处理器分支预测攻击研究综述

刘 畅[1] 杨 毅[2] 李昊儒[1] 邱朋飞[1),2)]
吕勇强[3] 王海霞[3] 鞠大鹏[1] 汪东升[1]

[1)](清华大学计算机科学与技术系 北京 100084)
[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 100867)
[3)](清华大学北京信息科学与技术国家研究中心 北京 100084)

摘 要 分支预测器是现代处理器的重要微架构组件,它可有效缓解流水线的控制流冒险问题,提升处理器性能.然而,尽管分支预测器的设计越发先进,设计细节也不被处理器厂商公开,但基于分支预测器的分支预测机制存在的安全问题仍不断被研究人员曝光.利用分支预测机制,攻击者能构建侧信道或隐藏通道,从而绕过软硬件的安全边界检查.在著名的 Spectre 攻击中,分支预测器还被用来构建瞬态执行窗口,这打破了被错误预测并执行的指令对软件程序员完全透明的错误安全假设.Spectre 攻击曝光后,分支预测的安全问题越来越受到重视,相关的攻击变种与防御措施成为学术界和工业界共同关注的课题.本文从分支预测器的设计角度出发,从已公开和被研究人员逆向工程出的分支预测器设计中总结了分支预测器的工作机制,然后按分支预测器填充方式、分支预测器索引方式和分支预测利用过程等特征对现有的分支预测攻击进行归纳和整理,并总结了这些攻击的攻击模型,包括攻击场景与攻击链.随后,本文结合 Intel、AMD 和 ARM 等主流商用处理器的典型微体系结构,从攻击模型深入分析了各分支预测攻击的关联性、创新点和可行性,并提出一种评价分支预测类瞬态执行攻击可行性的理论方法.最后,本文讨论了分支预测攻击未来的研究趋势、相关的防御策略以及安全分支预测器设计等诸多问题.

关键词 分支预测;处理器安全;计算机微体系结构;侧信道;瞬态执行
中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2022.02475

A Survey of Branch Prediction Attacks on Modern Processors

LIU Chang[1)] YANG Yi[2)] LI Hao-Ru[1)] QIU Peng-Fei[1),2)]
LYU Yong-Qiang[3)] WANG Hai-Xia[3)] JU Da-Peng[1)] WANG Dong-Sheng[1)]

[1)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2)](*Ministry of Education Key Laboratory of Trustworthy Distributed Computing and Service, Beijing University of Posts and Telecommunications, Beijing* 100867)
[3)](*Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing* 100084)

Abstract With the increasing demand for computer performance, many optimization techniques are adopted in modern processors. The branch predictors become significantly important components of computer micro-architecture for their efficiency to cope with the control hazards in pipeline. Although the design of the branch predictors has become more and more sophisticated and the details have not been disclosed by the vendors, their vulnerabilities have begun to expose. Using relevant mechanism of the branch predictors, attackers can construct side channels or covert

收稿日期:2022-01-28;在线发布日期:2022-09-27.本课题得到国家重点研发计划(2021YFB3100902)和国家自然科学基金(62072263)资助.
刘 畅,博士研究生,主要研究方向为计算机体系结构和处理器安全.E-mail:chang-li17@tsinghua.org.cn.**杨 毅**,硕士研究生,主要研究方向为硬件安全和可信系统.**李昊儒**,硕士研究生,主要研究方向为处理器安全.**邱朋飞**,博士,副教授,主要研究方向为处理器安全.**吕勇强**(通信作者),博士,副研究员,中国计算机学会(CCF)高级会员,主要研究方向为处理器安全.E-mail:luyq@tsinghua.edu.cn.**王海霞**,博士,副研究员,中国计算机学会(CCF)高级会员,主要研究方向为处理器体系结构和处理器安全形式化验证.**鞠大鹏**,博士,副研究员,主要研究方向计算机系统结构.**汪东升**,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为计算机体系结构、高性能计算和系统安全.

channels to bypass the check on security boundary of software and hardware. Spectre, the notorious speculative execution attack, exploits the branch predictor to construct a transient execution window and then exploits cache side channels to access the secret data of the victim. It disproves the security assumption that the mis-predicted instructions are completely transparent to the software programmers. With the exposure of Spectre attack, the security of branch predictors aroused wild attention, many related attack variants and defense measures have been proposed. Some variants apply Spectre in a specific attack scenario, such as NetSpectre aiming at breaking other virtual machine and SGXPectre aiming at breaking into SGX. Others exploit different microarchitectural side channels during transient execution, such as SpectreRewind and SMoTherSpectre which measure the difference of time consumption on specific execution port. To better understand how these attacks exploit the branch prediction, we summarize the principles of branch predictor through the published and reverse engineered branch predictor design from the perspective of branch predictor design, and then classify the existing speculative execution attacks according to their filling and indexing methods, as well as the utilization process. Furthermore, we extract the attack models of these attacks, including 8 attack scenarios and 2 attack chains. The attack scenarios include cross-process, cross-domain, cross-TEE, cross-VM, cross-hypervisor, cross-HT and cross-sandbox. The attack chains, consisting the side channel attack chain and the transient execution attack chain, can be merged into a model with 3 or 4 steps, including filling predictors, triggering branch instructions and exploiting the result of prediction. We use the attack model to describe the existing branch prediction attacks such as branch prediction side channels, branch prediction covert channels and transient execution attacks with or without branch prediction side channels. Furthermore, we analyse the relevance, innovativeness and feasibility of branch prediction attacks on modern processors such as Intel, AMD and ARM. Specifically, we present a theoretical method to evaluate the feasibility of transient execution attacks. Lastly, based on our analysis, we believe that automatically analysis of predictor security and secure predictor design will become the main research aspects in branch prediction security.

Keywords branch prediction; processor security; computer microarchitecture; side channel; transient execution

1 引 言

20 世纪 80 年代以来,为了提高性能,大部分处理器实现了流水线.流水线把指令执行分成取指、译码、执行等多个阶段,充分利用硬件资源,降低了执行每条指令的时钟周期数[1](Cycle Per Instruction, CPI).然而,由未解析的分支指令引起的控制冒险(hazard)和由数据依赖引起的数据冒险成为影响处理器性能的主要因素,这是因为在传统的流水线模型中,要想确保指令在冒险情况下正确执行,需要暂停流水线[2-4].为了解决流水线冒险问题,以进一步提高性能,现代处理器在对程序员透明的微体系结构(以下简称微架构)层面进行了全方面、多层次的优化,引入了包括分支预测器(Branch Preidictor)、缓存(Cache)和页表缓存(Translation Lookaside Buffer,TLB)在内的微架构部件[1],并实现了预测执行[4]和乱序执行[5]等多种微架构性能优化技术.表 1 总结了主流商用处理器中普遍实现的处理器性能优化技术.

表 1 处理器性能优化技术

技术名称	技术类型	目的
寄存器重命名[6]	乱序执行	减少数据冒险
指令重排序[7]	乱序执行	保证执行正确
指令多发射[8]	—	提高指令吞吐量
分支预测[9-10]	预测执行	减少控制冒险
指令预取[11]	预测执行	减少控制冒险
存储到加载转发[12-13]	预测执行	减少数据冒险
内存消歧预测[12,14]	预测执行	减少数据冒险
同步多线程[15]	—	提高指令吞吐量

其中，分支预测已成为预测执行中不可或缺的技术.在设计专用分支预测部件之前，处理器通过延迟槽[4]和分支折叠技术[16]等依赖于编译技术的静态方法解决控制流冒险问题.为提高程序的跨平台能力，减小编译器负担，研究人员提出使用硬件支持的分支预测器，通过执行流信息动态预测后续指令的地址.从 1981 年提出分支预测动态算法[9]至今，分支预测器在提高预测精度、增加速率和降低功耗等方面不断优化，不仅在工业界有成熟的实现[17-18]，而且在学术界也通过竞赛的方式逐年升级[19].

但是，在分支预测器的设计之初，设计者并没有特别关注其存在的安全问题.分支预测器对程序员透明的微架构特性，以及较为完善的错误检测与回滚(rollback)机制[14]，使部分处理器厂商存在“被错误预测并执行的指令对软件程序员完全透明”的错误安全假设.实际上，分支预测器与 Cache 一样，很早就被当成侧信道和隐藏通道的来源[20-22]，用于破解 RSA 密钥[21]、绕过虚拟地址随机化(Address Space Layout Randomization，ASLR)[23]或者窃取 SGX 中的私密数据[24].2018 年，著名的 Spectre 攻击[25]曝光，该攻击对以处理器安全为基石的系统安全造成严重的影响.根据 Xiong 等人的统计[26]，在 2018 至 2020 年间，Intel 在通用漏洞披露(Common Vulnerabilities and Exposures，CVE)中公开的破坏数据私密性的漏洞中，有超过半数是被 Spectre 相关攻击利用的瞬态执行漏洞.

Spectre 攻击曝光后，分支预测的安全问题越来越受到工业界和学术界的重视.在工业界，Intel①②、AMD③④和 ARM⑤⑥处理器厂商都分别公布了受影响的处理器类型，以及相应的防御方法.同时，学术界关于分支预测的攻防研究也迅速展开.随着攻击方法与防御策略的不断丰富，不少总结性的工作出现.Canella 等人[27]根据攻击模式总结了瞬态执行相关攻击与防御策略，Ragab 等人[14]以瞬态窗口的触发来源为切入点总结了所有瞬态执行攻击，Xiong 等人[26]根据侧信道类型、安全边界等分析了瞬态执行攻击的可行性和防御策略.然而，现有的分析都把重点放在瞬态执行攻击上.作为瞬态执行的重要来源之一，分支预测机制本身的安全问题还没有被全面地总结与分析.本文认为，除 Spectre 外，其余利用分支预测机制的攻击(本文简称为分支预测攻击)，包括分支预测侧信道与隐藏通道，与 Spectre 攻击的各类变种在分支预测器的利用原理上没有本质的区别.例如，BranchScope 攻击[28]与 Spectre V1 的原理相同.

因此，本文对 2007 年以来学术界主要安全会议和期刊的分支预测攻击进行了系统的归纳和分析.文章的主要创新点如下：

(1)首次从分支预测的工作原理出发，全面系统地总结了现有的主要分支预测攻击，归纳了这些攻击的基本原理，帮助计算机系统结构设计者深入地理解分支预测机制的安全问题来源.

(2)首次从攻击原理、攻击场景和攻击过程三个角度对分支预测攻击建立全新的攻击模型，并结合 Intel、AMD 和 ARM 等主流商用处理器的典型微架构，从攻击模型深入分析了这些处理器在分支预测器上的脆弱性，使硬件和系统安全研究者能对分支预测攻击有系统性的认识和比较.

(3)首次提出一种分支预测类型瞬态执行攻击可行性的评价方法，使用抽象处理器架构中攻击所需执行指令条数的理论模型分析各分支预测攻击的可行性，该评价方法对其余处理器微架构安全攻击同样具有参考价值.

本文第 2 节介绍分支预测器的基本原理和设计方法；第 3 节描述用于分析分支预测攻击的模型，包括攻击者视角下的分支预测器模型、攻击场景和攻击链；第 4 节根据分支预测器模型和攻击模型，对迄今为止主要的分支预测攻击进行了详细介绍；第 5 节分别从攻击原理、攻击场景和攻击过程出发，对各种分支预测攻击进行整体性的比较和分析；第 6 节讨论与展望分支预测机制的其它安全相关问题，包括分支预测攻击的研究趋势、分支预测攻击的防御策略以及安全的分支预测器设计；最后总结全文.

① Speculative Execution Side Channel Mitigations. https://software.intel.com/content/dam/develop/external/us/en/documents/336996-speculative-execution- side-channel-mitigations.pdf, 2018.5

② Intel Analysis of Speculative Execution Side Channels. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf, 2018.1

③ AMD Product Security. https://www.amd.com/en/corporate/product-security, 2020

④ Software Techniques for Managing Speculation on AMD Processors. http://developer.amd.com/wp-content/resources/Managing-Speculation-on-AMD-Processors.pdf, 2020.9

⑤ Vulnerability of Speculative Processors to Cache Timing Side-Channel Mechanism. https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability, 2021.5

⑥ Cache Speculation Side-channels. https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability/download-the-whitepaper, 2020.6

2 分支预测器设计

复杂的程序总是充斥大量的条件语句和循环语句,这些语句会根据程序上下文环境动态改变执行流.分支指令是用于修改执行流的指令,能通过一定条件决定分支方向,并附带跳转目标的地址信息.现代处理器的指令集大都实现了多类分支指令.根据分支指令是否附带跳转条件,可以分成条件分支和无条件分支,无条件分支必然跳转,条件分支则有可能跳转或不跳转,后者指的是分支被跳过且相邻的下一条指令被执行.根据分支目标是直接写在指令还是写在寄存器,还可以把分支指令分成直接分支和间接分支.

对于实现流水线的处理器,分支指令的跳转目标有时要在几个时钟周期后才被计算出来,在此期间取分支任何方向的指令预先执行都有可能出错,这种情况被称为控制流冒险[4].为了避免控制流冒险导致流水线阻塞,现代处理器使用分支预测器来预测分支指令的跳转目标.处理器在解析分支指令后判断预测结果的正确性,如果预测正确,将不造成任何性能损失;如果预测错误,则处理器通过回滚机制重新取另一方向的指令执行[14].

从 20 世纪 80 年代至今,为提高分支预测的精度并降低预测器的时延、面积与功耗开销,分支预测器的设计愈发复杂.为了使读者对分支预测的安全问题有全面系统的认识,本节对分支预测器设计的基本原理和关键部件的设计思想进行全面梳理和介绍.从结构角度看,尽管分支预测器的设计细节极为复杂,但都有类似的总体设计结构,即主要部件和工作原理是基本相同的.因此,2.1 节对分支预测器的主要部件进行介绍.从功能角度看,分支预测器的主要功能是实现对一条分支指令的分支方向和分支目标的预测.在分支预测器的发展过程中,实现这两类功能的部件基本是并行发展的.2.2 和 2.3 节分别对这实现这两类功能的分支预测部件的设计细节进行介绍.

2.1 主要部件

如今,主流的商用处理器实现了非常复杂的分支预测器和分支预测机制,并且实现细节未完全公开.图 1 根据部分技术手册[29-31]、部分逆向工程工作[25,28,32-34]和本研究团队在部分处理器上的实验结果,总结了现代商用处理器中主要的分支预测部件及其工作原理.图中用 ARMv8 精简指令集的分支指令举例,展示了现代处理器中主要的四个分支预测器部件,包括分支历史缓冲区(Branch History Buffer,BHB)、模式历史表(Pattern History Table,PHT)、返回栈缓冲区(Return Stack Buffer,RSB)和分支目标缓冲区(Branch Target Buffer,BTB).这四个部件是大部分现有分支预测安全问题的来源.

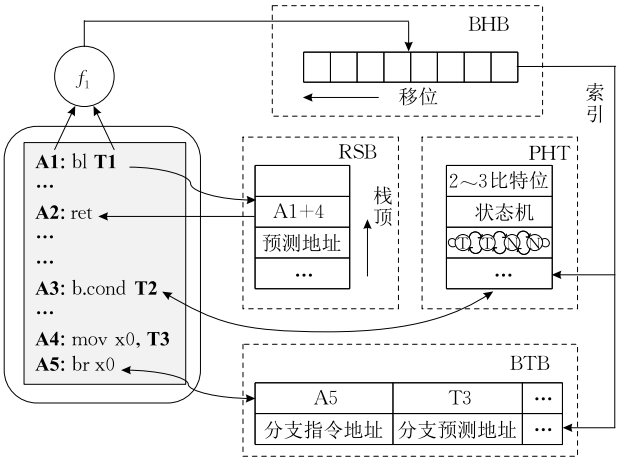


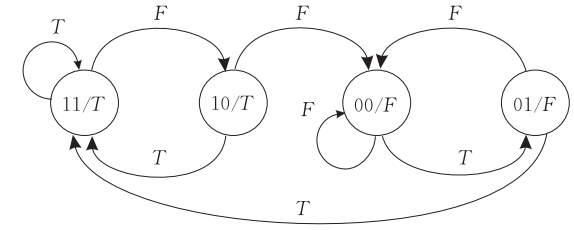
图 1 分支预测器主要部件及其工作原理示意图

BHB 记录了已执行分支的历史跳转信息.它有时作为一个系统寄存器实现,因此在部分设计中又被称为全局历史寄存器(Global History Register,GHR).对于每一条跳转的分支指令,其指令地址和跳转目标在通过一个复杂的逻辑运算后合并到 BHB.在分支预测中,BHB 中的内容可能会与分支指令的地址进行逻辑运算后作为访问 PHT 或 BTB 表的索引[28,34].也有部分文献分别把用来索引 PHT 和 BTB 的部件称为 GHR 和 BHB[35].

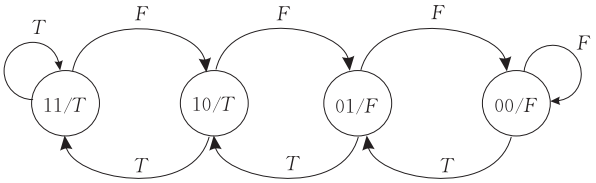
PHT 用于预测条件分支的方向.它由分支指令地址和分支历史索引,大部分处理器的 PHT 表项是 2 到 3 位的分支预测状态机[28],该状态机能够预测分支的跳转方向.2 位状态机的两种实现方式如图 2 所示[4,28].图中的四个状态中,大于 1 的两个状态表示跳转(Taken,记为 T),小于或等于 1 的两个状态表示不跳转(Not Taken,记为 F).对于索引到某个 PHT 表项的条件分支指令,该表项中状态机的状态表示了预测方向.

BTB 用于预测分支的目标.它的组织方式类似多路组相联的 Cache,记录了每条发生跳转的分支指令按照一定的哈希算法压缩后的地址和目标[34],并对索引到组(Set)且与标签(Tag)匹配的分支进行预测.

RSB 又名返回地址栈(Return Address Stack,RAS),是一种特殊的分支目标预测器,用于预测返



(a) 存在跳变的2位预测状态机(移位寄存器)



(b) 不存在跳变的2位预测状态机(饱和计数器)

图 2　2 位分支预测状态机的两种预测算法[4,28]

回指令的跳转目标. 它被设计成栈的结构. 处理器执行函数调用指令(如 x86 指令集的 call 指令或 ARM 指令集的 bl 指令)时,会把相邻的下一条指令地址写入 RSB 的顶部. 处理器执行返回指令时,栈顶地址将作为一个备选的预测目标被弹出[36-37].

表 2 总结了各类分支指令对分支预测部件的使用方式. 预测部件表示预测分支方向和分支目标使用的部件. 索引方式表示分支地址到 PHT 或 BTB 表项的映射方式,包括直接使用存放在程序计数器(Program Counter,PC)的分支指令地址和同时使用指令地址与分支历史(BHB)这两种方式. 更新部件表示分支执行完毕后更新的分支预测部件.

表 2　各类分支指令对分支预测部件的使用

分支类型	预测部件	索引方式[34]	更新部件
无条件直接分支	(BTB)*	PC	BHB,(BTB),(RSB)
条件直接分支	(BTB),PHT	PC	BHB,(BTB),PHT
无条件间接分支	BTB,RSB	PC	BHB,(BTB),(RSB)
条件间接分支	PHT,BTB	BHB,PC	BHB,(BTB),PHT

注:*表中添加括号表示不一定使用到该部件. 比如,只有函数调用指令更新 RSB;部分处理器的无条件直接分支能直接从指令得到分支目标而无需使用 BTB.

2.2　分支方向预测部件

分支方向预测主要由 PHT 实现. PHT 有两种设计思路,一是完全由硬件实现动态预测,二是硬件预测结合文件采样分析与编译优化[38-40]. 由于程序的复杂性和处理器微架构的多样性,现代处理器主要使用前一类设计思路. 表 3 总结了 1981 年到 2016 年间主要相关会议和期刊上的动态分支方向预测器设计,这些设计分别在分支指令的索引方式、分支预测表设计和预测算法方面有较大的创新. 表中标注的内容为创新点.

表 3　1981～2016 年的主要分支方向预测器设计

分支方向预测器	索引方式	表项设计	预测算法
Bimodal[41]	√	√	√
GAg[42]	√	√	
PAg[42]		√	
PAp[42]		√	
(m,n)-correlation[43]	√	√	
gselect[41]	√		
gshare[41]	√		
combine[41]			√
Skewed[44]	√	√	
Bi-mode[45]	√	√	√
Agree[46]		√	√
YAGS[47]		√	
Variable length path[48]	√		
Alloyed-index[49]	√		
perceptron-based[50]			√
GEHL[51]	√		√
TAGE[19,52-53]	√		
ITTAGE[54]		√	
TAGE-SC-L[55]			√

2.2.1　索引方式

使用 2 位状态机模型的 Bimodal 预测器是最早的动态分支预测器[9]. Bimodal 预测器实现了由 2 位状态机[41]组成的 PHT 表. 为了预测不同的分支指令,Bimodal 使用分支指令地址的低位索引表项.

在观察到分支历史是后续分支指令是否跳转的重要依据后,Yeh 等人[42]提出三种使用全局分支历史或按地址保存的分支历史(局部分支历史)索引 PHT 表的两级自适应预测器 GAg、PAg 和 PAp. gshare 和 gselect[41]等预测器则结合了分支历史和分支指令地址进行索引,这些预测器均使用了一个全局历史寄存器来记录历史分支跳转情况,每位表示一个历史分支是否跳转.

同时,研究人员注意到不同地址分支指令的别名(alias)问题是影响分支预测器预测精度的重要因素. 别名问题指不同分支指令索引到同一预测器表项的问题. 为了解决别名问题,处理器设计者设计了 Skewed[44]、Bi-mode[45]和 Alloyed-index[49]等抗别名干扰的分支预测器,这些预测器使用了更加复杂的索引逻辑,并增加了辅助部件来修正由 2 位状态机得到的预测结果.

除了别名问题,研究人员还引入变长历史索引的设计以适应不同的上下文环境,如 GEHL[51]和 TAGE[19],后者被认为是目前最精确的分支预测器之一. 这些设计思想被用在了部分商用处理器中[18].

现有的大部分主流商用处理器都同时实现了把指令地址与分支历史记录结合来索引 PHT 表的设计. 在索引粒度方面,Evtyushkin 等人[28]发现,Intel

系列处理器的 PHT 表按字节粒度索引，不同字节处起始的分支指令会被索引到不同的 PHT 表项.

2.2.2 表项设计与预测算法

除了索引机制，为了提高预测精度，研究人员还提出了更加复杂的表项设计. 比如，YAGS 预测器[47]在 PHT 表项中加入少量位作为标签，进一步减少了别名导致的冲突. Agree 预测器[46]则在 BTB 表项中加入偏置位用来纠正 PHT 表项的内容.

在预测算法方面，Choi 等人[56]和 Evers 等人[57]分别提出多线程和多进程环境下分支历史的更新算法. 更加智能化的算法也被引入分支预测器. Jiménez 等人[50]提出使用感知器模型代替状态机，并设计了预测算法与训练算法. Tarsa 等人[58]则引入神经网络来记录分支历史. Mittal[59]总结了各种用于分支预测的智能算法与优化策略. 基于感知器的方向预测器已经用于部分商用处理器[18].

为了应对复杂多变的执行流，McFarling[41]提出同时使用多种预测机制，然后根据之前预测的正确性动态选择预测机制的组合预测器. 这种锦标赛机制被沿用至今，Evtyushkin 等人[28]通过逆向工程发现，Intel Skylake、Haswell 和 Sandy 处理器使用了仅分支指令地址索引和同时使用地址和分支历史索引的混合预测器，并根据这两种索引方式的预测精度，动态选择预测更准确的索引方式.

2.2.3 其余设计理念

除了表 3 提到的几点外，分支方向预测还存在很多有价值的设计理念.

(1)引入提高预测精度的辅助部件，专门处理难预测的分支，如多层循环中的分支. 包括循环退出缓冲区[60]、最内层循环迭代计数器[61]和 Wormhole[62]在内的辅助部件专门处理复杂循环. 其中，循环退出缓冲区使用专门的缓冲区存放循环的迭代次数计数器，当计数器的值与循环边界值一致时修改预测方向，其设计思想被部分 Intel 处理器采用[32].

(2)提高预测器容量和访问效率. 相关设计包括引入多级 PHT[63]、使 PHT 流水线化[64]以及用傅里叶变换压缩过长的分支历史[65].

(3)引入专门的错误纠正部件，以尽快使分支预测器调整到正确的预测方向. 相关设计包括对高错误率预测状态机进行纠正的并行保守纠正器[66]和记录错误预测下分支路径的错误历史缓冲区[67].

2.3 分支目标预测部件

分支目标预测主要由 BTB 和 RSB 实现. 分支目标预测始于 Lee 等人[10]提出的 BTB. BTB 根据当前分支指令的地址预测其跳转目标. 由于 BTB 需要占用较大空间来存储指令地址，并且要能够快速响应访问，因此 BTB 的设计与多路组相联的 Cache 比较类似[68]. 为了节省存储空间，BTB 通常不会存储完整的指令地址. 比如，Intel 处理器中，BTB 只存储分支目标的低 32 位，为了组成合法的分支目标，处理器将分支指令地址的更高位和 BTB 中的 32 位拼接[34]，这类似相对寻址机制[1].

之后的大多数设计都是基于 BTB 的改进. 此外，也有部分根据指令特点特殊设计的预测部件. 表 4 总结了 1984 年至今主要相关会议和期刊上的动态分支目标预测器设计，这些设计主要在提高预测精度、提高容量与访问效率和减小功耗这几方面有较大的创新. 表中标注的内容为创新点.

表 4 1981～2021 年的主要分支目标预测器设计

分支目标预测器	提高预测精度	提高容量与访问效率	减小功耗
BTB[10]	✓		
I-Cache Based BTB[68]		✓	
RSB(RAS)[69-70]	✓		
Target Cache[71]	✓	✓	
FTB[72]	✓		
Rehashable BTB[73]	✓		
Lazy BTB[74]			✓
Phantom-BTB[75]		✓	
Leakage-Aware Speculative BTB[76]			✓
Shotgun BTB[77]	✓		
Micro BTB[78]	✓	✓	

2.3.1 提高预测精度

在提高预测精度方面，研究人员基于按指令类型使用不同预测部件的思想，设计了 RSB[70](在部分处理器中称为 RAS)，以及用于预测条件间接分支的 Rehashable BTB[73]和 Shotgun[77].

返回指令用于在函数调用结束后回到前一个上下文环境中. 处理器借助函数调用栈保存返回地址[1]，部分处理器也会使用一个专用寄存器存放返回地址[2]. 除了中断、异常和程序手动修改函数调用栈等特殊情况外，返回指令和函数调用指令是成对出现的[36]，即返回指令的跳转目标是与函数调用指令相邻的下一条指令地址. 根据这一特性，Kaeli 等人[69]设计了栈结构的 RSB. 这一部件在如今的大部分商用处理器中均有实现. 商用处理器常见的 RSB 表项有 8 条、16 条或 32 条等[31,37]. Mambretti 等人[79]发现，AMD 的可用 RSB 表项比实际大小少一项，这是因为 RSB 上保留了一个条目用于指针逻辑简

化.在RSB为空时,Intel Skylake 等处理器会用 BTB 来预测返回指令的目标[37].本研究团队也在部分 ARM 处理器上观察到同样的现象.Koruyeh 等人[36]未在 AMD 处理器观察到这一现象.

Chang 等人[71]参考分支方向预测中的分支历史索引,使用指令地址和分支历史索引 BTB,同时在 BTB 中加入分支指令的低位标签来减少别名问题.Zhang 等人[34]发现,Intel Haswell 和 Skylake 等处理器也使用了分支历史索引 BTB 的设计,并且同样实现了两种索引方式的锦标赛机制.

Ajorpaz 等人[80]分析了 BTB 替换策略对预测精度的影响,并提出基于分支历史和分支块重用情况的全局历史重用替换策略以提高预测精度.

2.3.2　提高访问容量与访问效率

BTB 表项比 PHT 表项更大,因此在不失访问效率的同时提高 BTB 容量也成为设计者关注的问题.Bray 等人[68]提出将 BTB 和 I-Cache 合并来减小存储开销,但这样放大了冲突缺失对分支预测精度的影响.Reinman 等人[72]设计了获取目标缓冲区(Fetch Target Buffer,FTB),FTB 只保存跳转分支的部分位,提高了存储空间利用率.Burcea 等人[75]设计的 Phantom-BTB 用 L2 Cache 来虚拟一个更大的 BTB 空间,从而增大了 BTB 的容量.Gupta 等人[78]设计了 Micro BTB,每个 BTB 条目存储多个分支,并且用不同类型表项来提高利用率.这些设计在商用处理器中有部分应用,但尚未全面部署.

2.3.3　减小功耗

为了减小分支预测器的功耗,Lazy BTB[74]跟踪进程并预先标定每个跳转分支的 BTB 查找,以减少未跳转分支的查找;Leakage-Aware Speculative BTB[76]则预测性地激活部分条目,将其余条目置于低功耗状态.这些设计在商用处理器上的应用较少.

2.3.4　分支目标预测中的静态策略

静态策略是指某些特殊情况下,分支预测器在设计之初就确定预测分支目标的预测策略,主流商用处理器的静态处理策略包括:

(1)当一个条件分支指令或间接分支指令在 BTB 中不存在表项时,Intel、AMD 等处理器会预测分支不跳转[24].

(2)如果分支预测器的目标地址不在指令 Cache 中,Intel 等处理器会预测分支不跳转[34].因为分支预测到分支指令解析的时间小于指令 Cache 内容缺失而访问内存的时间,所以即使预测分支跳转也会由于取指延迟而无法带来性能上的收益.本研究团队在部分 ARM 处理器上对 RSB 进行实验,也观察到同样的现象.

3　攻击模型

威胁模型或攻击模型都是描述信息安全的重要模型.前者强调攻击对系统造成的破坏性,需要综合考虑被攻击系统的安全特性和攻击者的认知与能力,后者则更强调攻击的原理和过程①.

在分支预测攻击中,有的攻击使用了不同的分支预测部件,但使用了相同的攻击过程;有的攻击则使用了相同的攻击原理,但攻击者想要突破不同类型的安全边界.为了归纳和分析这些复杂的分支预测攻击,本文使用攻击模型对攻击建模.

在攻击模型中,攻击原理、攻击者能力、攻击目标和攻击过程都是需要研究的对象.为了方便读者把第 2 节介绍的分支预测器设计和实际的分支预测攻击原理联系起来,3.1 节介绍了攻击者视角下的分支预测器模型.此外,3.2 和 3.3 节分别介绍了分支预测攻击中的攻击场景模型和攻击链模型.攻击场景很好地描述了攻击者的能力和目标,而攻击链很好地描述了攻击者的攻击过程.

3.1　攻击者视角下的分支预测器模型

为了提高预测精度、提高访问速度与减小功耗,分支预测器的设计越来越复杂,商用处理器也不再公开设计细节.然而,复杂且不透明的分支预测器设计无法保障分支预测器的安全性.实际上,在大多数利用分支预测器的攻击中,攻击者往往无需了解分支预测器的实现细节就能利用它完成攻击.

图 3 结合第 2 节的分支预测器设计和现有分支预测攻击的攻击原理,总结了攻击者视角下的分支预测器抽象模型.从整体看,分支预测器的输入包括历史分支的跳转情况、当前待预测分支指令的地址和一些解码信息,输出是当前分支的预测目标.分支预测部件包括存放分支历史信息的 BHB 等部件、预测分支跳转方向的 PHT 等部件、预测分支目标的 BTB 和 RSB 等部件以及复杂的选择部件.图中的虚线表示分支历史对分支预测部件的更新;输入分支预测部件的数据通路表示根据当前分支的地址和分支历史计算出的索引值,f_1 到 f_5 是用于计算索

① Attack Modeling vs Threat Modeling. https://www.techrepublic.com/article/attack-modeling-vs-threat-modeling/,2006.3

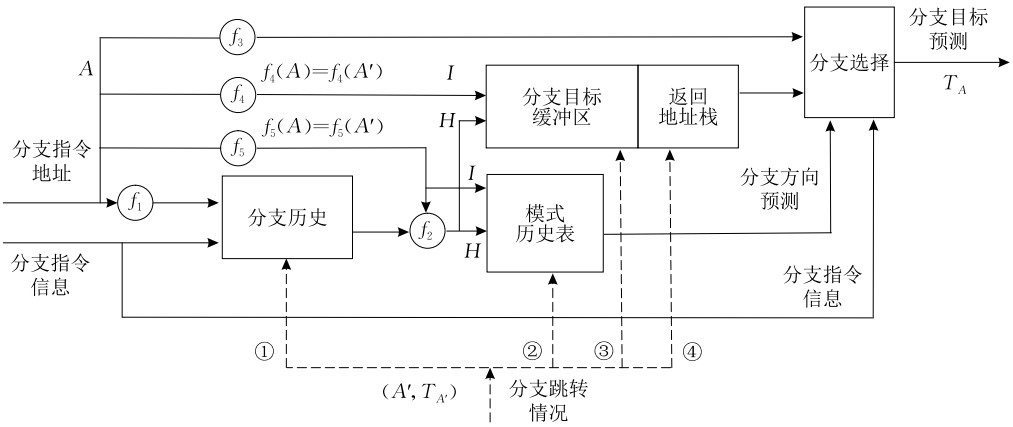


图 3 攻击者视角下的分支预测器模型

引的复杂逻辑函数. 索引方式包括仅根据分支指令地址的索引方式(用 I 表示)和结合分支地址与分支历史的索引方式(用 H 表示).

为了实现分支预测攻击,攻击者需要对预测部件的索引方式和表项规模等有比较清楚的了解. 在分支预测攻击中,攻击者通过精心构造的分支历史来影响分支预测部件的内容,这一过程通常被称为误训练(mistrain)或毒化(poisoning)[25]. 为了凸显攻击的底层原理,文章的后续章节把这一过程称为分支预测器填充. 图 3 用二元组 $(A', T_{A'})$ 来表示攻击者用于填充分支预测器的分支指令地址和该分支的实际跳转目标. 填充路径有四条:①是对 BHB 的填充;②是对 PHT 的填充;③和④分别是对 BTB 和 RSB 的填充.

在实际利用分支预测器时,攻击者需要触发一个待利用分支,该分支指令的地址为 A,攻击者的目标是控制分支预测器预测的跳转目标 T_A,或者把分支预测器的预测结果 T_A 作为侧信道来获取私密信息或作为隐藏通道传递信息. 为了能够利用之前的填充结果,当前分支地址 A 和用于填充的分支地址 A' 应该索引到同一表项,即索引值 $f(A)$ 与 $f(A')$ 应该相等.

图 3 的分支预测器模型是帮助读者理解分支预测攻击过程的重要工具. 具体的攻击过程将在 4.1 至 4.4 节中具体展开介绍.

3.2 攻击场景模型

在处理器安全研究中,权限等级是构建攻击场景的最重要指标. 现代处理器大都实现了访问控制,把处理器使用者分级,每级有不同的数据访问权限和操作权限.

Intel 处理器定义了权限环,ring0 表示操作系统权限,ring3 表示最底层的用户进程权限[29]. Intel 还实现了虚拟化[81],以支持虚拟机场景下虚拟机监视器(Hypervisor)和虚拟机(Virtual Machine,VM)的不同权限等级. 此外,Intel 还实现了软件防护扩展(Software Guard Extensions,SGX)技术[82]以提供可信执行环境(Trusted Execution Environment,TEE),SGX 提供了安全的加密内存空间,操作系统也无法访问这块内存的内容.

ARMv8 之后的 ARM 处理器通过异常等级(Exception Level,EL)实现访问控制[30]. EL0 为用户进程权限,EL1 为普通内核或虚拟机权限,EL2 为虚拟机监视器权限,EL3 为拥有最高权限的可信执行环境 TrustZone[83].

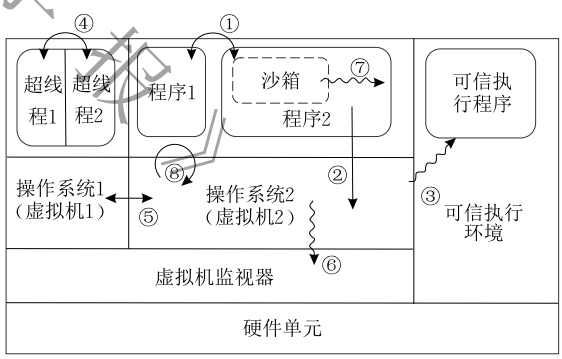


图 4 分支预测攻击场景模型

Xiong 等人[26]对瞬态执行攻击场景有很好的总结,图 4 结合 Intel 和 ARM 等商用处理器的权限等级架构,将其进一步扩展为分支预测攻击场景.

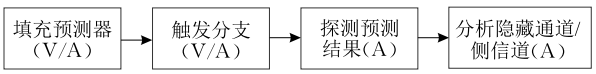
图中展示了由虚拟机监视器、操作系统(虚拟机)、普通应用程序、超线程、沙箱和可信执行环境等组成的复杂权限架构,以及 8 种不同的攻击场景. ① 是跨进程攻击场景,攻击者和攻击目标(本文简称为受害者)均是用户权限下的进程,攻击者想要获取受害者地址空间内的数据;② 是跨域攻击场景,攻击者只有用户权限,而受害者是内核进程,攻击者想要破

解一些内核防御或获取内核空间数据；③是跨 TEE 攻击场景，攻击者操控恶意操作系统攻击 TEE 中的数据；④是同步多线程攻击场景，攻击者和受害者同时在一个物理核的两个不同逻辑核上执行，攻击者目标与跨进程攻击场景相同；⑤是跨虚拟机攻击场景，拥有内核权限的攻击者操控恶意虚拟机往外发送任意数据包以攻击其余虚拟机，这一攻击场景多见于网络攻击；⑥是跨 Hypervisor 攻击，攻击者的目标是基于内核的虚拟机(Kernel-Based Virtual Machine，KVM)的攻击，攻击者想要跨过虚拟化机制的安全边界；⑦是跨沙箱攻击，又称为沙箱逃逸. 沙箱是应用程序内一个特殊的执行环境，访问和操作权限受应用程序控制，常见的跨沙箱攻击为使用某个带恶意 JavaScript 的网站来攻击执行这段代码的浏览器进程；⑧是使用恶意的内核木马程序攻击操作系统中的其它进程. 攻击者通过发布难以被分析工具检测到的木马程序，在一定条件下泄露操作系统或用户进程数据.

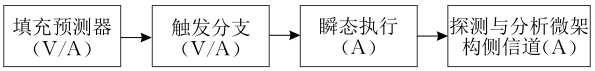
本文所构建的攻击场景涵盖了本文所整理的全部分支预测攻击. 5.2 节将结合这些攻击场景的特点，对分支预测攻击的攻击条件进行详细分析. 值得注意的是，图中没有包括的其它攻击场景，比如可信执行环境内部的跨进程攻击、跨虚拟机监视器攻击以及跨处理器攻击，也都是分支预测攻击的潜在应用场景. 受到实现难度的限制，尚未有工作在这些场景实现分支预测攻击，但不能排除它们受到分支预测攻击的可能.

3.3　攻击链模型

攻击链模型描述了攻击者的攻击过程，也是分析防御策略的重要工具. 本文参考 Deng 等人[84-85]提出的 Cache 和 TLB 侧信道攻击模型，按照图 5 的格式描述攻击链. 其中，每个阶段描述攻击者的一个操作，V 表示操作由受害者(如果存在)完成，A 表示操作由攻击者完成. 根据对现有分支预测攻击的总结，本文把分支预测攻击的攻击链抽象成两类，一类是侧信道或隐藏通道攻击链，另一类是瞬态执行攻击链.



(a) 基于分支预测的侧信道和隐藏通道攻击链



(b) 基于分支预测的瞬态执行攻击链

图 5　分支预测攻击的两类攻击链

攻击链 a 描述了将分支预测机制作为侧信道或隐藏通道的攻击. 第二阶段触发的分支指令隐藏了一些攻击者感兴趣的信息，比如程序的执行流[23-24]或者 RSA 密钥[20,86]，这些信息通过分支是否跳转来体现. 攻击者根据分支预测器的初始状态、对当前分支的预测结果和预测结束后分支预测器的状态来分析分支的跳转情况，从而恢复感兴趣或者想要传递的信息. 分支预测侧信道和隐藏通道的技术细节在 4.3 节中介绍.

攻击链 b 描述了利用分支预测机制进行的瞬态执行攻击. 这一类攻击中，攻击者通过第一阶段的预测器填充，使第二阶段触发的分支发生预测错误. 攻击者会利用从分支指令被错误预测到处理器发现预测错误的这一时间窗口，进行私密数据访问或控制流劫持，并影响一些微架构部件的状态. 处理器检测到错误的分支预测后，会通过回滚机制重新回到正确的执行路径. 之后，攻击者使用已存在的一些侧信道分析技术进一步窃取私密数据. 瞬态执行攻击和相关微架构侧信道技术细节在 4.4 节中介绍.

最新的一些分支预测攻击结合了分支预测侧信道和瞬态执行攻击，在瞬态执行中利用嵌套的分支预测及分支预测器的状态变化来窃取私密数据. 这类攻击相当于攻击链 b 的一个扩展，在攻击链 b 的第三步中，进一步利用了分支预测机制的特性. 相关的技术细节在 4.5 节中介绍.

4　分支预测攻击

本文调研了现有的各种分支预测攻击，并根据第 2 节描述的分支预测器设计和第 3 节介绍的攻击模型归纳了这些攻击的特征，最终结果如表 5 所示. 根据归纳结果，现有攻击主要使用了 PHT、BTB 和 RSB 这三个分支预测器部件，全面覆盖了 3.1 节中的八类攻击场景，攻击链也各有特点. 4.1 到 4.5 节对表 5 中分支预测攻击的技术细节进行了详细介绍.

第 4 节内容安排如下：4.1 节介绍攻击链第一阶段中 PHT、BTB 和 RSB 和 BHB 四个预测器部件的填充方式；4.2 节介绍攻击链第二阶段触发的分支如何索引到第一阶段填充的表项；4.3 节介绍利用分支预测机制实现侧信道和隐藏通道的原理；4.4 节介绍利用分支预测机制实现瞬态执行攻击的原理，以及相关微架构侧信道的一些技术细节；4.5 节介绍瞬态执行窗口中的分支预测机制，以及相应的瞬态分支攻击的基本原理.

表 5 分支预测攻击的攻击模型和原理*

攻击	攻击模型		利用部件	地址类型	PC索引	利用方式
	攻击链	攻击场景				
SBPA[20-22]	填充 BTB(A)→触发分支(V)→时间测量与分析(A)	④	BTB	别名	√	侧信道
Fault Attack[87]	填充 PHT(A)→触发分支(V)→事件测量与差分分析(A)	~~④~~**	PHT	同名	√	
HPC-based Attack[88]	离线训练(A)→填充 PHT(A)→触发分支(V)→事件测量与决策(A)	~~④~~	PHT	同名	√	
Jump over ASLR[23]	填充 BTB(A)→触发分支(V)→时间测量与分析(A)	①②	BTB	别名	√	
Branch shadowing[24]	填充 BTB(V)→触发分支(A)→LBR 采样与分析(A)	③	BTB	别名	—	
Template Attack[86,89]	离线训练(A)→填充 PHT(A)→触发分支(V)→事件测量与决策(A)	①	PHT	同名	√	
BranchScope[28]	填充 PHT(A)→触发分支(V)→时间或事件测量与分析(A)	①③	PHT	同名	√	
Bluethunder[90]	恢复 BHB(A)→填充PHT(A)→触发分支(V)→事件测量与分析(A)	③	PHT	同名	—	
RSC[91-92]	填充 PHT(A)→触发分支(A)→时间测量与分析(A)	①④⑦	PHT	多地址	—***	隐藏通道
CC[92-93]	填充 PHT(A)→触发分支(A)→时间测量与分析(A)	①④⑦	PHT	多地址	—	
Spectre V1[25]	填充 PHT(V)→触发分支(V)→瞬态执行(V)→侧信道分析(A)	②⑦	PHT	同名	—	瞬态执行
NetSpectre[94]		①②⑤	PHT	同名	—	
Speculative Interference[95]		⑦	PHT	同名	—	
SpectreRewind[96]		⑦	PHT	同名	—	
Spectre V3a[97]		②	PHT	同名	—	
Spectre V1.1[98]		⑦	PHT	同名	—	
SplitSpectre[79]		⑦	PHT	同名	—	
BlindSide[99]		②	PHT BTB	同名	√	
SepcHammer[100]		②	PHT	同名	—	
BranchSpectre[101]	填充 PHT(V)→触发分支(V)→瞬态分支嵌套(V)→侧信道分析(A)	④	PHT	同名	—	
Spectre V2[25]	填充 BTB(A/V)→触发分支(V)→瞬态执行(V)→侧信道分析(A)	①⑥	BTB	同名/别名	√	
SGXPectre[102]		③	BTB	同名/别名	√	
SMoTherSpectre[103]		④	BTB	同名/别名	√	
SpecROP[104]	填充 BTB(A/V)→触发分支(V)→瞬态分支嵌套(V)→侧信道分析(A)	④	BTB RSB	同名/别名	—	
ExSpectre[105]	填充 BTB(A)→触发分支(A)→瞬态执行(A)→侧信道分析(A)	⑧	BTB	同名/别名	√	
Distant Collision Torjans[34]	填充 BTB(A)→触发分支(A)→瞬态执行(A)→侧信道分析(A)	⑧	BTB	别名	√	
SpectreBHI[106]	填充BHB和BTB(A)→触发分支(A)→瞬态执行(V)→侧信道分析(A)	②	BTB BHB	别名	—	
Branch Skip Torjans[34]	缓存清除(A)→触发分支(A)→瞬态执行(A)→侧信道分析(A)	⑧	BTB RSB	—	—	
SpectreRSB[36]	填充 RSB(A)→触发分支(V)→瞬态执行(V)→侧信道分析(A)	①③④	RSB	—	—	
ret2spec[37]		①⑦	RSB	—	—	

注：*本文只注明了分支预测攻击的相关论文中实现的攻击模型和原理. 部分攻击仍可能存在额外的攻击场景，如 Spectre V2 攻击能够在同步多线程场景下实现，但由于论文中没有涉及，表中也未标注.

**删除线表示攻击场景与当前商用处理器不符. 比如 Fault Attack 属于跨进程攻击场景，但使用了性能计数器，而一般用户权限无法使用性能计数器.

***符号“—”表示攻击没有强制要求使用仅分支指令的索引方式. 比如，RSC 隐藏通道通过执行大量不同地址的分支来作为一轮迭代并传递单位信息，因此对索引没有严格要求.

4.1 分支预测器填充

大多数分支预测攻击都是从分支预测器的填充开始. 虽然 BHB 不能单独用于分支预测攻击，但也被一些基于分支历史索引的攻击利用[90,101,106]. 因此，本节依次介绍 PHT、BTB、RSB 和 BHB 四个微架构部件的填充方式. 分支预测器的填充过程对应图 5 中攻击链 a 和 b 的第一阶段.

4.1.1 PHT 填充

如表 5 所示，攻击者或受害者都可以填充 PHT 表项. 现代处理器的PHT表项大都是一个状态机[86]，因此，在反复执行往同一方向跳转的分支后，该分支对应的 PHT 表项中的状态机将被置为这一方向，在下一次预测时，这条分支指令的跳转方向也将被预测为这一方向[28]. 因此，通过反复执行相同方向的分支指令，攻击者能够把 PHT 表项填充为已知的初始状态. 但是，由于噪声和未知的微架构特性，目前的攻击技术还很难使 PHT 表项精确地初始化为任意状态[86]. 比如，对于图 2 所示的 2 位状态机，攻击者能轻松控制 PHT 表项落入状态 00 或 11，而控制 PHT 表项落入 01 或 10 就相对困难.

除了上述的单表项填充外，在把 PHT 作为隐藏通道的 CC[93]和 RSC[91]攻击中，攻击者通过同时执行大量有规律的分支，使多个 PHT 表项同时受到这些分支的影响. 这类攻击中，攻击者任意填充多个 PHT 表项且不关注具体填充的表项位置.

4.1.2　BTB 填充

由于 BTB 类似组相联 Cache 的特性，BTB 填充有两类目标，一是填充单个 BTB 块，二是填充一个 BTB 组. 在填充单个 BTB 块时，攻击者通过多次执行一个跳转到固定目标的分支，使 BTB 存入该分支的标签及其跳转目标，其中标签记录了这条分支指令的地址信息. 除了填充单个 BTB 块之外，Acıiçmez 等人[20]还在 BTB 上构造了类似 Cache 侧信道攻击中的 Prime+Probe 侧信道[107]. 在了解 BTB 的组索引和标签匹配机制后，攻击者可以构造一个驱逐集(Eviction Set)，驱逐集的内容是大量地址和跳转目标不同但索引到同一 BTB 组的分支指令. 通过反复执行驱逐集内的分支，攻击者能够用这些分支填满相应的 BTB 组.

4.1.3　RSB 填充

根据 2.1 节所述的 RSB 工作原理，在执行函数调用指令后，函数的返回地址(通常是与函数调用指令相邻的下一条指令地址)会被压入 RSB 栈顶[36]. 因此，攻击者在精心构造好的地址处执行函数调用指令，就能在 RSB 栈顶填入攻击者控制的地址.

但是，3.2 节中描述的攻击场景①③⑤等容易受到操作系统的干扰，因为这些场景涉及到操作系统切换或异常处理操作，这些操作需要切换上下文，并进行一些与攻击不相关的函数调用和返回操作，破坏攻击者的填充结果. 为了确保攻击者填充的地址不被覆盖或丢弃，RSB 的深度应该足够大，也就是说，RSB 深度会影响基于 RSB 的分支预测攻击在一些场景中的攻击效果[27].

如 2.3.1 节所述，有的处理器在 RSB 为空时用 BTB 来预测返回指令. Canella 等人[27]认为，通过这一机制，攻击者能绕过把间接分支替换成返回指令来抵御 BTB 瞬态执行攻击的防御策略 retpoline①. 这种情况下，深度的递归函数调用是清空 RSB 表项的一种主要策略[37].

4.1.4　BHB 填充

部分基于历史索引的攻击中，攻击者需要分支历史可控，从而使攻击者能够根据受害者分支指令的地址和已知的分支历史，计算出受害者执行该分支指令时索引的预测器表项.

为了研究 BHB 的填充方式，Huo 等人[90]对 Intel Haswell 处理器进行了逆向工程. 他们发现，在 Intel Haswell 等处理器上，BHB 的更新由最近执行的分支指令的地址和目标决定. 其中，指令地址的第 0x40 位、0x80 位以及分支目标的最低 2 位足够决定 BHB 本次更新的 2 位. 他们还发现，执行 93 个跳转的分支可以保证 BHB 完全更新. 因此，通过执行已知地址和跳转目标的 93 条分支指令，就能够使 BHB 内容可控.

Chowdhuryy 等人[101]对 Intel Skylake 处理器进行了逆向工程. 他们发现，只要执行 12 个跳转的分支，就能够保证下一次执行条件分支时，分支历史索引模式下 BHB 参与索引的 PHT 表项位置可控.

4.2　触发分支

本节介绍攻击链 a 和 b 的第二阶段. 这一阶段主要考虑的问题，是如何让前一阶段的填充结果可以被本阶段触发的分支使用，即如何让本阶段触发的分支被索引到之前填充的表项. 在上述部件中，由于 RSB 不存在索引机制，因此在 RSB 类型的攻击中，攻击者无需考虑索引问题.

4.2.1　同名索引和别名索引

本文使用图 3 来解释同名索引和别名索引. 同名索引是指攻击者填充预测器时使用的指令地址 A' 和第二阶段触发分支的指令地址 A 相等的索引方式；别名索引指的是二者不等，但经过哈希函数后的索引值 $f(A)$ 与 $f(A')$ 相等的索引方式.

因为 PHT 表项索引粒度为字节粒度[28]，所以大多数利用 PHT 的攻击都使用同名索引. 在跨进程场景下，攻击者和受害者使用不同的地址空间，这时攻击者进程和受害者进程用别名索引到同一 PHT 表项是有可能的，攻击者可以采用基于分支历史的索引方式达到这一点[101]，参见 4.2.3 节.

对于 BTB，同名索引和别名索引的攻击都已经出现. Evtyushkin 等人[23]发现，在 Intel 处理器中，一个分支指令地址只有低 30 位参与 BTB 索引. Zhang 等人[34]通过逆向工程技术进一步发现，这 30 位地址中，高 16 位作为标签，中间 9 位作为 BTB 索引，剩下 5 位作为偏移量. 因此，通过合理构造地址，攻击者可以用地址别名索引到同一 BTB 组并匹配

① Retpoline: A software construct for preventing branch-target-injection. https://support.google.com/faqs/answer/7625886, 2018

相同的 BTB 块.

4.2.2 指令地址索引方式的选择

在图 3 中,指令地址索引是通路 I 表示的索引方式,分支历史索引是通路 H 表示的索引方式,后者使用了分支历史作为索引值计算的一个输入.

如果能在相同上下文环境中重复执行往相同方向跳转的分支足够多次,如 Spectre V1,那么这两种索引方式都将预测分支往这一方向跳转,这时两种索引方式对攻击的成功率没有太大影响.其余情况下,攻击者大都需要避免两种索引方式的交错,仅使用单一的索引方式,从而避免第一阶段的填充位置和第二阶段被触发分支的索引位置失配.

对于 PHT,Evtyushkin 等人[28]发现,可以通过以下几种方式让分支预测器使用基于指令地址的索引方式:①确保使用分支历史索引的目标不在 PHT 表中;②使之前几次历史索引的预测结果不准确,或者还需要较长的训练时间才能达到准确;③使基于指令地址的索引正确率高于历史索引的正确率.实际攻击中,攻击者能通过构造包含大量随机跳转的分支块实现②和③.

对于 BTB,Zhang 等人[34]发现,可以通过以下方式让分支预测器使用基于指令地址的索引方式:①使用直接分支.因为只有间接分支会用到基于分支历史的索引;②使 BTB 中存在使用指令地址能索引到的表项,但不存在使用分支历史能索引到的表项;③使用图 6 所示的状态机,通过控制预测准确率使处理器倾向于使用基于指令地址的索引方式.图 6 中,状态 BHB 表示使用分支历史索引,状态 PC 表示使用分支指令地址索引,路径分别表示使用指令地址索引(PC-1)或使用分支历史索引(BHB-1)的预测结果正确.

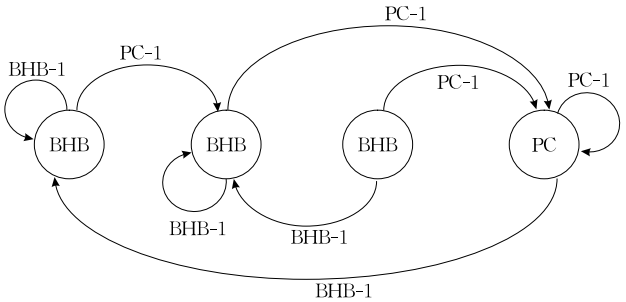


图 6 BTB 预测的索引机制选择状态机[34]

4.2.3 分支历史索引方式的选择

Huo 等人[90]提出一种仅使用分支历史索引来构造 PHT 侧信道的方法.为了使处理器选择这一索引方式,他们提出使用循环分支的方法,即构造一组足够长的随机方向分支序列后多次循环执行该序列.

为了令 PHT 使用基于分支历史的索引方式,攻击者需要了解受害者的分支历史.Huo 等人提出一种算法[90].首先,令攻击者与受害者均按照 4.1.4 节的方式主动执行相同的分支指令填充 BHB,这时攻击者和受害者在同一地址的分支将索引至相同的 PHT 表项.接着,令受害者减小主动填充的 BHB 位数,将这些 BHB 位改用其最近的几条历史分支填充.同时,攻击者主动调整相应的 BHB 位,使攻击者和受害者再次索引到相同的 PHT 表项,这时即可恢复部分受害者的分支历史.

Chowdhuryy 等人[101]更加细致地研究了使 PHT 从指令地址索引转移到分支历史索引的控制方式.他们发现,在基于指令地址的索引方式连续预测错误 3 次时,分支预测器的下一次预测会选择基于历史的索引机制.因此,只要按照“跳转、不跳转”相间的模式执行一个分支 6 次,就足以确保该分支的下次预测使用基于分支历史的索引方式.

Barberis 等人[106]利用 BHB 索引 BTB 表项,实现了比仅使用指令地址索引 BTB 更加复杂的分支预测攻击.

4.2.4 不同攻击方式下的地址索引方式

Canella 等人[27]也按照攻击方式对瞬态执行攻击进行了分类,根据他们的分类结果,瞬态执行攻击可以分为同进程同地址、同进程异地址、跨进程同地址和跨进程异地址四种.这种分类方式同时考虑了攻击场景和攻击原理,但没有深入到分支预测器的利用细节,比如没有考虑第二阶段被触发的分支是根据指令地址还是分支历史索引.

表 6 展示了把 Canella 等人[27]的分类映射到本文的分类方式后的结果.为方便起见,这里假设各种攻击方式都使用基于地址的索引方式.

表 6 从攻击方式到索引方式的映射情况

攻击方式		索引方式
同进程	同地址	同名索引
	异地址	别名索引
跨进程	同地址	同名索引/别名索引
	异地址	别名索引

同进程情况下,攻击者和受害者拥有相同的地址空间;跨进程情况下,攻击者和受害者的地址空间可能不同.同地址指攻击者和受害者使用相同的指令地址,异地址指攻击者和受害者使用不同的指令地址.在跨进程的情况下,即使是同地址,也有可能由于

ASLR 机制或不同的地址域而存在别名情况[108].

4.3　分支预测侧信道与隐藏通道

本节介绍图 5 中攻击链 a 的第三和第四阶段. 分支预测器的预测结果能间接反映一条分支指令的执行信息，比如分支指令的地址[23]、分支跳转方向[28]和指令粒度的程序执行流信息[24]. 同时，分支预测器的状态也能够用来传递信息[92]. 本节首先介绍分支指令可能泄露的信息，然后分别介绍 PHT 侧信道、PHT 隐藏通道和 BTB 侧信道的利用原理.

4.3.1　分支指令泄露的信息

通过分支指令的地址或跳转目标，攻击者能破解地址随机化(ASLR)或获取密钥. 图 7 展示了分支指令如何泄露信息.

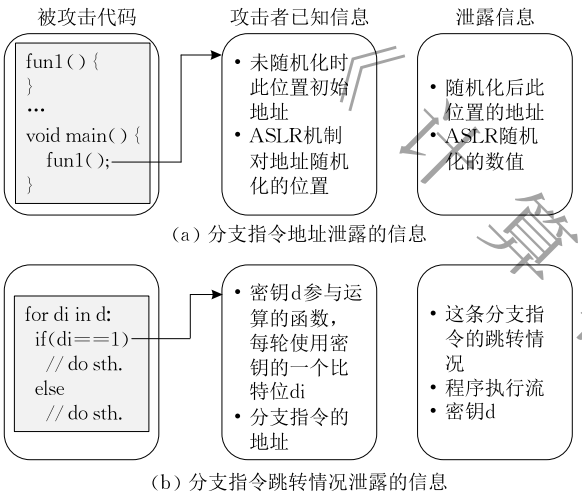


图 7　分支指令泄露信息的原理

(a)利用分支地址信息破解 ASLR. ASLR 用于进程地址空间随机化. 在代码编译并链接成可执行文件后，有固定的地址空间，如果开启 ASLR，那么每次执行的进程会通过一个随机偏移量映射到不同的地址空间[109]，在进程创建后，该偏移量固定不变. 部分操作系统在引导阶段也会生成一个随机序列作为偏移量，这些偏移量通过影响虚拟地址，使内核空间在物理内存中随机映射[110]. 为了破解 ASLR，攻击者首先通过静态二进制文件分析锁定一个待探测分支的初始地址，如(a)中主函数的 fun1 调用. 之后攻击者借助侧信道探测进程执行时 fun1 函数调用指令的实际地址，从而获取地址偏移量，即 ASLR 使用的随机化数值[23].

(b)利用分支的跳转情况破解密钥. 部分加密计算，如模幂算法或蒙哥马利阶梯算法[88]，会遍历私密数据，并在循环内部放置一个分支指令，根据每一位的值是 0 或 1 执行不同操作. 攻击者通过侧信道探测该分支跳转情况，就能获取密钥信息[20]. 该原理也被用于探测受害者进程的执行流[24].

4.3.2　PHT 侧信道

PHT 能反映分支指令的跳转信息. 为此，攻击者需要了解 PHT 表的匹配机制. 现有的商用处理器大都在 PHT 中实现了分支预测状态机[28]. 图 8 展示了 PHT 侧信道的基本原理. 攻击者填充 PHT 表项之后，控制了状态机的初始状态 S_0. 从初始状态出发，攻击者能探测每一步分支的跳转情况，使得状态机的状态在执行完每一轮分支后都是已知的. 图中用 S_1 表示探测新一轮分支时 PHT 表项的状态. 在阶段二，攻击者触发分支后，设分支实际跳转方向为攻击者未知的 D_1，同时分支预测器将给出攻击者已知的预测方向 P_1，攻击者能探测本轮分支预测器的预测正确性 T_1，并结合已知的 S_1 来判断分支实际跳转方向是否为 P_1，进而推导出 D_1. 探测 T_1 需要攻击者有很精确的同步能力，因此有的攻击选择在阶段三中执行部分受控的探测分支，来探测阶段二结束后 PHT 表项的状态 S_2. 这时，攻击者已知分支实际方向 D_2，通过探测分支预测的正确性 T_2 来判断分支预测器提供的预测方向 P_2，从而确定分支预测器状态 S_2，然后结合已知的 S_1 推导出 D_1.

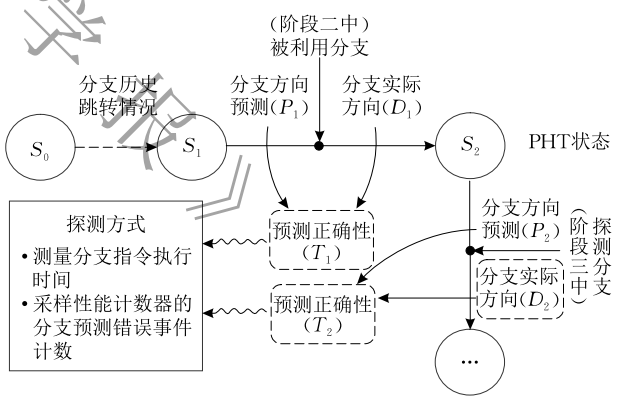


图 8　PHT 侧信道原理

根据攻击场景，攻击者选用不同的探测方式来探测分支预测的正确性，探测方式包括测量分支指令的执行时间和测量性能计数器中分支错误事件数. 当分支预测错误时，分支错误事件数加 1，并且处理器将回滚由于分支预测错误而错误执行的操作，从而造成更大的时间开销.

目前有五种分支预测攻击利用了 PHT 侧信道. BranchScope 攻击[28]通过探测 S_2 的方式利用侧信道. 触发分支后，攻击者继续执行两个跳转分支，并探测分支预测器的预测情况. 然后，攻击者再探测连

续执行两个不跳转分支时的预测情况，从而推断 S_2 的状态，进而推断阶段二触发分支的跳转情况 D_1 及密钥的其中一位，这种主动执行探测分支的探测方式可以避免复杂的同步操作. Fault Attack[87] 在阶段二注入单比特翻转故障，并继续执行三轮受害者进程中的分支，同时测量故障注入前后错误预测数的差值作为 T_1. 根据 T_1 和 S_1，攻击者能够从事先枚举出的所有可能情况中查询，从而恢复密钥零到三位数据. HPC-based Attack[88] 中，攻击者选择受害者进程中由密文输入值和密钥共同决定分支方向的分支指令作为阶段二触发的分支. 在已知 S_1 的前提下，攻击者预先构造四类不同的密文输入值，分别对应迭代的密钥位为 1 且预测错误、密钥位为 1 且预测正确、密钥位为 0 且预测错误、密钥位为 0 且预测正确四种情况. 然后，攻击者分别执行四类密文并通过测量执行时间的方式推测出 T_1，从而推测出 D_1 及密钥的其中一位. Template Attack[86,89] 基于分支预测采样，预先枚举所有可能的 T_1，然后和实际采样到的 T_1 进行匹配，根据最小二乘法推测出最有可能的 D_1 及密钥的其中一位. Bluethunder 攻击[90] 通过观测到的 P_2 动态调整下一次探测分支的跳转方向 D_2，以维护一个确定的 S_2 状态. 同时，攻击者观察动态调整的 D_2 序列来判断 D_1 及密钥的其中一位.

4.3.3 PHT 隐藏通道

在 PHT 隐藏通道中，存在一个位于高权限的木马进程和一个位于低权限的间谍进程. 木马进程先利用其高权限把私密数据编码到 PHT 表项，然后间谍进程再通过探测 PHT 表项恢复私密数据[92]. 图 9 展示了 PHT 隐藏通道的利用原理. 在完成攻击阶段二后，PHT 表项状态 S 编码了私密数据. 与 PHT 侧信道的不同之处在于，攻击者无需考虑 PHT 表项的初始状态，且在阶段三中通过探测分支恢复状态 S.

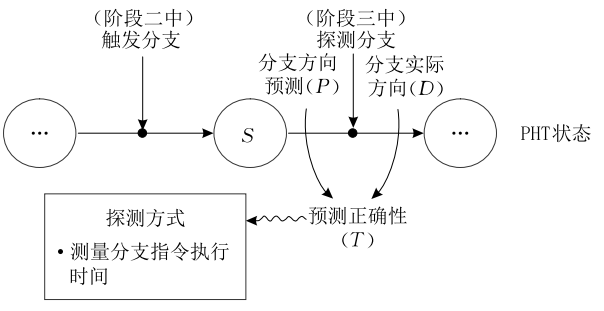


图 9 PHT 隐藏通道原理

目前，不依赖瞬态执行的 PHT 隐藏通道有两种实现方式. RSC 隐藏通道[91] 中，木马进程通过执行大量不同地址处的跳转分支来编码 1，通过执行大量不跳转分支来编码 0. Evtyushkin 等人[92] 认为，执行十万条分支能够同时兼具高传输率和高准确率. 间谍进程则执行与木马进程传递 1 时相同的跳转分支块，并测量执行分支块时的时间. 时间可以根据统计方法分成两个集合，短时间集合表示 T 为正确，木马传递 1；长时间集合表示 T 为错误，木马传递 0. CC 隐藏通道[93] 使用了 PHT 争用的原理，在传递 1 时，木马进程执行大量分支指令，其中有一半指令跳转，另一半指令不跳转；传递 0 时，木马进程执行大量空指令（nop）. 间谍进程执行与木马进程传递 1 时相同的分支块，并测量执行时间，时间短表示不存在 PHT 争用，木马传递 0，时间长表示存在 PHT 争用，木马传递 1.

4.3.4 BTB 侧信道

BTB 能反映阶段二中分支指令的地址和跳转方向等信息. 攻击者需要预先知道 BTB 的组索引与标签匹配方式，有时还需要知道 BTB 的规模. Uzelac 等人[32] 设计的 BTB 逆向工程的工具给 BTB 侧信道攻击提供了方便. BTB 侧信道的原理如图 10 所示. 图中给出了简化后的 BTB，每行表示一个 BTB 组，每格表示一个 BTB 块.

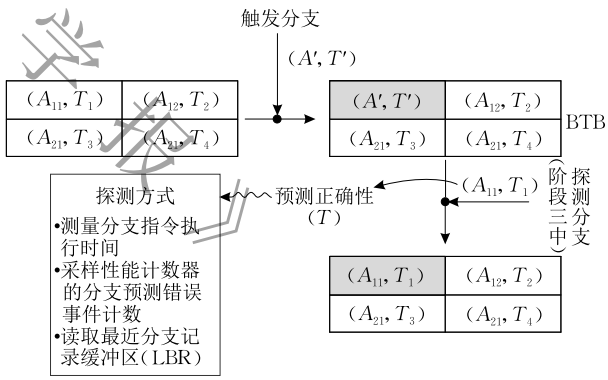


图 10 BTB 侧信道原理

如第 2 节所述，每个 BTB 块都存放了一个压缩后的指令地址 A 作为标签，并存放了压缩后的分支目标地址 T. 对于大部分处理器，阶段二中触发的分支如果跳转，就会把新的指令地址和目标地址对填入一个 BTB 块；若分支不跳转或未被执行，则不被填入 BTB 块[111]. 阶段三中，攻击者通过测量时间或计数器采样等方式观察 BTB 块是否发生替换.

Branch Shadowing 攻击[24] 等三种攻击利用了 BTB 侧信道. SBPA 攻击[20-22] 中，攻击者使用驱逐集填充 BTB 组，然后在阶段二触发受害者分支. 阶段三中，攻击者需要观察阶段二的分支是否驱逐了

BTB 组中的一个块. 为此，攻击者多次执行驱逐集中的分支指令，并测量执行时间，如果执行时间长，说明 BTB 组中的一个块被驱逐，进一步说明阶段二中的分支发生跳转. 反之，如果执行时间短，说明阶段二的分支不跳转. 根据上述分析，攻击者能够恢复密钥的其中一位. Jump over ASLR 攻击[23]通过 BTB 侧信道恢复分支指令的地址信息，然后根据 4.3.1 节所述原理破解 ASLR. 这一攻击中，攻击者只填充单个 BTB 块而非整个 BTB 组. 为了确定阶段二的分支指令地址，攻击者需要用不同地址的 BTB 块进行多次实验. Branch Shadowing 攻击[24]中，攻击者构造与 SGX 中受害者进程地址空间排布相同的进程，并在阶段一执行受害者进程使之填充 BTB 块. 在阶段二中，攻击者进程主动触发分支. 阶段三中，攻击者通过 Intel 处理器提供的最近分支记录缓冲区(Last Branch Record，LBR)观察前几次分支指令的预测结果，从而分析阶段二中分支的跳转方向，进而恢复受害者进程的执行流.

4.4　瞬态执行攻击

瞬态执行攻击是处理器的重大安全问题，它破坏了信息系统的数据私密性. 其中，分支预测器的错误预测是瞬态执行的重要来源. 本节首先介绍由分支预测错误引发瞬态执行的基本原理，然后介绍攻击者如何借助分支预测器和其余微架构技术构造合适的瞬态执行窗口，最后介绍攻击者如何在瞬态执行窗口泄露私密数据. 本节对应攻击链 b 的第三和第四阶段，且没有在瞬态窗口内部利用额外的分支预测机制.

4.4.1　由错误分支预测引发的瞬态执行

除了传统流水线中的取指、译码、执行等阶段外，现代高性能流水线引入分发(dispatch)、发射(issue)、提交(commit)和退出(retire)等新的阶段，以及保留站(Reservation Station，RS)和重排序缓冲区(Reorder Buffer，ROB)等微架构部件[4]. 在分发阶段，处理器把译码后从指令得到的操作(operation)或微操作传递到 RS 中，RS 用于存放由于源操作数未准备好而暂时无法执行的操作. 同时，处理器按照程序顺序把操作存入 ROB 中[95]. 在发射阶段，处理器把源操作数准备完毕的操作或微操作传送到空闲的执行端口. 因为发射操作到执行端口的时机是源操作数准备完毕，所以实际的执行过程是乱序的. 在提交阶段，处理器对 ROB 顶部的操作进行检查. 按程序顺序，该操作是最早写入 ROB 的操作，如果出现分支预测错误或异常等情况，处理器会压缩(squash)ROB 中剩余的操作，即把它们从微架构中清除[14]. 处理器通过这种方式实现了乱序执行中的精确异常处理[2]. 在退出阶段，由处理器在提交阶段确认正确性的操作正常退出，其造成的影响从微架构层面转移到系统结构层面.

处理器取得一条分支指令后，如果判断分支指令是否跳转或表示分支指令跳转目标的源操作数未就绪，处理器会使用分支预测器的预测结果[4]. 直到提交阶段，处理器才会对分支指令的预测结果进行检测. 如果发生预测错误，那么从预测错误到处理器发现错误存在一个窗口期，这一窗口期内执行的指令或操作将不会在系统结构层面提交，这些指令或操作的执行过程被称为“瞬态执行”[27]. 处理器检测到预测错误后，会通过回滚操作清除瞬态执行的指令或操作. 为提高性能，瞬态执行指令对 Cache 等部分微架构组件的影响，不会被回滚；此外，瞬态执行过程产生的端口争用也不可避免. 因此，攻击者能够通过一些微架构侧信道还原指令瞬态执行的结果，这是瞬态执行攻击的基本原理.

在错误的分支预测引发的瞬态执行中，瞬态执行的指令在 ROB 的位置均位于被错误预测的分支指令之后，因此这些指令触发的异常，如数组越界读写，不会马上被处理器处理. 反之，读写结果可能会通过微架构旁路被其余瞬态执行的指令利用. 根据这一原理，攻击者能够越权访问受害者进程的数据[25]，或者瞬态修改受害者进程的执行流[98]，这对系统安全造成了巨大威胁.

4.4.2　构造瞬态执行窗口

在瞬态执行攻击中，为构造合适的瞬态执行窗口，攻击者至少需要考虑两个问题. 一是如何构造瞬态执行窗口的起始指令地址；二是如何增加瞬态执行窗口内的指令数量.

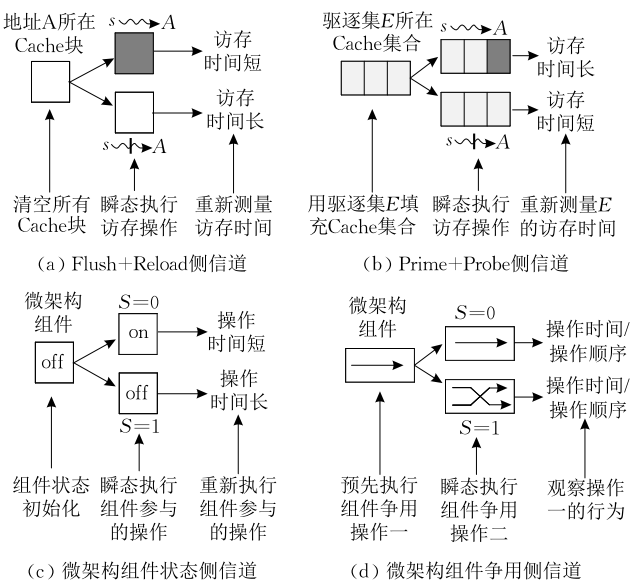
为了解决第一个问题，攻击者需要触发错误预测并且跳转到攻击者已知的代码片段(gadget). Spectre V1 等攻击中，攻击者利用 PHT 产生的错误方向预测触发瞬态执行窗口；Spectre V2 和 ret2-spec[37]等攻击中，攻击者利用 BTB 和 RSB 产生的错误目标预测触发瞬态执行窗口. 其中，RSB 的栈特性使攻击者能使用更多的方法触发错误预测. Maisuradze 等人[37]总结了利用 RSB 触发错误预测的几种方法，包括使用上下文切换、异常捕获、长跳转和直接修改程序调用栈. Zhang 等人[34]还利用了一种特殊的错误预测，他们称之为分支跳过木马. 攻击者预先把分支指令的跳转目标从指令 Cache 中清

除后，处理器将会错误地预测分支不跳转，本研究团队利用 ARM 处理器的 RSB 触发错误预测时，同样发现，如果 RSB 栈顶的地址在指令 Cache 中缺失，那么处理器会预测返回指令不跳转，并在瞬态执行窗口顺延执行地址空间中相邻的下一条指令.

为了解决第二个问题，攻击者需要尽可能延长错误的分支预测被处理器检测出来的时间，也就是该分支指令被解析(resolve)的时间. 现有的大多数攻击都是将决定分支指令跳转方向或跳转目标的源操作数从 Cache 中清除，使处理器花费多个时钟周期从内存中获取操作数，从而延长处理器解析此分支指令的时间，进而延长分支指令从发射到提交的时间. 处理器瞬态窗口的大小与处理器微架构的实现相关，Wampler 等人[105]通过实验观察到，在 Intel 和 AMD 处理器中，通过上述方式构造的瞬态窗口大约能执行 100 到 200 个微操作.

4.4.3 微架构侧信道

瞬态执行的指令无法在系统结构层面观察到结果，也就是说，攻击者无法从内存、寄存器等系统结构层面的数据单元中观察到瞬态执行. 然而，因为错误预测而造成的微架构状态，比如数据 Cache 的内容，没有在现代处理器中回滚. 因此，Cache 侧信道是攻击者利用瞬态执行攻击获取私密数据的主要手段[27]. 除此之外，由于微架构部件特性[94]或端口争用[24,96,103]引发的指令执行时间差异也是攻击者获取瞬态执行窗口数据的方式. 图 11 展示了瞬态执行攻击中常用的 4 类微架构侧信道.



(a) Flush+Reload侧信道　(b) Prime+Probe侧信道

(c) 微架构组件状态侧信道　(d) 微架构组件争用侧信道

图 11 瞬态执行攻击中使用的微架构侧信道

(a) 利用了 Cache 侧信道 Flush+Reload[112]. 在瞬态执行之前，攻击者通过一定方式清空全部 Cache 块. 在瞬态执行中，攻击者通过访存指令，把私密数据编码到某个数据 Cache 块中. 在瞬态执行之后，攻击者依次访问所有 Cache 块，如果某个 Cache 块的访问时间明显小于其余 Cache 块，则说明瞬态执行中私密数据被编码到这一 Cache 块. 攻击者通过多次测量取平均值的方式来减小其余访存指令造成的噪声. 使用该侧信道要求攻击者和受害者有一段共享的内存空间.

(b) 利用了 Cache 侧信道 Prime+Probe[107]. 与(a)的不同之处在于，攻击者预先使用驱逐集 E 填充一个 Cache 组，然后在瞬态执行中把私密数据编码到一个 Cache 块. 如果这一 Cache 块和驱逐集 E 映射到同一 Cache 组，那么将替换组中的一个 Cache 块. 攻击者在瞬态执行之后重新访问一遍驱逐集 E 并测量访存时间. 如果访存时间较长，说明私密数据被编码到这一 Cache 组；反之说明私密数据与这一 Cache 组无关. 利用这种侧信道，攻击者无需使用 Cache 块的清除操作或共享内存，这些操作往往需要攻击者拥有操作系统级的权限[26]. 使用该侧信道要求私密数据按照 Cache 组的粒度编码，因此攻击者需要了解机器的 Cache 参数.

(c) 利用了微架构部件状态以及相关操作的执行时间来传递私密数据. 在 NetSpectre 攻击[94]中，攻击者利用 AVX2 单元的特性作为微架构侧信道. 为了节省功耗，Intel i5-6200 处理器在 AVX2 单元空闲时间长达 1 ms 后掉电，而掉电后执行 AVX2 指令时需要重新上电，比正常状态下执行 AVX2 指令慢 100 个时钟周期左右. 攻击者首先等待 1 ms 使 AVX2 单元掉电，然后在瞬态窗口根据私密数据是否为 0 选择是否执行 AVX2 指令. 瞬态执行结束后，攻击者立刻执行另一条 AVX2 指令并测量其执行时间，通过执行时间长短确定私密数据的一位.

(d) 利用了组件争用侧信道. 攻击者启动一个争用组件的进程，然后在受害者的瞬态执行窗口中根据私密数据选择是否执行另一个争用组件的操作. 攻击者测量攻击者进程中组件争用操作的执行时间或者观察某些操作的顺序，从而确定单位的私密数据. 在 SMoTherSpectre 攻击[103]中，攻击者和受害者通过循环右移指令实现执行端口争用，并通过测量执行时间来确定私密数据的一位. 在 SpectreRewind 攻击[96]中，攻击者利用浮点运算争用来延迟被错误预

测的分支指令的操作数获取时间，并根据私密数据造成不同程度的延迟，攻击者通过测量延迟时间来确定私密数据的一位. 在 Speculative Interference 攻击[95]中，攻击者分别利用缺失状态处理寄存器争用、执行端口争用和保留站争用来影响两条访存指令的执行顺序，攻击者还利用 Cache 替换策略来恢复访存指令的执行顺序，进而确定私密数据的一位.

4.5　瞬态分支攻击

本节对应攻击链 b 的第三和第四阶段，并且攻击者需要在瞬态窗口内利用嵌套预测机制. 本节先介绍瞬态窗口内的分支预测机制，然后介绍攻击者如何利用这该机制完成攻击.

4.5.1　瞬态窗口内的嵌套预测

对于瞬态窗口内的分支指令，Intel、AMD 和 ARM 等主流商用处理器会在瞬态窗口做更深层的预测执行. 如果瞬态窗口内的分支更早被解析，那么不同处理器有不同的处理策略. 其中，Intel Skylake 等处理器的处理策略如图 12 所示.

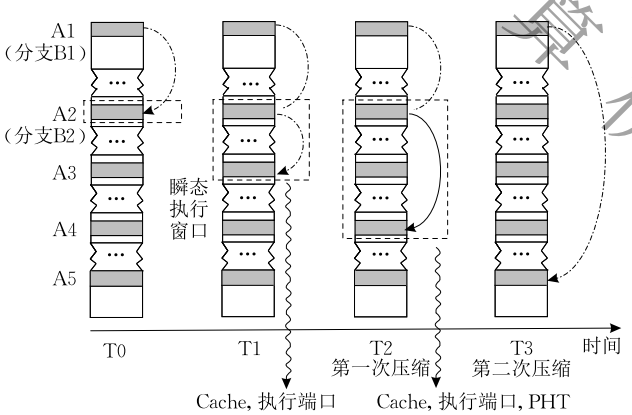


图 12　瞬态窗口内的嵌套预测机制

图中的可执行页有 5 个不连续的地址，即 A1 到 A5. 地址 A1 和 A2 处分别存在一条分支指令 B1 和 B2，不妨假设 B1 的解析时间为 T3，B2 的解析时间为 T2，其中 T3 大于 T2. B1 的正确跳转地址为 A5，B2 的正确跳转地址为 A4. 若分支预测器预测 B1 跳转到 A2，预测 B2 跳转到 A3，那么处理器将在 A1 处发生错误预测，产生一个以 A2 为起点的瞬态执行窗口，记该时刻为 T0.

在瞬态执行期间，对于小于 T2 的时刻 T1，分支 B2 也发生错误预测，使得瞬态窗口延伸到 A3 的位置. T2 时刻，B2 分支被解析，这时，Intel Skylake 和 Kaby Lake 等架构的处理器会在瞬态窗口进行一次压缩，但只回滚到 B2 分支的另一个方向，而不影响 B1 分支的解析. 之后，B2 将在瞬态窗口沿正确方向执行，使瞬态窗口扩展到 A4. 最后到达 T3 时刻，处理器完成 B1 的解析，发现 B1 预测错误，那么从 T0 到 T3 的处理器执行都将被回滚. 最终，B1 沿正确方向执行，到达地址 A5.

在上述执行过程中，由于瞬态窗口内存在分支，瞬态执行窗口在不断调整，期间对 Cache 等微架构的影响不会被消除. 特别地，Chowdhuryy 等人[101]发现，如果瞬态窗口内的分支 B2 提前于触发瞬态窗口的分支 B1 被解析，那么 PHT 表项也会朝着 B1 的正确跳转方向更新，因此 PHT 也能作为微架构侧信道的部件，按图 7(b)的原理泄露数据. 实际上，早在 1994 年的一篇工作中[113]，研究人员就发现，瞬态窗口内的 PHT 更新能够更好地提升分支方向预测的准确率. 这是因为，瞬态窗口内解析的分支可能和造成瞬态窗口的分支没有语义联系，反而提前反映了处理器后续执行过程中该分支的上下文环境，从而提前正确地调整了 PHT 表中该分支指令的方向.

目前为止，研究人员尚未在 Intel Haswell、AMD 和 ARM 处理器发现这一机制.

4.5.2　嵌套预测与瞬态分支攻击

嵌套预测意味着处理器有可能在一个瞬态窗口分别执行一个分支的两个方向，还可能在瞬态窗口执行任意其它地址的指令. 现有的攻击主要分为两类，一是使瞬态窗口内的分支朝着错误的预测方向执行而不被解析，二是尽可能早地解析瞬态窗口内的分支，使之朝正确方向瞬态执行. 前者能够尽可能多地扩展瞬态执行的指令组合，因为分支指令能够把不同地址的指令在瞬态窗口串联起来；后者能够帮助攻击者更精确地控制瞬态窗口的执行流.

第一类攻击中，攻击者可以同时从 Cache 中清除所有分支指令所依赖的操作数，使各个分支的解析时间都大大增加，且后面的分支指令不会早于第一条触发瞬态窗口的分支指令被解析. 典型的攻击包括 ExSpectre[105]、SplitSpectre[79] 和 SpecROP[104]. 这三个攻击都在瞬态窗口利用额外的分支指令来组合代码片段，使攻击者能够更简单地利用受害者代码进行其余类型的瞬态执行攻击. Bhattacharyya 等人[104]发现，瞬态窗口中额外执行 3 个分支，即连接 4 个代码片段的成功率能够到达 50%，但在连接 6 个代码片段时的成功率已不足 10%.

第二类攻击中，攻击者只延长触发瞬态窗口的分支的解析时间，而对于瞬态窗口内的分支，则通过

提前写入 Cache[99,114]或利用处理器微架构的存储到加载转发(Store-To-Load Forwarding,STL)机制[98]来加速解析.其中,Spectre V1.1[98]在瞬态窗口内把执行流改到攻击者在瞬态窗口自行注入的地址,而不像第一类攻击预先训练瞬态窗口内分支的跳转地址.BlindSide 攻击[99]把执行流改到攻击者想要探测的未知地址,并通过微架构状态变化观察该地址是否存在指令以及潜在的指令语义.Branch-Spectre 攻击[101]在瞬态窗口执行依赖于私密数据的分支,并根据该分支对 PHT 表项的修改来推断私密数据,增加了瞬态窗口内可能泄露数据的代码片段类型,扩展了瞬态执行攻击的可用性.

5 分析与评估

第 4 节介绍的分支预测攻击原理来自学术界主要的系统安全会议和期刊.表 5 所列举的 30 个攻击中,Spectre V1 和 Spectre V2 出自同一篇论文[25],两个分支木马出自同一篇论文[34],其余攻击都有独立的论文相对应.表 7 描述了论文的发表情况,从中可见,分支预测安全问题受到系统和信息安全会议和期刊的高度重视,超过半数的工作在 A 类会议和期刊发表.

表 7 本文所整理的分支预测攻击的论文发表情况*

分级	会议/期刊中涉及的分支预测攻击论文数量					合计
A	CCS	USENIX Security	ASPLOS	S&P	MICRO	16
	3	2	3	2	1	
	TC	NDSS	HPCA	CHES	—	
	2	1	1	1	—	
B	TACO	ASIACCS	ESORICS	RAID	ACSAC	5
	1	1	1	1	1	
其它	—	—	—	—	—	7

注:*会议和期刊分级参考自 2019 年清华大学计算机学科推荐学术会议和期刊列表(TH-CPL).部分攻击发表了多篇论文,本文只选取分级最高的一篇计入."其它"部分的论文可能发表在 TH-CPL 中未涉及的期刊或会议,也可能尚未发表.

根据本研究团队的了解,目前的分支预测攻击和相关综述中,尚未有工作全面系统地对各类分支预测攻击进行分析与比较.实际上,由于各类分支预测攻击的目标处理器、威胁模型和攻击目的有很大区别,给各类攻击寻找单一的分析和评价指标是非常困难且缺乏实际意义的,因此,第 5 节依据第 3 节建立的攻击模型和第 4 节的分析,从攻击原理、攻击场景和攻击过程三个维度对现有的分支预测攻击分析与评估.为了使读者更加清晰地把握文章脉络,图 13 展示了三个分析维度的层次关系,以及各维度的评估意义和价值.

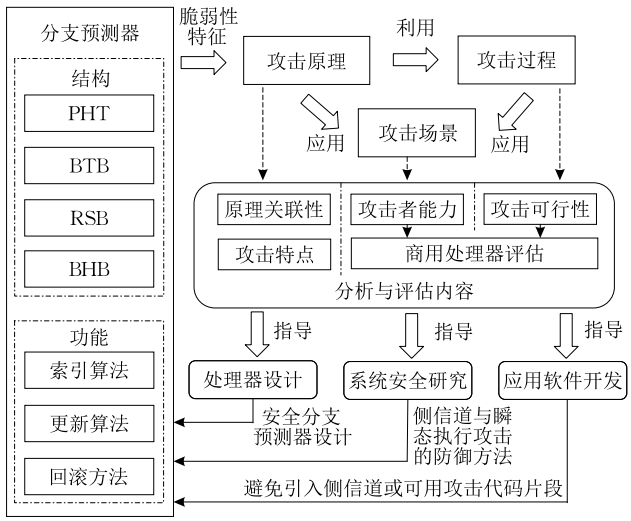


图 13 各类分析维度的层次关系和评估意义

本节内容安排如下:5.1 节基于攻击原理,分析分支预测攻击与分支预测机制之间的关联性以及不同利用方式的攻击特点;5.2 节分析和比较了第 3 节建模的 8 个攻击场景;5.3 节通过一个抽象的处理器模型,给出分支预测瞬态执行攻击可行性的一个评估指标;最后,5.4 节从三个维度综合评估了部分主流商用处理器的分支预测器脆弱性,从而指导研究人员开展更加安全的分支预测器设计工作.

5.1 攻击原理分析

在进行防御和安全设计时,针对一类攻击的防御比针对特定攻击的防御往往更有效.因此,研究分支预测攻击与分支预测机制的关联性,有助于从整体上设计覆盖全面的有效防御策略.

从攻击链 a 和 b 可以看出,填充预测器和触发分支是必要步骤.因此,无论是侧信道还是瞬态执行攻击,对分支预测器的利用方式从本质上看是相同的.以 BranchScope 攻击和 Spectre V1 为例,二者在攻击之前,都需要对 PHT 表项进行预填充,然后在受害者进程触发分支索引到预填充表项.在后续阶段,前者主要探测该分支的跳转方向,而后者主要利用该分支的错误预测进行瞬态执行攻击.因此,假设一种安全 PHT 能够抵抗攻击者恶意填充或索引表项,那么就能同时防御这两种攻击.根据这一原理,我们按照利用分支预测器的方式,对分支预测攻击进行了分类整理,如表 8 所示.

表 8　分支预测攻击利用预测器方式*

部件	索引	已知攻击	攻击特点			
			毒化成功率	抗噪声能力	可用攻击片段数量	攻击片段可测性
PHT	同名 PC	Fault Attack，HPC-based Attack，Template Attack，BranchScope，Spectre V1，Spectre V1.1，NetSpectre，SpecHammer，Speculative Interference，SpectreRewind，Spectre V3a，SplitSepctre	高	强	较少	高
	别名 PC	—	—	—	—	—
	分支历史	RSC，CC，Bluethunder，BranchSpectre	较低	较强	较少	低
BTB	同名 PC	SBPA，Spectre V2，SGXPectre，SMoTherSpectre，ExSpectre，SpecROP，BlindSide	高	较强	少	较高
	别名 PC	SBPA，Jump over ASLR，Spectre V2，SGXpectre，SMoTherSpectre，ExSpectre，Distant Collision Torjans，Branch Skip Torjans，SpecROP	较低	弱	较多	低
	分支历史	Branch Shadowing，SpectreBHI	较低	较弱	多	低
RSB	—	Branch Skip Torjans，SpectreRSB，ret2spec，SpecROP	高	弱	多	较高

注：*SBPA、Spectre V2、SMoTherSpectre、SGXpectre、ExSpectreSpecROP、Branch Skip Torjans 等攻击的论文中同时实现了多种利用方式，因此可能出现在多个表项中.

5.1.1　PHT 类型攻击的特点

表 8 表明，相比 BTB 类型攻击，针对 PHT 的攻击大都使用同名地址索引，而没有通过别名地址索引进行. 这与 PHT 的表项属性和索引特点有关. 如第 4 节所述，PHT 索引粒度是字节，且表项只有少数位，能容纳的表项远大于 BTB；此外，间接执行不同分支容易使 PHT 从基于指令 PC 索引转化成基于历史的索引方式，后者更加复杂，一个分支可能映射到多个 PHT 表项，影响攻击成功率.

相比基于指令地址的索引方式，使用分支历史索引方式有两个好处. 第一，攻击者更容易切换到历史索引模式. 切换预测器模式的方法是降低当前模式下预测的准确率. 然而，由于索引的复杂性，降低历史索引模式下的预测准确率是攻击者难以控制的，BranchScope 执行约 10 万条随机分支指令，才有较大概率切换到基于指令地址的索引模式；反之，在基于指令地址的索引模式触发错误预测是容易的，6 条分支指令足够实现这一点[90]. 第二，历史模式下的 PHT 的状态更丰富，支持攻击者通过侧信道获取更多执行流信息. Huo 等人[90]观察到，在基于指令地址索引时，PHT 表项只有 2 位被激活，在基于分支历史索引时则激活 3 位，因此额外增加了 4 个状态.

5.1.2　BTB 类型和 RSB 类型攻击的特点

由于 BTB 表项更少，BTB 的索引机制更加简单，因此构造 BTB 碰撞比 PHT 类型更容易. 此外，BTB 类型的攻击大都没有关注索引机制的切换，这是因为是索引的切换状态很少，在实际攻击时，攻击者只要持续不断地使训练分支朝一个方向跳转，通常就能保证索引到相应表项，参见图 6.

RSB 不存在索引问题，每次函数调用都会往 RSB 顶部写入一个预测目标. 因此，RSB 攻击面对的最大挑战是有限的 RSB 表项和未知的上下文切换噪声. 事实上，与利用 RSB 预测机制相反，SpecROP 和 SGXpectre 等攻击通过大量递归调用清空 RSB，强制返回指令使用 BTB 预测机制，从而避免 RSB 预测机制造成干扰.

5.1.3　主要攻击特点对比

表 8 对各利用方式的多项指标进行了横向比较. 其中，毒化成功率表示填充分支预测器的成功率. 在分支预测攻击中，无论是使用何种部件，多次使用同地址分支进行毒化都能够使分支预测器被填入目标地址. 使用基于别名或历史的索引方式时，由于分支历史是攻击者难以控制的，因此攻击者较难把别名分支索引到目标表项.

在分支预测攻击中，噪声主要来自受害者进程的其它分支指令、操作系统上下文切换时的分支指令以及其他无关进程的分支指令. 与攻击无关的分支被索引到攻击者填充位置的概率越大，说明该攻击抗噪声能力越弱. 由于 PHT 的字节粒度索引特点，PHT 类型攻击的抗噪声能力较强. 由于每次执行函数调用和返回指令都会影响 RSB 的内容，因此 RSB 类型攻击的抗噪声能力最弱. 历史索引方式下，多数分支都会影响 BHB，因此其抗噪声能力也较弱.

攻击片段的数量受到攻击原理、攻击过程和受害者程序特征的影响. 这些攻击片段大都遵循一定的模式，5.3 节将对攻击片段进行深入分析. 在实际攻击中，攻击者能用同名地址毒化和利用分支预测器的情况是很少见的，而别名分支和历史索引方式则能够充分利用不同分支，扩展了可用的攻击片段范围，但要求攻击者充分了解预测器的实现细节.

部分特征过于明显的攻击片段，如 PHT 类型

的分支后访存行为,以及 RSB 类型的返回指令后访存行为,容易被编译器或软件分析工具检测.可测性越高,则能在受害者进程利用分支预测攻击的可能性越低.需要注意的是,可测性与攻击成功率的关系不是绝对的.Kirzner 等人[115]指出,现有的编译器防御策略,如 JumpSwitches[116],虽然能检测到部分攻击代码片段,但也可能为了消除已知代码片段而在受害者进程中引入额外的可用攻击片段.

5.2 攻击场景分析

同样原理的分支预测攻击能在多个不同的攻击场景下应用.表 5 总结的分支预测攻击覆盖了图 4 所构建的 8 个攻击场景,这 8 个攻击场景也完整覆盖了本文所整理的所有分支预测攻击.这些场景的攻击者权限和攻击方式有较大区别,如表 9 所示.

表 9 攻击场景的性质比较

攻击者权限	受害者权限	攻击者与受害者关系	攻击场景
用户权限	用户权限	同进程	⑦ 跨沙箱攻击
		跨进程	① 跨进程攻击 ④ 同步多线程攻击
	内核权限	同进程	② 跨域攻击
		跨进程	② 跨域攻击
内核权限	用户权限	同进程	—
		跨进程	—
	内核权限	—*	⑤ 跨虚拟机攻击 ⑥ 跨 Hypervisor 攻击 ⑧ 恶意木马攻击**
	TEE	同进程	③ 跨 TEE 攻击
		跨进程	③ 跨 TEE 攻击

注:*这些攻击场景的攻击者和受害者都是内核本身,内核本身是具有多个线程且可以在不同核切换执行的,因此讨论同进程和跨进程无意义.

**本文把能够分析并部署程序的用户视为拥有内核权限.

5.2.1 用户权限、内核权限与 TEE

在分支预测攻击的威胁模型中,拥有用户权限的攻击者能够启动进程、执行系统调用、测量程序中某个片段的执行时间、甚至通过 nop 等指令达成 100 以内时钟周期粒度的跨进程同步[103].从用户权限发起的分支预测攻击绝大多数都需要进行时间测量,比如用分支指令执行时间来判断 PHT 或 BTB 表项的状态、用访存时间来判断 Cache 状态和用指令执行时机判断执行端口的状态等.因此,以干扰攻击者的时间测量为目的的防御策略被提出,如降低 Intel 处理器时间戳获取指令 rdtsc 的准确度[117-118]或干扰访存时间[119].为防御攻击场景⑦,Chrome、Firefox 等浏览器都降低了计时器的精确度[27].

如果攻击者拥有内核权限,那么将拥有更多操作系统级别的能力,从而大幅度提高攻击的成功率.内核权限下的攻击者一般拥有以下能力:

(1)使用性能计数器和其它硬件监测或跟踪单元,如分支记录单元[24],更加精确地获取分支指令的执行行为.比如,Intel、AMD 和 ARM 处理器都提供了分支预测错误事件的性能计数;Intel Skylake 等处理器还提供了记录最近几条分支执行情况的缓冲区[24].这些功能单元能比时间测量提供更加精确的攻击信息反馈,提高攻击的成功率.虽然 Intel 处理器的 SGX 禁用这些单元,但由于同步多线程技术,在同一物理核的另一普通进程能够通过这些单元间接获取 TEE 环境下的进程信息.

(2)精确的单指令粒度同步.内核环境下能够轻易改变进程的调度情况,甚至通过配置处理器提供的计时器来完成高频中断,比如 SGX-step[120]技术.精确的同步意味着,无论是同进程攻击还是跨进程攻击,攻击者都能"延缓"受害者进程的执行,使受害者每步操作都在攻击者的控制之内.之后,攻击者就可以通过微架构侧信道来恢复受害者每一步操作的细节,比如分支指令地址、分支指令的执行次数以及分支指令的跳转目标.

(3)控制受害者地址空间.在分支预测攻击中,无论是基于地址索引,还是基于分支历史索引,都与该指令的地址密切相关.因此,大部分分支预测攻击都要求提前获取目标分支的地址.对于拥有内核权限的攻击者,把受害分支固定到某个地址是容易的.

(4)关闭已有的系统安全补丁或处理器可配置的安全属性,比如地址空间随机化策略 ASLR、Intel 处理器的间接分支预测限制(IBRS)、单线程间接分支预测限制(Single Thread Indirect Branch Predictors,STIBP)和间接分支预测屏障(Speculative Store Bypass Disable,IBPB)[27].

(5)降低攻击噪声.跨进程攻击的一个主要噪声来源是进程的上下文切换.上下文切换期间内核线程执行的操作可能会污染攻击者对分支预测器的填充结果.攻击者拥有内核权限,可以减少不必要的上下文切换,或在上下文切换期间通过修改内核函数尽可能避免攻击噪声.

Intel 的 SGX 和 ARM 的 Trustzone 等可信执行环境都把安全边界建立在普通用户进程和不安全的内核上,因此跨 TEE 攻击的攻击模型中,攻击者大都具有内核权限[121].因为这个原因,现有需要内核权限的分支预测攻击中,超过半数选择了跨 TEE 的攻击场景③.

5.2.2　同进程、跨进程与同步多线程

同进程攻击多见于跨沙箱攻击场景，攻击者对受害分支的训练和利用在同一地址空间进行，该场景的攻击比较可控，因为攻击过程不需要切换进程，不会引入太大的噪声.然而，同进程攻击具有较大的局限性：受限于可执行页的共享性，同进程攻击难以构造别名索引，因此同进程攻击中攻击者用于填充预测器和触发攻击的分支往往是同一个分支，攻击者需要能够反复地执行这一分支，并且能够控制决定分支执行方向的条件.在实际的系统中，满足这种条件的攻击片段较少.

跨进程攻击指在同一核上通过交替执行攻击者进程和受害者进程，达到攻击目的.在分支预测场景中，这类攻击比较难以进行，主要原因是上下文切换带来的噪声不可控.

同步多线程攻击同时具有同进程和跨进程攻击的优点，因此是分支预测攻击的常见场景.Intel 和 AMD 等支持同步多线程的处理器中，分支预测器是物理核独有、逻辑核共享的.然而，操作系统往往不区分逻辑核与物理核，在两个逻辑核上执行的进程被视为跨核执行，能够并发执行.因此，同步多线程能够在满足跨进程的前提下不进行上下文切换，既降低了攻击噪声，又为攻击者提供了丰富的分支预测器填充方式.同步多线程的难点在于同步，在用户权限的分支预测攻击中，攻击者往往通过在攻击进程中不间歇地执行填充分支，来绕过复杂的同步问题.

5.2.3　恶意木马攻击场景

恶意木马攻击是分支预测攻击中比较特殊的攻击场景，在 ExSpectre、Distant Collision Torjans 和 Branch Skip Torjans 三个攻击中被使用.恶意木马攻击的一般流程为，攻击者把恶意代码隐藏在看似良性的内核补丁中发布，并绕过软件分析工具的检查，最终被受害用户部署并赋予内核权限，从而使攻击者能够从中窃取任意内核和用户进程的私密数据.这类攻击与其它的内核权限攻击不同，因为在通常意义下的内核权限攻击中，拥有内核权限的攻击者已经能够通过访存操作获取任意用户进程和其它内核线程的所有数据，而无需进行额外的攻击.而恶意木马攻击的难点在于欺骗用户部署此恶意内核程序，可以类比为一种跨内核的攻击，因此不违背威胁模型.结合上述考虑，本文在 Xiong 等人的模型[26]上增加了此场景.

5.3　攻击过程的可行性评估

由于攻击所使用的处理器和内核的配置不同，部分攻击甚至在模拟器进行[95]，因此目前没有工作对分支预测攻击进行统一的评估.事实上，脱离统一的微架构来评价各种分支预测攻击的可行性是不严谨的.为了深入比较和评估分支预测攻击，本节基于第 3 和第 4 节，构建一个简单的处理器微架构模型来对分支预测攻击可行性进行理论评估.

考虑到分支预测侧信道和隐藏通道的攻击方式和攻击目标复杂且多样，没有太大的可比性，因此本节只研究以获取私密数据为目标的瞬态执行攻击.这类瞬态执行攻击严格遵循图 5 的攻击链 b，且攻击者不主动注入数据，便于建模和比较.

需要注意的是，文章提出的评价指标只是比较各类攻击的一个维度.为了达成统一的评估标准，文章省略了大量的攻击准备和后续工作.但是，评估分支预测攻击的可行性，有助于安全研究人员对各分支预测攻击进行横向比较，也有助于安全系统设计者考虑各安全属性的优先级.

5.3.1　模型假设

(1) 处理器微架构

为简化分析，假设有一个被所有分支预测攻击影响的处理器，其 PHT、BTB 和 RSB 有无穷多表项，PHT 表项由 2 位状态机组成.参考 Chen 等人[102]、Evtyushkin 等人[28] 和 Maisuradze 等人[37] 的工作，为确保 PHT 表项可控，攻击者只需要预先执行 2 次条件分支；为确保 BTB 可控，攻击者需要执行 4 次间接分支；为确保 RSB 可控，攻击者需要执行 1 次函数调用.

处理器支持同步多线程技术.每个物理核有 4 个执行端口和 2 个除法部件.处理器只有一级 Cache，该 Cache 有 256 个 Cache 集，相联度为 4，每个 Cache 块大小为 8 字节，即 64 位.

(2) 指令集

假设处理器支持条件直接分支、无条件间接分支，如表 2 所示.特别地，函数调用指令和返回指令视为无条件间接分支.这些指令统称为分支指令.

处理器能够执行加法、减法、乘法、除法和移位运算.这些操作统称为计算指令.特别地，条件分支指令的操作数获取和条件比较视为 1 条计算指令，无条件直接分支指令执行前不需要计算指令.

处理器能够执行 load 和 store 等访存指令，访存地址的计算需要 1 条计算指令.此外，处理器支持时间戳获取指令、Cache 清空指令和读系统寄存器

指令,这些指令统称为系统指令.

(3)模型简化

为降低分析的复杂程度,本节不考虑复杂的执行环境,如上下文切换、进程间同步和缺页异常等造成的额外指令数量开销.此外,本节不考虑攻击者为掌握处理器微架构机制和受害者私密数据地址等信息而产生的额外指令数量开销.

在计算分支预测攻击为获取私密数据所需要执行的指令数量时,本节不计算为了满足攻击的控制流而引入的指令,比如在训练分支时使控制流不断回到该分支的分支指令;不计算对采样到的微架构数据进行保存和分析的指令,比如对多个 Cache 块的访存时间进行排序;不计算与攻击不相关的任何指令;也不计算 mov 指令等数据移动指令.

通过上述简化,本节把分支预测攻击的评估完全集中在攻击原理和攻击过程上,既包含了攻击目标,也绕过了处理器微架构特性、时钟频率和内核配置等复杂的攻击环境.

5.3.2 计算示例

本节以 Spectre V1 的计算为例进行说明,计算过程只参考相关论文的经典验证代码,比如 Spectre V1 利用 PHT 预测,使用同名分支索引,且使用 Flush+Reload 侧信道.

在 Spectre V1 攻击中,根据 5.3.1 节的假设,攻击者需要使用 2 条分支指令进行 PHT 填充,由计算指令的定义知,每条间接分支执行前需要计算分支条件.因此,填充预测器阶段至少需要 4 条指令.类似地,触发分支阶段至少需要 2 条指令.在瞬态执行阶段,攻击者需要先通过 load 指令加载出私密数据,把私密数据左移 8 位后通过 load 指令加载到 Cache 中,这个过程涉及 2 次访存,2 次访存地址计算和 1 次移位运算.最后,在利用 Flush+Reload 侧信道时,Spectre V1 每次能恢复 8 位二进制数,但需要在攻击前后分别访问 256 次 Cache,攻击前清空 256 个 Cache 块,攻击后测量访问 256 个 Cache 块的时间.测量时间需要执行 2 个时间戳获取指令,因此在微架构探测阶段共进行 256 次访存、512 次地址计算和 768 个系统指令.把所有阶段的指令数相加并除以 8,得到恢复单位比特需要执行的指令数 194;这些指令中绝大部分可以在攻击者进程空间执行,而每恢复 8 位数据需要受害者执行的指令只有填充预测器、触发分支和瞬态执行阶段的 12 条,再除以 8 得到 1.5 条.

5.3.3 结果分析

表 10 给出了 9 个基于分支预测的瞬态执行攻击中各阶段所需要执行的指令数量比较.不区分攻击者和受害者时,Cache 侧信道为恢复单位比特数据所要执行的指令数大于其它新侧信道.然而,在实际攻击中,相比总体攻击速度,窃取私密数据的可行性是攻击者优先关注的问题,也是系统防御者重点考虑的问题.因此,本文进一步计算了各个攻击中恢复单位比特数据需要执行的受害者指令数.

表 10 部分瞬态执行攻击各阶段需要执行的指令数量比较

	填充预测器		触发分支			瞬态执行			微架构探测				恢复单位比特数据需要执行的指令数	恢复单位比特数据需要执行的受害者指令数
	分支	计算	分支	计算	访存/系统	分支	计算	访存/系统	分支	计算	访存	系统		
Spectre V1	2	2	1	1	1	0	3	2	0	512	256	768	194	1.50
Spectre V2	4	0	1	0	1	0	3	2	0	512	256	768	193	1.36
Spectre V3a	0	0	1	1	1	0	1	2	0	512	256	768	193	0.75
SplitSpectre	2	2	1	1	1	1	3	2	0	512	256	768	194	1.63
SpectreRSB	1	0	1	0	1	0	3	2	0	512	256	768	193	1.00
SpectreRewind	2	2	1	2	0	1	2	1	0	0	0	2	13	13.00
BranchSpectre *	4	4	1	1	1	1	2	1	2	2	0	2	21	21.00
SMoTherSpectre	4	0	1	0	1	1	2	1	0	4	0	2	16	10.00
Spectre Interference **	2	2	1	3	4	0	2	1	0	15	15	4	49	15.00

注:*论文还提出一种基于分支历史索引的攻击思路,能进一步减少受害者进程需要执行的分支数,但索引过程比较复杂,这里使用更简单的一种.
**论文提出三种实现方式不同的攻击方法,这里取最简单的一种分析.

受害者指令数这一评价指标比总体指令数更有实际意义,因为这一指标更好地反映了攻击的可行性,原因包括:(1)即使存在一些技术方案[104],但在受害者代码空间找到符合攻击需要的代码片段目前仍然是困难的.因此,如果恢复单位比特的私密数据需要受害者执行的指令数越多,那么这一攻击能够在实际代码环境中利用的可能性就越低.实际上,现有的大多数端口争用攻击都是攻击者通过主动构造受害者代码进行验证的,而很难在实际程序中找到类似片段;(2)现有的防御策略中,有一类基于实时监测的防御技术,通过对受害者行为进行实时监控来观察是否有攻击发生[122-125].因此,在受害者进程

空间执行越多代码，就越有可能被监控程序发现；(3)现在已有静态二进制文件分析技术专门查找并修改潜在的风险代码[126-129]. 由于分支预测攻击的代码片段有特定模式，因此需要受害者执行的代码片段越长，这种代码片段就越有可能被分析攻击检测出来.

在这一评价指标下，Cache 侧信道的优势明显大于其余侧信道，这是因为每次执行受害者代码片段后，Cache 侧信道能在微架构探测阶段一次性恢复 8 位数据，而其余依赖私密数据参与进一步分支方向判断的攻击，受限于分支指令只有两个方向，每次只能恢复 1 位数据.

综上，从这一评价指标来看，Cache 侧信道目前仍是分支预测瞬态执行攻击中最具有可行性的侧信道. 当然，该方法只是评价分支预测攻击过程的一个角度. 虽然 Spectre Interference 等攻击提出的新微架构侧信道在可行性上不如 Cache 侧信道，但由于现有的侧信道防御主要集中在 Cache 侧信道，所以这类能绕过 Cache 侧信道防御的新侧信道攻击仍然是处理器安全的重要威胁.

5.4　商用处理器的分支预测器脆弱性综合评估

本文所调研的分支预测攻击全部以商用处理器为攻击对象，包括 Intel 处理器、AMD 处理器和 ARM 处理器. 表 11 总结了部分典型商用处理器架构所受到的分支预测攻击.

表 11　典型商用处理器已发现的分支预测攻击*

处理器	脆弱性部件			已知攻击数量	已知攻击场景	已知攻击过程	
	PHT	BTB	RSB			索引方式	侧信道**
Intel Haswell	RSC,CC,Jump Over ASLR,BranchScope,Bluethunder,Spectre V1,Spectre V1.1,NetSpectre,SpectreRewind,SpecHammer	Spectre V2,Distant Collision Torjans,Branch Skip Torjans	SpectreRSB,ret2spec,Branch Skip Torjans	16	①②③④⑤⑥⑦⑧	同名地址，别名地址，历史	PHT,BTB,Cache,TLB,执行端口状态，执行端口争用，DRAM
Intel Skylake	Branch Shadowing,BranchScope,Spectre V1,SpectreRewind,NetSpectre	Spectre V2,Distant Collision Torjans,Branch Skip Torjans,SGXpectre,ExSpectre,SpecROP,SMoTherSpectre	SpectreRSB,ret2spec,Branch Skip Torjans,SpecROP	16			
AMD Ryzen	Spectre V1,SpectreRewind	Spectre V2	SpectreRSB,ret2spec,Distant Collision Torjans,Branch Skip Torjans	7	①②③④⑧	同名地址，别名地址	Cache,执行端口争用
ARM Cortex-A72	Spectre V1,Spectre V3a,	Spectre V2	SpectreRSB,ret2spec,Branch Skip Torjans	6	①②	同名地址，别名地址	Cache,执行端口争用

注：*部分攻击所使用处理器与表中所列处理器架构均不相同，比如 BranchSpectre 是在 Intel Kaby Lake 处理器完成的.

**这里只总结分支预测侧信道以及在分支预测攻击中用到的瞬态侧信道.

从表中可以看出，Intel 处理器、AMD 处理器和 ARM 处理器的三个分支预测部件都存在安全问题. 由此可见，分支预测机制存在的安全问题是不同处理器架构普遍存在的，也间接反映这些处理器的微架构实现有相似之处. 其中，Intel 处理器所受攻击数量大于 AMD 处理器和 ARM 处理器.

5.4.1　Intel Haswell 和 Intel Skylake

表 11 指出，Intel Haswell 和 Intel Skylake 的分支预测机制没有很大的区别. Wampler 等人[105]发现的区别主要来自分支预测部件的规模，以及是否支持嵌套分支预测. 因此，4.5 节所述的瞬态分支攻击集中在 Intel Skylake 以及之后的 Kaby Lake 架构. 此外，由于 Intel Skylake 架构晚于 Haswell 架构发布，因此 RSC、CC 和 Jump Over ASLR 等攻击是在 Haswell 架构进行的，而未在 Skylake 架构验证.

5.4.2　Intel 和 AMD

Intel 和 AMD 均使用 x86 变长指令集，且均支持同步多线程(Simultaneous Multithreading，SMT)技术，因此能够支持图 4 的攻击场景④. 然而，Intel 的可信执行环境 SGX 的安全边界是不安全的操作系统，因为 SGX 的提出目标之一是在不可靠内核中保护用户进程数据. 因此，相比 AMD 处理器，Intel 处理器存在跨 TEE 的攻击场景③. 表 11 中 Intel 处理器所受攻击相比AMD处理器的增量，很大一部分来自跨 TEE 攻击，如 SGXpectre 和 Branch Shadowing 等. 此外，Intel 处理器更丰富的研究社区和技术手册，也使一些研究人员能够更轻易地逆向出其分支预测器的一些机制，从而提出一些新的利用方式，如 BranchScope 和 BlueThunder 等.

5.4.3　Intel 和 ARM

相比 Intel 处理器，主要面向低功耗设计目标的 Cortex-A72 等 ARM 处理器，不支持同步多线程，不存在图 4 中的攻击场景④，而单核跨进程的攻击难度较大，因此在 ARM 处理器上成功实现的攻击

相对较少.此外,尽管 ARM 处理器实现了可信执行环境 TrustZone,并且安全边界的设定与 SGX 类似,但二者在实现上有很大区别,比如 TrustZone 依托一个安全内核,并且处理器的隔离程度高于 SGX,后者可能与一个不安全进程通过 SMT 技术共享分支预测器等微架构组件.因此,目前没有 ARM 处理器上的跨 TEE 分支预测攻击出现.

Intel 处理器中 SGX 与 SMT 的结合,使得攻击场景更加丰富,研究人员也更倾向于在 Intel 处理器开发更多有效的攻击过程.同时,更多的分支预测侧信道和瞬态执行侧信道有更强的可控性和可测性,如 PHT 侧信道、BTB 侧信道和 TLB 侧信道.因此,对 Intel 处理器上的分支预测侧信道和瞬态侧信道的研究和发现也要多于 AMD 和 ARM 处理器.

6 讨论与展望

本文对现有分支预测攻击进行了详细的整理、总结与分析.分支预测攻击在 2016 年前后和 2019 年前后分别有两个研究热潮,前者主要围绕 SMT 环境下的分支预测侧信道,后者主要围绕 Spectre 等瞬态执行攻击.目前,瞬态执行攻击仍是分支预测攻击的一个主要研究点,而分支预测侧信道也随着分支预测机制细节的研究而有所发展.6.1 节将对分支预测攻击未来的研究方向进行讨论.

如何防御分支预测攻击,尤其是瞬态执行攻击,也成为系统结构安全的一个重要的研究点.6.2 节将讨论保持分支预测机制不变的情况下,现有分支预测攻击的防御策略及其未来的研究方向;6.3 节将介绍安全分支预测器设计与验证的已有成果以及未来的研究方向.

6.1 分支预测攻击研究方向

在分支预测安全研究中,各类攻击的开发和研究遵循的研究方法大体是相同的,本节主要讨论研究方法的发展方向.由于分支预测攻击与分支预测机制密切相关,因此对分支预测器的实现细节进行逆向工程,始终是分支预测攻击的一个重要研究方法.但是,囿于处理器设计的复杂性和不透明性、集成规模的巨大性以及微架构数据的不可监测和不可调试性,逆向工程面临巨大挑战.相比复杂的逆向工程,自动化的攻击生成技术能够更加高效和大批量地发现处理器潜在的安全问题[130],因此也将成为分支预测攻击研究的一类方法.同时,为拓展已有分支预测攻击的攻击面和攻击能力,也有研究人员在已公开分支预测安全问题的细节处挖掘新的攻击,这类研究也将伴随逆向工程和自动化攻击生成这两个研究方向同步推进.本节分别对这些研究方法和研究方向进行讨论.

6.1.1 逆向工程

通过逆向工程研究分支预测器的工作原理,然后构造分支预测攻击,是现有分支预测攻击的最主要研究方法.表 12 总结了部分分支预测攻击所做的逆向工程.

表 12 分支预测攻击所做的逆向工程

攻击	处理器架构	部件	描述
Jump Over ASLR	Intel Haswell	BTB	基于地址的索引方式
BranchScope	Intel Skylake, Intel Haswell, Intel Sandy Bridge	PHT	索引粒度; PHT 表项内容; 激活基于地址的索引方式
Bluethunder	Intel Haswell, Intel Skylake, Intel Coffee Lake	PHT BHB	BHB 更新算法
Spectre V1, Spectre V2	Intel Haswell, Intel Ivy Bridge, Intel Broadwell, Intel Skylake, Intel Kaby Lake, AMD Ryzen, ARM Cortex-A57	PHT BTB BHB	瞬态执行; BHB 更新算法; BTB 基于分支历史的索引方式
SpectreRSB, ret2spec	Intel Skylake, Intel Haswell	RSB	RSB 表项内容; 更新机制; 预测方式
ExSpectre	Intel Haswell, Intel Skylake, ARM Ryzen	BTB	BTB 嵌套预测
Distant Collision Torjans,Branch Skip Torjans	Intel Haswell, Intel Skylake, Intel Kaby Lake, AMD Ryzen	BTB	BTB 表项内容; 索引方式; 激活基于指令地址的索引方式; 激活基于分支历史的索引方式; 静态预测机制
SpecROP	Intel Skylake, Intel Coffee Lake, Intel Comet Lake	BTB	BTB 嵌套预测
SplitSpectre	Intel Kaby Lake, AMD Ryzen, Intel Broadwell, Intel Skylake	RSB PHT	RSB 规模; PHT 嵌套预测
BranchSpectre	Intel Skylake, Intel Coffee Lake, Intel Cascade Lake	PHT	PHT 嵌套预测; PHT 索引方式; 激活基于分支历史的索引方式

逆向工程的主要对象是 Intel 处理器,主要原因之一是 Intel 处理器的相关安全研究工作和设计文档比 AMD 和 ARM 更丰富.逆向工程的方法主要是

设计一段测试代码，并在测试代码的执行过程中使用性能计数器(Performance Monitor Counter，PMC)或时间戳指令观察执行情况. PMC 能够记录处理器微架构的一些事件，比如分支指令执行、分支指令预测错误和分支指令退出等[29-30]. 这些事件能够间接反映 PHT 和 BTB 等表项的内容，从而帮助研究人员推导分支预测器的预测机制.

Spectre V1 和 Spectre V2 攻击的研究是逆向工程工作的典型代表. Schwarz 和 Gruss 等人[121]最先考虑的是攻击场景③，即拥有内核权限的攻击者攻击 TEE，这时攻击者能使用 PMC 监测受害者进程的执行. 在研究过程中，PMC 计数表明错误的分支预测可能会导致部分指令的瞬态执行. 发现这一现象后，Schwarz 和 Gruss 等人进而结合 Cache 侧信道攻击开发出攻击场景①、②和⑦等用户权限下的分支预测攻击. 上述攻击场景的转移表明，逆向工作没有攻击模型的约束，在研究分支预测机制时，研究人员不需要考虑安全边界问题，而能够通过各种高权限技术来辅助分析. 实际利用分支预测机制时，研究人员再根据攻击所需要的权限要求，重新构建破坏处理器安全边界的攻击模型. 这是通过逆向工程的方法来开发分支预测攻击的通用研究范式.

6.1.2　自动化攻击生成

随着逆向工程的不断深入，分支预测机制的相关发现越来越丰富，但后续的逆向工程也将越发困难. 本文认为，自动化攻击生成技术将成为分支预测攻击的一类新的研究方向. 事实上，近两年内，自动化攻击生成技术已经被用于研究部分微架构攻击，如 Meltdown 类型的乱序执行攻击[131]、端口争用侧信道[130]、Cache 侧信道[84]、TLB 侧信道[85]和瞬态执行窗口的其它潜在的时间侧信道[132]. 该研究方法中，研究者不需要考虑处理器设计的细节，而是从自顶向下的角度，把工作重点放在软件层面，通过遗传算法等启发式算法自动构造测试用例，辅以 PMC 等监测工具来观察测试用例的执行情况，进而判断是否存在潜在的攻击.

这类研究方法的重点包括测试用例的生成、微架构状态的监测和潜在利用对象的分析等. 对于分支预测攻击，如何通过现有攻击链甚至潜在的新攻击链产生包含大量分支指令的有效测试代码、如何通过 PMC 的事件或者其它处理器监测工具反馈测试用例的执行情况以及如何判断测试用例包含潜在的攻击方式，都还没有被充分研究. 本文认为，这将是分支预测攻击发掘的一个有前景的研究方向.

6.1.3　其它研究方向

6.1.1 和 6.1.2 节的研究方法强调发现新的分支预测安全漏洞，即从原理角度挖掘新的分支预测安全问题. 此外，围绕已发现分支预测安全问题进行更加深入的利用，比如拓展攻击场景或结合其它攻击方式提高攻击的成功率，也是开发新的分支预测攻击的常用方法. 这部分研究不具有通用的研究范式，往往是针对攻击的特定场景或特定环节，采用特殊方法进行定向突破. 这些研究方法可以被归为以下四类：

(1) 目标导向的研究方法，即研究针对特定的攻击目标时如何实现分支预测攻击. 比如，Schwarz 等人[94]针对网络攻击，提出了基于数据包响应时间分布的时间测量方法，和基于 AVX2 指令的微架构侧信道来实现 Spectre 攻击. Chen 等人[102]针对 SGX 攻击，提出使用非安全区训练、安全区触发分支然后非安全区进行侧信道检测的攻击方法.

(2) 技术导向的研究方法，即研究如何利用特定处理器的其它机制实现或强化分支预测攻击. 比如，Lee 等人[24]利用 Intel 处理器的 LBR 观察历史分支的跳转情况，从而实现分支预测侧信道；Schwarz 等人[138]利用 Intel 处理器的微码辅助页表访问机制[82]在瞬态窗口进行可重复的数据泄露.

(3) 优化导向的研究方法，即研究如何进一步提高分支预测攻击的可行性与成功率. 该类研究往往采用与其它攻击相结合的手段达成目标. 比如，Bhattacharyya 等人将返回导向编程(Return Oriented Programming，ROP)攻击与 Spectre 攻击结合，提高发现可用代码片段的可能性，进而提高攻击的可行性；Tobah 等人[100]将 Rowhammer 攻击[134]与 Spectre 攻击结合，从而在将 Linux 5.6 版本内核的 Spectre 攻击可用代码片段从 100 个增加到 20000 个左右.

(4) 对抗导向的研究方法，即研究如何绕过现有的软硬件防御策略. 比如，Bhattacharyya 等人[103]、Fustos 等人[96]和 Behnia 等人[95]利用 ALU、除法部件、保留站等微架构部件的争用来构造新的瞬态侧信道，以绕过对 Cache 侧信道的防御；Wampler 等人[105]和 Zhang 等人[34]分别用控制流不可达代码和别名地址分支来填充预测器，以绕过软件分析工具对攻击片段的检查；Barberis 等人[106]利用分支历史索引来绕过 Intel 的 IBPB 对跨进程同名分支的检查.

6.2 防御策略

针对分支预测攻击的防御策略已经非常丰富.由于篇幅限制,本文不具体介绍防御方法的实现原理,只简单讨论现有防御策略的特点.防御方法的整理、归纳和分析将作为本团队的后续工作.

在不改变分支预测机制的前提下,防御策略需要考虑的核心问题是安全性和处理器性能的平衡.本节分别从软件和硬件的角度讨论现有防御策略.

6.2.1 软件防御策略

传统侧信道,如功耗侧信道和电磁侧信道等,通常要求攻击者能够物理接触受害者芯片,从而对其执行过程的部分物理量进行采样和分析.软件开发时,这些物理量是程序员或静态软件分析工具难以调试和控制的,因此对于这类传统侧信道很难通过软件防御,通常需要借助额外的处理器外部硬件单元辅助实现防御措施[135-136].相比于传统侧信道,分支预测侧信道和瞬态执行攻击中使用的其它微架构侧信道只需通过软件采集时间或 PMC 事件等数据即可完成分析,这些信息是软件程序员或代码分析工具能够预期和控制的,比如能够提前完成 Cache 或 PHT 等部件的填充来避免时间差异的产生.因此,对于分支预测侧信道和其它基于时间测量的微架构侧信道,软件防御策略是可以实现的.

从软件角度防御分支预测攻击,只需对处理器微架构做微小的修补甚至保留现有微架构.因此,在新一代处理器微架构被设计之前,软件防御策略是部署最简单、效率最高的防御策略.但是,软件防御策略大都只能防御已知攻击,并且因为硬件的设计细节存在安全缺陷而修改软件程序,也违反了计算机体系结构的分层设计原则,因此不能完全代替硬件防御方案.此外,软件防御策略需要处理器的指令集支持,尤其是序列化指令.序列化指令,如 x86 指令集的 lfence 指令和 ARM 指令集的 isb 指令,强制要求指令顺序执行,而阻止程序瞬态执行.

如果按照软件的生命周期来划分,现有的软件防御策略包括设计安全算法、编译时防御以及程序运行时防御.

(1)设计安全算法,尤其是加密算法,能从根本上防止私密数据通过分支预测器泄露.Acıiçmez[21]和 Bhattacharya[88]等人对此提出了一些建议,如不把私密数据作为分支方向的判断依据.但是,这种方法只针对特定算法,不具有普适性,不适合作为通用防御策略.

(2)编译时防御,是指在程序编译时主动替换一些依赖私密数据的高风险分支指令,并保持语义等价,比如把条件分支转化成条件移动指令(如 CMOV)[137]或位运算[138],把间接分支转化成一系列控制流固定的直接分支[24,123]或者一系列返回指令[116].对于瞬态执行攻击,可以在编译器的后端加入污点传播和符号执行等代码分析工具自动搜索可能的危险代码片段,然后通过插入序列化指令或修改分支指令等方式来阻止这类代码片段的瞬态执行[126-129,139].相比算法修改,这类策略能在保证语义的情况下进行防御.但是,由于程序的复杂性和路径爆炸等问题,这类分析很难做到程序的全面覆盖;反之,如果对所有指令都序列化,则可能带来超过 80% 的额外性能开销[26].

(3)运行时防御,包括运行时监测和添加测量噪声两种方式.运行时监测表示在程序运行时通过实时监测 PMC 的方式分析是否有潜在的恶意攻击者在填充分支预测器或探测微架构状态[122,125,140-141].这种防御策略无需对现有的软硬件实现做任何更改,适合在云服务器等远程环境下部署.但是,虽然实时监测对已知攻击检测的准确率已经能达到 99% 以上[141],但仍然存在误报和漏报的风险,且无法检测出新的攻击.添加测量噪声是另一种可能的防御策略,5.2.1 节对这种防御策略进行了简单介绍.该策略的性能开销几乎可以忽略,且部署简单,但是其安全性无法完全保证,因为仍可能有高精度时间获取方案替代现有的时间测量工具[142].

6.2.2 硬件防御策略

相比软件防御策略,硬件防御策略部署更加困难,大量方法目前仅在模拟器上进行验证,而没有实际应用.但是,如果应用硬件防御策略,那么部署后无需修改程序代码,符合计算机系统的分层设计原则.因此,从长远来看,硬件防御策略对整个系统生态更有利.在不修改分支预测器的前提下,硬件防御策略主要包括在程序局部位置禁用分支预测机制、防止瞬态执行过程中私密数据影响微架构状态以及防止攻击者从微架构恢复数据.

(1)在程序局部位置禁用分支预测器.这是 Intel 和 AMD 等商用处理器实际运用的方法,通过微码更新等方式添加新的安全属性,比如 5.2.1 节所述的 IBRS、STIBP 和 IBPB.IBRS 禁止低特权级的分支指令历史影响高特权级的分支指令结果;STIBP 禁止同步多线程之间分支预测机制的共享;IBPB 通

过清空 BTB，阻止 BTB 的恶意填充被后续分支利用.但是，禁用分支预测机制的策略对处理器性能造成的影响比较严重，某些测试用例下的性能开销甚至高达 440%[35].

(2)防止瞬态执行过程中私密数据影响微架构状态.这是针对瞬态执行攻击的防御方法，且主要针对的是 Cache 侧信道.比如，通过在流水线和 Cache 之前增加额外的缓冲区，防止瞬态执行过程中的访存指令影响 Cache 内容[143-145].这类防御策略重点关注了性能问题，增加的缓冲区在时间和空间上对处理器流水线的影响较小，一般开销不超过 10%[26].但是，这些防御策略大部分仅在模拟器上测试，而未在实际处理器上部署和测试，并且仅仅防御了 Cache 侧信道.尽管 5.3 节的评估结果表明，在分支预测攻击中，Cache 侧信道是可行性最大的微架构侧信道，但分支预测攻击还有很多微架构组件争用类型的侧信道，这类侧信道利用处理器的资源共享，除非增加微架构部件的数量，否则很难在保证功能正确性的前提下阻止私密数据影响这些共享部件的行为.

(3)防止攻击者从微架构恢复数据.与上一策略相同，该防御策略也主要针对瞬态执行攻击中的 Cache 侧信道，主要包括各种安全 Cache 的设计，包括 Cache 的进程隔离[146]、Cache 块索引随机化[147]以及 Cache 参数的动态调整[148]等.类似地，即使 Cache 是安全的，这些策略也只能防御 Cache 侧信道，攻击者仍有可能利用其它微架构的时间侧信道绕过现有防御.

6.2.3　防御策略的研究方向

目前商用处理器上部署的防御策略主要包括 6.2.1 节所述的编译时防御策略和 6.2.2 节所述的禁用策略.这些策略的特点为不需要大幅修改软硬件的实现，因此易于部署.但是，这些防御策略都可能带来较为明显的性能开销.因此，如何平衡安全和性能，将持续作为防御策略研究的最重要内容.

相比上述策略，6.2.2 节介绍的修改流水线的硬件实现的防御策略，是目前较理想的安全与性能折中方案，但在商用处理器上均未部署.如何把这些策略在处理器实际的硬件设计、综合与物理实现中应用起来，也将是防御策略的一个研究点.

此外，6.2.1 节所述的监测和添加噪声的策略，在当前的所有防御策略中具有最佳的性能表现.但是它们无法提供绝对的安全保证.因此，如何提高这些方法的安全性，也是防御策略的一个研究方向.

6.3　安全分支预测器的设计与验证

Hill 等人[149]提出，与不断发现处理器安全漏洞并修补相比，设计安全的处理器才是从根本上解决各种处理器安全问题的方法.对于分支预测器而言，如何平衡安全性和预测精度、预测速度及预测器容量等性能属性，是安全设计的最大难题.

综合比较表 8 整理的各个分支预测攻击原理，别名地址索引和分支历史索引的攻击实际上既影响分支预测器的安全性，又影响分支预测器的性能，因为这些攻击本质上用不同的分支挤占了同样的分支预测器表项.相反，同名地址索引的安全性和性能是冲突的，如果为了安全性而干扰同名地址索引的精度，就会降低分支预测器的性能.因此，现有的分支预测器安全设计策略全都集中在缓解别名地址索引和历史索引造成的安全问题，而把 Spectre V1 等同名地址索引以及 RSB 带来安全问题的防御放在其攻击链的后两个阶段.对于后者，使用 6.2 节讨论的防御策略，才能兼顾安全性和性能需求.

别名地址索引和分支历史索引造成表项碰撞的主要原因有两点，一是表项的有限性，二是同一物理核上分支预测器的共享性.表项的有限性是必然存在的，根据鸽巢原理[150]，当把大于表项个数的分支指令映射到有限表项的分支预测器时，一定存在映射到同一表项的不同分支，即发生表项碰撞.现有安全设计的主要思路是通过索引随机化使表项的碰撞不可逆向或不可控制.此外，在保留物理核共享分支预测器这一特性的同时，通过加密或上下文切换时清空预测器表项等方式，实现进程或线程粒度的逻辑隔离，也是现有设计主要采用的方法.下面讨论现有的 6 个安全分支预测器设计.

Tan 等人[151]针对分支预测攻击 SBPA 提出了一种安全的 BTB 设计，主要思路是给 BTB 表项添加一个锁标记，然后通过硬件监测每个进程的 BTB 占用率，把占用率高于既定阈值的可疑进程标记出来.可疑进程执行时，其余良性进程的 BTB 表项被上锁，该进程不可替换上锁的 BTB 表项.对于有 4096 条目的 BTB，该方案额外造成约 8 KB 存储开销，能在保证安全性的同时，使程序平均执行性能提高 0.12%.但是，该方案只能防御攻击场景④的 BTB 侧信道，防御点单一.

Vougioukas 等人[152]提出在上下文切换时清空预测器，为了缓解性能开销，他们设计了分支保留缓冲区(Branch Retention Buffer，BRB)存储每个进程

存储在分支预测器的关键信息，每次上下文切换后重新把 BRB 的信息写入分支预测器.这种设计方法实现了进程粒度的隔离，但是仍然造成约 21.8%的性能下降[153].

Lee 等人[153]提出的 BSUP 和 Zhao 等人[154]提出的 Noisy-XOR-BP 通过硬件对分支预测器的索引和分支预测器表项内容进行加密.与 BRB 相比，这两种方法造成的额外性能开销都更低，前者约 7.7%，后者约 2.5%.在加密算法方面，BSUP 的加密计算是较复杂的逻辑计算，而 Noisy-XOR-BP 仅使用异或计算，因此计算开销更小.就防御效果而言，BSUP 是针对 ARM 处理器设计的，用于加密的密钥以固定的周期更新，因此无法防御攻击场景④. Noisy-XOR-BP 的密钥是线程私有的，在上下文切换或权限转化时动态更新，但更新时机的确定性和加密方式的简易性有可能受到重放或逆向攻击.

Zhang 等人在提出分支木马攻击后，也提出了一种安全分支预测器：STBPU[35]. STBPU 提出使用软硬件协同的方式解决分支预测安全问题，每个需要隔离的进程或线程维护一个秘密令牌，该令牌决定分支指令如何索引到分支预测器.秘密令牌的产生和修改由可信的软件完成，该可信软件同时要通过性能计数器监测微架构事件，在判断可能受到攻击时更换令牌.更换令牌时机的不确定性增加了攻击者进行重放攻击或逆向工程的难度. STBPU 的造成的额外性能开销约 2%.但是，把 STBPU 把安全交付给软件的行为，使分支预测安全性变得不可控，且可能引出别的安全问题.比如，在不可信内核的环境下，即使令牌可以由 SGX 的用户进程单独维护，但性能计数器的配置和使用无论是交给不可信内核还是 SGX 进程都是有安全风险的，前者可能导致性能计数器不计数，使得令牌不更新，给攻击者逆向的机会；后者破坏了 SGX 不开启性能分析工具的安全假设，因为性能计数可能导致 SGX 进程的执行情况被直接泄露.

相比 STBPU，Chen 等人[155]提出一种把进程标识(Program ID，PID)直接加入分支指令索引过程参与哈希运算的设计 LS-BPU，从而实现分支指令的进程间隔离.该防御方法的平均性能损失小于 3%，最大性能损失为 6.57%.但是，这种进程粒度的防御方法，无法防御攻击场景③和⑦，也没有考虑使用分支历史的索引方式.

总体来看，安全分支预测器设计工作多见于 2019 年之后的学术会议和期刊，仍属于比较新的研究领域.现有设计仍存在性能或安全性的不足，该研究领域仍有很大的发展空间.

为证明设计是安全的，现有工作大都在模拟器或 FPGA 上证明分支预测攻击的无效性，一些基于加密的安全设计也从理论推导了受攻击的可能性.但是，这些方法只能说明设计能够防御已知攻击，而无法从信息安全的角度证明设计是绝对安全的.验证分支预测器设计的安全性，需要借鉴处理器设计中的安全验证方法.现有的处理器微架构安全验证方法包括：(1)从硬件描述语言层面验证信息流安全，比如 SecVerilog[156]和 SecChisel[157]，它们分别是对硬件描述语言 Verilog 和 Chisel 的扩展，在确保功能正确性的同时，对信息流安全进行严格验证；(2)对已设计好的处理器进行安全状态空间的建模和形式化验证. Cabodi 等人[158]对乱序执行处理器特性进行抽象与建模，然后使用模型检测方法验证乱序执行处理器在瞬态执行中的安全性.类似的思路可以用在设计分支预测器上；(3)从处理器设计代码和安全断言中直接挖掘潜在的漏洞. Coppelia[159]是一种从处理器的设计和安全断言出发，借助逆向符号执行，自动检测处理器潜在漏洞的技术框架.对于已开源的分支预测器设计代码，可以开发类似框架对其安全性进行检测.

在分支预测的安全验证方面，Ponce-de-León 等人[160]首次提出使用模型检验方法验证开源处理器设计是否可能受到 Spectre 攻击.除此之外，目前对分支预测器设计进行安全验证的工作还比较少，该领域具有较大的研究空间.

7 总 结

本文对分支预测技术的安全问题进行了系统的梳理.为提高分支预测精度，研究人员提出了日益复杂且高效的分支预测器设计，但却忽略了分支预测机制存在的安全问题.本文从攻击原理、攻击场景和攻击过程三个角度出发，建立了全新的攻击模型.同时，文章按攻击模型归纳整理了学术界主要系统与信息安全会议和期刊的分支预测攻击，并根据攻击链分阶段描述了每个阶段的攻击过程，同时总结了侧信道、隐藏通道和微架构侧信道的抽象利用过程.为了统一评估各分支预测攻击的可行性，文章在统一的理论模型下计算了分支预测类瞬态执行攻击恢

复单位比特数据需要执行的受害者指令数.结果表明,相比其它微架构时间侧信道,基于 Cache 侧信道的 Spectre 攻击仍具有较高的可行性.此外,文章结合 Intel、AMD 和 ARM 等主流商用处理器的典型微结构,从攻击模型深入分析了这些处理器的分支预测器的脆弱性.最后,本文讨论了分支预测安全问题研究的发展方向,包括逆向工程和自动化测试等,同时讨论了现有防御策略以及安全分支预测器的设计,提出用安全分支预测器解决别名地址索引和历史索引造成的安全问题,而用其它防御策略解决同名地址索引的相关安全问题.相信随着处理器设计技术的不断发展,基于安全分支预测器的安全处理器将成为未来信息化社会不可或缺的一部分.

参　考　文　献

[1] Bryant R E, David Richard O H. Computer Systems: A Programmer's Perspective. New Jersey, USA: Prentice Hall Upper Saddle River, 2003

[2] Patterson D A, Hennessy J L. Computer Organization and Design ARM Edition: The Hardware Software Interface. Burlington, USA: Morgan Kaufmann, 2016

[3] Stone H S. High-Performance Computer Architecture. Boston, USA: Addison-Wesley, 1987

[4] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach. Amsterdam, The Netherlands: Elsevier, 2011

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

[6] Palacharla S, Jouppi N P, Smith J E. Complexity-effective superscalar processors//Proceedings of the 24th Annual International Symposium on Computer Architecture. Denver, USA, 1997: 206-218

[7] Smith J E, Pleszkun A R. Implementation of precise interrupts in pipelined processors. ACM SIGARCH Computer Architecture News, 1985, 13(3): 36-44

[8] Smith J E. Decoupled access/execute computer architectures. ACM SIGARCH Computer Architecture News, 1982, 10(3): 112-119

[9] Smith J E. A study of branch prediction strategies//Proceedings of the 25 Years of the International Symposia on Computer Architecture (Selected Papers). Barcelona, Spain, 1998: 202-215

[10] Lee J K, Smith A J. Branch prediction strategies and branch target buffer design. Computer, 1984, 17(1): 6-22

[11] Kroft D. Lockup-free instruction fetch/prefetch cache organization//Proceedings of the 25 Years of the International Symposia on Computer Architecture. Barcelona, Spain, 1998: 195-201

[12] Stone S S, Woley K M, Frank M I. Address-indexed memory disambiguation and store-to-load forwarding//Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05). Barcelona, Spain, 2005: 182

[13] Hooker R E, Eddy C. Store-to-load forwarding based on load/store address computation source information comparisons. USA, 2013.9.10

[14] 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//Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). 2021: 1451-1468

[15] Tullsen D M, Eggers S J, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism//Proceedings of the 22nd Annual International Symposium on Computer Architecture. New York, USA, 1995: 392-403

[16] Ditzel D R, McLellan H R. Branch folding in the CRISP microprocessor: Reducing branch delay to zero//Proceedings of the 14th Annual International Symposium on Computer Architecture. Phoenix, USA, 1987: 2-8

[17] Fog A. The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers. Copenhagen University College of Engineering, 2012, 2

[18] Adiga N, Bonanno J, Collura A, et al. The IBM z15 high frequency mainframe branch predictor industrial product//Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 2020: 27-39

[19] Seznec A. A new case for the TAGE branch predictor//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. Porto Alegre, Brazil, 2011: 117-127

[20] Acıiçmez O, Koc Ç K, Seifert J-P. On the power of simple branch prediction analysis//Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security. Singapore, 2007: 312-320

[21] Acıiçmez O, Koç Ç K, Seifert J-P. Predicting secret keys via branch prediction//Proceedings of the Cryptographers' Track at the RSA Conference. San Francisco, USA, 2007: 225-242

[22] Acıiçmez O, Gueron S, Seifert J-P. New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures//Proceedings of the IMA International Conference on Cryptography and Coding. Cirencester, UK, 2007: 185-203

[23] Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Jump over ASLR: Attacking branch predictors to bypass ASLR//Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Taipei, China, 2016: 1-13

[24] Lee S, Shih M-W, Gera P, et al. Inferring fine-grained control flow inside SGX enclaves with branch shadowing//Proceedings of the 26th USENIX Security Symposium (USENIX Security 17). Vancouver, Canada, 2017: 557-574

[25] Kocher P, Horn J, Fogh A, et al. Spectre attacks: Exploiting speculative execution//Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP). San Francisco, USA, 2019: 1-19

[26] Xiong W, Szefer J. Survey of transient execution attacks and their mitigations. ACM Computing Surveys (CSUR), 2021, 54(3): 1-36

[27] Canella C, Van Bulck J, Schwarz M, et al. A systematic evaluation of transient execution attacks and defenses//Proceedings of the 28th USENIX Security Symposium (USENIX Security 19). Santa Clara, USA, 2019: 249-266

[28] Evtyushkin D, Riley R, Abu-Ghazaleh N C, et al. BranchScope: A new side-channel attack on directional branch predictor. ACM SIGPLAN Notices, 2018, 53(2): 693-707

[29] Guide P. Intel? 64 and IA-32 architectures software developer's manual. Volume 3B: System Programming Guide, Part, 2011, 2(11)

[30] Holdings A. ARM Architecture Reference Manual for ARMv8-A architecture profile. ARM Whitepaper, 2019

[31] Holdings A. ARM Cortex-A53 MPCore Processor Technical Reference Manual. ARM Whitepaper, 2014

[32] Uzelac V, Milenkovic A. Experiment flows and microbenchmarks for reverse engineering of branch predictor structures//Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software. Boston, USA, 2009: 207-217

[33] Milenkovic M, Milenkovic A, Kulick J. Microbenchmarks for determining branch predictor organization. Software: Practice and Experience, 2004, 34(5): 465-487

[34] Zhang T, Koltermann K, Evtyushkin D. Exploring branch predictors for constructing transient execution Trojans//Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland, 2020: 667-682

[35] Zhang T, Lesch T, Koltermann K, et al. STBPU: A reasonably safe branch predictor unit. arXiv preprint arXiv: 2108.02156, 2021

[36] Koruyeh E M, Khasawneh K N, Song C, et al. Spectre returns! speculation attacks using the return stack buffer//Proceedings of the 12th USENIX Workshop on Offensive Technologies (WOOT 18). Baltimore, USA, 2018

[37] Maisuradze G, Rossow C. ret2spec: Speculative execution using return stack buffers//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto, Canada, 2018: 2109-2122

[38] Ball T, Larus J R. Branch prediction for free. ACM SIGPLAN Notices, 1993, 28(6): 300-313

[39] Fisher J A, Freudenberger S M. Predicting conditional branch directions from previous runs of a program. ACM SIGPLAN Notices, 1992, 27(9): 85-95

[40] Young C, Smith M D. Improving the accuracy of static branch prediction using branch correlation. ACM SIGOPS Operating Systems Review, 1994, 28(5): 232-241

[41] McFarling S. Combining branch predictors. Digital Western Research Laboratory: Citeseer, Technical: TN-36, 1993

[42] Yeh T-Y, Patt Y N. Alternative implementations of two-level adaptive branch prediction. ACM SIGARCH Computer Architecture News, 1992, 20(2): 124-134

[43] Pan S-T, So K, Rahmeh J T. Improving the accuracy of dynamic branch prediction using branch correlation//Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems. Boston, USA, 1992: 76-84

[44] Michaud P, Seznec A, Uhlig R. Trading conflict and capacity aliasing in conditional branch predictors//Proceedings of the 24th Annual International Symposium on Computer Architecture. Denver, USA, 1997: 292-303

[45] Lee C-C, Chen I-C, Mudge T N. The bi-mode branch predictor//Proceedings of the 30th Annual International Symposium on Microarchitecture. Carolina, USA, 1997: 4-13

[46] Sprangle E, Chappell R S, Alsup M, et al. The agree predictor: A mechanism for reducing negative branch history interference//Proceedings of the 24th Annual International Symposium on Computer Architecture. Denver, USA, 1997: 284-291

[47] Eden A N, Mudge T. The YAGS branch prediction scheme//Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture. Dallas, USA, 1998: 69-77

[48] Stark J, Evers M, Patt Y N. Variable length path branch prediction//Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, USA, 1998: 170-179

[49] Skadron K, Martonosi M, Clark D W. A taxonomy of branch mispredictions, and alloyed prediction as a robust solution to wrong-history mispredictions//Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques (Cat. No. PR00622). Philadelphia, USA, 2000: 199-206

[50] Jiménez D A, Lin C. Dynamic branch prediction with perceptrons//Proceedings of the HPCA 7th International Symposium on High-Performance Computer Architecture. Monterrey, Mexico, 2001: 197-206

[51] Seznec A. Analysis of the O-GEometric history length branch predictor//Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05). Madison, USA, 2005: 394-405

[52] Michaud P. A PPM-like, tag-based branch predictor. Journal of Instruction Level Parallelism, 2005, 7(1): 1-10

[53] Seznec A. A 256 kbits l-tage branch predictor. Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2), 2007, 9: 1-6

[54] Seznec A. A 64-kbytes ittage indirect branch predictor//Proceedings of the JWAC-2: Championship Branch Prediction. San Jose, USA, 2011

[55] Seznec A. Tage-sc-l branch predictors again//Proceedings of the 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5). Seoul, Korea, 2016

[56] Choi B, Porter L, Tullsen D M. Accurate branch prediction for short threads//Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. Seattle, USA, 2008: 125-134

[57] Evers M, Chang P-Y, Patt Y N. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. ACM SIGARCH Computer Architecture News, 1996, 24(2): 3-11

[58] Tarsa S J, Lin C-K, Keskin G, et al. Improving branch prediction by modeling global history with convolutional neural networks. arXiv preprint arXiv:1906.09889, 2019

[59] Mittal S. A survey of techniques for dynamic branch prediction. Concurrency and Computation: Practice and Experience, 2019, 31(1): e4666

[60] Sherwood T, Calder B. Loop termination prediction//Proceedings of the International Symposium on High Performance Computing. Bangalore, India, 2000: 73-87

[61] Seznec A, San Miguel J, Albericio J. The inner most loop iteration counter: A new dimension in branch history//Proceedings of the 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Waikiki, USA, 2015: 347-357

[62] Albericio J, San Miguel J, Jerger N E, et al. Wormhole: Wisely predicting multidimensional branches//Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, UK, 2014: 509-520

[63] Jiménez D A, Keckler S W, Lin C. The impact of delay on the design of branch predictors//Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture. California, USA, 2000: 67-76

[64] Jiménez D A. Reconsidering complex branch predictors//Proceedings of the 9th International Symposium on High-Performance Computer Architecture. Anaheim, USA, 2003: 43-52

[65] Kampe M, Stenstrom P, Dubois M. The FAB predictor: Using Fourier analysis to predict the outcome of conditional branches//Proceedings of the Eighth International Symposium on High Performance Computer Architecture. Washington, USA, 2002: 223-232

[66] Lai C, Lu S-L, Chen Y, et al. Improving branch prediction accuracy with parallel conservative correctors//Proceedings of the 2nd Conference on Computing Frontiers. Ischia, Italy, 2005: 334-341

[67] Akkary H, Srinivasan S T, Lai K. Recycling waste: Exploiting wrong-path execution to improve branch prediction//Proceedings of the 17th Annual International Conference on Supercomputing. San Francisco, USA, 2003: 12-21

[68] Bray B K, Flynn M J. Strategies for branch target buffers//Proceedings of the 24th Annual International Symposium on Microarchitecture. Albuquerque, USA, 1991: 42-50

[69] Kaeli D R, Emma P G. Branch history table prediction of moving target branches due to subroutine returns. ACM SIGARCH Computer Architecture News, 1991, 19(3): 34-42

[70] Yeh T-Y, Patt Y N. A comprehensive instruction fetch mechanism for a processor supporting speculative execution. ACM SIGMICRO Newsletter, 1992, 23(1-2): 129-139

[71] Chang P-Y, Hao E, Patt Y N. Target prediction for indirect jumps. ACM SIGARCH Computer Architecture News, 1997, 25(2): 274-283

[72] Reinman G, Austin T, Calder B. A scalable front-end architecture for fast instruction delivery. ACM SIGARCH Computer Architecture News, 1999, 27(2): 234-245

[73] Li T, Bhargava R, John L K. Rehashable BTB: An adaptive branch target buffer to improve the target predictability of Java code//Proceedings of the International Conference on High-Performance Computing. Bangalore, India, 2002: 597-608

[74] Chang Y-J. Lazy BTB: Reduce BTB energy consumption using dynamic profiling//Proceedings of the Asia and South Pacific Conference on Design Automation. Yokohama, Japan, 2006: 6

[75] Burcea I, Moshovos A. Phantom-BTB: A virtualized branch target buffer design. ACM SIGPLAN Notices, 2009, 44(3): 313-324

[76] Khoshbakht S, Baniasadi A. Leakage-aware speculative branch target buffer. Journal of Low Power Electronics, 2012, 8(5): 595-603

[77] Kumar R, Grot B, Nagarajan V. Blasting through the front-end bottleneck with shotgun. ACM SIGPLAN Notices, 2018, 53(2): 30-42

[78] Gupta V, Panda B. Micro BTB: A high performance and lightweight last-level branch target buffer for servers. arXiv preprint arXiv:2106.04205, 2021

[79] Mambretti A, Neugschwandtner M, Sorniotti A, et al. Speculator: A tool to analyze speculative execution attacks and mitigations//Proceedings of the 35th Annual Computer Security Applications Conference. San Juan Puerto Rico, USA, 2019: 747-761

[80] Ajorpaz S M, Garza E, Jindal S, et al. Exploring predictive replacement policies for instruction cache and branch target buffer//Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). Los Angeles, USA, 2018: 519-532

[81] Uhlig R, Neiger G, Rodgers D, et al. Intel virtualization technology. Computer, 2005, 38(5): 48-56

[82] Costan V, Devadas S. Intel SGX explained. IACR Cryptology ePrint Archive, 2016, 2016(86): 1-118

[83] Ngabonziza B, Martin D, Bailey A, et al. Trustzone explained: Architectural features and use cases//Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC). Pittsburgh, USA, 2016: 445-451

[84] Deng S, Xiong W, Szefer J. Cache timing side-channel vulnerability checking with computation tree logic//Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy. New York, USA, 2018: 1-8

[85] Deng S, Xiong W, Szefer J. Secure TLBs//Proceedings of the 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). Phoenix, USA, 2019: 346-359

[86] Bhattacharya S, Maurice C, Bhasin S, et al. Template attack on blinded scalar multiplication with asynchronous perf-ioctl calls. IACR Cryptology ePrint Archive, 2017, 2017: 968

[87] Bhattacharya S, Mukhopadhyay D. Fault attack revealing secret keys of exponentiation algorithms from branch prediction misses. IACR Cryptology ePrint Archive, 2014, 2014: 790

[88] Bhattacharya S, Mukhopadhyay D. Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on Intel platforms//Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems. Saint-Malo, France, 2015: 248-266

[89] Bhattacharya S, Maurice C, Bhasin S, et al. Branch prediction attack on blinded scalar multiplication. IEEE Transactions on Computers, 2019, 69(5): 633-648

[90] Huo T, Meng X, Wang W, et al. Bluethunder: A 2-level directional predictor based side-channel attack against SGX. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 321-347

[91] Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Covert channels through branch predictors: A feasibility study//Proceedings of the 4th Workshop on Hardware and Architectural Support for Security and Privacy. Portland, USA, 2015: 1-8

[92] Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Understanding and mitigating covert channels through branch predictors. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(1): 1-23

[93] Hunger C, Kazdagli M, Rawat A, et al. Understanding contention-based channels and using them for defense//Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). Burlingame, USA, 2015: 639-650

[94] Schwarz M, Schwarzl M, Lipp M, et al. NetSpectre: Read arbitrary memory over network//Proceedings of the European Symposium on Research in Computer Security. Luxembourg, 2019: 279-299

[95] Behnia M, Sahu P, Paccagnella R, et al. Speculative interference attacks: Breaking invisible speculation schemes//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 1046-1060

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

[97] Sternberger M. Spectre-NG: An avalanche of attacks//Proceedings of the 4th Wiesbaden Workshop on Advanced Microkernel Operating Systems (WAMOS'18). Wiesbaden, Germany, 2018: 21-26

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

[99] Göktas E, Razavi K, Portokalidis G, et al. Speculative probing: Hacking blind in the spectre era//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 1871-1885

[100] Tobah Y, Kwong A, Kang I, et al. SpecHammer: Combining spectre and Rowhammer for new speculative attacks//Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP). San Francisco, USA, 2022: 1362-1379

[101] Chowdhuryy M H I, Yao F. Leaking secrets through modern branch predictor in the speculative world. IEEE Transactions on Computers, 2022, 71(9): 2059-2072

[102] Chen G, Chen S, Xiao Y, et al. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution//Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P). Stockholm, Sweden, 2019: 142-157

[103] Bhattacharyya A, Sandulescu A, Neugschwandtner M, et al. SMoTherSpectre: Exploiting speculative execution through port contention//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK, 2019: 785-800

[104] Bhattacharyya A, Sánchez A, Koruyeh E M, et al. SpecROP: Speculative exploitation of ROP chains//Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020). San Sebastian, Spain, 2020: 1-16

[105] Wampler J, Martiny I, Wustrow E. ExSpectre: Hiding malware in speculative execution//Proceedings of the Network and Distributed Systems Security (NDSS) Symposium. San Diego, USA, 2019

[106] Barberis E, Frigo P, Muench M, et al. Branch history injection: On the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks//Proceedings of the 31st USENIX Security Symposium (USENIX Security 22). Boston, USA, 2022: 971-988

[107] Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: The case of AES//Proceedings of the Cryptographers' Track at the RSA Conference. San Jose, USA, 2006: 1-20

[108] Peterson J L, Silberschatz A. Operating System Concepts. Boston: Addison-Wesley, 1985

[109] Shacham H, Page M, Pfaff B, et al. On the effectiveness of address-space randomization//Proceedings of the 11th ACM Conference on Computer and Communications Security. Washington, USA, 2004: 298-307

[110] Cook K. Kernel address space layout randomization. Linux Security Summit, 2013

[111] Kong J, Aciicmez O, Seifert J-P, et al. Architecting against software cache-based side-channel attacks. IEEE Transactions on Computers, 2012, 62(7): 1276-1288

[112] Yarom Y, Falkner K. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack//Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14). San Diego, USA, 2014: 719-732

[113] Hao E, Chang P-Y, Patt Y N. The effect of speculatively updating branch history on branch prediction accuracy, revisited//Proceedings of the 27th Annual International Symposium on Microarchitecture. San Jose, USA, 1994: 228-232

[114] Chowdhuryy M H I, Liu H, Yao F. BranchSpec: Information leakage attacks exploiting speculative branch instruction executions//Proceedings of the 2020 IEEE 38th International Conference on Computer Design (ICCD). Hartford, USA, 2020: 529-536

[115] Kirzner O, Morrison A. An analysis of speculative type confusion vulnerabilities in the wild//Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). 2021: 2399-2416

[116] Amit N, Jacobs F, Wei M. JumpSwitches: Restoring the performance of indirect branches in the era of spectre//Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19). Renton, USA, 2019: 285-300

[117] Vattikonda B C, Das S, Shacham H. Eliminating fine grained timers in Xen//Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. New York, USA, 2011: 41-46

[118] Martin R, Demme J, Sethumadhavan S. TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks//Proceedings of the 2012 39th Annual International Symposium on Computer Architecture (ISCA). Portland, USA, 2012: 118-129

[119] Trilla D, Hernandez C, Abella J, et al. Cache side-channel attacks and time-predictability in high-performance critical real-time systems//Proceedings of the 55th Annual Design Automation Conference. San Francisco, USA, 2018: 1-6

[120] Van Bulck J, Piessens F, Strackx R. SGX-Step: A practical attack framework for precise enclave execution control//Proceedings of the 2nd Workshop on System Software for Trusted Execution. Shanghai, China, 2017: 1-6

[121] Schwarz M, Gruss D. How trusted execution environments fuel research on microarchitectural attacks. IEEE Security&Privacy, 2020, 18(5): 18-27

[122] Depoix J, Altmeyer P. Detecting spectre attacks by identifying cache side-channel attacks using machine learning. Advanced Microkernel Operating Systems, 2018, 75

[123] Hosseinzadeh S, Liljestrand H, Leppänen V, et al. Mitigating branch-shadowing attacks on Intel SGX using control flow randomization//Proceedings of the 3rd Workshop on System Software for Trusted Execution. New York, USA, 2018: 42-47

[124] Zhang T, Zhang Y, Lee R B. CloudRadar: A real-time side-channel attack detection system in clouds//Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses. Paris, France, 2016: 118-140

[125] Harris A, Wei S, Sahu P, et al. Cyclone: Detecting contention-based cache information leaks through cyclic interference//Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. Columbus, USA, 2019: 57-72

[126] Oleksenko O, Trach B, Silberstein M, et al. SpecFuzz: Bringing spectre-type vulnerabilities to the surface//Proceedings of the 29th USENIX Security Symposium (USENIX Security 20). Boston, USA, 2020: 1481-1498

[127] Qi Z, Feng Q, Cheng Y, et al. SpecTaint: Speculative taint analysis for discovering spectre gadgets//Proceedings of the Network and Distributed Systems Security (NDSS) Symposium. San Diego, USA, 2021

[128] Yu J, Yan M, Khyzha A, et al. Speculative taint tracking (STT) a comprehensive protection for speculatively accessed data//Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. Columbus, USA, 2019: 954-968

[129] Daniel L-A, Bardin S, Rezk T. Hunting the haunter—efficient relational symbolic execution for spectre with haunted RelSE//Proceedings of the Network and Distributed Systems Security (NDSS) Symposium. 2021

[130] Gras B, Giuffrida C, Kurth M, et al. ABSynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures//Proceedings of the Network and Distributed Systems Security (NDSS) Symposium. San Diego, USA, 2020

[131] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading kernel memory from user space//Proceedings of the 27th USENIX Security Symposium (USENIX Security 18). Baltimore, USA, 2018: 973-990

[132] Weber D, Ibrahim A, Nemati H, et al. Osiris: Automated discovery of microarchitectural side channels. arXiv preprint arXiv:2106.03470, 2021

[133] Schwarz M, Lipp M, Moghimi D, et al. ZombieLoad: Cross-privilege-boundary data sampling//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK, 2019: 753-768

[134] Kim Y, Daly R, Kim J, et al. Flipping bits in memory without accessing them: An experimental study of DRAM

disturbance errors. ACM SIGARCH Computer Architecture News, 2014, 42(3): 361-372

[135] Shan W, Zhang S, Xu J, et al. Machine learning assisted side-channel-attack countermeasure and its application on a 28-nm AES circuit. IEEE Journal of Solid-State Circuits, 2019, 55(3): 794-804

[136] Shan W, Fu X, Xu Z. A secure reconfigurable crypto IC with countermeasures against SPA, DPA, and EMA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(7): 1201-1205

[137] Agosta G, Breveglieri L, Pelosi G, et al. Countermeasures against branch target buffer attacks//Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007). Vienna, Austria, 2007: 75-79

[138] Coppens B, Verbauwhede I, De Bosschere K, et al. Practical mitigations for timing-based side-channel attacks on modern X86 processors//Proceedings of the 2009 30th IEEE Symposium on Security and Privacy. Washington, USA, 2009: 45-60

[139] Duta V, Giuffrida C, Bos H, et al. PIBE: Practical kernel control-flow hardening with profile-guided indirect branch elimination//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 743-757

[140] Gianvecchio S, Wang H. Detecting covert timing channels: An entropy-based approach//Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, USA, 2007: 307-316

[141] He Z, Lee R B. CloudShield: Real-time anomaly detection in the cloud. arXiv preprint arXiv:2108.08977, 2021

[142] Schwarz M, Maurice C, Gruss D, et al. Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript//Proceedings of the International Conference on Financial Cryptography and Data Security. Sliema, Malta, 2017: 247-267

[143] Yan M, Choi J, Skarlatos D, et al. InvisiSpec: Making speculative execution invisible in the cache hierarchy//Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Fukuoka City, Japan, 2018: 428-441

[144] Ainsworth S, Jones T M. MuonTrap: Preventing cross-domain spectre-like attacks by capturing speculative state//Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 2020: 132-144

[145] Kim S, Mahmud F, Huang J, et al. ReViCe: Reusing victim cache to prevent speculative cache leakage//Proceedings of the 2020 IEEE Secure Development (SecDev). Atlanta, USA, 2020: 96-107

[146] Kiriansky V, Lebedev I, Amarasinghe S, et al. DAWG: A defense against cache timing attacks in speculative execution processors//Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Fukuoka City, Japan, 2018: 974-987

[147] Qureshi M K. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping//Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Fukuoka City, Japan, 2018: 775-787

[148] Bandara S, Kinsy M A. Adaptive caches as a defense mechanism against cache side-channel attacks. Journal of Cryptographic Engineering, 2021, 11(3): 239-255

[149] Hill M D. Technical perspective: Why 'correct' computers can leak your information. Communications of the ACM, 2020, 63(7): 92

[150] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms. Cambridge, UK: MIT Press, 2022

[151] Tan Y, Wei J, Guo W. The micro-architectural support countermeasures against the branch prediction analysis attack//Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. Beijing, China, 2014: 276-283

[152] Vougioukas I, Nikoleris N, Sandberg A, et al. BRB: Mitigating branch predictor side-channels//Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). Washington, USA, 2019: 466-477

[153] Lee J, Ishii Y, Sunwoo D. Securing branch predictors with two-level encryption. ACM Transactions on Architecture and Code Optimization (TACO), 2020, 17(3): 1-25

[154] Zhao L, Li P, Hou R, et al. A lightweight isolation mechanism for secure branch predictors. arXiv preprint arXiv: 2005.08183, 2020

[155] Chen C, Shen C, Zhang J. Lightweight and secure branch predictors against spectre attacks//Proceedings of the 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC). 2022: 25-30

[156] Zhang D, Wang Y, Suh G E, et al. A hardware design language for timing-sensitive information-flow security. ACM SIGPLAN Notices, 2015, 50(4): 503-516

[157] Deng S, Gümüşoğlu D, Xiong W, et al. SecChisel framework for security verification of secure processor architectures//Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy. Phoenix, USA, 2019: 1-8

[158] Cabodi G, Camurati P, Finocchiaro F, et al. Model-checking speculation-dependent security properties: Abstracting and reducing processor models for sound and complete verification. Electronics, 2019, 8(9): 1057

[159] Zhang R, Deutschbein C, Huang P, et al. End-to-end automated exploit generation for validating the security of processor designs//Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Fukuoka City, Japan, 2018: 815-827

[160] Ponce-de-León H, Kinder J. Cats vs. Spectre: An axiomatic approach to modeling speculative execution attacks//Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP). San Francisco, USA, 2022: 1415-1428

LIU Chang, Ph. D. candidate. His main research interests include computer architecture and processor security.

YANG Yi, M. S. candidate. His main interests include hardware security and trusted system.

LI Hao-Ru, M. S. candidate. His research interests mainly focus on processor security.

QIU Peng-Fei, Ph. D., associate professor. His research interests mainly focus on processor security.

LYU Yong-Qiang, Ph. D., associate professor. His research interests mainly focus on processor security.

WANG Hai-Xia, Ph. D., associate professor. Her research interests include processor architecture and formal verification of processor security.

JU Da-Peng, Ph. D., associate professor. His research interests include computer architecture.

WANG Dong-Sheng, Ph. D., professor, Ph. D. supervisor. His research interests include computer architecture, high performance computing and system security.

Background

With the adoption of various aggressive optimizations for modern processor performance, the price of security detriment is beginning to tell. For example, the out-of-order processors can dispatch instructions regardless of the dependent data ready or not. There are numerous speculative mechanisms to enhance processor performance, representatively, the branch predictors.

However, the pursuit of extreme performance indicates moderate access control, which increases the risk of information confidentiality undermining. Recently, the basic security assumption that changes of speculative execution states should not be observed by programmers is disproved by researchers. Combining the branch predictor mechanisms and side or covert channels, attackers can transmit the transient data to the observer outside and even break the security boundary. Since the notorious attack, Spectre, coming into surface, more and more speculative attacks are discovered. Researchers are now focusing on exhausting all potential attacks and getting a complete picture. To thwart such vulnerabilities at the least cost of performance, many mitigations are proposed.

In this paper, we provide a comprehensive overview of the research progress in finding new attacks and evaluate the defensive measures. We summarize the branch predictors in the perspective of their designs and classify them into several clusters. Then we systematically generalize the mitigations based on the attack chains extracted from their building blocks. Lastly, we discuss several uncovered corner cases with potential attacks from our classification and prospect the development of offensive and defensive technologies.

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2021YFB3100902 and the National Natural Science Foundation of China under Grant No. 62072263.