列, Q_2 是预测执行情况下的分支操作序列, 两个操作序列最终达成的目的一致, 即执行结果相同(预测执行序列可能预测错误并回滚)。 Q_2 预测非干涉当且仅当 Q_2 中的任意两个操作在分支预测时不产生相较于 Q_1 的额外信息流, 形式化描述如公式(3-2)所示:

$$\forall op_i \in Q_1, \forall op_j \in Q_2, I(op) = (L \rightarrow H \cup H \rightarrow L) \quad (3-2)$$

$$I(op_j^*) \leq I(op_i)$$

其中, $I(op)$ 表示当前 op 执行时产生的信息流, $\rightarrow$ 表示信息流向, H 和 L 分别表示高低安全级, $\leq$ 代表信息量小于等于, $*$ 表示预测执行。

CTL 逻辑公式描述的安全规约如公式(3-3)所示:

$$AG(I(op_j^*) \leq I(op_i)) \quad (3-3)$$

3.2.2 微架构数据采样类漏洞违反的安全属性

微架构数据采样类漏洞的主要原因在于: 英特尔处理器流水线在瞬态窗口内访存操作在缺少部分权限检查的情况下, 依据部分地址匹配转发处理器微架构一些临时缓冲区内的私密数据。例如 MSBDS 漏洞就是利用存储缓冲区中的一个机制: Write-transient-forwarding^[4]。在存储缓冲区中, 会暂存流水线前端解码的 Store 指令相关的信息, 包括指令和数据, 直到 Store 指令提交。Load 指令在检索数据时会首先检索存储缓冲区, 查找存储缓冲区是否有地址相等的 Store 指令数据, 如果有则在存储缓冲区拿到这个数据。存储缓冲区中的地址匹配是分阶段进行匹配的, 首先进行低位地址的比对, 然后对比虚拟地址, 最后校验物理地址。当 Load 指令地址解析出现问题时, 暂时无法解析得到有效的物理地址时, 存储缓冲区会依据之前的低位地址对比的结果转发数据, 等到得到物理地址时再进行校验, 这个机制即 Write-transient-forwarding。攻击者首先构造一个会引起故障的地址, 访问这个地址即可拿到存储缓冲区里低地址相同的暂存数据, 此时 Load 指令暂时得到了一个地址不等的的数据。而在行填充缓冲区中也存在类似的部分地址匹配转发的机制, 这类机制是微架构数据采样类漏洞的主要成因。

产生微架构数据采样类漏洞的根本原因是缺乏足够的权限检查, 使得 Load 指令读到了自己本没有权限读取的数据。因而安全属性可以定义为: 读写序列中每个 Load 指令只能读取到与其地址完全匹配的

数据。在我们的模型中, 地址与数据绑定, 因为数据的私密性在处理器内部是与所在的地址相关的, 其数据本身并无私密意义。

CTL 逻辑公式描述的安全规约如公式(3-4)所示:

$$AG(load_address=data_address) \quad (3-4)$$

3.3 建模方法

3.3.1 分支预测器的建模

现代分支预测器不再由单个部件组成, 而是采用众多子部件构成, 根据部分技术手册、逆向工程以及研究综述^[23-29], 攻击者视角下的分支预测器设计如图 5 所示:

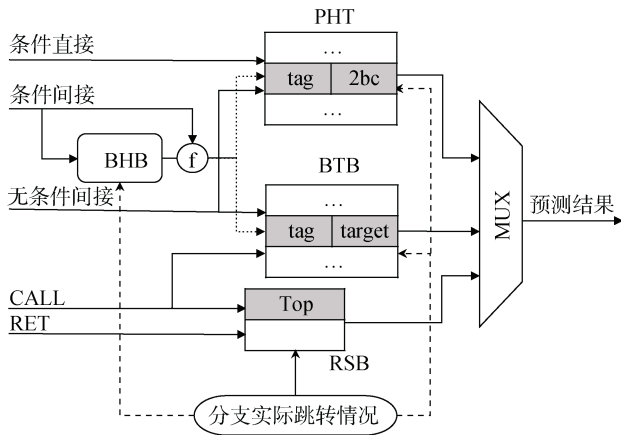


图 5 分支预测器设计

Figure 5 Branch predictor design

主要包括四大部件: 预测分支方向的 PHT、用于预测 Return 指令返回地址的 RSB、用于预测分支目标的 BTB 以及用于提高预测精度的分支历史缓冲区 (Branch History Buffer, BHB)。分支指令可根据其有无跳转条件和跳转地址的存储方式分为: 无条件直接(Unconditional Direct, UD)分支指令、条件直接(Condition Direct, CD)分支指令、无条件间接(Unconditional Indirect, UI)分支指令和条件间接(Conditional Indirect, CI)分支指令, 不同的分支指令会由不同的预测部件处理, 各类分支指令对分支预测部件的使用方式, 如表 2 所示。

分支指令地址被送往分支预测器时, BHB 会使用该地址(部分)索引分支历史表, 无条件直接分支指令的地址会直接被送往最终结果选择器; 条件间接分支指令地址(部分)会和 BHB 中对应的分支历史做哈希运算, 运算结果作为 PHT 和 BTB 的分支索引; 条件直接分支地址的地址(部分)会直接作为 PHT 的索引; 无条件间接分支指令的地址(部分)会直接作为 BTB 的索引; CALL 指令会从 BTB 中得到预测结果, RTE 指令会从 RSB 中返回预测结果, 最终的预测结

果由多路选择器决定。

表 2 各类分支指令对分支预测部件的使用

Table 2 Use of branch prediction units by various branch instructions

分支类型	索引方式	预测部件
无条件直接分支	PC	(BTB)
无条件间接分支	PC	(BTB), PHT
条件直接分支	PC	BTB, RSB
条件间接分支	BHB, PC	PHT, BTB

分支预测器的建模主要为输入的建模和上述四个部件的建模, 如图 6 所示。对模型验证时, 所有类型的抽象分支操作会被依次送往抽象模型, 抽象模型根据输入操作的类型迁移状态, PHT 输出对跳转方向的预测, BTB 和 RSB 则输出对目标地址的预测, 模型检验工具配合模型中其余状态变量的安全标签检查当前状态是否符合安全规约, 即预测错误并发生信息流动时违反安全规约, 当前状态为危险状态, 从初始状态迁移至当前状态的输入操作序列构成一条攻击路径。

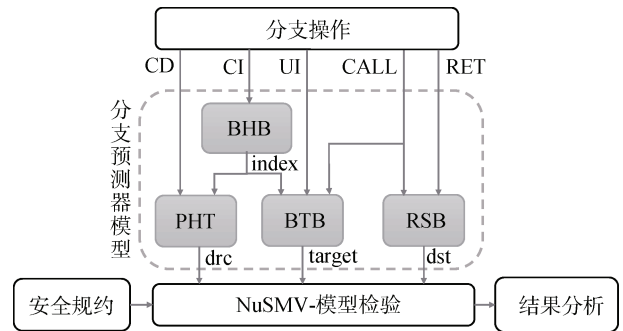


图 6 分支预测器模型

Figure 6 Branch predictor model

输入的抽象分支操作 op 由六个属性组成, 如表 3 所示。

BHB 建模为存储分支历史和输出哈希运算结果。分支历史的存储有三种类型, 高安全级分支历史、低安全级分支历史和无效历史。单独建模哈希运算过程会导致模型复杂度过高, 且哈希运算过程和设计的安全性无关, 因此运算后索引直接在 BHB 中输出。对于 a 类型的地址运算后为 b , $alias$ 类型的地址运算后同样为 b , b 类型的地址运算后为 d , d 类型的地址运算后为 d 。

PHT 建模为接受 CD 、 CI 类型的分支操作, 给出分支跳转方向的预测, 并检查预测的正确性。PHT 表项由索引、预测跳转方向和安全等级组成, 其中 CD 类型分支操作的索引由分支地址类型决定, CI 类型

表 3 抽象分支操作的属性说明

Table 3 Description of abstract branch operation properties

属性	取值情况	说明
操作类型	<i>UI</i> 、 <i>CI</i> 、 <i>CD</i> 、 <i>CALL</i> 、 <i>RET</i>	无条件间接分支、条件间接分支、条件直接分支、 <i>Call</i> 指令 ^① 、 <i>Return</i> 指令
行为类型	<i>Multi</i> 、 <i>Single</i>	<i>Multiple</i> 更改子模型的预测行为和安全级别, <i>Single</i> 不更改状态
地址类型	<i>a</i> 、 <i>alias</i> 、 <i>d</i> 、 <i>b</i>	<i>a</i> :任意地址, <i>alias</i> :与 <i>a</i> 相关地址 ^② , <i>d</i> :无关地址, <i>b</i> : <i>a</i> 和 <i>alias</i> 哈希运算后地址
跳转方向 ^③	<i>T</i> 、 <i>NT</i>	<i>T</i> :跳转, <i>NT</i> :不跳转
目标地址 ^③	<i>Normal</i> 、 <i>Gadget</i> 、 <i>Secret</i>	<i>Normal</i> :正常地址, <i>Gadget</i> :恶意代码地址, <i>Secret</i> :私密地址
安全等级	<i>H</i> 、 <i>L</i>	<i>H</i> :高安全级, <i>L</i> :低安全级

注:

① *Call*、*Return* 指令属于无条件间接分支类指令, 这两条指令会对 *RSB* 产生影响, 与其他无条件间接分支指令存在差异。

② *alias*、*a* 与 *BHB* 哈希运算后结果一致。

③ 本模型将分支的跳转方向和目标地址写入分支操作内, 这样做有两个原因: 一是模型复杂度限制无法建模乱序执行相关的部件, 直接将分支结果写入输入操作内有利于快速判断预测的正确性; 二是本模型服务于安全验证, 该差异虽然有悖于实际设计, 但不影响模型的安全性。

分支操作的索引由 *BHB* 决定。同时只有分支行为类型为 *Multi* 时, 表项内容才会更新为当前输入操作的地址类型、分支跳转方向和安全等级, *Single* 类型的分支操作不改变 *PHT* 表项的内容。

BTB 建模为接受 *CI*、*UI*、*CALL* 类型的分支操作, 给出分支目标地址的预测, 并检查预测的正确性。*BTB* 表项由索引、预测目标地址和安全等级组成, 其中 *CI* 类型分支操作的索引由 *BHB* 决定, *UI*、*CALL* 类型分支操作的索引由分支地址类型决定。*BTB* 表项内容的更新逻辑和 *PHT* 类似。

RSB 建模为接受 *CALL*、*RET* 类型的分支操作, 给出 *RET* 操作的预测地址, 并检查预测的正确性。*RSB* 表项仅由预测目标地址和安全等级组成, *CALL* 类型的分支操作会将目标地址和安全等级压栈, *RET* 类型的分支操作会从 *RSB* 栈顶取得预测地址, 并弹栈。

下面以 *PHT* 为例说明 NuSMV 工具下的建模细节, *PHT* 模型使用 Kripke 结构的实现如下:

令 *AP* 为原子命题集合, 则 *AP* 上的 Kripke 结构 *M* 是一个四元组 $M=(S, S_0, R, L)$, 令输入集为 *I*, 自动机在输入 *I* 的控制下进行状态迁移, 其中:

• *S* 是状态的有限集合, 存在三个状态变量: 索引标签(*tag*)与抽象分支操作地址类型相关, 有 5 种取值 [*inv*, *a*, *alias*, *b*, *d*], *inv* 表示初始空值; 预测分支跳转方向(*drc*)有 2 种取值 [*T*, *NT*]; 安全级别(*sec_level*)有 2 种取值 [*H*, *L*], 共有 17 种状态, 取值集合如表 4 所示:

表 4 PHT 状态集合

Table 4 PHT state set

状态标识	索引标签	跳转方向	安全级别
<i>s0</i>	<i>inv</i>	-	-
<i>s1</i>	<i>a</i>	<i>T</i>	<i>H</i>
<i>s2</i>	<i>alias</i>	<i>T</i>	<i>L</i>
<i>s3</i>	<i>b</i>	<i>NT</i>	<i>H</i>
<i>s4</i>	<i>d</i>	<i>NT</i>	<i>L</i>
...	...	...	...
<i>s16</i>	<i>d</i>	<i>T</i>	<i>L</i>

• 输入集 *I* 是输入操作的集合, 由五个属性构成:

操作类型(*op_type*): *CD*、*CI*

行为类型(*op_bhv*): *Multi*、*Single*

地址类型(*op_adr*): *a*、*alias*、*b*、*d*

跳转方向(*op_drc*): *T*、*NT*

安全级别(*op_level*): *H*、*L*

PHT 是负责分支方向的预测, 与目标地址无关, 因此输入操作中没有目标地址的属性。

• 初始状态: $S_0=\{s_0\}$ 。

• *R* 是迁移关系, 迁移规则如下:

$$PHT.tag \xrightarrow{I.op_type[CD]} I.op_adr;$$

$$PHT.tag \xrightarrow{I.op_type[CI]} BHB.tag;$$

$$PHT.drc \xrightarrow{I.op_bhv[Multi]} I.op_drc;$$

$$PHT.sec_level \xrightarrow{I.op_bhv[Multi]} I.op_level;$$

其中, []表示取值情况, *BHB.tag* 为 *BHB* 经哈希运算输出的索引值。

• AP 是原子命题集, 即符合安全属性预测非干涉的安全规约, PHT 模型存在一个规范:

$$P \equiv !(tag = (op_tag \setminus BHB.tag) \& sec_level \neq op_level \& drc \neq op_drc)$$

• L 为标记函数, 表示该状态下原子命题为真的集合:

$$L = \{si | L(si) = P\}$$

NuSMV 工具下检查的安全规约为:

$$SPEC \quad AG(!(tag = (op_tag | BHB.tag) \& sec_level \neq op_level \& drc \neq op_drc))$$

3.3.2 微架构访存组件的建模

微架构数据采样类漏洞中涉及的微架构组件较多, 根据专利以及漏洞攻击过程中的逆向工程, 主要泄露信息的组件为存储缓冲区(Store Buffer)和行填充缓冲区(Line Fill Buffer)。存储缓冲区用于保存前端解码得到的 Store 指令, 行填充缓冲区位于一级缓存和二级缓存中间, 用于保存一级缓存缺失的请求。

前端解码得到 Load 指令后, 会依次访问不同层次存储。首先在存储缓冲区中检索是否存在数据转发, 命中后读取数据并停止检索; 否则访问一级缓存, 同样进行检索, 命中后读取数据并停止检索; 如果缺失, 则在行填充缓冲区检索是否存在相同地址的条目, 如果还没有命中, 则在行填充缓冲区分配条目, 进行二级缓存以及下级存储的检索。访问的存储顺序依次是存储缓冲区、一级缓存、行填充缓冲区、二级缓存、三级缓存及主存。

本文建模了这些微架构组件正常匹配转发流程和异常指令的部分地址匹配转发机制。图 7 所示为访存层次模型, 输入的指令序列会根据类型分发到不同的组件上, 各组件会依据指令的类型进行状态变迁来模拟数据的存储过程, 并输出访存结果, 最终模型会依据安全规范进行安全属性的检查。

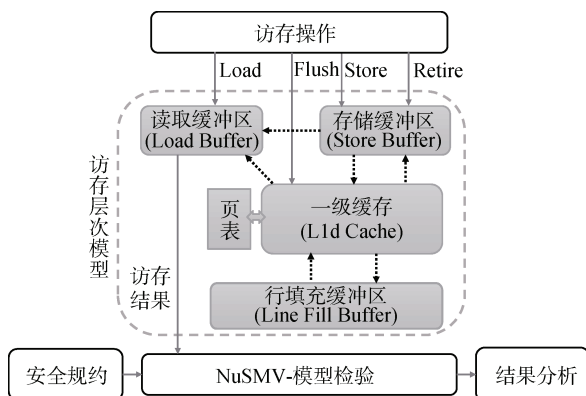


图 7 访存层次模型

Figure 7 Fetching hierarchical model

输入的抽象访存操作 op 由四个属性组成, 如表 5 所示:

表 5 访存操作的属性说明

Table 5 Attribute description of fetch operation

属性	取值情况	说明
操作类型	Store、Load、Retire、Flush	存储操作、访存操作、模拟指令 retire 的操作、刷新操作
地址类型 ^①	NC Normal Fault	非标准地址、正常地址、页表触发异常的地址
写回类型 ^②	WB、WT	写回类型、直写类型
偏移类型	a、alias、d	a:任意偏移, alias:与偏移低位相同, d:无关地址

注:

① 根据地址类型的异常分为非标准地址与标准地址, 而标准地址分为正常地址以及页表项触发异常的地址。

② 写回类型与页内偏移也会影响漏洞攻击的表现, 且与是否触发异常属于不同层面的属性。

不同组件的行为建模如下所示:

存储缓冲区: 接受 Store 操作, 暂存 Store 操作数据; 接受 Retire 操作, 将对应 Store 操作的数据由存储缓冲区写入下级内存; 接受 Load 操作, 根据地址检索数据(如果正常的指令地址完全匹配, 则进行全地址匹配的数据转发; 异常 Load 操作或触发微码辅助的 Load 操作, 依据低 12 地址匹配结果进行数据转发)。

读取缓冲区: 接受 Load 操作, 向不同层级存储发送检索请求; 最终依据不同层级存储的访存情况, 输出访存数据。

一级数据缓存: 接受 Retire 操作, 写回(Write-Back)的数据由存储缓冲区写入缓存; 接受 Load 操作, 根据地址检索数据, 正常指令中地址完全匹配的进行数据转发; 接受 Flush 操作, 将对应数据由一级缓存写入行填充缓冲区。

行填充缓冲区: 接受 Retire 操作, 写直(Write-Through)的数据由存储缓冲区写入行填充缓冲区; 接受 Load 操作, 根据地址检索数据(如果正常的指令地址完全匹配, 则进行数据转发; 异常 Load 操作或触发微码辅助的 Load 操作, 依据低 12 地址匹配结果进行数据转发); 接受 Flush 操作, 接受数据由一级缓存写入行填充缓冲区。

二级及以下缓存: 接受 Load 操作, 由于该层级为模型建立的最后内存层级, 因此到该层级的 Load 操作默认会通过访存得到与地址完全匹配的数据。

4 验证结果

本文对第 3 章所述的分支预测器和微架构数据

采样类漏洞进行了验证, 使用了基于 SMV(Symbolic Model Verification)^[28]建模语言对上述行为模型进行了建模, 并使用经过修改的模型检验工具 NUSMV (version 2.6.0)对模型状态空间进行了遍历, 分支预测模型的状态数为 62208, 微架构数据采样类漏洞模型的状态数为 35937, 验证的时间开销均在 3 秒内。实验平台的配置为 8 核、16 线程、1.8GHZ 的 AMD Ryzen 7 4800U CPU, 16GB 内存和 Windows 11 22H2 版本的操作系统。

4.1 分支预测器的验证结果

4.1.1 验证结果

对分支预测器模型的验证结果如表 6 所示。表中所列漏洞为本文调研的 2018 年至今的基于瞬态执行的分支预测攻击漏洞, 这些漏洞在触发时均利用了分支预测误用缺陷。本文对分支预测模型的验证结果覆盖了上述所有漏洞, 其中需特别说明的是 BranchSpectre^[30]不再使用缓存时间侧信道泄露私密数据, 而是基于 PHT 的表项冲突, 利用 PHT 作为隐蔽信道传送私密数据, 但训练误预测的原理是和 Spectre V1 一致的。

表 6 分支预测模型攻击覆盖情况

Table 6 Branch prediction model attack coverage

攻击	部件	攻击	部件
Spectre V1 ^[1]	PHT ✓	BranchSpectre ^[30]	PHT ✓
Spectre V2 ^[1]	BTB ✓	SmoTherSpectre ^[31]	BTB ✓
SpecROP ^[32]	BTB RSB ✓	SpectreRewind ^[33]	PHT ✓
NetSpectre ^[34]	PHT ✓	ExSpectre ^[35]	BTB ✓
BlindSpectre ^[36]	PHT ✓ BTB ✓	SpectreBHI ^[37]	BHB ✓ BTB ✓
SplitSpectre ^[38]	PHT ✓	BranchSkipTojans ^[39]	RSB ✓ BTB ✓
SpecHammer ^[40]	PHT ✓	SpectreRSB ^[41]	RSB ✓
SGXPectre ^[42]	BTB ✓	Ret2spec ^[43]	RSB ✓

分支预测器模型结合了 BHB、PHT、BTB 和 RSB 四个模型, 实验结果共发现 2 步攻击 200 个和 3 步攻击 18828 个, 除上述已覆盖的攻击外, 本实验还发现了一类利用分支历史缓冲区(BHB)泄露分支历史信息的攻击方式, 这在以往的攻击中没有出现过, 是一种全新的攻击方式。下面将详细介绍分支预测模型中发现的攻击序列的攻击原理。表 7 范例别为各子模型泄露信息的攻击序列。

攻击序列由攻击者或者受害者发出的分支操作组成, 其中 A 代表攻击者, 其安全等级为 L , V 代表受害者安全等级为 H 。每个分支操作右上角的值代表分

支地址与分支跳转情况, 右下角的值代表分支操作类型和行为类型, 其取值和模型输入的抽象分支操作的取值相同。例如, $A_{CD: Multi}^{a: NT}$ 表示 L 安全级的攻击者发出了多条($Multi$)条件直接(CD)分支指令, 其地址为 a 、实际跳转情况为不跳转(NT)。

表 7 分支预测器不同子模型攻击序列

Table 7 Branch predictor different submodel attack sequences

例 1: 分支预测器攻击序列

1. $A_{CD: Multi}^{a: NT} \rightarrow V_{CD: Single}^{a: T}$
2. $A_{UI: Multi}^{a: Gadget} \rightarrow V_{CI: Single}^{alias: Normal}$
3. $V_{CALL: Single}^{Normal} \rightarrow V_{CALL: Multi}^{Gadget} \rightarrow V_{RET: Single}^{Normal}$
4. $A_{UI: Multi}^{Normal} \rightarrow V_{CI: Multi}^{alias: NT / Secret} \rightarrow A_{UI: Single}^{Normal}$

攻击序列 1: 攻击者使用多条 CD 类型的分支操作训练 PHT 的预测方向为不跳转, 并使 PHT 的安全级变为 L ; 受害者发出 CD 类型的实际跳转的分支操作时, 此时 CD 类型的分支操作安全级为 H 。PHT 预测不跳转, 形成错误信息流($L \rightarrow H$), 触发错误预测, 受害者跳转向错误方向瞬态执行, 造成信息泄露。

攻击序列 2: 攻击者使用 UI 类型的多条分支操作填充 BTB 预测目标地址为 $Gadget$, 使 BTB 安全等级为 L ; 受害者发出 UI 类型的分支操作, 此时 UI 类型的分支操作安全级为 H 。BTB 表项预测目标地址为 $Gadget$, 形成错误信息流($L \rightarrow H$), 触发错误预测, 受害者在高权限下执行恶意代码, 造成信息泄露。

攻击序列 3: 受害者执行 $CALL$ 分支操作在 RSB 中压入了正常返回地址, 使 RSB 安全等级为 H ; 攻击者使用多条 $CALL$ 类型的分支操作覆盖 RSB 的预测目标地址为 $Gadget$, 使 RSB 安全等级为 L , 并从 BTB 获取 $CALL$ 操作的无关目标地址。当受害者发出 RET 类型的分支操作, 此时 RET 类型的分支操作安全级为 H 。RSB 返回的目标地址为 $Gadget$, 形成错误信息流($L \rightarrow H$), 触发错误预测, 受害者在高权限下执行恶意代码, 造成信息泄露。

攻击序列 4: 攻击者使用多条 UI 分支操作填充 BTB, 使 BTB 安全等级为 L ; 受害者执行多条 CD 类型的分支操作填充 PHT 和 BHB, 使 PHT 和 BHB 的安全等级为 H ; 攻击者使用 CI 类型的分支操作访问 BTB, 由于 CI 类型的分支操作的索引标签由分支地址(L)和 BHB 值(H)经哈希计算产生, 因此索引标签的安全级为 H , CI 分支操作的安全级同样升级为 H , 并从 L 安全级的 BTB 获取了错误目标地址, 因此产生了($H \rightarrow L$)的信息流, 造成信息泄露。

攻击序列 4 是一种利用 BHB 泄露信息的新攻击类型, 一种在实际分支预测器上可能的利用方式如图 8 所示, 共有布局、填充、探测三个阶段:

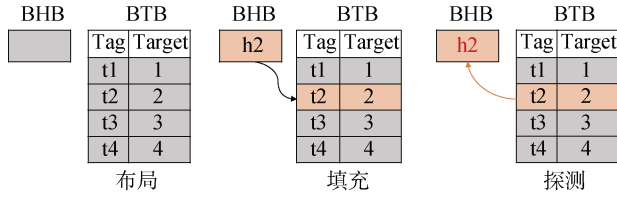


图 8 泄露 BHB 信息的新攻击

Figure 8 New attack to leak BHB information

布局阶段: 低安全级攻击者使用 *UI* 类型的分支操作, 填充 BTB 的所有表项, 并控制每个 BTB 表项的内容不一致, 即 BTB 每个表项的预测地址能够由攻击者区分;

填充阶段: 由高安全级受害者执行部分 *CD* 类型的分支操作, 该类型的分支操作会更改 PHT 和 BHB 的内容, 将分支跳转信息 *h2* 遗留在 BHB 中;

探测阶段: 攻击者执行 *CI* 类型的操作, BHB 的值 *h2* 和分支指令地址 *PC* 计算的索引标签为 *t2*, 攻击者得到的预测地址为 *B*, 此时攻击者可根据预测地址知道索引标签为 *t2*, 最后攻击者根据分支指令地址 *PC* 即可反推出受害者的分支历史 *h2*, 达成攻击目的。这样的攻击在以往的攻击方式中没有被发现, 是一种新的攻击类型。

4.1.2 防御措施评估

针对分支预测攻击已有多种防御措施被提出, 部分防御措施已应用至实际处理器设计, 现有的硬件层防御思路主要有隔离策略和刷新策略两种。然而 SpectreBHI 等工作表明, 现有的防御措施仍然存在被攻击的可能。

隔离策略的实现基础是对不同进程的做权限标记, 不同权限级的用户不共享分支预测器内的预测信息, 以实现高权限级用户和低权限级用户的互不干涉。在隔离策略下, 工业界提出了 IBRS、eIBRS、STIBP 等防御机制, 逆向工作表明英特尔考虑到性能影响, 权限分级的措施未在 BHB 中应用。实现刷新策略是在每次发生不同权限级的上下文切换时, 将分支预测器内的表项信息清空, 以阻止不同权限级进程的互相干涉。在清空策略下, 工业界提出了 IBPB 的防御机制, 通过清空整个 BTB, 防止 BTB 的预先填充被后续分支误用。

基于上述两种防御思路, 本文在分支操作的输入和 BTB 模型的表项中新加入了权限等级属性, 有 *Kernel* 和 *User* 两种取值。在隔离策略下, 仅当输入

分支操作的权限等级和 BTB 表项的权限等级一致时, BTB 才会做出预测。在清空策略下, 当前后两条输入的分枝操作权限等级不同时, 会清空 BTB 以及 PHT 的表项内容。与英特尔的应用方式一致, 未对 BHB 实现权限分级。

验证结果发现基于刷新的防御策略能有效防御分支预测攻击, 但基于隔离的防御策略仍然存在被攻击的可能。如图 9 所示, 步骤①中攻击者在内核代码中寻找某些可以利用的恶意代码片段, 而后多次调用执行以填充 BTB 表项, 此时 BTB 表项标记的权限等级为 *K(kernel)*。步骤②中攻击者在 *U(user)* 权限等级下执行无关分支, 以填充 BHB, 使得 BHB 产生的索引指向步骤①中填充的 BTB 表项。步骤③受害者执行 *K* 权限等级分支操作, BHB 产生的索引指向预期的 BTB 表项, 由于 BTB 表项和分支操作的权限等级相同, BTB 做出错误预测, 受害者执行恶意代码路径泄露数据。这种攻击方式和 SpectreBHI^[40] 的攻击原理是一致的。

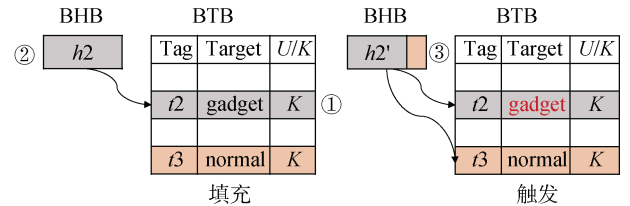


图 9 隔离策略下的防御漏洞

Figure 9 Defense vulnerabilities under isolation policy

4.2 微架构数据采样类漏洞验证结果

4.2.1 验证结果

基于微架构数据采样类漏洞的行为模型, 验证结果得到了 MSBDS 和 MFBDS 这两类微架构数据采样类漏洞的三种攻击路径, 其中包含两步攻击序列数 162 个, 三步攻击序列数 135 个, 四步攻击序列数 6634 个。验证得到的攻击序列覆盖了 Zombieload 和 Fallout 的多种变种, 但未覆盖 2023 年 8 月 10 日发布的 Downfall^[44] 漏洞, 具体覆盖情况如表 8 所示。

表 8 访存层次模型攻击覆盖情况

Table 8 Access hierarchy model attack coverage

步数	序列数	利用部件	覆盖攻击
2 步	162	Store Buffer	Fallout ^[4] ZombieLoad v2 ^[5]
3 步	135	Line Fill Buffer	Fallout ZombieLoad v1/v2/v3 ^[5]
4 步	6634	Line Fill Buffer	Fallout ZombieLoad v1/v2/v3

攻击序列示例如表 9 所示:

表 9 微架构数据采样类漏洞不同路径序列

Table 9 Different path sequences of microarchitecture data sampling vulnerabilities

范例 2: 攻击序列示例

1. $Store_{Normal}^{wba} \rightarrow Load_{nc}^{wba}$
2. $Store_{Normal}^{wfa} \rightarrow Retire_{Normal}^{wfa} \rightarrow Load_{fault}^{wfb}$
3. $Store_{Normal}^{wba} \rightarrow Retire_{Normal}^{wba} \rightarrow Flush_{Normal}^{wba} \rightarrow Load_{fault}^{wfb}$

这三种攻击路径根据所需要的操作步数进行划分, 分别为两步攻击、三步攻击和四步攻击, 如图 10 所示, 下面将分别介绍这三种攻击路径。

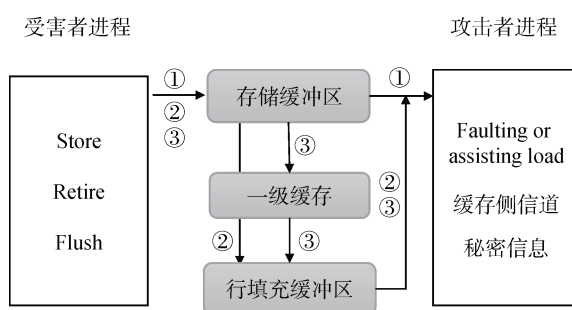


图 10 微架构数据采样类漏洞验证结果

Figure 10 Verification results of microarchitecture data sampling vulnerabilities

攻击序列 1: 如图 10 中 1 号路径所示。受害者程序进行存储操作并将秘密数据暂存在存储缓冲区中。此时攻击者程序构造异常访存指令或需要微码辅助的访存指令, 并在瞬态窗口内通过存储缓冲区的部分地址匹配转发路径窃取到了秘密数据。在该攻击路径下, 受害者程序存储指令与攻击者程序的访存指令的地址页内偏移相等。这种攻击路径对应了 MSBDS 攻击。

攻击序列 2: 如图 10 中 2 号路径所示。受害者程序进行存储操作, 该存储操作完成后进行提交 (retire), 由存储缓冲区直接写入行填充缓存区, 此时攻击者同样构造异常访存指令或需要微码辅助的访存指令, 在行填充缓冲区窃取了受害者程序保存的秘密数据。在该攻击路径下, 受害者程序指令地址的写回属性为写直属性, 且其地址与攻击者程序的访存指令地址低六位相等, 即缓存行内偏移相等。这种攻击路径对应了 MFBDS 攻击。

攻击序列 3: 如图 10 中 3 号路径所示。受害者程序进行存储操作, 该存储操作完成后进行提交, 由存储缓冲区写入一级缓存, 再由一级缓存写入行填充缓冲区。此时攻击者通过构造异常访存指令或

需要微码辅助的访存指令, 在行填充缓冲区窃取了受害者程序保存的秘密数据。在这种攻击路径下, 受害者程序指令地址的写回属性为写回, 且其地址与攻击者访存指令地址的低六位相等。这种攻击路径也对应 MFBDS 攻击。

这三种攻击路径, 完整地覆盖了触发 MFBDS 和 MSBDS 这两类微架构数据采样类漏洞变种的所有攻击路径。此外, 除了对已有的微架构数据采样类攻击的路径进行了验证, 还可以对该漏洞的部分防御措施进行评估。

4.2.2 防御措施评估

针对微架构数据采样类漏洞, 英特尔厂商发布的防御措施, 主要包括两种: 第一种防御措施从根本上解决了问题, 即从 Whiskey Lake 架构开始, 英特尔从架构层面对微架构数据采样类漏洞进行了防御。该防御措施具体为: 当处理器流水线中预测执行的访存指令出现异常或需要微码辅助的情况时, 微架构组件转发数据零给该访存指令作为占位, 不再根据部分地址匹配的结果进行数据转发作为占位。但之前的处理器无法通过架构层面的设计进行防御, 因此英特尔第二种防御措施是通过微码更新即打补丁的方式进行防御措施的增加。该防御措施为: 在特定场景下, 当处理器由高特权级进程切换到低特权级进程时, 处理器会使用 `verw` 指令对微架构的临时缓冲区进行清空, 从而避免临时缓冲区的数据发生泄露。

第一种防御措施从根本上防御了该漏洞, 但是无法对现有的处理器进行防御。基于上述的微架构数据采样类漏洞的行为模型, 可以实现对英特尔厂商所提出的第二种防御措施的验证评估。在现有的行为模型基础之上, 增加指令操作的属性, 用于表示操作的特权级即区分该指令是属于攻击者程序还是受害者程序。在增加该属性后, 可以建模第二种防御措施, 即当指令操作特权级发生变化时(由受害者程序切换到攻击者程序时), 对临时缓冲区进行清空。在建了防御措施后, 利用模型检验工具对状态空间进行检索, 查找是否存在反例, 即可实现对防御措施的验证评估。

验证结果发现在四步攻击中, 攻击者程序可以通过将数据从一级缓存中驱逐至行填充缓冲区, 从而绕过清空缓冲区的防御措施, 如图 11 所示。传统的微架构数据采样类漏洞只能通过攻击泄露临时缓冲区的数据, 而在该攻击中攻击者可以主动将数据填充进临时缓冲区, 实现了一种新的攻击路径。不过该漏洞已经在现有的工作中通过测试的方法被发现, 并被命名为 `CacheOut`^[45]攻击。但是通过建立微架构数据采样类漏洞的行为模型, 也可以实现对防御措

施的验证,发现这种区别于传统微架构数据采样类漏洞的攻击路径。

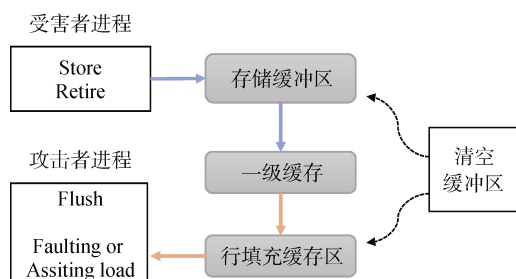


图 11 CacheOut 攻击流程
Figure 11 CacheOut attack process

此外,在此攻击中,攻击者程序窃取数据的范围也得到了提升。传统的微架构数据采样类漏洞只能窃取临时缓冲区的数据,该攻击通过构造不同的驱逐集可以将一级缓存内的数据驱逐至行填充缓冲区并实现秘密信息的窃取,而一级缓存的容量是远大于行填充缓冲区的,因此相比于之前攻击者窃取秘密数据的能力也有所增强。

5 结论

本文阐述了基于模型检验对瞬态漏洞验证的方法,主要包括验证系统、行为模型实现方式和验证实验结果的内容。

针对分支预测类攻击,本课题的实验结果覆盖了目前现有的所有攻击方式,说明目前的分支预测器设计均存在被攻击的可能。此外,本实验还发现了一种新的利用分支历史缓冲区泄露分支跳转历史的攻击方式,该攻击方式在模型上可以攻击成功,对于实际的分支预测器理论上存在着攻击成功的可能,未来的工作有待进一步验证其可行性。

针对微架构数据采样类漏洞,验证结果表明该方法对微架构数据采样类漏洞中的 MFBDS 和 MSBDS 的攻击路径进行了全面的枚举。此外,还实现了对微架构数据采样类漏洞防御错的验证评估。针对英特尔的第二种防御措施,对微架构数据采样类漏洞的行为模型进行修改后验证,发现了与传统的微架构数据采样类漏洞不同的攻击路径,该路径可以绕过英特尔的第二种防御措施。

参考文献

[1] Kocher P, Horn J, Fogh A, et al. Spectre Attacks: Exploiting Speculative Execution[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 1-19.
[2] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading kernel memory from user space[J]. *Communications of the ACM*, 2020,

63(6): 46-56.
[3] Van Schaik S, Milburn A, Österlund S, et al. RIDL: Rogue in-flight data load[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 88-105.
[4] Canella C, Genkin D, Giner L, et al. Fallout: Leaking Data on Meltdown-Resistant CPUs[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 769-784.
[5] Schwarz M, Lipp M, Moghimi D, et al. ZombieLoad: Cross-privilege-boundary data sampling[C]. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 753-768.
[6] Van Bulck J, Moghimi D, Schwarz M, et al. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 54-72.
[7] Moghimi D, Lipp M, Sunar B, et al. Medusa: Microarchitectural data leakage via automated attack synthesis[C]. *Proceeding of the 29th USENIX Security Symposium*, 2020.
[8] Yarom Y, Falkner K. Flush+ reload: A high resolution, low noise, L3 cache side-channel attack[C]. *23rd {USENIX} Security Symposium*, 2014: 719-732.
[9] Osvik D A, Shamir A, Tromer E. Cache Attacks and Countermeasures: The Case of AES[C]. *Topics in Cryptology*, 2006: 1-20.
[10] Advanced Micro Devices, Inc. Store-to-load forwarding[P]. US Patent. US20140181482A1, Dec. 20, 2012.
[11] Sebastien Hily, Zhongying Zhang, and Per Hammarlund. Resolving False Dependencies of Speculative Load Instructions. US Patent. 2009. 7.603,527
[12] Clarke E M. Model checking[C]. *Foundations of Software Technology and Theoretical Computer Science*, 1997: 54-56.
[13] Denning D E. A Lattice Model of Secure Information Flow[J]. *Communications of the ACM*, 1976, 19(5): 236-243.
[14] Bell D E. Looking back at the Bell-La Padula Model[C]. *21st Annual Computer Security Applications Conference*, 2005: 15pp.-15351.
[15] Biba K J. Integrity considerations for secure computer systems[R]. MITRE CORP BEDFORD MA, 1977.
[16] Guanciale R, Balliu M, Dam M. Inspectre: Breaking and fixing microarchitectural vulnerabilities by formal analysis[C]. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020: 1853-1869.
[17] McIlroy R, Sevcik J, Tebbi T, et al. Spectre is here to stay: An analysis of side-channels and speculative execution[J]. arXiv preprint arXiv:1902.05178, 2019.
[18] Guarnieri M, Köpf B, Morales J F, et al. Spectector: Principled Detection of Speculative Information Flows[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 1-19.
[19] Cheang K, Rasmussen C, Seshia S, et al. A Formal Approach to Secure Speculation[C]. *2019 IEEE 32nd Computer Security Foundations Symposium*, 2019: 288-28815.
[20] Disselkoen C, Jagadeesan R, Jeffrey A, et al. The Code that never Ran: Modeling Attacks on Speculative Evaluation[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 1238-1255.
[21] Cauligi S, Disselkoen C, Gleissenthall K V, et al. Constant-Time Foundations for the New Spectre Era[C]. *The 41st ACM SIGPLAN*

- Conference on Programming Language Design and Implementation*, 2020: 913-926.
- [22] Cimatti A, Clarke E, Giunchiglia E, et al. NuSMV 2: An Open-Source Tool for Symbolic Model Checking[C]. *Computer Aided Verification*, 2002: 359-364.
- [23] Jimenez D A, Lin C. Dynamic Branch Prediction with Perceptrons[C]. *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2002: 197-206.
- [24] Liu C, Yang Y, Li H R, et al. A Survey of Branch Prediction Attacks on Modern Processors[J]. *Chinese Journal of Computers*, 2022, 45(12): 2475-2509.
(刘畅, 杨毅, 李昊儒, 等. 处理器分支预测攻击研究综述[J]. *计算机学报*, 2022, 45(12): 2475-2509.)
- [25] Guide P. Intel® 64 and IA-32 Architectures Software Developer Manuals[S]. Volume 3B: System Programming Guide, Part, 2011, 2(11).
- [26] Holding A. ARM Architecture Reference Manual for ARMv8-Architectural profile[S]. ARM Whitepaper, 2019.
- [27] Uzelac V, Milenkovic A. Experiment Flows and Microbenchmarks for Reverse Engineering of Branch Predictor Structures[C]. *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009: 207-217.
- [28] Milenkovic M, Milenkovic A, & Kulick J. Microbenchmarks for determining branch predictor organization[M]. *Software: Practice and Experience*, 2004, 34(5): 465-487.
- [29] Burch J R, Clarke E M, Long D E, et al. Symbolic Model Checking for Sequential Circuit Verification[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1994, 13(4): 401-424.
- [30] Chowdhury M H I, Yao F. Leaking Secrets through Modern Branch Predictors in the Speculative World[J]. *IEEE Transactions on Computers*, 2022, 71(9): 2059-2072.
- [31] Bhattacharyya A, Sandulescu A, Neugschwandtner M, et al. Smotherspectre: exploiting speculative execution through port contention[C]. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 785-800.
- [32] Bhattacharyya A, Marin A S, Koruyeh E M, et al. SpecROP: Speculative exploitation of ROP chains[C]. *23rd International Symposium on Research in Attacks, Intrusions and Defenses*, 2020.
- [33] Fustos J, Bechtel M, Yun H. SpectreRewind: Leaking Secrets to Past Instructions[C]. *The 4th ACM Workshop on Attacks and Solutions in Hardware Security*, 2020: 117-126.
- [34] Schwarz M, Schwarzl M, Lipp M, et al. NetSpectre: Read Arbitrary Memory over Network[M]. *Computer Security – ESORICS 2019*. Cham: Springer International Publishing, 2019: 279-299.
- [35] Wampler J, Martiny I, Wustrow E. ExSpectre: Hiding Malware in Speculative Execution[C]. *NDSS*, 2019.
- [36] Göktas E, Razavi K, Portokalidis G, et al. Speculative probing: Hacking blind in the Spectre era[C]. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020: 1871-1885.
- [37] Barberis E, Frigo P, Muench M, et al. Branch History Injection: On the Effectiveness of Hardware Mitigations Against {Cross-Privilege} Spectre-v2 Attacks[C]. *31st USENIX Security Symposium*, 2022: 971-988.
- [38] Mambretti A, Neugschwandtner M, Sorniotti A, et al. Speculator: a tool to analyze speculative execution attacks and mitigations[C]. *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019: 747-761.
- [39] Zhang T, Koltermann K, Evtushkin D. Exploring branch predictors for constructing transient execution trojans[C]. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020: 667-682.
- [40] Tobah Y, Kwong A, Kang I, et al. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks[C]. *2022 IEEE Symposium on Security and Privacy*, 2022: 681-698.
- [41] Koruyeh E M, Khasawneh K N, Song C Y, et al. Spectre Returns! Speculation Attacks Using the Return Stack Buffer[C]. *IEEE Design & Test*, 2024: 47-55.
- [42] Chen G X, Chen S C, Xiao Y, et al. SgxPectre: Stealing Intel Secrets from SGX Enclaves via Speculative Execution[C]. *2019 IEEE European Symposium on Security and Privacy*, 2019: 142-157.
- [43] Maisuradze G, Rossow C. Ret2spec: Speculative Execution Using Return Stack Buffers[C]. *The 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 2109-2122.
- [44] Moghimi D. Downfall: exploiting speculative data gathering[C]. *32th USENIX Security Symposium*, 2023.
- [45] van Schaik S, Minkin M, Kwong A, et al. CacheOut: Leaking Data on Intel CPUs via Cache Evictions[C]. *2021 IEEE Symposium on Security and Privacy*, 2021: 339-354.



叶青瑜 于 2023 年在北京邮电大学计算机技术专业获得硕士学位。现任中关村实验室助理工程师, CCF 会员。研究领域为处理器安全。研究兴趣包括: 形式化验证, 微架构安全, 硬件漏洞分析和检测。Email: yeqy@zgclab.edu.cn



柯谦 于 2023 年在北京邮电大学网络安全专业获得硕士学位。研究领域为处理器安全, 研究兴趣包括: 形式化验证。Email: keqian_9@163.com



王海霞 于 2004 年在中国科学院计算技术研究所获得博士学位, 现任清华大学信息科学与技术国家研究中心副研究员, CCF 会员。研究领域为计算机系统结构, 研究兴趣包括: 处理器结构设计、处理器安全验证。Email: hx-wang@tsinghua.edu.cn



邱朋飞 于 2020 年在清华大学计算机科学与技术专业获得博士学位, 现任北京邮电大学特聘副研究员, CCF 会员。研究领域为计算机硬件安全。研究兴趣包括: 处理器硬件漏洞挖掘与利用、处理器安全模型设计。Email: qpf@bupt.edu.cn



刘畅 于 2021 年在清华大学计算机科学与技术系获得学士学位。现在清华大学计算机科学与技术系攻读博士学位。研究领域为处理器安全。研究兴趣包括: 处理器微架构安全、处理器漏洞挖掘和检测。Email: chang-li17@tsinghua.org.cn



汪东升 于 1995 年在哈尔滨工业大学获得博士学位。现任清华大学计算机系长聘教授, CCF 会员。研究领域为计算系统安全、处理器设计。Email: wds@tsinghua.edu.cn