vmscape:云环境中不完全分支预测器隔离的暴露与利用

Jean-ClaudeGraf[†] 苏黎世联邦理工学院
SandroRu¨egge[†] 苏黎世联邦理工学院
AliHajiabadi 苏黎世联邦理工学院
KavehRazavi 苏黎世联邦理工学院

†合著第一作者

***摘要*——虚拟化是现代云基础设施的基石，为客户提供必要的隔离。然而，这种隔离正受到推测执行攻击的威胁，CPU厂商试图通过将隔离扩展到分支预测器状态来缓解这一问题。我们的系统分析表明，这种扩展不幸是不完整的：虽然现有硬件缓解措施已解决宿主中客户机控制分支预测的最明显案例，但我们发现AMD Zen 1-5和Intel Coffee Lake CPU上存在若干新的Spectre*分支目标注入*（Spectre- BTI）攻击原语，其中一些允许恶意客户机在用户空间执行时控制宿主的间接分支预测。利用上述原语，我们设计了vmscape，这是首个Spectre-BTI 攻击，使恶意 KVM 能够以154B/s的速度从运行在AMD Zen 5服务器系统上的未修改 QEMU 进程中泄漏任意内存，从而暴露用于磁盘加密和解密的加密密钥。我们对可能的缓解策略分析表明，在常见场景中，通过选择性刷新分支预测器以最小性能影响来缓解vmscape是可行的。**

1. 介绍

Spectre v2或*分支目标注入*（BTI）[1]攻击最近受到了极大的关注，通常会从非特权用户[2]泄露特权内存，[3],[4]，[5]，[6]，[7]，[8]虽然这些攻击展示了对安全边界的侵犯，但由于它们假设用户系统上执行本地代码，因此在现实世界中的影响有限。更有趣的威胁模型出现在云中，恶意*虚拟机*（VM）可以利用Spectre- BTI 从主机或另一个VM泄露信息。然而，这些场景中已发布的攻击方式，都需要攻击者在主机中控制代码[1]。[5]，[8] 这是一个不切实际的假设。通过对跨虚拟化边界分支预测状态隔离的系统分析，我们发现了基于Spectre-BTI 的若干新型攻击原语。随后，我们利用其中一种原语构建了首个Spectre- BTI 漏洞利用程序，该程序可从运行未修改默认配置软件的主机中泄露信息。

云中的Spectre- BTI 。虚拟化技术通过虚拟机（VM）的形式，是云环境中安全隔离共置工作负载的主要机制，既能保护客户数据，又能保障底层基础设施安全[9]。Spectre攻击（尤其是Spectre- BTI）会通过滥用CPU内部的共享分支预测器状态，破坏这种隔离机制[1]。在此场景中，常见的威胁模型包括恶意虚拟机攻击虚拟机监控程序（hypervisor）或攻击其他虚拟机。因此，硬件和软件供应商特别关注通过以下方式缓解此类威胁：对虚拟机和虚拟机监控程序中学习到的分支预测条目进行差异化标记，并在不同虚拟机之间切换时刷新分支预测器状态。通过混淆代理攻击虚拟机监控程序所形成的残余攻击面，在不假设代码变更的情况下实际难以被利用[5]。本文研究的问题是：是否存在其他可能来自恶意虚拟机的未被探索的攻击原语

云的新Spectre- BTI 原语。我们发现一个关键观察结果，即现有的分支预测隔离机制过于粗粒度——它们将虚拟机管理程序和虚拟机视为整体实体，而实际上它们包含自己的特权级别。为探究现有分支预测隔离机制如何处理不同虚拟化域中的特权层级，我们系统分析了x86 CPU中攻击者与受害者保护域的所有可能组合，即*主机监督者*（HS）、*主机用户*（HU）、*客户监督者*（GS）和*客户用户*（GU）。分析表明，尽管存在硬件防护措施，所有AMD Zen处理器（Zen 1至Zen 5）和Intel Coffee Lake仍易受新型*基于虚拟化的Spectre- BTI*（vBTI）攻击原语攻击，这些攻击发生在客户用户与主机用户之间，我们将其称为$\text{vBTI}_{\text{GU}\rightarrow\text{HU}}$和$\text{vBTI}_{\text{HU}\rightarrow\text{GU}}$。$\text{vBTI}_{\text{GU}\rightarrow\text{HU}}$使得从恶意虚拟机对主机上运行的用户进程发起新的Spectre- BTI 攻击成为可能，而$\text{vBTI}_{\text{HU}\rightarrow\text{GU}}$则在假设主机为恶意且客户机已部署硬件辅助隔离（如*安全加密虚拟化 - 安全嵌套分页*（SEV -SNP）[10]或*信任域扩展*（TDX）[11]）的场景中，使得新的Spectre- BTI 攻击成为可能。我们还发现了其他需要考虑的vBTI原语，这些原语将在本文中讨论的其他攻击场景中发挥作用。

vmscape.为展示我们vBTI原语的实用性与严重性，我们提出vmscape——首个基于Spectre的端到端漏洞利用方案，其中恶意客户用户可在主机域的管理程序中无需任何代码修改且在默认配置下泄露任意敏感信息。我们以广泛使用的*内核虚拟机*（KVM）/ QEMU [12][13]作为管理程序，并特别以 QEMU 作为主机中管理程序的用户空间组件为目标。虽然$\text{vBTI}_{\text{GU}\rightarrow\text{HU}}$原语验证了该漏洞利用的可行性，但我们针对若干额外挑战展开研究，以实现对AMD Zen 4和Zen 5 CPU的可靠端到端攻击。例如，实现任意内存泄漏需要足够大的推测窗口。然而，对于访客用户而言，获取较大的推测窗口颇具挑战性，因为由于无法直接刷新受害分支指针，这源于缺乏对物理内存的访问权限或代码修改QEMU。为解决这一问题，我们通过逆向工程AMD Zen 4和Zen 5的缓存层级结构，为其非全包含的*最后一级缓存*（LLC）构建了驱逐集，从而为vmscape成功利用 QEMU 中发现的漏洞提供足够大的推测窗口。

vmscape在AMD Zen 5处理器上可以154B/s的速率泄露 QEMU 进程的内存。我们利用vmscape定位机密数据并进行泄露，整个过程仅需102秒，以提取用于磁盘加密/解密的加密密钥为例。分析表明，缓解vmscape攻击需要在特定条件下显式清空*分支预测单元*（BPU）的状态。具体而言，每个 VMEXIT 进入用户空间虚拟机管理程序前，必须设置*间接分支预测屏障*（IBPB）。我们的评估表明，这种由Linux内核维护者针对vmscape开发的缓解措施，在常见场景中仅引入了边际性能开销。

贡献。我们做出以下贡献：

- 首次对跨越所有可能虚拟化与权限边界的分支预测器隔离进行了系统性分析，从而发现了即使在近期CPU缓解措施下仍有效的新型原语。
- 针对未修改软件的首个从客到主的Spectre- BTI攻击，称为vmscape，利用我们新发现的$\text{vBTI}_{\text{GU}\rightarrow\text{HU}}$原语，并对AMD Zen 4和Zen 5上缓存驱逐的新见解，以增加推测窗口。
- 针对不同微架构的缓解方案进行分析，最终以IBPB -on- VMEXIT 作为缓解vmscape攻击的基础。

负责任的披露。我们已于2025年6月7日向AMD和英特尔 PSIRT 披露了vmscape，该问题在2025年9月11日之前一直处于保密状态。Linux维护者根据我们的 IBPB on VMEXIT 建议，通过补丁缓解了vmscape的影响。我们验证了这些补丁能有效阻止vBTI原语。vmscape的追踪编号为 CVE -2025-40300。更多实验源代码及漏洞利用代码等详细信息，可访问https://comsec.ethz.ch/vmscape获取。

2. 背景

我们提供了一些关于Linux中基于*内核虚拟机*（KVM）的虚拟化堆栈的背景（第2.1节）、分支预测机制（第2.2节）、Spectre- BTI 攻击（第2.3节）及其相应缓解措施（第2.4节）。最后，我们通过探讨云场景中Spectre- BTI 攻击的必要性来总结全文（第2.5节）。

2.1. KVM 虚拟化

硬件虚拟化扩展技术，例如AMD SVM [10]和Intel VT-x[14]，为在同一处理器上同时运行多个操作系统提供硬件支持，以最小的性能开销提供强大的隔离保证。*虚拟机管理程序*为每个*客户虚拟机*（VM）呈现对物理系统的完全控制的假象。在Linux系统中，KVM 充当虚拟机监控程序，并以内核模块的形式实现运行在*主机*上。KVM 依赖于一个用户空间组件，负责管理虚拟机的生命周期和模拟设备。我们将这个用户空间进程称为*虚拟机监控程序的用户模式组件*（Hypervisoruser）。QEMU [13]是Linux系统中广泛使用的$\text{Hypervisor}_{\text{user}}$ [15]。它执行诸如创建、启动、停止和迁移虚拟机的任务，并模拟所需的硬件设备。每个客户*虚拟CPU*（vCPU）映射到Hypervisoruser中的$\text{Hypervisor}_{\text{user}}$，使得主机内核能够像普通主机进程一样调度客户执行，因为Linux调度器对线程和进程的处理方式相似。因此，客户线程的执行会与其他客户线程和主机进程交错进行。

进入和退出虚拟机。vCPU线程运行在“客户机模式”或“主机模式”。当从主机模式切换到客户机模式时，系统会向 KVM 内核模块发送KVM_RUN *ioctl*命令，传递指向虚拟CPU的指针。根据平台不同，KVM 会执行vmlaunch/vmresume指令（英特尔平台）或 VMRUN 指令（AMD平台）。CPU从虚拟CPU恢复自身状态后，开始执行客户机程序。当客户机执行敏感或特权指令时，CPU会拦截该指令并触发 VMEXIT。随后CPU将内部状态保存回虚拟CPU结构，继续执行 KVM。根据 VM-EXIT 类型，KVM 会直接处理退出操作，或通过KVM_EXIT将退出任务委托给$\text{Hypervisor}_{\text{user}}$。

保护域。为确保机密性和完整性，计算系统必须强制隔离不同保护域中的实体。在多数场景中，重点在于区分*用户*域与*管理员*域（即环0与环3）。然而在考虑虚拟化环境时，必须扩展这一抽象概念以涵盖客户机执行。如图1所示，用户域包含*主机用户*（HU）和*访客用户*（GU）域，而管理员域则包含*主机管理员*（HS）和*访客管理员*（GS）域。

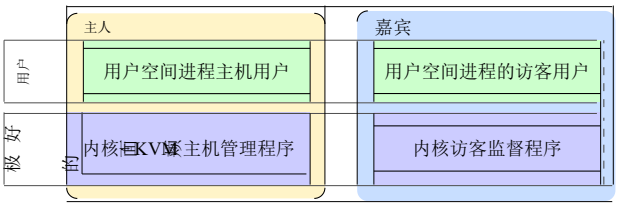


图1. 在搭载虚拟机的x86-64系统中，保护域包含HU、HS、GU和GS，其范围超越用户模式与管理员模式。

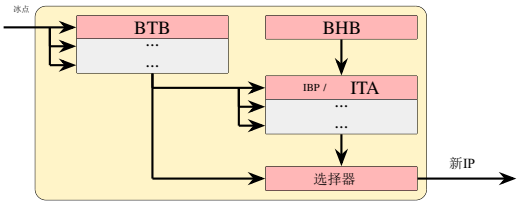


图2。预测间接分支的 BPU 组件。 BTB 提供静态预测，而 IBP /ITA 则将目标与存储在英特尔/AMD BHB 中的路径历史相关联。

2.2. 分支预测

CPU优化的核心在于对分支指令及其目标的精准预测。现代CPU中的*分支预测单元*（BPU）结构极为复杂，由针对不同分支类型和上下文定制的多种预测器组成。静态预测器仅考虑分支的源地址，而*动态预测器*则将分支路径历史作为附加上下文[2]。[3]，[6]，[7],[16]，[17]*间接分支*通过寄存器或内存将控制流转移至目标地址。这使得它们能够拥有多个目标地址， BPU 会尝试将这些目标地址与分支路径历史相关联。英特尔CPU使用*分支目标缓冲区*（BTB）进行静态预测，以及*间接分支预测器*（IBP）进行动态预测。当可用时，动态预测优先于静态预测[8]，[17]在AMD CPU上，动态预测器被称为*间接目标数组*（ITA），当分支遇到多个目标时使用该机制。否则仅使用 BTB [18]。路径历史记录在*分支历史缓冲区*（BHB）中，适用于Intel和AMD。图2展示了预测间接分支时涉及的结构。

2.3. Spectre-BTI

幽灵分支目标注入攻击或幽灵- BTI[19]是一类推测执行攻击，攻击者通过利用间接分支预测错误，跨越保护域边界泄露数据。攻击者使 BPU 预测*目标分支*（作为*推测装置*的一部分）指向*泄露装置*，从而诱导*瞬时执行*。在泄露装置中，秘密以某种方式被访问，使得攻击者后续能够通过*侧通道*（通常基于缓存的刷新+重载[20）推断出该秘密。为导致目标分支的误预测，攻击者*训练* BPU 使用一个*训练分支*，该分支*与目标预测器中的目标分支发生冲突*。若要针对特定预测器，攻击者可能还需模拟训练分支中目标分支的路径历史。

Spectre- BTI 的先前研究主要集中在非特权主机用户进程攻击主机管理程序的场景[1]。[2]，[3]，[5]，[6]，[21]相比之下，用户进程之间的攻击相对较少受到关注[7]。

2.4. 幽灵 - BTI 缓解措施

幽灵-*分支目标注入*（BTI）攻击要求攻击者控制用于预测受害者的 BTB 状态，这意味着攻击者与受害者域之间需要共享 BPU 状态。*主要隔离*技术旨在通过阻止关键保护域边界间 BPU 状态的共享来缓解 BTI 攻击。*间接分支受限推测*（IBRS）[22]可防止低权限域（如用户）的分支影响高权限域（如管理员）的分支，从而保护内核免受用户模式攻击者的影响。英特尔和 AMD 已部署*增强型 IBRS*（eIBRS）[22]和*自动 IBRS*（AutoIBRS）[23]以改进 IBRS；这两种缓解措施均为*始终开启*且无需在权限转换时进行昂贵的模型特定寄存器写入。*消毒* BPU 的缓解措施可在潜在恶意条目影响其他保护域的受害者之前将其清除。*间接分支预测屏障*（IBPB）允许开发者在非充分隔离的域转换期间显式清空 BPU 状态。在启用*同步多线程*（SMT）的CPU上，BPU 状态也可能在*兄弟线程*之间共享。*单线程间接分支预测器*（STIBP）[24]是一种限制 SMT 线程间 BPU 状态共享的缓解措施。

不同微架构的可用缓解措施各不相同，默认缓解措施取决于硬件支持和内核配置。表1展示了我们评估的系统上可用的缓解措施，并标明了这些措施在Ubuntu 24.04中是否默认启用。

2.5. 动机

虽然用户对内核的Spectre- BTI 攻击展示了安全违规，但它们需要在受害机器上执行代码，这限制了此类攻击的影响。可以说，Spectre- BTI 更有趣的威胁模型是在云中，攻击者可以很容易地租用一个虚拟机，并对虚拟机管理程序或其他虚拟机执行攻击。之前的工作[1]，[5] 研究者已深入探讨恶意访客攻击虚拟机管理程序的案例。然而这类攻击需要通过修改虚拟机管理程序代码来注入恶意组件，这种技术手段极大限制了其实用性与攻击范围。这引发了一个关键问题：*是否能在不强加过度安全假设的前提下，跨虚拟化边界实施幽灵 BTI 攻击？*

表1。评估了微架构及其针对幽灵-BTI的隔离机制。

供应商CPU		年份代码	微结构 微码			隔离机制			
						IBPB	IBRS	eIBRS/AutoIBRS	STIBP
智力	酷睿i7-8700	2017	咖啡湖S	咖啡湖	0分	◓	●	○	◓
	酷睿i7-13700K	2022	猛禽湖	掠夺者湾格拉西	0x12c	◓	○	●	●
				蒙特	0x12c	◓	○	●	●
	锐 龙 5 1600X	2017	罗马峰脊	禅宗1	0x8001137	◓	●	○	○
	EPYC 7252	2019		禅宗2	0x8301038	◓	●	○	●
空中运动指示器	EPYC 7413 瑞安	2021	米兰拉斐尔	Zen 3	0xa0011d3	◓	●	○	●
	7 7700X	2022	花岗岩山脊	禅宗4	0A601209	◓	◓	●	●
	锐龙5 9600X	2024		Zen 5	0xb404023	◓	◓	●	●

○ 不可用◓ 可用● 可用与默认（Ubuntu 24.04）

3. 威胁模型

攻击者的目标是推断跨越主机客户机虚拟化边界的机密信息。具体而言，我们考虑以下三种威胁模型：第一，$TM_{G\rightarrow H}$描述了攻击者控制客户机虚拟机并针对主机系统的场景。第二，$TM_{H\rightarrow G}$表示恶意主机针对系统上运行的客户机的攻击行为。该威胁模型在考虑依赖英特尔的*信任域扩展*（TDX）或AMD的*安全加密虚拟化-安全嵌套分页*（SEV -SNP）来防御恶意主机的客户机时尤为相关。第三，恶意客户机并非直接攻击主机，而是可能攻击另一个客户机，这体现在$TM_{G1\rightarrow G2}$模型中。我们假设使用硬件虚拟化扩展：AMD采用 SVM [10]，英特尔采用VT-x[14]。此外，我们假设所有组件（特别是主机和客户机内核以及虚拟机监控程序用户$Hypervisor_{user}$。最后，我们假设系统采用表1所列针对Spectre-BTI 的全部默认安全缓解措施。

4. 挑战概述

现有的幽灵 BTI 缓解措施旨在隔离 BPU 状态跨越保护域边界。虽然过去已探索过恶意客户攻击虚拟机管理程序的最明显案例[1]，[5]目前仍缺乏对虚拟化环境中 BPU 状态隔离的系统性分析，特别是在考虑客户机与主机不同权限级别时。这引出了我们的首要挑战：

挑战C1。识别虚拟化环境中 BPU 保护域隔离的漏洞。

为应对这一挑战，我们需要仔细评估 BPU 中各独立结构的隔离效果。在第5节中，我们系统测试了 BTB 和 IBP /ITA在考虑不同权限级别时所提供的隔离能力。系统分析表明，近期开发的许多微架构都存在易受*基于虚拟化的幽灵- BTI*（vBTI）原语攻击的漏洞，这表明BPU 中缺乏完善的主机-客户机隔离机制。

然而，利用这些新型vBTI原语进行攻击的具体后果及潜在影响尚不明确。我们面临的下一个挑战是确定攻击者在不同威胁模型中构建相关攻击的精确条件。

挑战C2。识别并理解虚拟化环境中多孔 BTB 隔离所影响的不同威胁模型。

要应对这一挑战，需要深入分析攻击者利用已识别的虚拟化边界跨域攻击（vBTI）原语所需满足的条件。在第6节中，我们基于第3节提出的三种威胁模型，提出了五种新型攻击场景。这些攻击场景使攻击者能够跨越虚拟化边界，利用已识别的原语对目标系统发起攻击。其中，我们重点研究了一种新型的跨权限vBTI原语攻击——恶意客户机攻击$Hypervisor_{user}$以泄露敏感信息，例如云基础设施机密。尽管底层推测原语在概念上较为简单，但构建实用的端到端攻击方案仍面临重大挑战。

挑战C3。允许跨虚拟化边界进行实际利用和数据泄露。

由于宿主与访客软件层间的交互有限，这使得寻找合适的推测工具和可靠的侧信道变得具有挑战性。要成功实现数据泄露，关键在于确保拥有足够大的推测窗口。然而，在现代AMD Zen微架构（尤其是Zen 4和Zen 5）上实现有效的缓存驱逐仍是一个未解难题。在第7节中，我们详细阐述了在AMD Zen 4和Zen 5架构上构建首个可靠缓存驱逐方案。基于这些研究成果，第8节开发了vmscape——一款实用的Spectre漏洞利用工具，该工具通过我们新发现的vBTI原语，使虚拟机能够跨越虚拟化边界泄露主机内存。

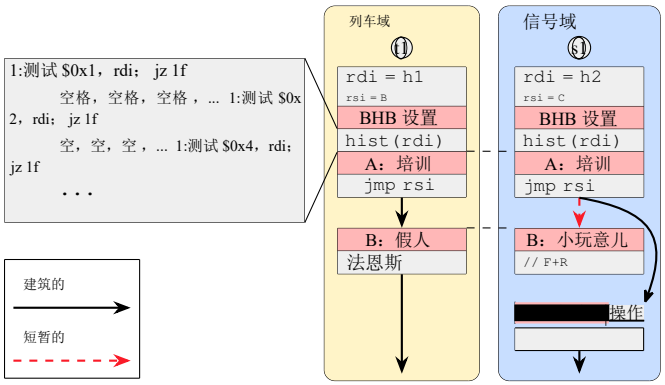


图3. vBTI实验的训练（t1）与信号传输（s1）轨迹使我们能够评估BPU 状态在训练-信号传输域边界处的隔离情况。间接分支的源节点*A*和目标节点*B*在两个域中映射到相同的虚拟地址。每个间接分支前都存在 BHB 设置——通过rdi中的种子值，一系列分支序列将 BHB 初始化为两个轨迹中的相同状态。这确保了 BPU 无法区分训练分支与信号传输分支，因为两者在 BTB 和ITA/ IBP 中会发生冲突。若隔离缺失，信号传输分支会偏离由rsi给出的架构目标，并在*B*处短暂执行训练目标。我们通过披露装置检测这种误判。

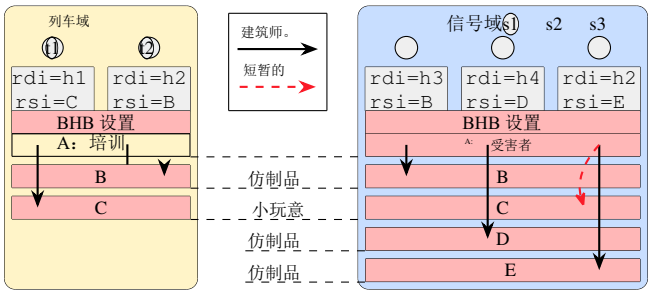


图4.通过组合不同的训练轨迹（t1、t2、s1、s2），我们可以诱导 BPU使用来自 BTB 或 IBP /ITA的信号轨迹s3的预测。分支源*A*和目标*B*和*C*的虚拟地址在两个域中是相同的。

5. 虚拟化环境下的 BPU 隔离

我们分析了间接分支预测的隔离问题在虚拟化环境中的保护域边界跨越多种微架构时，要评估整体隔离保障，我们需要分别考察静态和动态间接分支预测器的隔离效果，并测试x86架构中硬件虚拟化技术对四大基本保护域（硬件虚拟化、硬件安全、通用保护和通用安全）的干扰情况。所有评估系统均列于表1。以近期的英特尔Raptor Lake处理器为例，其采用异构设计架构，包含两种微架构：一种专为高性能任务优化，另一种则针对低功耗任务设计。

5.1. 方法学

为评估跨域 BPU 对间接分支的干扰，我们采用经典的Spectre- BTI [1]实验，如图3所示。我们在一个域进行训练，并在另一个域中使用flush+reload侧信道[20]进行信号干扰。

该分析要求我们在所有四个评估域中执行任意代码。在硬件安全（HS）代码执行方面，我们开发了一个内核模块，允许用户以管理员身份运行任意用户代码。这需要在主机系统上禁用*管理员模式执行防护*（SMEP）和*管理员模式访问防护*（SMAP）。对于客户机执行，我们依赖于Linux内核自带的 KVM 自检机制。这个基础的虚拟机监控程序允许我们将代码映射到任意客户机虚拟地址并执行。但 KVM 自检默认以GS权限运行，因此我们扩展了框架，通过新增权限转换机制实现了GU权限。

在评估过程中，我们必须对所有涉及间接分支预测的结构进行隔离验证。这些结构包括英特尔架构中的 BTB 和 IBP，以及AMD架构中的 BTB 和ITA。逆向工程需要我们控制 BHB 状态，以引发或避免 IBP /ITA中的冲突。如图3所示，我们通过执行伪随机分支序列来初始化 BHB，该序列取决于控制种子。这段历史设置代码直接置于训练分支和受害者分支之前，从而对这些分支处的 BHB 状态进行控制。

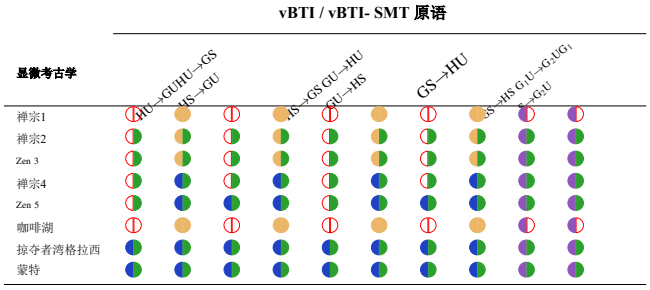
注意，本节中提到的所有修改仅用于逆向工程。第8节中提出的最终攻击不需要任何软件修改，并且运行时使用表1中列出的默认缓解措施。

BTB 预测。如图2所示，多个结构参与间接分支的预测。因此，需要精确的实验设置以确保预测来自 BTB 而非 IBP /ITA。英特尔始终检查 BTB 中的命中，但当可用时优先考虑 IBP 命中[17]。AMD使用 BTB 来预测所有仅见过单个目标的间接分支，否则它更倾向于来自ITA的预测[18]。

在训练域中，从*A*到*B*的训练分支执行会在Intel和AMD上向 BTB 插入条目（图3 t1）。切换到信令域后，我们在*A*到*C*处执行信令分支（s1）。由于*A*在训练域和信令域中是相同的虚拟地址，BTB 无法区分两者。此外，使用种子值h_1实现的不同 BHB 状态$\neq h_2$确保英特尔 IBP 无法提供预测结果，转而采用 BTB。对于AMD，由于分支源*A*目前仅遇到目标*C*，ITA中尚未创建任何条目，因此它也被 BTB 预测。为确保实验环境清洁，我们在每次迭代之间设置 IBPB。本实验中检测到的信号表明存在vBTI易感性。

IBP /ITA预测。测试动态预测器的核心挑战在于确保预测结果不源自 BTB。为此需要执行特定的

表2。所有评估系统观测 VBTI 原始数据的汇总。空半圆表示易受攻击，带色圆表示默认防泄漏缓解措施。



同线程是：易感的，或受以下因素保护的：利托波林 IBPB, AutoIBRS/eIBRS

交叉 SMT 是：易受伤害的，或受以下保护的：利托波林 STIBP

训练轨迹，如图4所示。在训练域中，我们在A处执行训练分支两次，每次针对不同的目标。为了迫使 BPU 区分两次调用，我们设置了不同的历史种子$h_1 \neq h_2$。因此，不仅在 BTB 中创建了一个条目，还在 IBP /ITA内部创建了一个条目。

从训练域到信号域的转换可能会使 BTB 条目失效或被隔离。然而，在英特尔和AMD处理器上， BTB 中的有效条目是 IBP /ITA获取预测结果的必要条件。因此，我们需要确保在切换到信号域后存在相应的 BTB 条目。在英特尔架构中，间接分支在A（s1①处的单次相遇就足以让后续相遇②s3）被 IBP 预测。而在AMD架构中，我们需要额外的分支相遇（s2）指向不同目标，才能让最终分支相遇（s3①的预测由ITA提供。如果本实验中检测到信号出现在当前域而非先前域，这表明 BTB 与 IBP /ITA之间存在隔离失衡。

5.2. 原始评估

为评估隔离保障效果，我们系统测试了虚拟化与权限边界之间所有保护域的组合方案。表1中列出的所有默认缓解机制均已启用。此外，我们分别以间接跳转和间接调用作为训练指令与信号指令各重复实验一次，从而验证两种方式的预测效果是否一致且隔离是否到位。我们的结果表明间接跳转与间接呼叫之间没有差异，这表明它们在缓解措施中被同等处理。因此，在本研究的后续部分中我们不再区分二者。此外，我们观察到干扰信号在客户与主机之间是对称的，这表明隔离机制在双向传输中具有等效性。我们使用符号$\text{vBTI}_{X \to Y}$表示在域X中进行训练、在域Y中进行信令的vBTI原语。

表2展示了各微架构对vBTI原语的易感性。由于 BTB 与 IBP /ITA之间未发现漏洞差异，表2

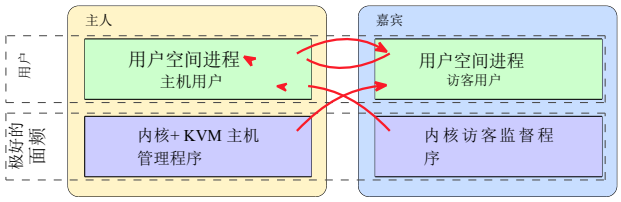


图5. Zen 4处理器的有效保护域隔离机制，易受 $\text{vBTI}_{GU \to HU}$，$\text{vBTI}_{GS \to HU}$，$\text{vBTI}_{HU \to GU}$，以及$\text{vBTI}_{HS \to GU}$.

无需区分它们。我们观察到，除英特尔Raptor Cove和Gracemont外，所有微架构都易受某些vBTI原语攻击。最值得注意的是，Zen 1至Zen 5易受$\text{vBTI}_{HU \to GU}$和$\text{vBTI}_{GU \to HU}$攻击，而Zen 1至Zen 4还易受$\text{vBTI}_{HS \to GU}$和$\text{vBTI}_{GS \to HU}$攻击，如图5所示。虽然Raptor Cove和Gracemont防护充分，但Coffee Lake对上述针对Zen 1至Zen 4的原语仍存在漏洞。然而我们发现，没有任何微架构易受针对HS和GS域的原语攻击。

观测O1。尽管启用了最新的缓解措施，所有AMD Zen微架构和Intel Coffee Lake仍易受$\text{vBTI}_{GU \to HU}$和$\text{vBTI}_{HU \to GU}$的影响。此外，Zen 1至Zen 4和Coffee Lake受到$\text{vBTI}_{GS \to HU}$和$\text{vBTI}_{HS \to GU}$的影响。

BPU 隔离机制。AMDZen4和Zen5采用AutoIBRS作为Spectre- BTI 缓解措施。在监督模式下，它会阻止对间接分支目标的推测性执行，直到这些目标被解析[5]。[6]通过禁用AutoIBRS重新运行先前实验，我们能更深入地了解Zen 4和Zen 5微架构的隔离机制。Zen 4在缺乏AutoIBRS时会暴露于所有vBTI原语攻击，而Zen 5则不受跨用户-监督者边界原语的影响。这表明AMD Zen 5 CPU在AutoIBRS基础上还具备额外的原生隔离机制，相当于为预测条目设置了单比特特权级别标签。

观察点O2.在Zen 5架构中，分支预测条目通过单比特特权级别进行标记，从而实现用户域与管理域的隔离。

然而，单个比特位不足以区分虚拟化环境中存在的四个域。因此，仍需依赖AutoIBRS来分离例如HS域和GS域。

观察结果O3.尽管 BTB 中添加了隔离位，Zen5的 BPU 仍无法区分主机域和客户域，因此仍依赖AutoIBRS来保护管理域。

SMT 隔离。为补充先前测试同线程隔离的实验，我们采用改进实验来验证vBTI- SMT 原语。我们设计了一个训练过程，该过程迭代执行训练分支（图3 t1）。随后将信号处理㊀程绑定至 SMT 的兄弟核心，并以不同历史记录执行信号分支（t2）。信号处理完成后㊁我们使用 IBPB 刷新 BPU，使其*忘记*设备目标。如表2所示，虽然Zen 2至Zen 5及Raptor Lake架构上的 STIBP 能成功隔离兄弟线程，但Zen 1架构上缺乏 STIBP，导致vBTI-$SMT_{HU \rightarrow GU}$、vBTI-$SMT_{GU \rightarrow HU}$、vBTI-$SMT_{HS \rightarrow GU}$和vBTI-$SMT_{GS \rightarrow HU}$原语无法实现隔离。此外我们发现，尽管Coffee Lake支持 STIBP，但默认并不启用该功能，从而允许使用相同的跨 SMT 原语。

观测O4。Zen 1上完全缺失 STIBP 且仅在Coffee Lake上条件性启用，这两种微架构似乎都易受vBTI SM-$T_{GU \rightarrow HU}$、vBTI-$SMT_{HU \rightarrow GU}$、vBTI-$SMT_{GS \rightarrow HU}$以及vBTI-$SMT_{HS \rightarrow GU}$的影响。

6. 不同威胁模型下的开采

我们的实验表明，部分虚拟化边界未能得到充分隔离，导致跨边界分支目标干扰。本节将结合第3节提出的三种威胁模型，探讨五个现实且此前未被探索的场景，这些场景将导致具体的安全违规行为。我们将每个攻击场景表示为$A \rightarrow B \mid V$，其中A和B分别代表攻击者和受害者的防护域，V指代具体目标（例如主机进程）。

6.1. 从客到主的攻击

我们分析的第一个威胁模型是$TM_{G \rightarrow H}$，该模型考虑了恶意访客针对主机系统的攻击。我们在此模型中识别出三种场景。

S1 G → H|主机进程。在此场景中，恶意访客针对主机用户进程泄露机密，例如SUID进程的密码哈希[7]。受害进程可能在访客被抢占后被调度到同一硬件线程（即*同线程*），或被调度到同一物理核心上的 SMT 子线程（即*跨 SMT 线程*）。在同线程情况下，除了缺乏访客到主机的 BTB 隔离外，攻击还需要在任务切换期间不存在 BTB 净化。根据我们的逆向工程观察，我们发现Zen 1至Zen 5以及Intel Coffee Lake缺乏足够的隔离，容易受到$vBTI_{GS \rightarrow HU}$和$vBTI_{GU \rightarrow HU}$原语的影响。对于跨 SMT 攻击，必须将客户机到主机的 BTB 干扰与兄弟线程之间的共享预测器状态耦合，这由vBTI-$SMT_{GS \rightarrow HU}$和vBTI $SMT_{GU \rightarrow HU}$原语表示。Zen 1和Coffee Lake微架构上满足这一条件。

虽然这些攻击在理论上可行，但实际操作中面临诸多挑战。基于客户机的攻击者通常对宿主用户进程的控制力有限，难以可靠地操控或与之交互。在客户机与任意宿主用户进程之间建立可靠的侧信道尤为困难，因为攻击者既缺乏空间控制也缺乏时间控制。特别是攻击者无法决定目标进程的执行时间和执行位置，这进一步增加了精准利用攻击的难度。

S2 G → H|$\mathbf{Hypervisor_{user}}$。此场景考虑了一个恶意访客针对其自身的$Hypervisor_{user}$。这需要在VMEXIT和KVM退出操作中进行不充分的 BTB 消毒，同时在同一线程场景下缺乏客户机与主机之间的 BTB 状态隔离。与一般的G → H|主机进程攻击相比，此处的利用更为简单。客户机可故意触发由其$Hypervisor_{user}$处理的VMEXIT，从而在相同 SMT 场景下实现频繁的攻击者-受害者交互。交叉 SMT 场景也更具实际操作性。通常情况下，调度机制通过降低攻击者与受害者共处的可能性来缓解交叉SMT BTI 攻击。然而在此场景中，受害者执行同一客户机的$Hypervisor_{user}$，使得共处可能性大幅增加。尽管攻击者只能访问映射到$Hypervisor_{user}$地址空间的内存，但其中仍可能包含敏感数据，例如用于网络或存储的云基础设施机密[25]。我们观察到Zen 1至Zen 5及Coffee Lake架构存在此场景的漏洞，其中Zen 1和Coffee Lake还存在交叉 SMT 场景的漏洞。

观测O5。由于攻击者和受害者属于同一客户执行环境，交叉 SMT G → H|$Hypervisor_{user}$攻击更容易发生，这使得共址攻击的可能性大大增加。

S3 G → H|Guest Kernel.在此场景中，攻击者利用$Hypervisor_{user}$作为被误导的代理，从攻击者自身的客户机实例中提取内核内存。因此，此场景假设攻击者是运行在客户机中的无权限进程。这是 G → H |$Hypervisor_{user}$场景的变体，但泄漏的不是$Hypervisor_{user}$秘密，而是客户机自身的内存，这些内存正在$Hypervisor_{user}$中映射。除了要求$Hypervisor_{user}$必须映射客户机内存外，其前提条件与 G → H |$Hypervisor_{user}$相同。具体而言，这些前提条件包括缺乏客户机到主机的 BTB 隔离，以及对 VMEXIT 和 KVM_EXIT 的消毒不足。

观测O6。包括Zen5在内的多种近期微架构都容易受到G → H|GuestKernel攻击。

表3。按威胁模型划分的攻击场景，包含需求、目标及易受攻击的微架构。

场景	目标	SMT	所需的基本图形	可开采的 禅宗1	禅宗2	Zen 3	禅宗4	Zen 5	咖啡湖	猛禽湾	格雷斯蒙特
S1 G → H | 主机进程	主机用户进程	同一的	vBTIGS →HU或 vBTIGU →HU	✓	✓	✓	✓	✓	✓	✗	✗
		穿越	vBTI-SMT$_{GS \rightarrow HU}$或 vBTI-SMT$_{GU \rightarrow HU}$	✓	✗	✗	✗	✗	✓	✗	✗
S2 G→H | Hypervisor$_{user}$	管理程序用户	同一的	vBTIGS →HU或 vBTIGU →HU	✓	✓	✓	✓	✓	✓	✗	✗
		穿越	vBTI-SMT$_{GS \rightarrow HU}$或 vBTI-SMT$_{GU \rightarrow HU}$	✓	✗	✗	✗	✗	✓	✗	✗
S3 G → H | 客户内核	客座主管	同一的	vBTI$_{GU \rightarrow HU}$	✓	✓	✓	✓	✓	✓	✗	✗
		穿越	vBTI-SMT$_{GU \rightarrow HU}$	✓	✗	✗	✗	✗	✓	✗	✗
S4 H → G | SEV -SNP过程	访客用户进程	同一的	vBTIHU →GU或 vBTIHS →GU	- -	- -	✓	?	✗	- -	✗	✗
		穿越	vBTI-SMT$_{HU \rightarrow GU}$或 vBTI-SMT$_{HS \rightarrow GU}$			✗	✗	✗		✗	✗
S5 G_1 → G_2| 客户进程 其他客户用户进程		同一十字	vBTI$_{G_1U \rightarrow G_2U}$	✗	✗	✗	✗	✗	✗	✗	✗
			vBTI-SMT$_{G_1U \rightarrow G_2U}$	✓	✗	✗	✗	✗	✓	✗	✗

✓ 易损X不易损 - 不适用

？ 该功能在测试系统上不受支持

6.2. 从主机到客户机的攻击

我们分析的第二个威胁模型是TM$_{H \rightarrow G}$，其中恶意主机针对的是访客。

S4 H → G|SEV -SNP 过程。在此场景中，恶意主机攻击目标为一个假设为敌对主机并采用基于硬件隔离技术（如 SEV -SNP [10] 或 TDX [11]）的客户机。与TM$_{G \rightarrow H}$不同，攻击者控制着客户线程的调度，从而对受害者拥有更多控制权。由于这些技术会施加额外的安全措施，仅凭在未部署 SEV -SNP/ TDX 的系统上存在vBTI$_{HS \rightarrow GU}$、vBTI$_{HU \rightarrow GU}$、vBTI-SMT$_{HS \rightarrow GU}$以及 vBTI SMT$_{HU \rightarrow GU}$原语，不足以判定不同微架构的漏洞。然而，我们确认在Zen3架构中，默认情况下 SEV -SNP不会隔离客户机与主机之间的 BTB 。在Zen5架构中，我们注意到预测结果在主机与 SEV -SNP客户机之间呈现了正确的隔离。遗憾的是，我们没有Zen4服务器来验证这种隔离效果。虽然Zen1和Zen2不支持 SEV -SNP，但在使用 SEV 或 SEV -ES等较弱技术的场景中，它们可能存在安全漏洞。

观测结果O7.AMD Zen 3易受H → G| SEV -SNP攻击。

SEV -SNP为访客提供了多种可选保护机制，我们将在第9.3节讨论。虽然我们的分析表明较旧的、pre-eIBRS Intel微架构对vBTI$_{HS \rightarrow GU}$、vBTI$_{HU \rightarrow GU}$、vBTI-SMT$_{HS \rightarrow GU}$以及vBTI SMT$_{HU \rightarrow GU}$原语的易感性，但这些系统不支持 TDX 。

6.3. 宾客之间的攻击

我们分析的第三个威胁模型是TM$_{G1 \rightarrow G2}$。在此模型中，恶意客户端攻击同一主机系统上运行的另一客户端。

S5 G_1 → G_2|客户进程。在此场景中，恶意客户针对另一个客户的用户进程，凸显了不可信虚拟机共存的风险。这要求不同客户之间缺乏 BTB 隔离。与早期场景类似，我们区分了同线程和跨 SMT 两种情况。在同线程情况下，受害者在攻击者线程抢占后，于攻击者执行的同一线程上运行。这还要求虚拟机交换机缺乏 BTB 净化。然而，Linux内核通过虚拟机交换机的 IBPB 缓解了这一问题。跨 SMT 情况则要求 BTB 状态在兄弟线程间共享。根据我们在第5.2节的逆向工程，我们发现Zen 1和Coffee Lake均缺乏 SMT 线程隔离，使其容易受到G_1 → G_2| 客户进程攻击。

观测O8。AMD Zen 1和Intel Coffee Lake容易受到G_1 → G_2| Guest Process攻击。

然而，实际应用中仍存在诸多挑战。通常情况下，公共云主机上缺乏攻击者与客户机虚拟机之间的直接交互通道，也缺乏用于侧信道攻击的共享内存。

6.4. 摘要

表3概述了攻击场景，总结了其前提条件、目标对象及易受攻击的微架构。我们的分析表明，vBTI原语可在第3节介绍的三种威胁模型中引发具体安全漏洞。这些包括恶意客户机对主机的攻击（TM$_{G \rightarrow H}$）、恶意主机对客户机的攻击（TM$_{H \rightarrow G}$），甚至两个客户机之间的攻击（TM$_{G1 \rightarrow G2}$）。

为验证vBTI攻击的可行性，我们提出vmscape——一种针对G → H|Hypervisor$_{user}$场景的端到端漏洞利用方案。该场景代表

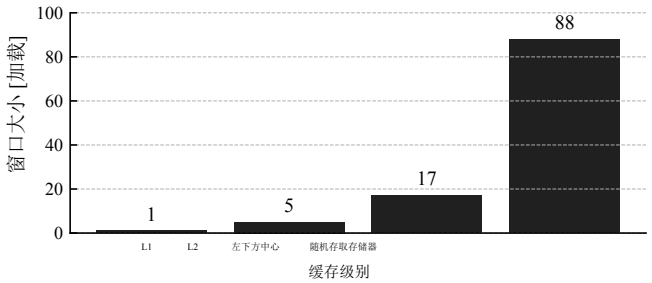


图6. Zen 4架构中推测窗口大小与执行的依赖加载最大数量的关系，其中间接分支目标的指针位于相应缓存层级。

对于云提供商来说，这是一个特别严重的案例，因为泄露的$\text{Hypervisor}_{\text{user}}$内存针对的是他们的基础设施。

7. 扩大投机窗口

我们的目标是在G → H|$\text{Hypervisor}_{\text{user}}$场景中构建一个端到端的内存泄漏漏洞利用，目标是最新的AMD Zen 4和Zen 5系统。在构建该漏洞利用时，我们发现受害分支的推测窗口太短。我们首先解释了如何评估推测窗口的大小，然后讨论了如何通过缓存驱逐来扩大它。

推测窗口大小。为测量推测窗口的大小，我们设计了以下实验：采用从内存获取目标地址的间接分支，使 BPU 能够预测并推测目标地址。推测窗口的大小取决于目标地址在架构层面的解析速度，而该速度又取决于存储间接分支目标地址的缓存层级。我们通过在推测过程中连续执行n次依赖性加载操作，并检查最后一次加载的缓存命中情况来测量推测窗口的大小。我们确保除最后一次加载外的所有加载操作都位于L1缓存中。针对每个$n \in [1,127]$，我们重复实验4096次，并改变存储间接分支目标地址的缓存层级。

如图6所示，当目标地址位于L1层级时，推测窗口期间仅执行一次依赖性加载。这种推测窗口对于基于刷新+重载的侧信道攻击而言并不充分，因为该攻击至少需要两次依赖性加载。虽然L2层级的实验结果表明其推测窗口已足够大以实现完美推测装置，但某些攻击仍需要更大的推测窗口。这可能源于先前指令产生的额外争用，导致披露装置运行变慢，或是因为受害软件在指令流中更早阶段就开始加载目标地址（例如验证其是否为有效指针）。这些因素会压缩披露装置的推测时间窗口，迫使攻击者不得不扩大整体推测时间窗口。

通过清除存储间接分支目标地址的内存，可延长推测窗口。

表4。缓存详情在AMD ZEN 3、ZEN 4和ZEN 5上。该尺寸可能在同一代处理器之间存在差异。

核心	L1			L2			*最后级缓存*（LLC）		
	尺寸	集合	方法	尺寸	集合	方法	尺寸	集合	方法
Zen 3	32KB	64	8	512KB	2048	8	32兆字节	32768	16
禅宗4	32KB	64	8	1兆字节	2048	8	32兆字节	32768	16
Zen 5	48千字节	64	12	1兆字节	1024	16	32兆字节	32768	16

从缓存中读取数据，从而延缓目标分支的解析[1]。然而，虚拟机内部的攻击者无法直接清除目标指针，因为其对应的物理内存是分配给主机进程的，虚拟机无法直接访问。因此，攻击者需要将目标指针从缓存层级中驱逐。我们注意到，驱逐操作应仅针对必要的缓存集，以避免在推测阶段拖慢整个目标执行速度。

7.1. AMD Zen 4和Zen 5的LLC驱逐

Wang等人的近期研究[26] 我们验证了在AMD Zen 3架构上构建LLC（局部缓存）淘汰集的可行性。基于他们的研究成果，我们首次为Zen 4和Zen 5架构实现了LLC淘汰集构建。为达成目标，我们遍历了缓存层级的所有层级，识别出与Zen 3架构的差异，并逐步淘汰更大层级的缓存，从而获得更长的推测窗口。随后，我们展示了如何将淘汰集构建过程移植到虚拟机中，并利用XOR 索引技术使该过程高效可靠。表4列出了所评估处理器的相关缓存结构。为简洁起见，我们先专门讨论Zen 4架构，再总结其与Zen 5架构的差异。

为实现逆向工程，我们采用1GB分页机制以确保虚拟地址与物理地址之间的位匹配。我们进一步使用AMD的精确 APERF 计时器，该计时器可通过rdpru指令访问[27]。默认情况下，1GB分页和精确计数器对客户机不可用。因此，我们仅在逆向工程期间使用这些功能，而不用于第8节的端到端漏洞利用。

缓存访问延迟。我们首先对每个缓存层级的内存访问时序函数进行校准。这使我们能够确定内存访问是从L1、L2、LLC还是主内存中获取的。为确定延迟，我们按以下步骤进行：从包含N条缓存行的内存区域开始，以伪随机顺序访问所有缓存行并记录访问时间。随着N值的增加，重复此过程。在图7中，我们绘制了内存区域大小N增加时的中位访问时间。延迟平台表示L1、L2、LLC和RAM的访问延迟。

L1淘汰集。根据我们先前的缓存访问测量数据，可以确认Zen 4的L1索引方案与Zen 3完全一致。我们可通过访问8个不同的缓存行（对应于L1路数）来清除任何L1条目，这些缓存行均共享匹配位[6:11]。

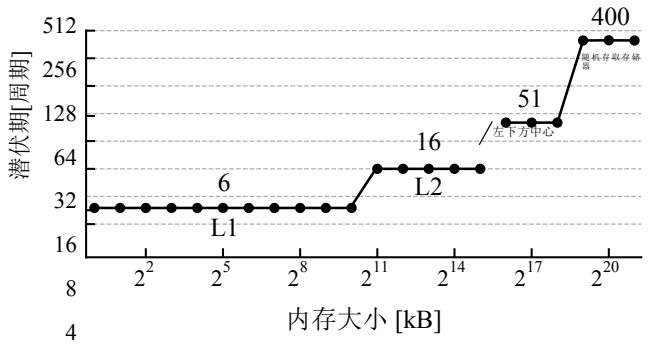


图7. Zen 4处理器上不同大小内存区域的中位存取延迟（以对数坐标显示）。平台期的延迟读数分别表示L1、L2、LLC和RAM的延迟。

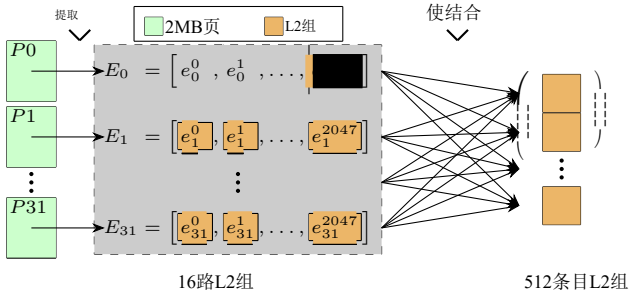


图8.基于虚拟机的攻击者的LLC缓存驱逐集构建策略。第一步，我们*提取*不同2MB页面中包含的L2驱逐集。第二步，我们将来自不同2MB页面的L2驱逐集*合并*。

L2淘汰集设计。假设L2索引的架构设计使得2MB大页内存均匀分布于所有L2缓存集。因此，需要在2MB页边界下方至少11个索引位才能映射到全部2048个L2缓存集。基于先前对Zen 3处理器缓存逆向工程的研究[26]，我们进一步假设2MB页边界下方的部分高位未用于L2索引。考虑到2MB页面包含21个地址位，其中11位用于索引，6位用于64B缓存行内的偏移量，我们推测最上方的4位未被用于缓存索引。这将导致每个L2缓存集包含16个条目，我们预计这足以实现所有8路的淘汰操作。

我们通过以下方式验证上述假设。我们随机选取一个2MB的巨型页面，根据[6:16]位的数值将该页面内所有缓存行对齐地址划分为多个集合。通过测量重复访问集合内所有条目的平均延迟，我们确认每个生成的集合都具有自驱逐特性。进一步通过测量重复访问每个集合中半数地址的延迟，我们验证了任意两个生成集合之间不存在相互驱逐现象。基于实验结果，我们得出结论：AMD Zen 4架构的二级缓存索引机制并未使用[17:20]位。

观测结果O9.在AMD Zen 4架构的L2缓存索引中，位[17:20]未被使用。

AMDZen4的L2缓存有8个替换通道，根据*最近最少使用*（LRU）替换策略，我们预期最小的替换地址集大小应为8个地址。然而，实验发现至少需要12个替换地址才能可靠地将目标地址从L2缓存中替换出去。我们认为这种现象是由于替换策略比 LRU 更为复杂所致。

LLC驱逐设置。AMD Zen 4架构中的LLC（乱序缓存）是L2缓存的非包含式缓存。因此，要将缓存行插入LLC进行驱逐，必须先将这些缓存行从L2中驱逐出去。基于L2驱逐集，我们可识别出同一1GB大页内的所有地址。通过检测地址是否被驱逐到该驱逐集，即可确定其映射关系。假设地址在1GB大页内存的2048个L2集内均匀分布，理论上应有8192个地址映射到同一L2集，实验结果也印证了这一预期。我们进一步确认，前512个地址已能为同一L2缓存集内的任意地址提供可靠的LLC驱逐。虽然这还不构成最小驱逐集，但对我们的攻击已足够高效，并显著延长了推测窗口的长度，如图6（RAM）所示。

Zen 5架构由于L1和L2缓存的扩展方式，其缓存淘汰集的容量也相应增加。相较于Zen 4，Zen 5通过减少L2缓存集数量，使容量再次与Zen 3持平。因此我们发现，与Zen 3相同，2MB内存页中的[16:20]位段仍处于未使用状态。最后，与Zen 4相比，Zen 5需要处理的地址数量翻倍，且每个地址需访问两次，这使得推测窗口期的可靠扩展成为可能。

7.2. 从虚拟机中驱逐

我们的LLC驱逐机制使攻击得以成功，但其采用的精确计数器和1GB分页机制默认不提供给 QEMU 访客。此外，通过暴力破解新地址集合成员来增大每个L2驱逐集合的规模既耗时又易受噪声干扰。首先，我们通过同时测量驱逐集内所有条目的访问时间而非单独测量，克服了计时器限制，从而放大信号。其次，我们提出了一种显著更高效的驱逐集生成方法，无需依赖1GB大页。

我们注意到，Linux系统默认采用透明2MB大页机制来支持虚拟机页面，以此提升性能。因此，我们基于2MB页面构建L2缓存集的方法在虚拟机内部仍然适用。此外，我们提出：当存在大量此类2MB页面时，可以像图8所示那样，为每个页面单独*提取*L2缓存集。虽然这为LLC（页表冲突）驱逐提供了充足地址，但我们仍需找到一种高效方法来*整合*不同页面生成的L2缓存集。

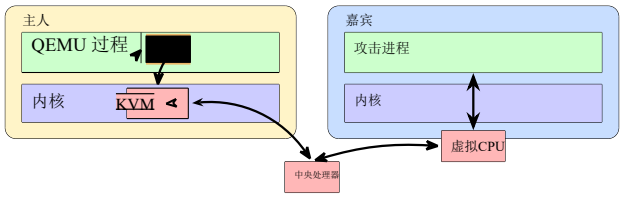


图9.基于 KVM 的客户机运行于支持硬件虚拟化的CPU上，由 QEMU 管理。

令*eij*为源自页面*j*的第*i*个驱逐集，其中*i*由L2集合索引位的低位[6:16]确定。令E_j= {*eij* |*i* ∈[0,2047]}为页面*j*的驱逐集集合。我们的目标是合并E_j中的所有L2集合（$j \neq 0$) 到E_0的 L2 集合。现在，假设对于某个*j*、*a*和*b*，集合*eaj*与集合*eb0*发生碰撞。我们希望高效地确定与哪个集合*ecj*会发生碰撞。我们假设 L2 和 LLC 使用基于 XOR 的索引函数，这将允许按如下方式计算碰撞集合*ed0*：

$$d=a \text{ 与 } b \text{ 与 } c \text{ 的并集}$$

采用这种方法，为每个E_j（*j*）的单个元素找到匹配的L2集$\neq 0$)足以确定所有其他元素的冲突驱逐集。Zen 3架构[26]的先前方法需要逐个采样地址并测试*每个*地址是否被驱逐，从而增加初始L2集的规模。我们的新方法仅需*一次*执行该过程，利用2MB页面内的整个第一L2集，即可无需进一步测试直接为同一页面的其他32752个地址分配各自的驱逐集。此外，该方法具有极高的可靠性，在100次运行中首次尝试即以100%成功率创建LLC驱逐集，而先前研究[26]在Zen 3架构使用1GB页面时报告的成功率约为60%。

观测O10。基于 XOR 的缓存索引功能可高效合并来自不同2MB页面的L2集，形成更大的L2集。

随着推测窗口的成功扩展，我们现在着手构建vmscape。

8. VMSCAPE

我们在第5节介绍的跨域实验揭示，所有AMD Zen CPU和Intel Coffee Lake CPU都容易受到$\text{vBTI}_{\text{GU}\rightarrow\text{HU}}$原语的攻击。在本节中，我们介绍vmscape——作为恶意用户端的端到端漏洞利用工具，用于泄露虚拟机管理程序机密。图9展示了相关组件。我们的攻击目标是QEMU，这是Linux生态系统中工业领域常用的$\text{Hypervisor}_{\text{user}}$[15]。然而，威胁模型（参见第3节）或漏洞利用设计中的所有假设，都无法从根本上阻止使用稍作修改的相同技术来攻击其他$\text{Hypervisor}_{\text{user}}$。我们在搭载Linux的AMD Zen 5服务器CPU上运行了该攻击。

内核版本6.8.0-85-generic和Ubuntu25.10软件包，搭载最新 QEMU v10.0.2。我们进一步验证了vmscape在AMD-Zen4系统上成功泄露信息。

在第7节中，我们解决了该攻击的推测窗口挑战。然而，要开发端到端的漏洞利用，还需解决两个额外挑战：

- 首先，我们需要确定一个由侧信道、推测装置和披露装置组成的合适漏洞利用链（第8.1节）。
- 其次，我们需要对*地址空间进行去随机化*和*布局随机化*（ASLR）QEMU 以确定受害分支地址及重载缓冲区地址（第8.2节）。

最后，我们将解决方案与推测窗口延长技术相结合，构建针对 QEMU 的任意内存泄漏攻击（第8.3节）。

8.1. 漏洞利用链

我们的目标是在 QEMU 未修改的代码中发现一个漏洞利用链。要构建这样的链，需要在 QEMU 二进制中找到合适的侧信道、推测装置和披露装置。

侧信道。我们首先定义用于泄露主机机密的侧信道。通过调查，我们发现 QEMU 将所有客户机内存映射到其虚拟地址空间。主机与客户机之间的共享内存支持flush+reload操作，这可作为侧信道[20]。

观测O11。QEMU 将所有客户机内存映射到$\text{Hypervisor}_{\text{user}}$虚拟地址空间，支持刷新+重新加载操作。

flush+reload通过检测最近被访问的内存位置（与受害者共享）来泄露机密信息。作为侧信道的共享内存称为*重载缓冲区*。

推测装置。我们的漏洞利用链需要触发推测操作，为此vBTI原语采用了间接调用或间接跳转。我们通过在可控制寄存器的区域寻找此类推测装置，从而影响被推测目标的执行。通过flush+reload操作，我们试图同时控制两个寄存器：一个存储秘密指针，另一个存储重载缓冲区地址。此外，理想的推测装置能够被重复触发、稳定运行且高频次执行，从而实现高带宽泄漏。

QEMU 主要负责 KVM 虚拟机管理中的I/O和设备模拟。因此， QEMU 的活跃代码量相对较小。我们仅搜索通用I/O处理代码，避免针对特定设备模拟，以免对威胁模型施加额外假设。

我们确定了 QEMU 的*内存映射I/O*（MMIO）写入处理中一个合适的受害分支，如图所示

```
静态MemTxResult内存区域写入访问器（Memor-
yRegion *mr，hwaddr addr，uint64__t*value，uns-
igned size，signed shift，uint64__t mask，MemTx-
Attrs attrs）{
    uint64__t tmp = 内存区域移位写入访问(
                值、移位、掩码);
    //  [...]
    mr->ops->write（mr->opaque，addr，tmp，
    size）；返回MEMTX OK；
}
```

清单1. QEMU 系统内存.c源文件片段，显示第12行的受害者分支，其中tmp变量存储着攻击者控制的数值。

在清单1中，攻击者能够控制临时变量tmp中的数值，该数值对应于 MMIO 操作中写入的值。虽然攻击者在函数调用时仅直接控制单个64位寄存器（*RDX*），但动态分析表明，第二个寄存器（*R12*）中也残留着先前使用的相同数值。此外，我们可以通过共享内存传递额外数值，但需要在披露装置中增加一次额外的加载操作。

观测O12。MMIO 实现了客户机与虚拟机管理程序之间可重复、可靠且高频次的寄存器控制交互。

需要注意的是，虽然通用 MMIO 支持8B写入，但被写入的设备也必须支持这种粒度的写入。在我们的主要攻击中，我们使用了 QEMU 虚拟机默认配备的hpet高精度事件计时器。

泄露装置。为从目标程序中获取机密信息，我们需要一种基于flush+reload机制的泄露装置。我们基于Wikner和Razavi[2]提出的泄露装置扫描器进行改进。由于原始扫描器无法在 QEMU 二进制中识别所需装置，我们通过优化内存指令的污染逻辑（针对使用不同污染状态基址寄存器和索引寄存器的指令）来扩展扫描器功能。此外，我们新增了链式装置检测能力——通过将两个独立装置组合，其中第一个装置以间接调用结尾，经过训练可对第二个装置进行误判。这些改进使我们能够精准定位目标泄露装置。如清单2所示，这是本攻击中使用的最终装置链。

该装置不会对密钥应用任何乘数，这可能对flush+reload造成问题。然而，先前的研究表明，我们可以通过依赖重装缓冲区基址的移位技术，并检查密钥是否最终位于不同的缓存行来克服这一限制[2]。

分支碰撞。当到达 QEMU 中的目标分支时，控制流刚通过一个 VMEXIT，随后执行KVM退出。因此，分支历史会受到客户机、 KVM 和 QEMU 中分支的影响。

```
// 零件链的首段
0xac334e移动rdi，qword ptr [r12+ 0x2b58] 0
xac3356调用 qword ptr [r12+ 0x2b50]
// 电子配件的第二部分
0x8260b5 加载 CL，字节指针[RDI] 0x8260
b7 测试十六字节指针[RDX+ RCX ]，ESP
ő
```

清单2. 分两部分的披露装置。第2行加载秘密指针，第6行检索一个字节的秘密，第7行访问重装缓冲区。

取决于历史缓冲区的大小。由于AMD的历史分支预测器优先于静态 BTB 预测，因此需要覆盖或使注入目标的历史预测失效才能观察到其效果。虽然在训练过程中准确复现历史预测是可行的，但难度较大。根据AMD文档[18]，[28]该机制仅对遭遇过多个攻击目标的间接分支进行历史记录分析。通过在训练前使用 IBPB 清空 BPU，恶意攻击者可确保目标分支仅观察到训练目标。因此，该分支无需依赖最后两个分支之外的匹配历史记录即可被误判，而这两个分支相对容易复现。虽然允许访问 IBPB 的攻击者看似是合理假设，但事实上这正是预期的——攻击者本应能够自行实施域隔离。

观测项O13。客户端软件可利用对缓解控制的访问权限，影响主机软件可获取的分支预测条目。

然而，AMDZen4的Zen5及后续微代码更新修改了IBPB，使其同时使 BTB 中的直接分支类型信息失效，以帮助缓解*推测性返回栈溢出*（SRSO）[6]。[29]这种设计无意中缩小了我们攻击的推测窗口范围，推测是因为在第一个受害者指针加载与受害者分支之间设置直接分支会减慢推测执行速度。因此，我们必须让 BTB 忘记受害者分支是具有多个目标的间接分支，而无需使用IBPB 。在AMD Zen 4和Zen 5架构中，我们可以通过以下方式实现：(1) 将训练分支替换为直接跳转指令，(2) 执行训练流程，(3) 再将原始训练分支写回。该方案与AMD针对Retbleed漏洞的Jmp2Ret缓解措施[30]保持一致。

现在我们能够可靠且重复地劫持 QEMU 中的受害分支，将临时执行重定向至我们的披露装置。

8.2. 打破 ASLR

Linux默认使用 ASLR 随机化用户空间代码、堆和栈的虚拟地址。要成功发起攻击，我们需要知道推测装置和泄露装置的位置。

该虚拟地址是 QEMU 映射虚拟机内存时所使用的，而我们的重载缓冲区就位于该地址处。

部分代码 ASLR 去随机化。Linux将 ASLR 实现为相对于程序非 ASLR 基地址的随机偏移量。最大偏移量是一个可配置参数，默认为Linux选择的某个动态值。目标系统使用默认的最大偏移量为2^{32}。

与先前研究类似，我们旨在通过检查受害分支的所有可能位置[8]来确定推测装置的地址。[31]他们会在攻击者场景中训练一个分支B_i，该分支位于某个假设的受害者位置V_i，触发目标执行并测量对B_i的预测是否被驱逐。这种方法在我们的案例中不适用，主要基于两个原因。首先，与英特尔处理器不同，AMD Zen 4和Zen 5的 BTB 不会在每次预测错误时覆盖目标，而是将新目标插入ITA（指令传输区），同时将旧目标保留在 BTB 中。其次，我们需要单独检查多达约2^{32}个位置，这将耗费难以承受的时间。

我们通过反转攻击者与受害者的角色来优化上述方法，该思路源自先前研究[7]。给定假设的受害分支位置V_i，我们即可计算出对应的靶地址T_i。具体操作流程为：首先触发受害方训练分支预测器，随后在攻击者域的V_i位置执行匹配分支操作，目标为非信号目标。通过在攻击者域的假设受害目标位置T_i额外部署信号装置，即可检测碰撞事件。采用新方法时，仅需执行一次受害分支操作并进行一次信号重载缓冲区重载，即可验证多个潜在受害位置（例如1024个位置）。一旦检测到信号，即可确定哪个位置是正确的。

结果。虽然这种方法能让我们在Zen4上定位到单一受害者——直接破解代码 ASLR，但在Zen5上却会得到大量候选对象。由于所有候选者共享相同的24位低位，我们得出结论：这是由于混叠效应以及 BTB 提供部分目标值（类似英特尔[8]）所致。[17]虽然这种方法能揭示目标分支的低24位信息，但仅能部分解密代码 ASLR——该 ASLR 对页偏移位以上的32位进行随机化处理。我们发现，当仅使用匹配的低24位训练预测器时，其输出结果也仅能提供目标的低24位信息。因此，仅凭部分解密的代码 ASLR，我们只能推测目标分支与目标地址之间存在24位偏移范围内的关系。然而，我们选定的披露装置链距离目标分支较远（超过2^{24}个单位），这要求我们必须完全破解代码 ASLR。

定位重装缓冲区。在我们能够继续完全破解代码 ASLR 之前，需要确定 QEMU 映射攻击者虚拟机所用内存以支持重装缓冲区的虚拟地址。重装缓冲区最终也需要通过flush+reload泄露任意数据。与先前工作[2]类似，我们采用暴力攻击法定位重装缓冲区。部分去随机化通过分析先前数据，我们推测目标地址可能位于受害者分支指令24位偏移范围内。在此区域中，我们发现了一个加载攻击者指定地址内存的装置。值得注意的是，正如先前观察到的，Linux和 QEMU 系统会协作将虚拟机页面映射为2MB大页。因此，通过将重载缓冲区映射为2MB大页，我们能大幅减少潜在重载缓冲区的位置数量。此外，由于在推测窗口中仅需单次加载即可完成重载缓冲区搜索，这意味着我们需要延长该步骤的推测窗口。

结果。经过100次重复测试，我们能够以100%的成功率对Zen 5和Zen 4的重装缓冲区位置进行去随机化处理，其中Zen 5的中位耗时为46秒，Zen 4为83秒。

完全破解代码 ASLR。定位到重载缓冲区后，我们即可彻底解随机化代码 ASLR 并定位推测获取函数。由于 ASLR 仅对可执行文件的基地址进行随机化处理而保留偏移量，这同时也揭示了泄露装置的位置。Zen4无需这额外步骤，因为 BTB 未出现混叠现象，可直接获取目标位置。

在目标地址分支处，寄存器*RAX*存储着指向某个代码段的指针。掌握该数值后，我们就能计算出完整的分支地址。通过在24位偏移量中植入一个特殊装置，可以泄露这个指针。该装置会将攻击者控制的寄存器*RDX*与*RAX*相加，并将结果地址加载到目标地址。我们通过暴力测试*RDX*的取值，使*RAX*的值泄露，从而在重载缓冲区中记录命中地址。

结果。在Zen5处理器上，完全破解代码 ASLR 耗时中位数12秒，准确率达到75%，不考虑定位重载缓冲区所需时间。而在Zen4处理器上，由于第一步已能定位目标分支，破解 ASLR 耗时238秒，准确率降至70%。时间差异源于代码 ASLR 破解的第一步：在Zen5架构中，我们只需从2^{12}种可能性中筛选出单一候选地址；而在Zen4架构中，需要检查全部2^{32}个可能位置才能找到正确地址。Zen4架构在第一步需要更大的搜索空间，其耗时远超Zen5架构所需的第二步——后者通过泄露剩余20位 ASLR 熵来实现破解。

8.3. 任意内存泄漏

我们现已具备构建任意内存泄漏攻击所需的所有组件：漏洞利用链、反动态链接随机化（ASLR）破解原语，以及提供足够大推测窗口的可靠方法。

我们的vmscape攻击流程如下：首先，通过 ASLR 破解技术定位目标分支。确定gadgets后，即可确定 QEMU 的重载缓冲虚拟地址。接着构建LLC驱逐集，通过测试哪些驱逐集能使依赖加载操作退出缓存跟踪来确定正确集合。最后利用长推测窗口gadgets，通过 QEMU 内部数据结构的泄露实现攻击。

获取秘密数据的地址后，随即泄露该数据。

结果。针对Zen5的端到端攻击，我们测得中位攻击时长为102秒，成功泄露4KiB QEMU 机密对象（以磁盘加密解密密钥为例）。攻击总耗时中，58秒用于定位目标分支和重载缓冲区。获取目标后，攻击程序以154B/s的速率泄露任意 QEMU 内存，字节准确率达99.85%。整体攻击成功率达68%。

9. 讨论

我们讨论了vmscape攻击的NUMA注意事项（第9.1节）、我们在最初负责任披露后发现的其他vBTI变体（第9.2节）以及vBTI原语的缓解策略（第9.3节）。

9.1. NUMA注意事项

我们注意到，vmscape无法在采用NUMA的Zen 5系统上泄漏某些内存页。追踪该问题后发现，Linux NUMA平衡机制[32]会定期标记不存在的页面，以跟踪这些页面是否被同一NUMA节点中的CPU核心最频繁访问。被标记为不存在的页面无法在推测阶段被访问——这实际上会阻止vmscape对虚拟机管理程序不常访问的内存页面进行访问。不过，云虚拟化平台和提供商通常会在同一NUMA节点上为虚拟机分配物理内存以提升性能[33]。[34], [35]在实际应用场景中，Linux内核建议关闭NUMA平衡功能以降低不必要的跟踪开销[32]。

9.2. BHI 变体

英特尔对经典幽灵 BTI [1]的应对措施是eIBRS，这是一种硬件缓解方案，能有效隔离用户和管理员 BTB 中的条目。*分支历史注入*（BHI）[3]却通过利用用户与管理员域间缺乏 BHB 隔离的漏洞，重新激活了跨权限幽灵攻击。如我们在第5节所示，eIBRS在虚拟化环境中成功实现了 BTB 隔离。然而，与 BHI 类似，eIBRS本身并不能完全阻止客户机对分支历史的（部分）控制。虽然针对历史存储[5]的攻击已得到缓解，[36]我们相信，对于HU没有缓解措施。虽然我们没有验证这一点，但英特尔已经证实了这一点，这表明最近的英特尔cpu可能会受到*基于虚拟化的幽灵 BHI*（vBHI）原语的攻击。

9.3. 缓解

如第5节所述，我们的逆向工程揭示了近期微架构中存在多种vBTI原语。本节将探讨相应的缓解策略。

防止此类原始操作，从 IBPB -on- VMEXIT 开始，该操作是已部署vmscape补丁的基础。

IBPB -on- VMEXIT 。该缓解措施通过在每次 VMEXIT 时用 IBPB 刷新 BPU 状态，来保护$TM_{G \to H}$模型中的主机。因此，它在每次客户机到主机的转换中强制执行客户机与主机之间的 BTB 隔离。然而，基于 IBPB 的缓解措施通常会带来显著的性能开销[2]，[6]当应用于 VMEXIT 等快速路径时，这一问题会变得尤为棘手。

为量化这一开销，我们采用UnixBench测试套件[37]对缓解措施进行了基准测试，方法遵循先前研究[2]的方案。[6], [8]我们在配备4GB内存和双核处理器的虚拟机中以多线程模式运行工作负载。基于五次重复测试，性能开销的几何平均值在Zen 5架构上为57%。

Linux内核开发团队基于IBPBon- VMEXIT 提出缓解方案，但通过优化策略——仅在KVM退出后执行用户空间操作时触发 IBPB ——实现了性能优化。由于并非所有 VMEXIT 都会触发KVM退出并进入用户空间，该方案将 IBPB 移出快速路径，同时保持其安全特性。开发团队将此方案命名为“*用户空间退出前 IBPB*”。该补丁适用于所有受影响系统，包括AMD Zen 5处理器及近期发布的英特尔Lunar Lake和Granite Rapids等处理器。

除验证补丁有效性外，我们还在相同配置下对其性能开销进行了基准测试。UnixBench显示，Zen 4处理器仅产生1%的边际开销。但作为计算导向型基准测试，UnixBench导致用户空间的退出次数相对较少。为评估频繁导致用户空间退出的工作负载，我们采用fio[38]工具生成磁盘I/O测试，通过在虚拟机磁盘上随机读写10 GB数据重复10次。在Zen 4架构上，优化后的缓解方案开销为51%，接近 IBPB -on-KVM_EXIT方案的57%开销。当使用主机内核管理的设备（vhost）或直通设备时，用户空间退出现象显著减少。实验结果表明，缓解方案对性能的影响既取决于工作负载类型，也受 QEMU 配置影响，但在大多数实际应用场景中预计仍处于边际水平。

Retpoline。Retpoline 是一种经典的基于软件的 Spectre-eBTI 缓解方案，它通过使用不同的预测器[39]预测某些间接分支（jmp 和 call）为其他间接分支（ret）来实现替换。在未受到早期针对返回点攻击的系统上，使用 Retpoline 编译$Hypervisor_{user}$可以保护它们免受 $G \to H \mid Hypervisor_{user}$情景下的攻击者攻击[6]。虽然在考虑$G \to H$|主机进程的情况下未保护其他主机用户进程，但其性能开销较 IBPB -on- VMEXIT 方案更低。通过UnixBench工作负载对retpoline的性能评估显示开销为3.9%。根据威胁模型的不同，将retpoline与使用prctl的PR_SET_specULATION_CTRL进行选择性主机进程隔离相结合，可作为高开销 IBPB -on- VMEXIT 的替代策略。我们的研究表明

流行的开源Hypervisor$_{user}$部署均未采用retpoline[13][40], [41], [42]该缓解措施仅适用于不受幻影推测[43]影响的CPU，例如Zen 5。

保护 SEV -SNP客户机。虽然受vBTI原语影响的英特尔系统不支持 TDX，但Zen3至Zen5的 SVM 为 SEV -SNP 启用的客户机提供了多种可选保护措施[18]。[28](1)*分支目标缓冲隔离*机制可防止客户域外的执行操作影响客户域内的分支执行。(2)*入口间接分支预测屏障*迫使CPU在每次 VMRUN 时执行 IBPB 。(3)*SMT 保护*机制要求当客户域在另一分支 SMT 上运行时，其兄弟线程必须处于空闲状态。我们在第5节的实验表明，Zen 5架构默认已为 SEV -SNP客户域提供充分隔离。因此，此类缓解措施在早期CPU上同样适用。

保护 SMT 。STIBP 是一种硬件缓解措施，可隔离 SMT 核心的 BPU 。我们发现大多数系统默认启用 STIBP，以防止vBTI- SMT 原语。然而， STIBP 在某些CPU（如CoffeeLake）中并非*始终开启*。不支持 STIBP 的系统（如Zen1）只能完全禁用 SMT，这将导致显著的性能影响[2]。[44].

基于硬件的隔离。归根结底，vBTI原语的根本原因在于主机域与客户域之间缺乏 BPU 状态隔离。Zen5CPU引入特权域标签技术以区分用户域和管理员域。类似地，通过标签区分主机域与客户域可防止vBTI原语的产生，这一机制应被纳入Zen微架构的后续版本设计中。

10. 相关工作

关于微架构安全的研究已有数十年历史，其研究范围涵盖多个领域，最早可追溯至时序侧信道技术，尤其在缓存[45]方面。[46]Acıic ·梅兹等人[47], [48] 发表了类似的研究，但针对早期分支预测因子。Evtyushkin等[31] 后续研究证实，现代BTBs（生物靶向载体）同样可被用于 ASLR 。Ge等[49] 提出了一种通过共享资源和上下文对这类（以及其他许多）可利用时序侧信道的分类方法。

光谱。Kocher 等人[1] 首次提出了由分支预测错误引发的瞬态执行攻击。随后又出现了一系列其他瞬态执行攻击[50]。[51], [52],[53], [54], [55], [56], [57]Canella等[58] 我们根据攻击的保护对象将推测性执行攻击分为三类:虚拟机监控程序攻击、管理程序攻击和用户空间攻击。

(1) 虚拟机监控程序攻击。关于Spectre的原始研究[1]，以及后续研究[5]，[8]，针对虚拟机监控程序软件的修改后管理组件进行了概念验证攻击。谷歌研究人员基于Inception[6]，部分公开了针对 KVM 的攻击[59]。我们的工作，对比之下，这是首个真正的端到端攻击案例，恶意KVM 客体针对虚拟机监控程序的*用户空间*组件发起攻击。

(2) 监督员攻击。监督模式很早就成为 BTI 漏洞攻击的诱人目标[1]。当CPU厂商和操作系统开发者忙于寻找缓解措施时，研究人员仍在新防御中发现漏洞。维克纳和拉扎维通过推翻预测器训练的假设，在Retbleed[2]中绕过了retpoline，并在Phantom[43]和Inception[6]中绕过了硬件层面的防御。 BHI 和后续研究反复证明，尽管存在各种防范措施，仍可实现samemode训练[3]，[4], [5]打破交叉权限 BTI 缓解措施所建立的假设。

(3) 用户空间攻击。已有多种针对用户空间的攻击方法被提出，尤其是针对浏览器沙箱技术的攻击。Ragab等人[60] 广义机器清除作为推测性执行的来源被清除，并识别出浏览器中利用的机器清除新原因。Ret2spec[61]和Spectre Returns[21]都探索了*返回栈缓冲区*（RSB）预测在各种场景中攻击用户空间受害者。Spring[62]随后证明，尽管存在各种系统和浏览器层面的防护措施，RSB 预测仍可能被利用。近期攻击还针对其他预测器，以类似方式跨沙箱边界泄露机密[63][64]。Wikner和Razavi[7]率先展示了完整的端到端跨进程攻击。与以往针对用户空间的攻击不同，我们识别并利用了不仅跨越上下文边界，还能同时跨越权限和虚拟化边界的原语。

11. 结论

我们推出了vmscape，这是首个在 QEMU 上由恶意KVM 客户机发起的实际SpectreBTI攻击，可在AMDZen 5处理器上以154B/s的速率泄露内存。与以往由恶意客户机发起的Spectre- BTI 攻击不同，vmscape无需修改任何虚拟机监控程序代码。该攻击通过一种新颖的vBTI原语实现，该原语是通过系统性地探索不同AMD和Intel处理器在不同特权级别下客户机与主机之间分支预测器状态的隔离漏洞而发现的。缓解vmscape问题需要在VMEXIT 进入用户空间虚拟机管理程序后执行 IBPB，这会带来轻微的性能开销。

致谢

我们衷心感谢匿名评审专家和项目导师的宝贵意见。同时感谢英特尔 PSIRT 、AMD PSIRT 及Linux安全团队在本项目中的协调与合作。此外，我们还要感谢谷歌的AlexandraSandulescu对缓解方案分析提出的宝贵建议。本研究由瑞士联邦教育、研究与创新部资助，合同编号MB22.00057（ERC-StG承诺）。

参考文献

[1] P. Kocher、J. Horn、A. Fogh、D. Genkin、D. Gruss、W. Haas、M. Hamburg、M. Lipp、S. Mangard、T. Prescher、M. Schwarz和Y. Yarom合著的《幽灵攻击：利用推测执行》一文，发表于2019年5月的*2019 IEEE安全与隐私研讨会（SP）*，第1-19页。

[2] J. Wikner和K. Razavi，《retbleed：带有返回指令的任意推测性代码执行》，收录于*第31届 USENIX 安全研讨会（USENIX 安全22）*，2022年，第3825–3842页。

[3] E.Barberis、P.Frigo、M.Muench、H.Bos和C.Giuffrida合著的《分支历史注入：硬件缓解措施对抗交叉特权幽灵-v2攻击的有效性》，收录于*第31届 USENIX 安全研讨会（USENIX 安全22）*，2022年，第971-988页。

[4] S. Wiebing、A.d.F.Tron、H.Bos和C.Giuffrida合著的《InSpectre Gadget：检测Cross-privilege Spectre v2的残余攻击面》，发表于2024年*第33届 USENIX 安全研讨会（USENIX Security 24）*，第577-594页。

[5] S. Wiebing和C. Giuffrida，《 Training Solo ：域隔离对抗Spectre-v2攻击的局限性》，收录于*2025 年IEEE 安全与隐私研讨会（SP）*。IEEE计算机学会，2025年5月，第3599–3616页。

[6] D.特鲁希略、J.维克纳和K.拉扎维，《Inception:通过瞬态执行训练揭示新攻击面》，收录于*第32届 USENIX 安全研讨会（USENIX 安全23）*，2023年，第7303-7320页。

[7] J. 维克纳与 K.Razavi，“打破障碍：后障碍幽灵攻击”，*S&P*，2025年5月。

[8] S. Rüege、J. Wikner 和 K. Razavi，《分支特权注入：通过利用分支预测器竞态条件破坏 Spectre v2 硬件缓解措施》，收录于*USENIX Security* 2025年8月，英特尔悬赏奖励，BlackHat USA 演讲。[在线]。可获取：Paper=https://comsec.ethz.ch/wp-content/files/bprc sec25.pdfURL=https://comsec.ethz.ch/bprc

[9] M.-M. Bazm、M. Lacoste、M. S¨udholt和J.-M. Menaud，《云计算基础设施中的隔离：新的安全挑战》，*《电信年鉴》*，第74卷，第3期，第197-209页，2019年4月。

[10] Advanced Micro Devices，Inc.，《AMD64 APM 第2卷：系统编程》，技术报告24593,2024年3月。

[11] Intel信任域扩展，技术报告

[12] “内核虚拟机”，https://linux-kvm.org/page/Main Page。

[13] QEMU :一个通用的开源机器模拟器和虚拟化程序，https://www.qemu.org//。

[14] 《Intel64 SDM 第3卷（3A、3B、3C和3D）：系统编程指南》，技术报告。

[15] “Proxmox:简化您的数据中心，” https://proxmox.com/en/。

[16] H. Yavarzadeh、A. Agarwal、M. Christman、C. Garman、D. Genkin、A. Kwong、D. Moghimi、D. Stefan、K. Taram 与 D. Tullsen合著的《Pathfinder：利用条件分支预测器的高分辨率控制流攻击》，收录于《第29届ACM国际编程语言与操作系统架构支持会议论文集》第3卷（*第29届ACM国际会议论文集，第3卷*），美国加州拉霍亚：ACM出版社，2024年4月，第770–784页。

[17] 李立、雅瓦扎德和塔尔森在《间接分支预测器攻击：高精度分支目标注入攻击》一文中指出，该文发表于2024年*第33届 USENIX 安全研讨会（USENIX 安全24）*，第2137-2154页。

[18] 超微半导体公司（AMD）发布的《AMD Zen5微架构软件优化指南》，技术报告编号58455,2024年8月。

[19] Y.Zhang与R.Sion合著的《 Speculative Execution 攻击与 Cloud Security 》收录于《2019年ACM SIGSAC 云计算安全研讨会论文集》（CCSW ‘19系列），该论文集由美国纽约州纽约市的计算机协会于2019年11月出版，第201页。

[20] Y. Yarom 和 K.Falkner在《flush+reload：一种高分辨率、低噪声的L3缓存侧信道攻击》一文中，发表于2014年*第23届 USENIX 安全研讨会（USENIX 安全14）*，第719-732页。

[21] E.M.科鲁耶、K.N.哈萨瓦内、C.宋和N.阿布-加扎勒，《幽灵归来！利用返回栈缓冲区的推测攻击》，收录于*第12届 USENIX 进攻技术研讨会（WOOT 18）*，2018年。

[22] 《间接分支限制性投机》技术报告

[23] 超微半导体公司（Advanced Micro Devices，Inc.）发布的《Amd 64技术间接分支控制扩展》技术报告，2018年7月。

[24] 单线程间接分支预测器，技术报告

[25] “向 QEMU 提供秘密数据，” https://www.qemu.org/docs/master/system/secrets.html。

[26] H. Wang、M. Tang、Q. Wang、K. Xu 和 Y. Zhang，《ZenLeak：针对AMD Zen处理器的实用级缓存侧信道攻击》，2025年。

[27] 超微半导体公司（AMD）发布的《AMD64 APM 第三卷：通用指令与系统指令》技术报告（编号24594,2024年3月）。

[28] ——，《AMD Zen4 微架构软件优化指南》，技术报告57647,2023年1月。

[29] “Speculative Return Stack Overflow（SRSO）— Linux 内核文档，” https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/srso.html。

[30] 超微半导体公司（Advanced Micro Devices，Inc.）发布的《缓解分支类型混淆的技术指南》，技术报告。

[31] D.Evtyushkin、D.Ponomarev 和 N.Abu-Ghazaleh 合著的《跳过 ASLR：攻击分支预测器以绕过 ASLR》，发表于*2016年第49届 IEEE/ACM国际微架构研讨会*（2016年10月），第1-13页。

[32] Linux内核文档。Linux内核文档。[在线]。可用地址：https://docs.kernel.org/admin-guide/sysctl/kernel.html#numa-balancing

[33] 配置NUMA感知虚拟机——Google分布式云（仅限软件）用于裸机。Google Cloud。[在线]。可用：https://cloud.google.com/kubernetes-engine/distributed-cloud/bare-metal/docs/vm-runtime/numa

[34] ESXi NUMA调度的工作原理。[在线]。可用地址：https://techdocs.broadcom.com/us/en/vmware-cis/vsphere/vsphere/8-0/vsphere-resource-management-8-0/using-numa-systems-with-esxi/how-esxi-numa-scheduling-works.html

[35] NUMA - Proxmox VE。[在线]。可用网址：https://pve.proxmox.com/wiki/NUMA

[36] 分支历史注入与模式内分支目标注入 https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/branch-history-injection.html。

[37] “UnixBench 测试套件”，https://github.com/kdlucas/byte-unixbench。

[38] “灵活I/O测试器”，https://github.com/kdlucas/byte-unixbench。

[39] Retpoline：一种用于防止分支靶向注射的软件构建体，https://support.google.com/faqs/answer/7625886。

[40] “VirtualBox:强大的开源虚拟化个人和企业使用，” https://www.virtualbox.org/。

[41] Firecracker：专为无服务器计算打造的安全高效微虚拟机

[42] Crosvm：ChromeOS虚拟机监控程序

[43] J. Wikner、D. Trujillo和K. Razavi在《幻影：利用解码器可检测的误预测》一文中指出，该研究发表于2023年10月在加拿大安大略省多伦多市举行的第56届IEEE/ACM国际微架构年会（*56th Annual IEEE/ACM International Symposium on*

Microarchitecture），论文编号49-61。

[44] P. 米肖与 A.塞兹内克《全球历史动态分支预测的综合研究》，INRIA 报告，2001年。

[45] W.-M. Hu，“Lattice scheduling and covert channels，” in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*，1992，pp. 52–61.

[46] P. C. Kocher，“针对Diffie-Hellman、RSA、DSS等系统的时序攻击，” 见*Advances in Cryptology — crypto ’96*，N. Koblitz，Ed. Berlin，Heidelberg: Springer Berlin Heidelberg，1996，pp. 104–113.

[47] O. Acıic·梅兹、C·K·科克和J.-P·塞弗特合著的《通过分支预测预测密钥》，收录于M.阿贝主编的《密码学专题——CT-RSA 2007》（柏林/海德堡：斯普林格出版社，2006年），第225-242页。

[48] O. Aciic¸mez、c. K. Koc¸与J.-P. Seifert合著的《关于简单分支预测分析的效能》，收录于*第二届会议论文集 ACM信息、计算机与通信安全研讨会*，第12届ACIACS ‘07系列会议。美国纽约州纽约市：美国计算机协会，2007年，第312–320页。[在线]。可获取于：https://doi.org/10.1145/1229285.1266999

[49] Q. Ge、Y. Yarom、D. Cock 和 G. Heiser，《当代硬件中微架构时序攻击与对策综述》，2018年。

[50] M. Lipp，M.Schwarz，D. Gruss，T. Prescher，W. Haas，A. Fogh，J. Horn，S. Mangard，P. Kocher，D. Genkin，Y. Yarom，和 M. Hamburg，“Meltdown: 从用户空间读取内核内存，” 在*第27届 USENIX 安全研讨会（USENIX Security18）*，2018 年，第973-990页。

[51] J.V.Bulck、M.Minkin、O.Weisse、D.Genkin、B.Kasikci、F.Piessens、M.Silberstein、T.F.Wenisch、Y.Yarom和R.Strackx合著的《Foreshadow：通过瞬态乱序执行提取Intel SGX 王国的密钥》，收录于*第27届 USENIX 安全研讨会（USENIX 安全18）*，2018年，第991页。

[52] V. Kiriansky和C. Waldspurger，“推测性缓冲区溢出：攻击与防御”，2018年。[在线]。可获取：https://arxiv.org/abs/1807.03757

[53] J. Stecklina 与 T. Prescher，《Lazyfp：利用微架构侧信道泄露浮点运算单元（fpu）寄存器状态》，2018年。[在线]。可获取于：https://arxiv.org/abs/1806.07480

[54] D. Moghimi，“Downfall: Exploiting speculative data gathering，” in *32nd USENIX Security Symposium（USENIX Security 23）*. Anaheim，CA: USENIX Association，Aug. 2023，pp. 7179–7193. [Online]. Available:https://www.usenix.org/conference/usenixsecurity23/presentation/moghimi

[55] S.范沙伊克，A.米尔本，S.奥斯特伦德，P.弗里戈，G.梅苏拉泽 K. Razavi、H. Bos和C. Giuffrida，《RIDL：飞行数据异常负载》，载于*S&P*，2019年5月。

[56] C. 卡内拉，D. 根金，L. 吉纳，D. 格鲁斯，M. 利普，M. 明金，D. 莫吉米 *等人*，“Fallout: Leaking data on meltdown-resistant cpus”，收录于*ACM SIGSAC 计算机与通信安全会议论文集（中国化学会）*。ACM，2019年。

[57] M. Schwarz、M. Lipp、D. Moghimi、J. Van Bulck、J. Stecklina、T. Prescher 和 D. Gruss，《ZombieLoad：跨特权边界数据采样》，载于*中国化学会*，2019年。

[58] C. Canella、J. V. Bulck、M. Schwarz、M. Lipp、B. von Berg、P. Ortner、F. Piessens、D. Evtyushkin和D. Gruss，《瞬态执行攻击与防御的系统评估》，收录于*第28届 USENIX 安全研讨会（USENIX 安全19）*。美国加州圣克拉拉：USENIX 协会，2019年8月，第249–266页。[在线]。可获取地址：https://www.usenix.org/conference/usenixsecurity19/presentation/canella

[59] “谷歌CPU安全研究”，https://github.com/google/security-

[61] G. Maisuradze和C. Rossow在*2018年ACM计算机与通信安全 SIGSAC 会议论文集*（系列：中国化学会 ‘18）中发表了题为“ret 2spec: Speculative execution using return stack buffers”的论文。该论文由美国纽约州纽约市的计算机协会于2018年出版，页码未提供。2109–2122。[在线]。可获取于：https://doi.org/10.1145/3243734.3243761

[62] J.维克纳、C.朱弗里达、H.博斯和K.拉扎维，《Spring：Spectre漏洞在浏览器中重现，涉及推测性负载队列与深度栈》，载于*WOOT*，2022年。

[63] J. Kim、D. Genkin 和 Y. Yarom，《Slap：基于苹果硅芯片加载地址预测的数据推测攻击》，载于*S&P*，2025年。

[64] J. Kim、J. Chuang、D. Genkin和Y. Yarom合著的《Flop：通过错误的加载输出预测破解苹果M3 CPU》，发表于*USENIX 安全*期刊，2025年。

research/tree/master/pocs/cpus。

[60] H. Ragab、E. Barberis、H. Bos和C. Giuffrida，“对机器清除的愤怒：对机器清除及其对瞬态执行攻击影响的系统分析”，见*第30届*
USENIX 安全研讨会（USENIX 安全21）。USENIX 协会，2021年8月，第1451–1468页。[在线]。可获取：https://www.usenix.org/conference/usenixsecurity21/presentation/ragab

附录A。
Meta-综述

本元综述由2026年IEEE安全与隐私研讨会（S&P）项目委员会编制，作为论文征集公告中详述的评审流程组成部分。

A.1. 摘要

本文中，作者深入研究了x86处理器对Spectre- BTI漏洞的易感性，重点关注虚拟化边界。研究发现，此类边界隔离机制存在若干问题，导致主机与客户机之间、客户机与主机之间存在攻击面。论文针对不同CPU架构分析了该攻击面，并展示了针对Zen4处理器的攻击案例。

A.2. 科学贡献

- 通过有限前期研究对重要结果的独立验证
- 识别重大漏洞
- 在已确立的领域中迈出宝贵一步
- 其他的

A.3. 接受的理由

1) 本文首次分析了跨越虚拟化边界的Spectre- BTI 。研究结果对虚拟化技术普遍应用的云服务器领域具有重要影响，且所提出的攻击方式具有现实可行性。CPU厂商与操作系统开发者已确认存在这些漏洞，并正在开发补丁进行修复。
2) 本文提出针对Zen系列CPU的LLC驱逐新技术，该技术可进一步辅助分析此类CPU面临的其他侧信道攻击。
3) 本文提出了一种针对Zen4 CPU的端到端攻击方法，该处理器被广泛应用于云计算设备中。

A.4. 需要关注的问题

1) 尽管该论文对跨虚拟化边界及不同CPU的各类攻击进行了分析，但仅在客户机到主机用户场景及AMD Zen4 CPU上进行了攻击演示。这使得评审者对其他场景的直接适用性产生了一些疑虑。