分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题　目：投机执行攻击综述

学　　号　M202173674

姓　　名　张艺

专　　业　电子信息

课程指导教师　施展

院（系、所）　计算机学院

2021 年 **12** 月 **24** 日

投机执行攻击综述

张艺 [1)]

[1)](华中科技大学计算机学院，武汉 430074)

摘 要 推测执行漏洞影响大多数现代处理器。他们利用推测执行、乱序（000）执行、硬件预测和缓存——所有加速程序执行的基本功能。它们允许无特权的对手绕过用户内核隔离或用户定义的边界。在推测执行攻击中，推测窗口被引入以允许非法访问秘密的瞬时指令，然后执行一些本质上“发送秘密”的微架构状态变化，以便攻击者可以观察到它。在检测到错误推测时，架构状态的变化会被丢弃，但一些微架构状态的变化不会——因此泄露了秘密。本文通过对投机执行攻击进行综述报告简要介绍多种攻击方式，以及目前行业所拥有的安全防御策略。并选取三篇相关论文进行解读，从不同方面来应对投机执行攻击。

关键词 计算机安全；投机执行攻击；安全防御；

Overview of Speculative Execution Attacks

Zhang Yi[1)]

[1)](HUST SCHOOL of COMPUTER of SCIENCE and TECHNOLOGY, Wuhan 430074)

Abstract Speculative execution vulnerabilities affect most modern processors. They exploit speculative execution, Out-of-Order (OOO) execution, hardware prediction and caching – all essential features for speeding up program execution. They allow an unprivileged adversary to bypass the user-kernel isolation or user-defined boundaries. In a speculative execution attack, a speculation window is induced to allow transient instructions that illegally access a secret, then perform some micro-architectural state changes which essentially “send out the secret” so that it can be observed by the attacker. Upon detecting mis-speculation, architectural state changes are discarded, but some micro-architectural state changes are not thus leaking the secret. This article provides an overview of opportunistic execution attacks and provides a brief overview of the various types of attacks and the security strategies currently available in the industry

Key words Speculative execution; computer security; Defense strategy;

1 引言

预测执行（Speculative Execution）技术，它和 multiple branch prediction（多分支预测）、 data flow analysis（数据流分析）三项技术，一起构成了 out-of-order execution （乱序执行，OOOE）的技术基石。它是现代高性能计算的基础技术之一，广泛被应用在高端 ARM CPU（包括各种定制 ARM 芯片：高通、Apple、etc）, IBM 的 Power 系列 CPU，SPARC 和 X86 CPU 中。

之前，总是假定如果预测错误（分支被丢弃），预测执行的结果是不可见的，这是因为精心的 CPU 设计会在预测结果和体系结构状态更新之间保持严格的区分。然而，最近的研究揭示了违反这种分离的侧通道。研究者已经表明可以通过多种方式从预测执行中泄露结果。Spectre 和 Meltdown 两种实际的攻击方法就是这一安全问题典型代表！

乱序执行和预测执行在遇到异常或发现分支预测错误时，CPU 会丢弃之前执行的结果，将 CPU 的状态恢复到乱序执行或预测执行前的正确状态，然后选择对应正确的指令继续执行。这种异常处理机制保证了程序能够正确的执行，但是问题在于，CPU 恢复状态时并不会恢复 CPU 缓存的内容，而这两组漏洞正是利用了这一设计上的缺陷进行侧信道攻击。

NDSS 2019 安全会议上提出的研究中，UCB 学者们表明，预测执行可以用于数据窃取以外的其他事件，这表明预测执行线程可以作为隐藏恶意命令的秘密场所。

密歇根大学的计算机科学家，旁路攻击的主要研究者 Daniel Genkin 说：“通常，在设计算法时，我们会考虑输入和输出。我们不会考虑程序运行时会发生的其他事情。” 。“但是计算机不是在纸上运行的，它们是在物理上运行的。当您从纸张转向物理时，计算具有各种物理效果：时间，功率，声音。旁通道利用这些效果之一来获得更多信息并收集算法中的秘密。”

对于一个足够聪明的黑客来说，几乎可以收集任何偶然的信息泄漏来学习他们不应该知道的东西。随着时间的推移，计算变得越来越复杂，组件被推到了物理极限，并在各个方向抛出了意外的信息，侧信道攻击变得越来越丰富且难以预防。过去两年来，英特尔和 AMD 一直在努力修补一系列 bug，例如 Meltdown，Spectre，Fallout，RIDL 或 Zombieload，这些 bug 全都使用了旁道攻击作为其秘密窃取技术的一部分。

本次主要调研了三篇文献。第一篇《Speculator: A Tool to Analyze Speculative Execution Attacks and Mitigation》在 35th Annual Computer Security Applications Conference (ACSA)由 Mambretti, A 等人介绍了 SPECULATOR，这是一种用于研究这些新的微架构攻击及其缓解措施的新工具，旨在成为推测执行的 GDB。使用推测执行标记，发现在 CPU 推测期间可以通过性能计数器观察到的一组指令，SPECULATOR 可以研究单个代码片段的微架构行为，或更复杂的攻击者和受害者场景（例如分支目标注入 (BTI) 攻击）。还在多个 CPU 平台上展示了他们的发现，展示了 SPECULATOR 及其模板提供的精度和灵活性。。

第二篇《Defeating Speculative-Execution Attacks on SGX with HYPERRACE》在 2019 IEEE CONFERENCE ON DEPENDABLE AND SECURE COMPUTING (DSC)由 Chen, Guoxing 等[错误!未找到引用源。]提出了

使用 HYPERRACE 击败 SGX 上的推测执行攻击。提出了一种基于软件的解决方案来解决推测执行攻击，即使在强烈假设 enclave 内存的机密性受到损害的情况下也是如此。他们的解决方案扩展了一项名为 HyperRace 的现有工作，这是一种编译器辅助工具，用于检测针对 SGX enclave 的基于超线程的侧信道攻击，以阻止来自 SGX enclave 的推测执行攻击。

第三篇《NDA: Preventing Speculative Execution Attacks at Their Source》来自于 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) 由 Weisse, O 等[错误!未找到引用源。]提出了不是一一对抗漏洞利用技术和隐蔽渠道，而是寻求从源头上关闭推测执行攻击。他们的主要观察结果是，这些攻击需要一系列相关的错误路径指令来访问和传输秘密数据。他们提出了 NDA，这是一种限制推测数据传播的技术。 NDA 打破了攻击的错误路径依赖链，同时仍然允许推测和动态调度。描述了 NDA 变体的设计空间，这些变体的不同之处在于它们对动态调度的约束以及它们防止的推测执行攻击的类别。 NDA 保留了乱序执行

的大部分性能优势：在 SPEC CPU 2017 上，NDA 变体缩小了有序和无约束（不安全）乱序执行之间的性能差距的 68-96%。

接下来，本文将系统的介绍一些与投机执行相关的一些理论，以及一些经典的投机执行方法，最后将详细的介绍这三篇文章的贡献点及不足。

2 理论

这一章将介绍与投机执行以及与攻击执行有关的攻击方法。

2.1 投机执行

尽管英特尔等供应商一直在努力通过 SGX（软件防护扩展）等缓解措施在其处理器设计中实施新的安全机制，但似乎几乎没有采取措施实施永久性硬件缓解措施以完全消除所有这些漏洞。这确实质疑是否有可能在不完全重新设计推测执行架构的情况下完全缓解漏洞。研究人员论文中显示的漏洞利用技术似乎也绕过了英特尔实施的缓解措施，Google 也支持像 Spectre 这样的推测执行攻击将存在很长时间的观点

Exploit name	V.	CVE Entry	Known as	NVD Release date.
Spectre	1	CVE-2017-5753	bounds check bypass	April 2018
Spectre	2	CVE-2017-5715	branch target injection	April 2018
Branchscope (variant 2 Spectre)	n/a	CVE-2018-9056	branch target injection(pattern history table PHT)	March 2018
Meltdown	3	CVE-2017-5754	rogue data cache load	April 2018
Spectre	3a	CVE-2018-3640	Rogue System Register Read(RSRE)	May 2018 (under review)
Spectre	4	CVE-2018-3639	*Speculative Store Bypass (SSB)*	May 2018
Foreshadow	n/a	CVE-2018-3615	*L1 Terminal Fault SGX*	Aug 2018
Foreshadow	n/a	CVE-2018-3620	*L1 Terminal Fault OS/SSM*	Aug 2018
Foreshadow	n/a	CVE-2018-3646	*L1 Terminal Fault VMM*	Aug 2018
SgxPectre	n/a	CVE Pending	*Leaking SGX Enclave Secrets.*	Pending

图 1 投机执行 CVE

最近 Spectre 和 Meltdown 漏洞利用的变种正在演变（见表 1）。变体 1、2 和 3 的初始记录 CVE 条目之间的时间尺度反映了在 NIST（美国国家标准与技术研究院）通过国家漏洞数据库 (NVD) “正式”发布之前对这些漏洞的冗长分析。在减轻这些攻击方面，实施范围从包含 KAISER 和 LFENCE 等程序的软件补丁到更复杂的微代码更新，例如减轻崩溃和 Spectre v.2 攻击的程序。英特尔第 8 代处理器提供了硬件机制，可以完全缓解 Spectre 变体 2-3a 和 Meltdown（变体 3）。但是，Spectre 变体 1 仍然是一个漏洞。这是因为只有完全重新设计推测执行才能完全缓解漏洞。这对优化与安全之间的芯片设计权衡提出了质疑。

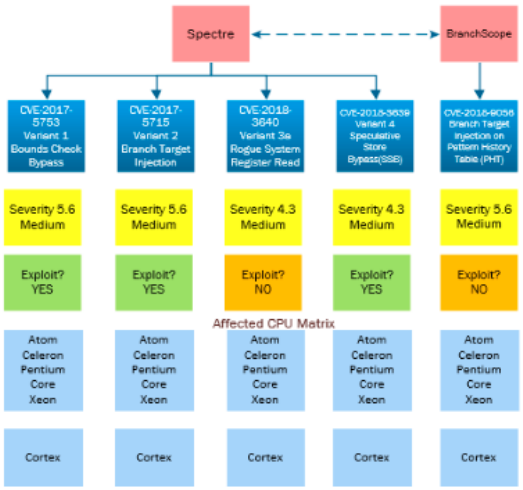

图 2 常见的投机执行攻击

2.2 与投机执行相关的计算机技术

2.2.1 分支预测

推测执行方法的一部分称为分支预测。当程序包含条件（例如“if”语句）时，处理器将预测要使用的条件指令，以推测性地执行先前存储在内存中的指令。执行 BP 的组件是 BPU - 分支预测单元，它具有进一步的子组件，例如分支目标缓冲区 (BTB) 和返回堆栈缓冲区 (RSB)。

2.2.2 乱序执行

除了推测执行和分支预测之外，微处理器还乱序执行指令。这种从线性方式执行指令的转变也提高了 CPU 的性能，因为指令不会“等待”前一条指令退出。

2.2.3 隐式缓存

隐式缓存可以描述为“将没有真正被要求的东西放入缓存中”。例如，由于错误预测而保存在缓存中的数据将成为易受攻击的数据。

2.3 推测执行的攻击方法

2.3.1 缓存时序分析

为了检索敏感或秘密信息，攻击程序需要使用一种技术来从错误预测中识别缓存中保存的秘密数据。为此，需要测量执行特定指令所花费的时间。由于缓存数据的检索速度比未缓存数据（保存在物理内存中）快得多，因此存在可识别和可测量的时间差。这个时间差可以区分缓存或物理内存中保存的数据。以前使用的攻击类型包括 Prime-Probe 、Flush-Reload [和 Evict & Reload 技术。

2.3.2 用户空间到内核

就像内核内存被“映射”到虚拟内存页面，有一个漏洞可以通过破坏虚拟内存的权限级别被恶意程序利用。在这种攻击场景中，从用户空间运行的程序可以打破用户空间和内核空间之间的安全边界，以获取内核内存数据。该漏洞存在于“机会窗口”中，当用户内存读取到特权内存空间导致异常时可利用该漏洞。当抛出异常时，攻击程序可以捕获异常并通过处理“窗口”从用户空间缓存的虚拟内存中提取数据，然后再将其恢复。这是 Meltdown 工作原理的前提。

2.3.3 记忆阴影

Meltdown 的防御性缓解措施，例如英特尔的软件保护扩展 (SGX)，涉及实施进一步的防御机制，以限制对只有特权级别用户（root）才能访问的部分内存的访问。但是，在 Foreshadow 攻击的情况下，程序可以将受保护的内存复制或“隐藏”到 L1 缓存等内存位置，其中未实现 SGX 并且非特权程序可以利用。这是 Foreshadow 工作原理的前提。

2.3.4 处理器培训

以有界数组为例，处理器训练本质上涉及通过通过条件 for 循环重复传递数组索引的合法值来欺骗 CPU。经过一些迭代后，CPU 将期望一个合法值，但攻击程序将传递一个越界的索引值。

2.3.5 隐秘通道

隐蔽通道是将数据发送回对手的攻击程序。这些攻击的初始阶段涉及通过内存和/或缓存操作设置隐蔽通道。要执行的瞬态指令是隐蔽通道设置的一部分。

2.3.6 乱序瞬时执行

现代处理器远离传统的线性获取、解码、执行、退役管道。例如，存储在缓存中并先前用于条件的指令可以推测性地执行，而不是按任何特定顺序执行。推测中使用的指令被称为“瞬态”指令，容易被恶意程序滥用。

2.3.7 传输微架构

微架构层的推测执行和分支预测工作。因此，任何旁道攻击都需要将微架构状态转移到更高级别的状态——完成数据检索的架构状态。

2.3.8 内存操作

Spectre (CVE-2018-3638) 的变体 4 被称为 Speculative Store Bypass，这是攻击如何操纵内存的一个例子。在这种情况下，内存加载的推测执行可能会被错误预测，从而导致可能通过侧信道泄露数据。

3 论文分析

3.1 Defeating Speculative-Execution Attacks on SGX with HYPERRACE

3.1.1 研究背景

英特尔处理器中引入了英特尔软件防护扩展 (SGX)，旨在保护安全区域内的数据和代码免受其不受信任的主机操作系统 (OS) 甚至流氓系统管理员的侵害。由于其对屏蔽执行的承诺，研究人员和从业人员都构建了具有这些功能的各种软件工具和应用程序。最近披露的推测执行攻击，即 Meltdown 和 Spectre 攻击使恶意程序能够读取其安全域之外的内存内容（例如，从用户空间读取内核数据）。他们的变种 SgxPectr 和 Foreshadow，专门针对 Intel SGX 读取 enclave 内存内容，完全破坏了 SGX 的机密性保证。新披露的微架构数据采样 (MDS) 硬件漏洞也可能使对手能够即时读取飞地内存。尽管已经发布了微码补丁来缓解这些攻击，但修复并没有消除漏洞的根本原因——超出安全边界检查和合法控制流的推测执行指令。因此，未来可能会发现此类攻击的新变种。在本文中，作者旨在解决针对英特尔 SGX 的推测执行攻击。现有的解决方案通常建议防止可疑内存负载的推测执行或关闭缓存侧通道，但作者的解决方案是基于软件的。它（1）请求不受信任的操作系统（OS）为飞地创建一个特殊的执行条件，在该条件下，推测执行攻击是不可能的； (2) 在运行时动态验证此类执行条件是否满足，并通过远程证明提供证明；(3) 利用可在微代码中实现的扩展 SGX 功能来保护证明密钥免受内存泄漏

3.1.2 设计实验

作者假设处理器制造商英特尔是可信的，并且处理器的内部存储是安全的。关于对手执行推测执行攻击的能力，假设必须满足以下条件之一：

一、通过并发执行攻击：当启用超线程时，恶意代码可以与受害者飞地在同一物理内核上并发执行。

二、 通过 AEX 攻击：恶意操作系统需要通过 AEX 直接或间接产生中断 1 或触发异常来明确抢占飞地的执行。

三、 EEXIT 后的攻击：即使飞地代码未运行，也可能发起一些攻击。例如，Foreshadow 可以通过调用 EWB 和 ELD 将数据加载到 L1 缓存中，然后在不触发 enclave 代码的情况下执行攻击。

作者的设计分为以下三个方面。第一个是检测一个飞地内的推测执行攻击。第二个是增强现有的证明循环，以向远程客户端证明证明飞地没有此类攻击。三是保护可信平台服务，防止通过重放进行推测执行攻击。

1、 攻击检测：

飞地程序通常提供一个或多个 ECall 功能来完成敏感工作。每项工作都可以通过 对这些 ECall 函数的一次或多次调用来完成。攻击者可能会在一次调用期间或两次调用之间发起推测执行攻击。作者首先描述如何检测一次调用中的推测执行攻击，然后介绍通过多次调用保护秘密的机制。HYPERRACE 自然是针对假设 的。但它解决 假设 2 的能力有限，因为 HYPERRACE 的检测策略必须允许少量块免费中断，以容忍操作系统的正常调度活动。为了解决这些问题，作者通过提出防御要求在操作系统的帮助下，飞地执行期间应消除所有中断。基本上，操作系统必须提供额外的支持，包括但不限于禁用定时器中断和提前分配安全区内存以避免潜在的页面错误。来实现满足假设 2。对于假设 3，提出如下算法来保证实施：

Algorithm 2: Sealing-protected intervals

Input: Inputs (might include sealed secrets)
Output: Outputs
if *sealed secrets is in Inputs* **then**
  secrets ← Unseal(sealed secrets);
  Clear the seal key;
else
  secrets ← Gen_Secrets();
Outputs ← ECall_Func(Inputs, [secrets]);
Derive a seal key with a new random `KeyID`;
Seal secrets with the seal key;
Clear the seal key;
return Outputs

图 3 算法 Sealing-protexted intervals

2、 认证增强

为了实现认证增强就要实现两个认证：本地认证和远程认证。与此同时，提出了两个防御要求：（1）永远不应在飞地内存中公开报告密钥。相反，为了在不暴露报告密钥的情况下验证报告，我们提出了一个名为 EVERIFY 的新 SGX 叶指令。它将报告和相应的报告签名作为输入，导出报告密钥，验证报告签名，并仅输出验证结果。（2）远程证明密钥不应在飞地内存中公开。我们建议扩展 EREPORT 的功能，以允许仅使用根供应密钥（没有根密封密钥）来派生特殊报告密钥，以签署证明数据。

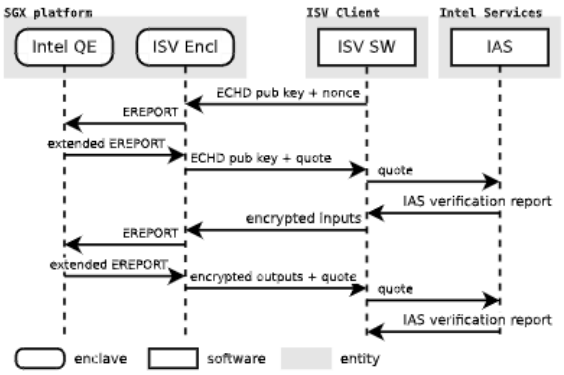

图 4 Enclave 程序工作流

3、 回放预防

为了应对可信平台服务本身就容易受到攻击提出了以下安全策略：在执行初始密封的一次调用中，飞地首先增加单调计数器值。然后，推导出具有随机生成的 KeyID 的新密封密钥来密封秘密。密封后，密封键被清除。如果未检测到攻击，则生成要附加到密封数据的

密封数据报告。 • 在执行开封和重新密封的一次调用中，enclave 也先增加单调计数器值，然后验证所附报告并检查密封计数器值是否比当前单调计数器值小 1（由于增量开头）。如果所有检查都通过，它会解封秘密，处理秘密并使用新的密封密钥重新密封秘密。如果最终没有检测到攻击，它会生成一个报告并将其附加到密封数据中。

3.1.3 总结

作者所提出的安全策略，可以解决以下安全问题：

英特尔的秘密：英特尔的秘密包括远程证明密钥和英特尔发布的飞地的报告密钥，用于保护 ISV 飞地和远程客户端之间的证明循环。当对手能够获得这样的秘密时，她可以伪装成任何有效的飞地。ISV 飞地秘密：ISV 飞地秘密包括随机生成的秘密密钥、密封密钥、报告密钥和由 ISV 飞地处理的远程用户的秘密。

为了实现假设一要求操作系统消除在 enclave 模式下运行的逻辑核心的中断，作者对内核进行了两项修改：首先，在 Linux 内核配置中设置调度程序选项 isolcpus 以隔离可能运行 enclave 代码的逻辑核心内核调度程序，以便操作系统不会在运行飞地代码以触发中断的逻辑内核上调度其他任务。其次，为了消除本地定时器中断，实现了一个内核模块来提供用于配置本地高级可编程中断控制器 (APIC) 的 API。具体来说，根据英特尔手册 ，使用伪中断向量寄存器中的 APIC 软件启用/禁用标志禁用和启用 APIC。因此，在进入 enclave 的临界区之前，将通过内核模块禁用本地 APIC。离开 enclave 的临界区后，将再次启用本地 APIC。实现了 HYPERRA 中引入的 co-location 测试和 AEX 检测机制作为共享库，为 enclave 开发人员提供 API。

3.2 Speculator: A Tool to Analyze Speculative Execution Attacks and Mitigations

3.2.1 研究背景

推测执行攻击利用 CPU 微架构级别的漏洞，直到最近，这些漏洞仍然隐藏在指令集架构之下，CPU 供应商基本上没有记录。每月都会发布新的推测执行攻击，展示如何利用迄今为止尚未探索的微架构攻击面的各个方面。在本文中，作者介绍了 Speculator，这是一种新工具，用于研究这些新的微架构攻击及其缓解措施，旨在成为推测执行的 GDB。使用推测执行标记，发现在 CPU 推测期间可以通过性能计数器观察到的一组指令，Speculator 可以研究单个代码片段的微架构行为，或更复杂的攻击者和受害者场景（例如分支目标注入 (BTI) 攻击）。还在多个 CPU 平台上展示了作者的发现，展示了 Speculator 及其模板提供的精度和灵活性。

3.2.2 设计实验

文章的主要工作:

1. 提出了一种基于性能计数器的方法和工具，speculator，用于分析攻击和缓解策略

2. 利用 speculator 验证了 RAS 的大小，嵌套的推测执行的有效性，推测执行不会跨越系统调用，clflush 在推测执行过程中无效等。同时也测量了对于间接分支，间接控制流转移和 store-to-load 前递导致的推测窗口的大小。最后测试了页面权限，内存保护扩展和特殊指令（例如 lfence）对推测执行的影响.

3. 给出了一些利用 speculator 分析的攻击和缓解示例.

Speculator 的目的

准确测量代码片段中推测部分相关的微体系结构状态属性。精度是指工具能够将代码片段所引起的微体系结构状态变化和工具本身以及系统其余部分隔离开的程度（外界的影响）。可观察的信息：

代码中那些部分是被推测执行的

引发推测执行和结束推测执行的原因

影响推测执行指令数量/指令窗口的参数

特定的指令如何影响推测执行的行为

那些安全边界可以有效的阻止推测执行

CPU 在同一个体系结构内以及跨体系结构和供应商的行为如何保持一致

还可以为代码片段的生成和操作（manipulation）提供一个工具。Speculator 可以在推测执行期间检查单个代码片段或者代码片段组，从而使得用户只需要专注于特定用例相关的指令组合。

Speculator 的设计和实现：

(1)预处理单元：将提供的输入编译成合适的执行格式，从而引入性能监视器接口所需要的代码/工具，以便能够观察所选的计数器的值。

输入：一段 C 或者汇编代码

处理：将预先编译的 json 文件中的指令添加到

输入代码的特定位置（在 source template 中定义的位置中）

(2)Runtime 单元：主要由 speculator monitor 负责初始化和读取生成代码中使用到的计数器

生成的代码片段和 monitor 会放在不同的核上执行，以减少相互之间的干扰

当 Monitor 建立好环境后，就会加载并在另一个进程中执行 snippet，直到结束

代码片段的开头和结尾会对计数器进行 reset，start 和 stop 的操作.

(3)后期处理单元: 当代码片段执行完成后，OS 通知 monitor。之后 Monitor 将获取这些计数器的值，将结果写入文件中. 在多次运行之后，统计收集所有结果，并且去除其中的可疑点。

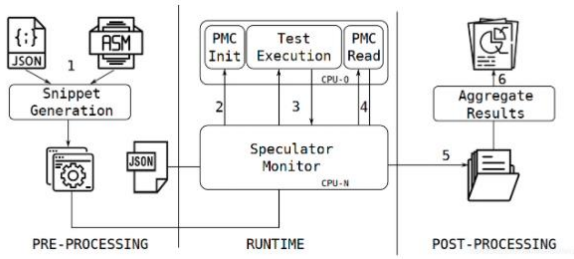

图 5 工作流程

为了自动生成测试样例，speculator 为用户提供了一系列的模板，能够用于重现各种触发推测执行的方式，包括 BTI（branch target injection），spectre 等 。

利用 Speculator 分析微体系结构状态的变化：

（1）Return Stack Buffer Size ：RSB 的大小通常已知，因此可以用于衡量 speculator 的准确性

设计：snippet 代码片段包含在 call 和 ret 之间，在代码中调用 filler 函数，该函数会增加嵌套调用的层数。每一层 call 都会在 RSB 中增加一个表项，如果表项数使用完了，最外层的 call 指令将找不到对应的 ret 指令，导致 CPU 无法执行实现设置的标记

在 Intel 的 Kabylake 处理器中的测试结果

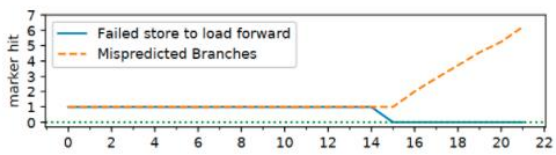

图 6 测试结果

（2） 嵌套推测执行

实验：三个嵌套的条件分支。最外层最为复杂，包括源数据不在缓存中，同时需要进行除法操作，因此解决时间最长。中间的分支操作很简单，但是数据不在缓存中。最里层的分支数据是被缓存的，最容易解决。BP 的状态会实现被训练好，三个分支都会被预测错误

结论：内层推测执行会发生。如果内层分支在上一层分支解决之前被解决了，会在另一个方向进行推测执行。如果最外层分支被解决了，所有的指令都会被取消。

（3）跨越系统调用推测执行

实验：推测执行的代码片段中发射一条系统调用指令（sys_getppid）。使用计数器记录执行的微操作的数量，并且将其调整为用户模式和内核模式分开的执行指令数量

结论：在系统调用之前的推测执行的指令数量不会增加，尽管增加了系统调用指令之后的指令数量。在内核模式下的执行的指令数量和未进行推测执行的情况下一致。因此系统调用指令可以有效的停止推测执行，当系统调用从内核模式返回时。

（4）Flushing the Cache

实验：在代码片段中，首先将值从 cache 中刷新出去，然后再访问该值（lfence 保证 clflush 的操作都可以正常执行，并且停止推测执行）。在两次执行中，分别在 setup 阶段保证该值在 cache 中和不在 cache 中。通过测量执行时间来反映结果

结论：在推测执行过程中的 clflush 不会影响 cache，直到该指令提交了，因此攻击者无法使用 clflush 扩展推测执行的窗口。

（5）内存保护拓展

MPX 的功能：efficiently keeping track of bounds information associated with pointers and corresponding spatial memory checks before dereferencing pointers 。如果边界检查失败，会触发一个#BR 异常，CPU 自陷入 kernel。

实验：测量在边界检查分支之后有多少代码会被推测执行

结果：MPX 的 FNOP 指令在增加到 22 个之后，执行的指令数量不在发生变化

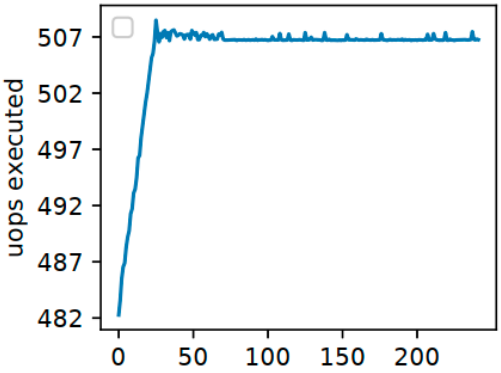

图 7 MPX 实验结果

（6）推测执行的指令窗口的大小：

在分支条件取决于不同类型的指令的情况下，条件分支引发推测执行的指令窗口的大小

Conditional branch	Broadwell	Skylake	Zen
Register access	14	16	7
Access to cached memory	19	17	9
Access to uncached memory	144	280	321
Mul with register	19	19	2
Mul with cached memory	33	33	8
Mul with uncached memory	154	290	362
Div with register	35	41	17
Div with cached memory	34	39	30
Div with uncached memory	164	306	353

图 8 窗口大小

Indirect branch target location	Broadwell	Skylake	Zen
Register	28	22	24
Cached memory	41	34	35
Uncached memory	154	303	301

图 9 间接控制转移指令

store-to-load 前递：在 Broadwell 处理器中，推测窗口大小平均为 55 周期

（7）可执行页权限

问题：在推测执行过程中，是否可以推测执行不具有执行权限的页内的指令

实现：在推测执行过程中（branch mis-prediction），增加控制流转移指令跳转到 non-executable 的内存区域。在实验中，首先保证该内存区域中要被使用的数据会在 L2 cache 中，同时地址也在 TLB 中

结论：研究的所有架构中，页的可执行权限检查都会被满足，即如果该页面是不可执行的，则该页面中的指令也不会被推测执行。

3.2.3 总结

在本文中，阐明了与安全相关的推测执行和微架构行为。展示了 Speculator，这是一种新颖的工具，可以有针对性地精确测量微体系结构特征。使用 Speculator，我们然后调查推测执行。研究了各种推测执行触发器的推测窗口等方面，这是推测执行攻击有效载荷的重要因素。还展示了哪些事件会停止推测执行，以及一些安全控制（如 NX）在推测执行期间仍然有效，而其他安全控制则不会作为障碍，如英特尔的 MPX 边界检查。还展示了使用 Speculator 编写 PoC 攻击和测试缓解措施。编写 Speculator-ready 测试，大大提高了 PoC 和通用测试的可移植性。预计 Speculator 将用于共享可重现的测试，这些测试可用于以更精确和可靠的方式验证系统保护的状态，以及进一步对 CPU 未记录的功能进行逆向工程。

3.3 NDA: Preventing Speculative Execution Attacks at Their Source

3.3.1 研究背景

像 Meltdown 和 Spectre 这样的推测执行攻击通过访问错误路径执行中的秘密数据来工作。然后攻击者通过隐蔽通道传输和恢复秘密。现有的缓解措施要么需要修改代码、仅解决特定的漏洞利用技术，要么仅阻止缓存隐蔽通道。为了不是一一对抗漏洞利用技术和隐蔽渠道，而是寻求从源头上关闭推测执行攻击。作者的主要观察结果是，这些攻击需要一系列相关的错误路径指令来访问和传输秘密数据。提出了 NDA，这是一种限制推测数据传播的技术。 NDA 打破了攻击的错误路径依赖链，同时仍然允许推测和动态调度。描述了 NDA 变体的设计空间，这些变体的不同之处在于它们对动态调度的约束以及它们防止的推测执行攻击的类别。 NDA 保留了乱序执行的大部分性能优势：在 SPEC CPU 2017 上，NDA 变体缩小了有序和无约束（不安全）乱序执行之间的性能差距的 68-96%

首先，根据所有已知攻击中存在的三个基本攻击阶段对推测执行攻击进行分类。描述了现有的缓解技术、它们如何阻止攻击以及它们的缺点。最后，证明关闭特定的侧信道是不够的。

3.3.2 设计实验

NDA 的设计目标：减轻控制转向和选择代码攻

击，同时尽可能多地获得 OOO 推测执行的好处。通过讨论 NDA 的不同变体，它们根据威胁模型为推测性数据传播提供不同的策略。不同的 NDA 数据传播策略提供不同的安全保证并具有相应的性能影响。基于基线物理寄存器的 OOO 微架构上构建 NDA 。

NDA 设计背后的关键见解是，只要输入是安全指令的结果，推测性指令（无论是在正确的还是错误的路径中）都可以安全地执行而不会泄露机密。将指令定义为相对于我们的威胁模型是安全的，这样错误路径执行不会比正确路径指令泄漏更多信息到侧信道。因此，通过消除了推测性侧信道攻击和非推测性侧信道之间的差距，这是具有安全意识的程序员已经必须推理的。

（1） Strict Data Propagation

NDA 通过将未解析的分支和未解析的存储（预测可能不正确）定义为安全和不安全推测之间的边界来解决控制转向攻击。当一个分支微操作进入 ROB 时，它是未解决的。由于获取单元预测在分支之后获取哪些指令（通过 BTB、RSB 等），因此随后分派的微操作可能是错误的路径。类似地，当一个 store micro-op 进入 ROB 时，它是无法解析的，直到它的地址被计算出来。如果一个存储的地址没有被计算出来，如果它们的地址重叠，则跟随该存储的加载可能会错误地访问陈旧数据。考虑关于控制转向攻击的数据传播限制的两种变体：严格和宽容。两种变体都利用绕过限制机制来击败 SSB 攻击。

严格定义：NDA 将所有在未解决的分支机构或商店之后分派的微操作标记为不安全。不安全的指令可能会像基线 OoO 一样被唤醒并竞争发出（即，当它们的操作数准备好时，它们可能会发出）。但是，当一个不安全的微操作完成执行时，它会写回其目标物理寄存器，但不会将其目标标签广播给相关指令，不会将其目标寄存器标记为就绪，并且不会在旁路网络上转发其输出值。因此，相关指令不会发出并且无法观察到不安全值。当最年长的未解决微操作解析时，它会在 ROB 保险箱中标记指令，直到下一个最年长的未解析分支/存储。 ROB 条目扩展了三个位：不安全跟踪，如果指令跟随一个仍未解析的微操作，exec 跟踪如果指令已执行，以及 bcast 跟踪，如果指令已将其标记广播到唤醒依赖项。指令完成后，如果不安全，则延迟标签广播。当微操作解析时，后续 ROB 条目的不安全位将被清除，直到下一个未解析的分支/存储。 !unsafe && exec && !bcast 指令仲裁标签广播端口，与当前周期内完成的指令竞争（完成指令优先避免流水线停顿）；广播时设置 bcast。当安全指令将它们的标签广播到问题队列时，它们将它们的目标寄存器标记为就绪，唤醒它们的依赖项。没有在基线 OoO 上向 ROB 添加额外的标签广播端口；广播的数量不变，广播是时移的，直到前面的微操作解决。例如，假设广播带宽为 4，并且有两条指令完成了这个周期。如果另外三个指令被标记为安全的，那么这些新安全的指令中的两个可以唤醒依赖项；第三个等待下一个周期。

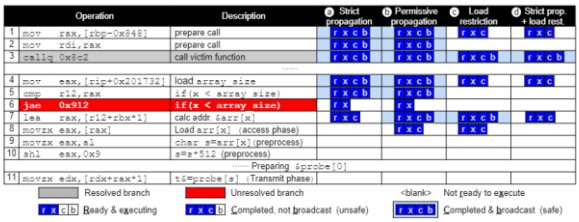

图 10 NDA 数据传播政策

（2） Permissive Data Propagation

非严格定义：对于 NDA 仅保护内存或特殊寄存器中的机密的威胁模型，我们可以通过许可传播安全地优化性能，这仅将未解析分支/存储之后的加载指令标记为不安全。算术和控制指令在发货时无条件标记为安全。此策略的关键直觉是只有加载才能将新的秘密值引入微体系结构。在最年长的未解决微操作之前的负载将把它们的价值提交给架构 GPR，而架构 GPR 不受此威胁模型的保护。请注意，在此威胁模型下也未解决由于异常（如 Meltdown 或 Fore-shadow）导致的错误路径执行；我们将这些称为选择代码攻击。

为了击败 SSB 攻击，我们引入了一种新的安全存储绕过机制，我们将其与严格和许可传播结合使用。在此方案中，与英特尔的 SSBD 不同，即使加载绕过加载存储队列 (LSQ) 中的存储，也允许加载执行。但是，在解析所有绕过存储的地址之前，加载会被标记为不安全。如果绕过的存储以生成顺序违规的方式解析其地址，则内存依赖单元会压缩违规的加载和较年轻的指令。

（3） Load Restriction

NDA 通过阻止来自推测性负载的数据传播来防止选择代码攻击，例如 Meltdown、Foreshadow 、LazyFP 和 MDS 攻击 。这些攻击利用处理器实现

中的特定缺陷，其中数据从最终会出现故障的负载传播。这些缺陷中的每一个都已单独修补。然而，鉴于现代处理器实现的复杂性，人们可能会预料到未来会出现类似的实现错误。此外，在选择代码上下文中，有无数种方法可以导致错误路径执行（错误加载、英特尔 TSX 事务中止、中断传递、断点和系统调用指令、性能计数器溢出、由于内存顺序错误推测 等）正如先前的工作所表明的那样，有效的防御必须解决选择代码攻击背后的常见问题。

我们提出了一个全面的 NDA 保护策略，即负载限制，它既可以阻止所有 11 种记录在案的 选择代码攻击，也可以提供防止未来变体的潜力。例 NDA 的负载限制阻止了我们提交后发现的 MDS 攻击。在负载限制下，负载被认为是不安全的，直到它们成为最年长的未退休指令（即，在 ROB 的头部）。通过负载限制，微架构保证当且仅当负载立即退出时，负载才会唤醒其依赖项。

（4） Preventing All Classes of Attacks

为了击败控制转向和选择代码攻击，NDA 的最终策略构成了严格的传播和负载限制。这个 NDA 政策是最具防御性的，所以我们称之为全面保护。

（5） Security Analysis

具有旁路限制的严格传播：该策略保护内存中的机密并阻止通过控制转向攻击泄露 GPR 中的机密。 Spectre v1、v1.1、v2、v4 (SSB)和 ret2spec 被阻止。最重要的是，NetSpectre、SMoTher Spectre 和我们的 BTB 不是 2.0 GHz 核心 (OoO) 上的参数值架构 X86-64 8 期，无 SMT，32 个加载队列条目，32 个 Store Queue 条目，192 个 ROB 条目，4096 个 BTB 条目，16 个 RAS 条目 Core (in-order) TimingSimpleCPU from gem5 L1-I/L1-D Cache 32kB, 64B line, 8-way set associative (SA), 4 cycle 往返 (RT) 延迟，1 端口 L2 缓存 2MB，64B 线，16 路 SA，40 周期 RT 延迟 DRAM 50ns 响应延迟 表 3：Gem5 模拟配置对于驻留在内存中的秘密，访问阶段的输出能被传输阶段 2〇在同一错误路径执行窗口中使用。对于从 GPR 泄漏内容的攻击者，成功攻击中的传输阶段必须仅包含不相互依赖且仅依赖于分支前指令值的微操作。我们注意到所有现有的攻击都需要多个相关的微操作来传输秘密。

具有旁路限制的许可传播：此策略保护内存中的机密，但不保护 GPR（例如 rax）中的机密。这种保护级别有阻止所有隐蔽通道的额外好处。所有 14 次记录在案的控制转向攻击都被阻止。在未解析的分支或存储之后的任何负载都被标记为不安全。因此，传输阶段 2〇 将无法读取负载的输出。然而，与严格传播不同的是，非负载微操作被标记为安全的。如果秘密已经存在于 GPR 中，攻击者可以使用一系列错误路径操作来预处理和传输秘密。

负载限制：载限制策略解决了所有已知的选择代码攻击，包括 Spectre v3、v3a、v4 、LazyFP 、Foreshadow/NG 和 MDS 攻击 。在选择代码威胁模型中，攻击者已经控制了执行的代码，因此可以轻松访问他们自己的 GPR 和内存空间的内容。负载限制保护特权内存和特殊寄存器中的秘密。具体来说，任何依赖于负载（或类负载指令）的微操作只有在负载退出后才会准备好。退役后，负载返回的值不再是推测性的，因此可以安全读取。负载限制还有可能阻止未来访问内存和特殊寄存器的选择代码攻击。此外，鉴于现有的 25 种推测执行攻击都没有从 GPR 泄露秘密，负载限制策略可以防止所有已知的控制转向攻击。

全面保护：将负载限制与严格的传播策略相结合，提供了 NDA 最具防御性的设计点。全面保护策略可以击败所有 25 种已知的控制转向和选择代码攻击，这些攻击从内存、特殊寄存器中窃取数据，并阻碍攻击者传输 GPR 内容的能力。

实验：

运行 SPEC CPU 2017 基准套件的 gem5 [9] 上评估 NDA。它反映了类似 Haswell 的微体系结构，并与最近的推测执行攻击 的体系结构研究中使用的匹配。为了获得代表具有统计置信度保证的 SPEC 基准性能的结果，通过扩展 gem5 以启用类似于 SMARTS 的模拟采样方法。在真实硬件 (Haswell Xeon E5-2699) 上运行 SPEC 基准测试，并使用 gdb 以固定时间间隔转储其执行状态的快照。开发了一种新工具来将这些快照转换为 gem5 检查点并在模拟中恢复它们的执行。从每个检查点，为 500 万条指令预热模拟状态并测量 100,000 条指令的性能。验证测量期间未知缓存引用的数量（对并非所有标签都在预热中初始化的缓存集的引用）可以忽略不计（即，由于未知缓存引用导致的最坏情况性能错误要小得多比抽样误差）。图 中报告了 CPI 的 95% 置信区间。使用原作者提供的源代码，将 NDA 的性能与具有相同 SMARTS 方

法和 gem5 配置的 InvisiSpec 的两种变体进行比较。 NDA 和 InvisiSpec 在 SPEC 17 上的基线配置的性能在置信区间内相似。由于基准（SPEC 06 与 SPEC 17）和采样方法（单个十亿指令段与 SMARTS 采样）不同，InvisiSpec 的绝对性能数字与原始论文不同。

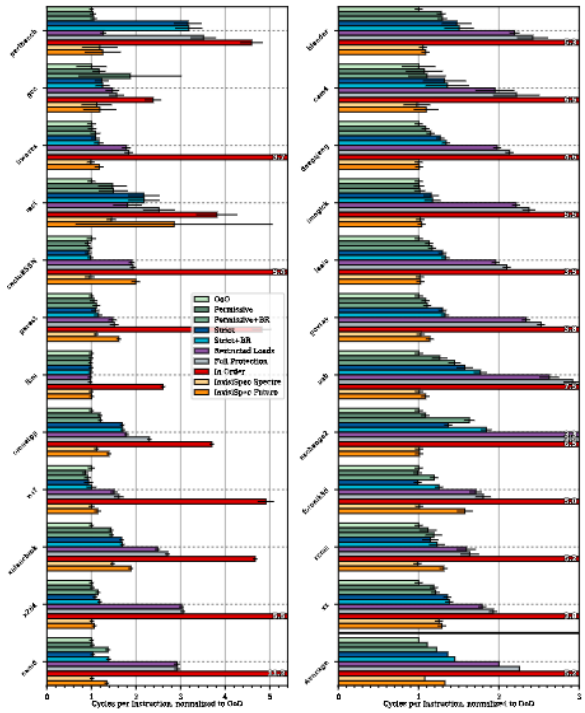

图 11 实验评估结果

3.3.3 总结

推测执行攻击难以缓解。仅仅阻止单个隐蔽通道或特定的利用技术是不够的。为了设计有效的缓解措施，作者根据每次攻击如何导致错误路径执行引入了一种新的推测执行攻击分类，用于控制推测数据传播的新技术 NDA 击败了所有已知的推测执行攻击，并大大减少了未来变体的攻击面。

4 总结

本文概述了有关与投机执行攻击的相关概念，并选取三篇发表的有关投机执行侧信道攻击主题的论文进行分析。

HYPERRACE、Speculator、NDA 都是近年来有关作者所提出的关于投机执行攻击的相关方法工具，通过对防御工具的不断研究，做到“魔高一尺、道高一丈”，积极的应对目前的安全威胁，保护计算机中的相关安全数据不被泄露，是每一位安全从业者的目标和心愿。

通过对攻击变体、攻击目标和攻击方法，防御工具的论文进行了集体分析。可用于未来研究推测执行侧信道攻击的可能组合以及如何帮助减轻它们。

5 参考文献

[1] Chen G， Li M， Zhang F，et al. Defeating Speculative-Execution Attacks on SGX with HyperRace[C]// 2019 IEEE Conference on Dependable and Secure Computing (DSC). IEEE, 2019.

[2] Andrew Johnson , Dr Ross Davies Speculator: A Tool to Analyze Speculative Execution Attacks and Mitigation [C]// 35th Annual Computer Security Applications Conference (ACSA), 2019.

[3] Ofir Weisse, Ian Neal, Kevin Loughlin,et al. NDA: Preventing Speculative Execution Attacks at Their Source [C]// 52nd Annual IEEE/ACM International Symposium on Microarchitecture ,IEEE, 2019.