doi:10.3772/j.issn.1002-0470.2020.05.004

AnC 和 Xlate 攻击防御研究①

李小馨②*** 侯 锐③* 孟 丹*

(*中国科学院信息工程研究所信息安全国家重点实验室 北京 100093)

(**中国科学院大学网络空间安全学院 北京 100049)

摘 要 分析了 AnC 和 Xlate 类由内存管理单元(MMU)和中央处理单元(CPU)核共享高速缓存(cache)导致的侧信道攻击,指出防御的关键在于隔离 cache 中的页表项和普通数据。在操作系统层面,首先基于页面属性表(PAT)不缓存(uncache)全部页表项,进一步结合透明大页,将平均性能损失由 82.35% 降至 26.95%。在芯片层面,首先在 uncache 全部页表项的基础上,改进了页表缓存(PTC)以缓存各级页表项,在 PTC 增大到 256 项时,平均性能损失为 1.59%;然后在 cache 中按路分区缓存页表项和普通数据(页表项占一路),平均性能损失为 6.61%;进一步探索了和各级页表项局部性相适应的混合隔离机制(高级页表项缓存在 PTC 中,最低级页表项缓存在分区 cache 中),在 PTC 大小为 64 项时,平均性能提升 0.81%。

关键词 高速缓存;内存管理单元(MMU);侧信道攻击;地址随机化;加密算法;页面属性表(PAT);分区

0 引 言

近年来高速缓存(cache)作为系统中所有进程共享的功能单元,受到了攻击者的普遍关注。利用 cache 侧信道攻击,攻击者打破了操作系统通过控制虚拟地址实现的不同进程间的访问隔离。自 cache 侧信道攻击出现至今十余年,攻击手段不断丰富,攻击平台由单核到多核至云平台[1,2],攻击架构由 x86[3,4] 至 ARM[5],攻击目标由加密密钥[3,4] 至地址随机化(address space layout randomization,ASLR)[6,7]。但一直以来的 cache 侧信道攻击利用的都是 CPU 核中不同进程对 cache 的共享。

最近有研究者利用内存管理单元(memory management unit, MMU)和 CPU 核对 cache 的共享实现了新型的 cache 侧信道攻击。AnC[8] 攻击中攻击者使用 JavaScript 控制 CPU 核的访存操作,探测出 MMU 对 cache 的访问踪迹,在 150 s 内破解了浏览器中的地址随机化。Xlate[9] 攻击中攻击者利用 MMU 的访存操作,绕过所有已有的软件侧信道防御措施,探测出 CPU 核对 cache 的访问踪迹,破解了基于 T-table 实现的 AES 的加密密钥。攻击者已在 Intel、AMD、ARM 等的 22 种机器上发现了这一侧信道的存在。

防御此类攻击的关键在于隔离 MMU 和 CPU 核对 cache 的访问。针对这类新的 cache 侧信道攻击,已有的基于隔离的防御手段[10-16] 已不适用。这些防御手段只隔离了来自 CPU 核的不同进程(数据或代码)对 cache 的访问,并没有考虑 MMU 对 cache 的访问。Stealthmem[13] 等通过置一页表项中的保留位来触发缺页中断(page fault)以实现页粒度的 cache 访问控制,但会产生 L1 终端故障(L1 terminal

① 优秀青年科学基金(Y710051102)资助项目。

② 女,1991 年生,博士生;研究方向:计算机体系结构安全;E-mail: lixiaoxin@iie.ac.cn

③ 通信作者,E-mail: hourui@iie.ac.cn

(收稿日期:2019-03-11)

fault,L1TF)[17,18]等新的侧信道泄露。

本文分析了 AnC 和 Xlate 类基于 MMU 和 CPU 核共享 cache 的侧信道攻击,指出防御的关键在于隔离 cache 中的页表项和普通数据,并分析了已有的基于隔离的防御措施的不足,研究探索了软件、硬件 2 种隔离方案。

(1)在软件层面,首先基于页面属性表设置所有页表项属性为不可缓存(uncacheable),隔离了页表项和 cache 中的普通数据;进一步结合透明大页,减少了旁路转换缓冲(translation lookaside buffer, TLB)的失效率,将平均性能损失由 82.35% 降至 26.95%。

(2)在硬件层面,首先实现了不缓存所有页表项的基本方案,并研究了随机地不缓存部分页表项的性能及安全性。进一步地,探索了在 cache 中隔离页表项和普通数据的防御方案,最后提出了与各级页表项局部性相适应的混合隔离方案,页表缓存(page table cache, PTC)大小为 64 项时,平均性能提升 0.81%。

1　研究背景

如图 1 所示,CPU 核访问内存数据时,首先向 cache 发送请求,cache 失效时再向内存发送请求,从内存中取回的数据会被缓存在各级 cache 中,以减少后续的内存访问次数,加速数据的再次访问。MMU 进行地址翻译时,首先查询 TLB,TLB 失效后向 PTW(page table walker)发送请求。若访问的页表项不在 PTC 中,PTW 进一步向 cache 发送请求。cache 失效后从内存中取回的页表项也会和普通数据一样缓存在各级 cache 中。对 cache 的共享,使得 MMU 和 CPU 核可以互相探测对方的访存操作在 cache 中留下的踪迹。当这一访存踪迹依赖于隐私数据时,攻击者便可获取隐私数据。

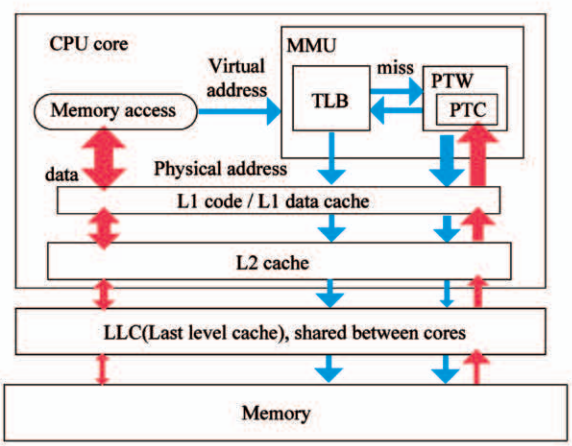


图 1　MMU 和 CPU 核共享 cache

1.1　AnC 攻击利用普通数据探测隐私页表项信息

现代系统中普遍采用多级页表的页表组织结构,虚拟地址可以分解为各级页表页及数据页的索引(如图 2 所示)。AnC 攻击利用 cache 侧信道获得各级页表项在 cache 中的位置,进而获得各级页表项在页表页中的索引,最终获得关键数据或代码的虚拟地址(即破解 ASLR)。

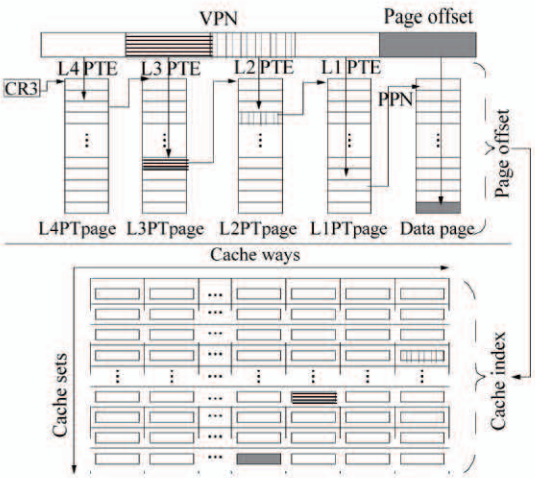


图 2　虚拟地址分解为各级页表项的 cache 索引

组相联 cache 的索引来自于被缓存数据的地址,cache 索引相同的数据会落入同一 cache 组(set)。每个 cache 组容量有限,当某个 cache 组已被占满时,缓存新的数据需要驱逐该 cache 组中缓存的旧数据。在 cache 侧信道攻击中,可以占满某个 cache 组,即驱逐该 cache 组中所有旧数据的数据集被称为该 cache 组的驱逐集(eviction set)。

MMU 和 CPU 核共享 cache 时,MMU 访问的页表项和 CPU 核访问的普通数据在 cache 中存在互替关系。AnC 攻击中攻击者通过控制其普通数据的页内偏移部分(即 cache 索引)为每个 cache 组构造驱逐集。攻击者使用驱逐集探测 MMU 访存操作在 cache 中踪迹(见图 3),其具体步骤如下。

(1)攻击者触发受害者访问隐私地址,TLB 失

效后,MMU 进行 PTW 将隐私虚拟地址对应的各级页表项加载到 cache 中。

(2)攻击者访问某个驱逐集。

(3)攻击者再次触发受害者访问隐私地址,并测量受害者访问时间。

若访问时间较长,则表明受害者访问的页表项已被攻击者的驱逐集驱逐出 cache,从而获得页表项对应的 cache 索引。

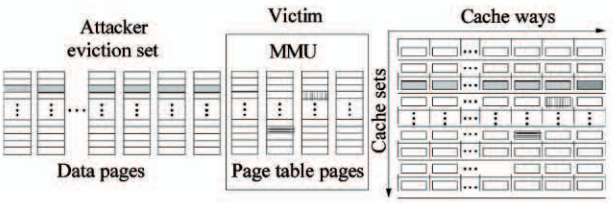


图 3 AnC 利用普通数据构建驱逐集探测 MMU 访存在 cache 中的踪迹

1.2 Xlate 攻击利用页表项探测隐私数据信息

基于 T-table 实现的 AES 加密算法,第一轮访问查找表的索引由密钥 k 和明文 p 异或得到。已知明文攻击通过探测加密算法访问的 T-table 的 cache 索引破解密钥。在传统破解 AES 密钥的侧信道攻击中,攻击者用普通数据构造驱逐集探测 AES 加密过程在 cache 中留下的访问 T-table 踪迹。现有防御策略已隔离攻击者的普通数据和受害者的普通数据对 cache 的访问,以阻止由于不同进程的普通数据对 cache 共享导致的侧信道泄露。但 MMU 对 cache 的访问一直未被考虑在内。Xlate 攻击利用页表项来构造驱逐集(如图 4 所示),绕过了已有的隔离防御机制。

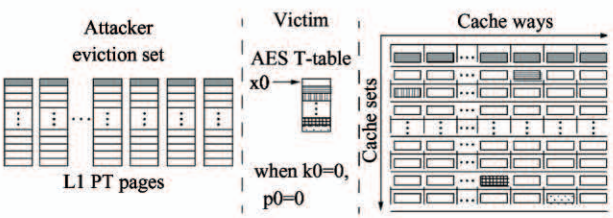


图 4 Xlate 攻击使用最低级页表项构建的驱逐集探测 AES T-table 的 cache 访问踪迹

Xlate 攻击使用驱逐集探测 AES T-table 访问在 cache 中踪迹的具体步骤如下。

(1)攻击者访问驱逐集。

(2)攻击者触发受害者执行 AES 加密算法,受害者访问的 T-table 会被加载到 cache 中。

(3)攻击者再次访问驱逐集,并测量访问驱逐集的时间。

若访问时间较长,则表明受害者访问的 T-table 项驱逐了驱逐集中的页表项,从而获得 T-table 对应的 cache 索引。

2 防御策略

从前文分析可见,AnC 和 Xlate 类攻击的根本在于 MMU 和 CPU 核对 cache 的无差别共享,防御的关键在于将 MMU 和 CPU 核对 cache 的访问隔离。

隔离包括多个层面。本文首先探索在软件层面的隔离。隔离 MMU 和 CPU 核对 cache 的访问,需要区分页表项和普通数据。页表由操作系统创建维护,用户程序不可见,最低级页表项指向的为普通数据页,其他级页表项指向的为页表页。Intel 提供了页面属性表(page attribute table, PAT)特性,可通过配置页表项,指定页表项所指向页面(包括页表页)的 cache 访问属性,且各级页表项可独立配置[19]。因而可以利用 PAT 为页表页和数据页配置不同的 cache 使用策略,达到隔离 MMU 和 CPU 核对 cache 访问的目的。基于 PAT 的不缓存策略会给 MMU 访存操作占比大且 TLB 失效率高的程序带来较大的性能损失。使用大页可以减少这类程序的页表项数量和页表级数,进而减少不缓存页表项策略的性能损失。

不缓存页表项策略的性能损失大。Page coloring[15]等其他软件 cache 访问隔离策略,虽然性能损失会较少,但无法完全防御 AnC 和 Xlate 类攻击。

(1)页着色(page coloring)利用物理页号和 cache 索引间存在的重叠位(即颜色位 color bits),将不同安全域的地址映射到不同的 cache 组。AnC 类的攻击可以实现基于 L1 cache 的攻击,page coloring 无法控制不存在 color bits 的 L1 cache。

(2)Intel 的缓存分配技术(cache allocation tech-

nology,CAT)[12]可以基于 cache 路隔离不同进程的数据,但没有区别对待同一进程的普通数据和页表项,且目前的 CAT 并未在 L1 cache 实现。

(3)StealthMem 等通过修改页表项中的保留位来触发 page fault 以实现页粒度的 cache 访问控制,会产生 L1 终端故障等新的侧信道漏洞。

本文进一步研究在芯片层面的隔离,探索性能损失更小的防御策略。芯片层面的隔离无需借助操作系统来区分页表项和普通数据,可以实现软件透明的防御。给 MMU 发出的 cache 访问请求打标签,可使 cache 区分来自 MMU 和 CPU 核的请求。对于页表项的访问请求,cache 失效后从下级 cache 或内存取回来的页表项不缓存在 cache 中直接返回给 MMU,为基本的不缓存策略。随机地不缓存部分页表项的策略可以减少性能损失,但缓存的页表项仍存在侧信道泄露。MMU 中的 PTC 也缓存页表项,将页表项只缓存在 PTC 中也可以实现页表项和普通数据对 cache 访问的隔离。cache 分区策略可以在共享 cache 中隔离页表项和普通数据。本文根据页表项访问请求占 cache 总访问请求的比例,评估页表项占一路的按路隔离策略。因各级页表项的 cache 局部性差异较大,本文最后探索和各级页表项局部性特征相适应的混合隔离策略。

3 软件防御机制

AnC 和 Xlate 类攻击的关键在于 MMU 和 CPU 核无差别地共享 cache,使得页表项和普通数据可以在 cache 中互相驱逐。本文研究了在现有系统下如何打破这种共享关系。

3.1 基本方案:基于 PAT 不缓存页表项

本文基于 Intel 提供的 PAT,为页表页和数据页配置不同的 cache 访问属性,隔离 MMU 和 CPU 核对 cache 的访问。

Intel 从 Pentium III 开始引入 PAT 特性,可以在虚拟地址空间以页为单位设置数据的访存类型(ARM 和 AMD 也提供了类似的特性)。每级页表项中都增加了 PWT、PCD 位,最低级页表项中还增加了 PAT 位,指向最高级页表页的 CR3 寄存器中也增加了 PWT、PCD 位,以设置其指向页面的访存类型。

本文通过设置寄存器 CR3 和指向页表页的页表项(除最低级页表项)中的 PCD 位和 PWT 位,将页表页的访存属性设置为 UC(uncacheable),普通数据页的访存属性设置为 WB(write-back),从而隔离了 MMU 和 CPU 核对 cache 的访问。

3.2 优化方案:结合透明大页减少性能损失

TLB 失效时,使用 4 级页表的系统完成地址翻译需要访问 4 次内存。不缓存页表项的机制会给 TLB 失效多的程序带来较大的性能损失。降低 TLB 失效率或减少每次 TLB 失效后的内存访问次数,可以减少 TLB 失效引起的内存访问总量。

大页增大了 TLB 覆盖的地址范围,从而减少 TLB 失效率。同时,大页对应的页表级数也减少,因而也减少了每次 TLB 失效后访问内存的次数。本文进一步结合系统提供的 2 M 透明大页,评估基于 PAT 的防御机制对于性能的影响。

4 硬件防御机制

软件防御机制可以方便地应用于现有系统,但可以实施的性能优化措施有限。本文进一步研究在芯片层面的隔离,实现性能损失更小的防御机制。

4.1 基本方案:不缓存全部页表项

为使 cache 可以区分来自 MMU 和 CPU 核的请求,本文在 PTW 发往 cache 的请求中增加标志位 PTW _ uncache,以通知 cache 不缓存该页表项。

操作系统创建页表项时会进行写页表项的访存操作,此类页表项访问请求由 CPU 核发起,会被正常缓存在 cache 中。对于写回型 cache,写操作的数据只有在数据被替换出 cache 时,才会被写入内存。若此类页表项未被写回内存,直接去内存查找 PTW 请求的页表项,会出现缺失错误。为避免该缺失错误发生,在本文的硬件不缓存页表项机制中,PTW 的访存操作仍会查找 cache。页表项只有被创建时会被缓存在 cache 中,被驱逐集替换出 cache 后,后续被 PTW 请求时,将不再被缓存在 cache 中。成功实施 AnC 或 Xlate 攻击都需要多次驱逐 cache,因而

页表项短暂地被缓存在 cache 中并不会影响安全性。

当 PTW 访存请求发生 cache 命中时,cache 采取和处理普通数据 cache 命中相同的策略,即将命中的页表项返回给 PTW。如图 5 所示,当 PTW 访存请求发生 cache 失效时,cache 将从内存取回的页表项直接返回给 PTW,不缓存在任一级 cache 中。

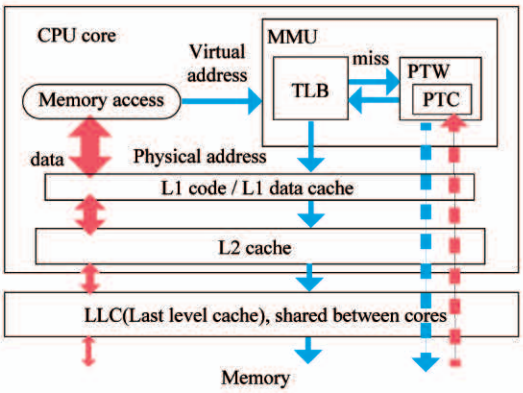


图 5　MMU 请求的数据不缓存在 cache 中

4.2　优化 1:随机不缓存页表项减少性能损失

为减少不缓存全部页表项的性能损失,本文进一步在 cache 中随机地缓存部分页表项。本文在 PTW 中增加一个访存请求计数器:PTW 每向 cache 发送一次请求,计数器都加 1,饱和后清零。当计数值为 0 时,PTW 将访存请求中的 PTW _ uncache 标志位清零,否则置 1(即不缓存在 cache 中)。

实验表明,随机地缓存 1/4 和 1/64 页表项在 cache 中均存在侧信道泄露,且缓存的页表项越多侧信道泄漏的风险越大。缓存 1/4 页表项时,3 级页表项的索引皆被破解;缓存 1/64 的页表项时,第 2 级页表项的索引被破解。因为 AnC 具有抗噪音的特点,难以找到完全不存在侧信道泄漏的随机不缓存方案。

4.3　优化 2:改进 MMU 中的 PTC 缓存各级页表项

系统中除了 L1-L3cache 外,MMU 中也有专用于缓存页表项的 PTC。普通数据无法驱逐 PTC 中的页表项,将页表项只缓存在 PTC 中也可以实现页表项和普通数据对 cache 访问的隔离。多级页表中最低级页表项数量最多,而现有的 PTC 未缓存最低级页表项(由 TLB 缓存)。为减少 TLB 失效引起的内存访问总量,本文改进 PTC 来缓存所有级别的页表项(含最低级页表项)。

4.4　优化 3:按路分区(way partitioning)cache 隔离页表项和普通数据

组相联 cache 的每个 cache 组都有多路(way)缓存行,将页表项和普通数据缓存到不同的 cache 路可以实现隔离。考虑到动态分区仍存在部分信息泄露[20,21],而且会使某一类数据强占 cache 影响其他类型数据对 cache 的使用,本文采用静态的按路分区机制。

根据 cache 访问请求中页表项访问请求所占的比例,本文采用为页表项分配一路的按路分区方案:页表项使用第 1 路,普通数据使用其他路。本文在 PTW 发往 cache 的访存请求以及 cache 的每个缓存行中添加标志位 isPTE,以区分是页表项还是普通数据。

按路分区只改变替换逻辑,不影响查找逻辑。考虑到本文并未对操作系统创建页表项时的写页表项进行标记,页表项和普通数据仍可以命中在任意 cache 路。发生 cache 失效时,PTW 请求的页表项只能被缓存在第 1 路缓存行,CPU 请求的普通数据只能被缓存在其他路缓存行;若上级(离 CPU 核更近)cache 中的数据被驱逐,isPTE 位为 1 的缓存行只能被缓存在下级 cache 中的第 1 路,isPTE 位为 0 的缓存行只能被缓存在下级 cache 中的其他路。如图 6 所示,按路分区后,页表项只缓存在 cache 的第 1 路,

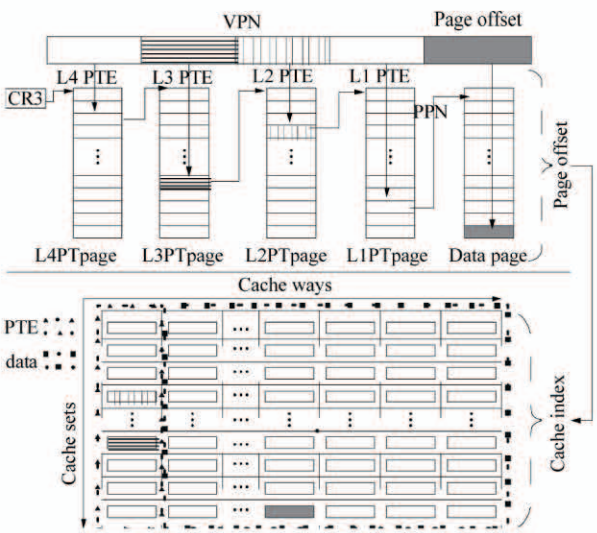


图 6　按路分区方案

普通数据只缓存在 cache 的其他路，页表项和普通数据在 cache 中不再有互替关系，无法互相探测对方的 cache 访问踪迹。

4.5 优化 4：与各级页表项局部性相适应的混合隔离方案

各级页表项的局部性不同，高级页表项局部性好于低级页表项。本节探索与各级页表项局部性特征相适应的隔离方案。本文评估了不同配置下各级页表项的失效率（从内存取各级页表项的次数/TLB 失效次数）。图 7 显示，高级页表项的局部性远好于低级页表项。

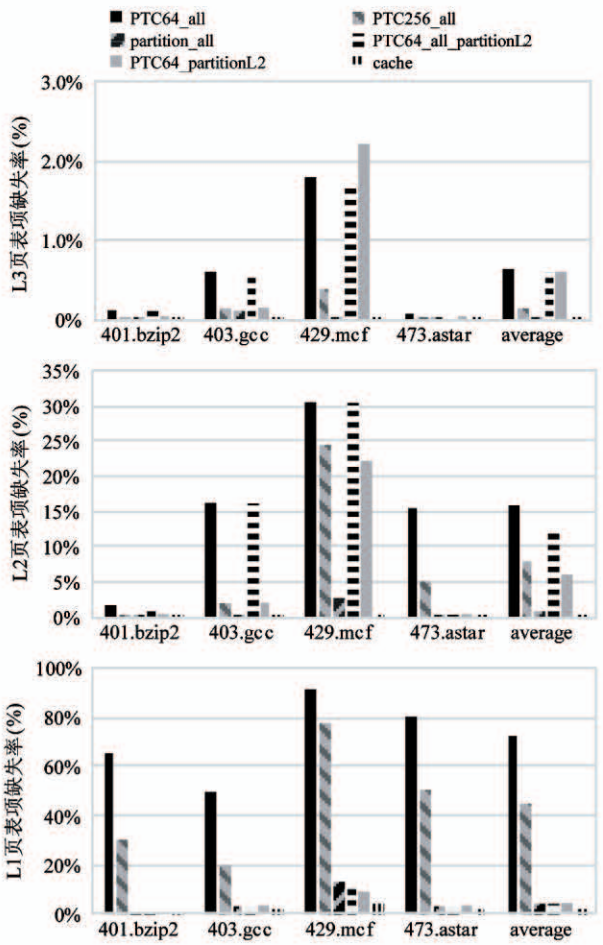


图 7 各级页表项失效率差异比较

只在 PTC 中缓存各级页表的方案中，PTC 为 64 项时，L3 页表项失效率已较低，平均为 0.65%，L2 页表项的平均失效率为 15.91%，L1 页表项平均失效率为 71.79%。PTC 增大至 256 项时，L2、L1 页表项失效率显著下降，L2 页表项平均失效率为 7.9%，但 L1 页表项的失效率仍较高，为 44.71%。采用按路分区时，各级页表项的失效率普遍较低，L3 页表项的平均失效率为 0.05%，L2 页表项的平均失效率为 0.78%，L1 页表项的平均失效率也仅为 4.99%。

为减少不同级页表项间的干扰，本文将只在 PTC 中缓存各级页表项的机制与按路分区 cache 的机制相结合。首先，在按路分区 cache 中只缓存 L1 页表项，PTC 中缓存各级页表项。PTC 为 64 项时，L3 页表项平均失效率为 0.59%，L2 页表项平均失效率为 12.09%，L1 页表项平均失效率为 4.43%。相较于在按路分区 cache 中缓存所有级页表项的方案，L1 页表项的失效率有一点下降，但 L2 页表项的失效率增加较大。为进一步减少 PTC 中 L1 页表项对其他级页表项的干扰，本文在 PTC 中只缓存 L3、L2 页表项，不缓存 L1 页表项，L2 平均失效降为 6.11%，L1 平均失效率为4.37%，L3 平均失效率为 0.6%。

5 性能分析

5.1 软件方案性能分析

软件方案实验环境配置如表 1 所示。使用大页的不缓存页表项方案相较于基本的不缓存页表项方案其性能损失（分别归一至使用 2 M 大页的正常缓存方案和未使用大页的正常缓存方案）由平均 82.35% 降至平均 26.95%。不缓存页表项的性能损失来源于 TLB 失效后的 PTW 访存操作。性能损失大小具体取决于正常缓存时 PTW 访存时间占程序总执行时间的比例、不缓存页表项后增加的 TLB 失效数量和平均 PTW 执行时间。

以 cactusADM 为例，未使用大页时，正常缓存机制中 PTW 访存时间占程序总执行时间的比例为 38.85%、每千条指令 TLB 失效数为 13、平均每次 PTW 执行时间为 23 个周期，不缓存页表项机制中每千条指令 TLB 失效数为 36、平均每次 PTW 执行时间为 187 个周期，计算得性能损失为 815.25%。使用大页时，正常缓存机制中 PTW 访存时间占程序总执行时间的比例为 11.38%、每千条指令 TLB 失

效数为7、平均每次 PTW 执行时间为 13 个周期,不缓存页表项机制中每千条指令 TLB 失效数为 16、平均每次 PTW 执行时间为 113 个周期,计算得性能损失为 211.41%。

表 1　实验环境配置

CPU	Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz
cache	8-way 32 kB L1 cache, 8-way 256 kB L2 cache, 20-way 6 M LLC
TLB	32 项 4-way L1 DTLB(2 M 页或 4 M 页) 64 项 4-way L1 DTLB(4 k 页) 8 项全相联 L1 ITLB(2 M 页或 4 M 页) 128 项 4-way L1 ITLB(4 k 页) 512 项 4-way L2 TLB(4 k 页)
内核版本	3.19.8
测试程序	SPECCPU 2006 (reference)
编译器	gcc4.8.4
编译选项	-O2

5.2　硬件方案性能分析

硬件方案的实验环境配置如表 2 所示。如图 9 所示,高级页表项缓存在 PTC、最低级页表项缓存在按路分区 cache 的混合隔离方案中,PTC 大小为 64 项时,平均性能提升 0.67%,其中 mcf 的性能损失最大,为 5.85%。

本文实验用到的 PTC 为全相联结构,达到的性能提升是理想化的最大值。采用组相联结构后,需

表 2　实验环境配置

FPGA	VC709
指令集	RISCV
CPU	Rocket
流水线	5 级
cache	8-way 32 kB L1 cache, 16-way 1 MB L2 cache,
TLB	16 项全相联 ITLB,DTLB
PTC	3 项

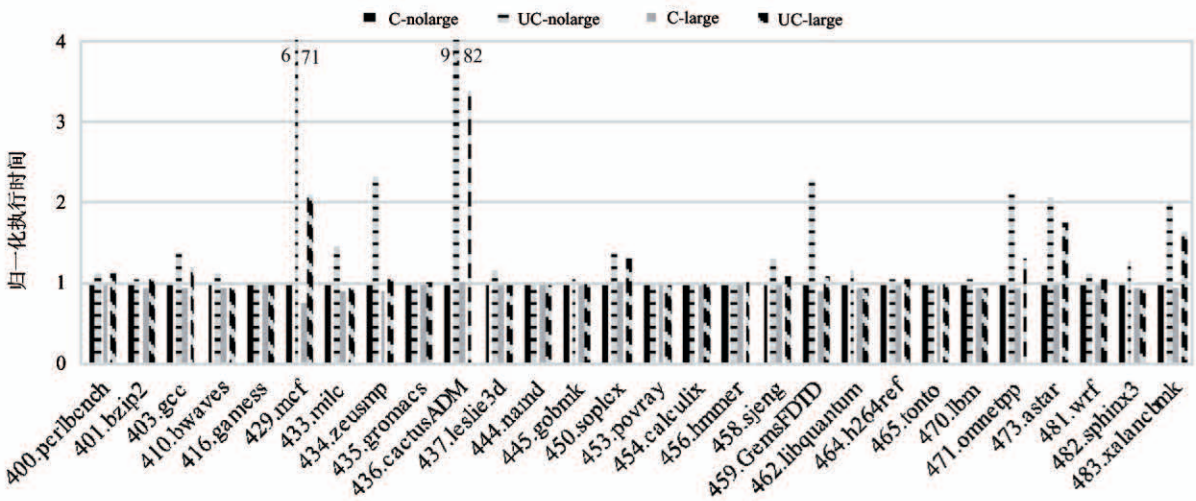


图 8　软件方案执行时间(归一到未使用大页时的正常缓存方案)

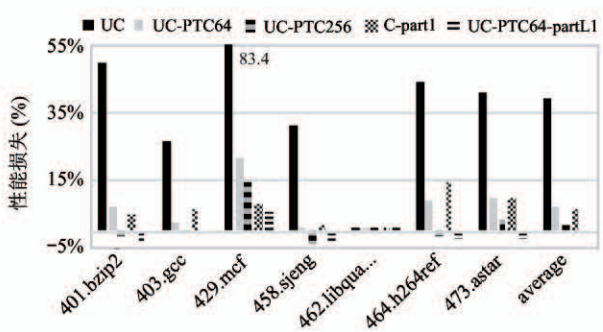


图 9　硬件方案性能损失

进一步增大 PTC 才能达到同等的性能提升效果。如图 10 所示,PTC 大小同样的情况下增加 PTC 的路数的效果要好于增加 PTC 的组数。

6　安全性分析

AnC 和 Xlate 类攻击都依赖于页表项和普通数据在 cache 中的互替关系,隔离的防御机制打破了这种关系,使得页表项和普通数据无法探测对方的

访存操作在 cache 中留下的踪迹。

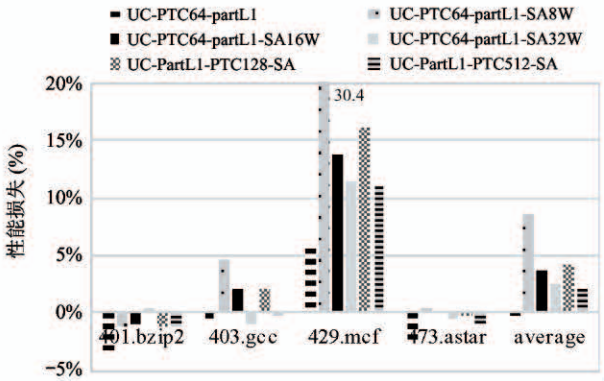
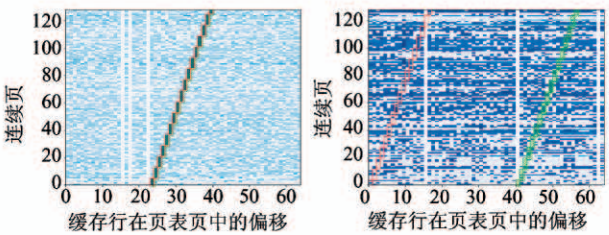


图 10　组相联 PTC 性能损失

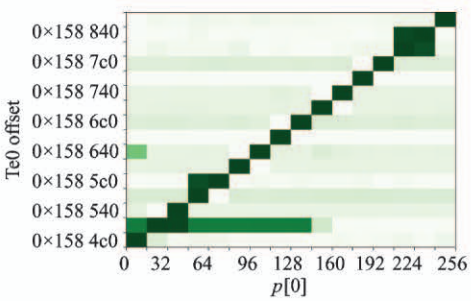
图 11 和图 12 分别为软件平台上 AnC 和 Xlate 攻击防御前后的攻击结果(硬件防御机制中实验结果类似,不再赘述)。



颜色越深代表访问延迟越大,猜测结果与实际结果重叠代表攻击成功。

图 11　防御前后 AnC 攻击结果(以最低级页表项为例)

实施防御前,隐私地址对应的页表项访问延迟较其他地址对应的页表项访问延迟大。AnC 攻击中攻击者可以观测到该时间差,获得隐私页表项的页内索引。实施防御后,所有地址访问延迟均增加,攻击者无法观察到 MMU 的访存踪迹。



注:以 $k[0]=0$ 为例,对于 $p[0]$ 从 0 到 256,访问的 T0 从缓存行 0 到缓存行 15,出现如图所示的阶梯状深色部分代表攻击成功

图 12　防御前 Xlate 攻击结果

实施防御前,Xlate 攻击者可以用页表项构造驱逐集,探测 AES 加密过程访问的 T-table 在 cache 中的位置。实施防御后,页表项未被缓存在 cache 中,Xlate 攻击者无法使用页表项创建驱逐集,后续无法使用驱逐集进行 cache 访问踪迹的探测,因而未生成驱逐集探测结果图。

7　相关工作

目前基于隔离的 cache 侧信道防御措施可以分为 3 类:按组分区、按路分区和将敏感数据锁存。

按组分区　Page coloring 通过控制 cache 索引实现 cache 按组分区。如图 13 所示,物理页号和 cache 索引存在重叠部分为 color bits。Page coloring 为不同安全域的数据分配不同的 color bits,不同安全域的数据便缓存在不同的 cache 组,不存在互替关系。

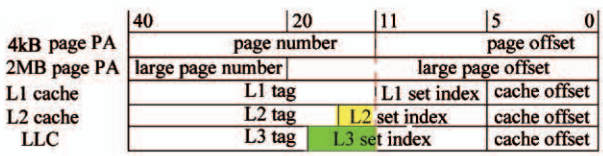


图 13　Page coloring 工作机制

按路分区　IntelCAT 将 cache 按路分区,操作系统通过控制位掩码将 cache 路与 CLOS(class of service)绑定,再进一步将 CLOS 与 CPU 核绑定以实现不同进程对 cache 的隔离访问。

NoMocache[21] 针对具有 2 个硬件线程的处理器,为每个线程静态分配 Y 路 cache,其他路为 2 个线程共享,每个线程都不能独占 cache。

利用硬件机制将敏感数据锁存在 cache 中　Intel TSX(事务内存)[10] 使用时涉及到 2 个集合:写集合和读集合。当写集合超过 L1 Dcache 大小时或读集合超过 LLC 大小时,事务终止率会急剧增加。Cloak[10] 使用 TSX 访问敏感数据,监测 TSX 的事务终止率,将敏感写数据锁在 L1 Dcache 中或敏感读数据锁在 LLC 中。

ARM Autolock[11] 机制具有锁住私有 cache 数据的特性:当核间共享 cache 中的缓存行在核私有

cache 中留有备份时,该缓存行不可以被从共享 cache 驱逐。利用该特性可以阻止基于共享 LLC 的跨核攻击。

PLcache[22] 为每个缓存添加 L 标志位,被标记为 Lock 的缓存行只能被同进程的 Lock 行替换出 cache,未标记为 Lock 的缓存行可以被任意缓存行替换。

利用软件机制将敏感数据锁存在 cache 中。映射到同一 cache 组的物理地址在 cache 中具有互替关系,StealthMem[13] 和 CacheBar[16] 通过限制各进程可占有的 cache 路数,阻止不同安全域数据的互相驱逐。StealthMem 和 CacheBar 跟踪不同安全域数据的物理页访问踪迹(将这些页对应页表项中的保留位置 1 以引起 page fault,监控对这些物理页的每一次访问),当非安全域占有的 cache 路数超过阈值时,重新加载安全域的数据至 cache,达到将安全域数据锁在 cache 中的目的。

已有防御机制都未考虑 MMU 对 cache 的访问。即使考虑了页表项,上述这些防御机制也不能完全防御该类攻击。MMU 和 CPU 共享 cache 引起的攻击包括核内攻击和跨核攻击。Page coloring 机制只适用于存在 color bits 的情况,从而无法防御基于 L1 cache 的核内攻击,且 Page coloring 也无法在大页开启时使用。IntelCAT 只在 LLC 或 L2 cache 中实现,也无法防御核内攻击。基于 ARM Autolock 的机制同样无法防御核内攻击。NoMocache 中共享部分的存在,仍会造成部分信息泄露。在基于 TSX 的防御机制中,受保护的进程其敏感数据大小需和 L1 Dcache 或 LLC 大小吻合。PLcache 需借助软件标记保护数据,不能实现软件透明的防御。StealthMem 和 CacheBar 等通过置一页表项中的保留位来触发 page fault 的方式会引入新的侧信道漏洞,例如 L1 终端故障。

8 结 论

AnC 和 Xlate 类基于 MMU 和 CPU 核共享 cache 的攻击防御关键在于隔离页表项和普通数据,已有隔离防御机制并不能完全防御此类攻击。本文提出的隔离机制包含软件和硬件 2 个层面。软件隔离机制可以方便地应用于现有系统,但性能损失大。在硬件层面可以实现更细粒度的隔离方案,探索性能更优的防御机制。

在软件层面,本文基于 PAT 不缓存页表项,进一步结合透明大页,减少了 TLB 失效率,平均性能损失由 82.35% 降至 26.95%。在硬件层面探索了缓存页表项的隔离机制,改进了 PTC 缓存各级页表项,只在 PTC 中缓存页表项;按路分区 cache,将页表项和普通数据缓存在不同 cache 路。基于各级页表项局部性存在差异的观察,本文提出了与各级页表项局部性相适应的混合隔离方案。下一步的工作是将访存特性不同的程序区别对待,探索适合不同程序的更细粒度的隔离机制。

参考文献

[1] 梁鑫, 桂小林, 戴慧珺, 等. 云环境中跨虚拟机的 Cache 侧信道攻击技术研究[J]. 计算机学报, 2017, 40(2): 317-336

[2] 李保琲, 李斌, 任望, 等. 面向可信云计算的资源安全管理机制研究[J]. 信息安全学报, 2018, 3(2): 76-86

[3] Liu F, Yarom Y, Ge Q, et al. Last-level cache side-channel attacks are practical[C]// Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, USA, 2015: 605-622

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

[5] Lipp M, Gruss D, Spreitzer R, et al. ARMageddon: cache attacks on mobile devices[C]// Proceedings of the 25th USENIX Security Symposium, Austin, USA, 2016: 549-564

[6] Gruss D, Maurice C, Fogh A, et al. Prefetch side-channel attacks: bypassing SMAP and kernel ASLR[C]// Proceedings of ACM Conference on Computer and Communications Security, Vienna, Austria, 2016: 368-379

[7] Hund R, Willems C, Holz T, et al. Practical timing side channel attacks against kernel space ASLR[C]// Proceedings of the 2013 IEEE Symposium on Security and Privacy, San Francisco, USA, 2013: 191-205

[8] Gras B, Razavi K, Bosman E, et al. ASLR on the line: practical cache attacks on the MMU[C]// Proceedings of the 24th Annual Network and Distributed System Security Symposium, San Diego, USA, 2017

[9] Van Schaik S, Giuffrida C, Bos H, et al. Malicious management unit: why stopping cache attacks in software is harder than you think[C]// Proceedings of the 27th USE-

NIX Security Symposium, Baltimore, USA, 2018: 937-954

[10] Gruss D, Lettner J, Schuster F, et al. Strong and efficient cache side-channel protection using hardware transactional memory[C]//Proceedings of the 26th USENIX Security Symposium, Vancouver, Canada, 2017: 217-233

[11] Green M, Rodrigues-Lima L, Zankl A, et al. AutoLock: why cache attacks on ARM are harder than you think[C]//Proceedings of the 26th USENIX Security Symposium, Vancouver, Canada, 2017:1075-1091

[12] Intel CAT. Improving real-time performance by utilizing cache allocation technology[R]. Santa Clara: Intel Corporation, 2015

[13] Kim T, Peinado M, Mainar-Ruiz G. STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud[C]//Proceedings of the 21th USENIX Security Symposium, Bellevue, USA, 2012: 189-204

[14] Liu F, Ge Q, Yarom Y, et al. Catalyst: defeating last-level cache side channel attacks in cloud computing[C]//Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture, Barcelona, Spain, 2016: 406-418

[15] Ye Y, West R, Cheng Z, et al. COLORIS: a dynamic cache partitioning system using page coloring[C]//Proceedings of International Conference on Parallel Architectures and Compilation, Edmonton, Canada, 2014: 381-392

[16] Zhou Z, Reiter M K, Zhang Y. A software approach to defeating side channels in last-level caches[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 2016: 871-882

[17] Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: extracting the keys to the Intel SGX kingdom with transient out-of-order execution[C]//Proceedings of the 27th USENIX Security Symposium , Baltimore, USA, 2018: 991-1008

[18] Weisse O, Van Bulck J, Minkin M, et al. Foreshadow-NG: breaking the virtual memory abstraction with transient out-of-order execution[EB/OL]. https://foreshadowattrack.eu:KU Leuven, 2018

[19] Intel 64 and IA-32 architectures software developer's manual. Volume 3A: system programming guide[R]. Santa Clara: Intel Corporation, 2016

[20] Kong J, Aciicmez O, Seifert J, et al. Deconstructing new cache designs for thwarting software cache-based side channel attacks[C]//Proceedings of the 2nd ACM Workshop on Computer Security Architecture, Alexandria, USA, 2008: 25-34

[21] Domnitser L, Jaleel A, Loew J, et al. Non-monopolizable caches: low-complexity mitigation of cache side channel attacks[J]. *ACM Transactions on Architecture and Code Optimization* (*TACO*), 2012, 8(4): 35

[22] Wang Z, Lee R B. New cache designs for thwarting software cache-based side channel attacks[J]. *ACM SIGARCH Computer Architecture News*, 2007, 35(2): 494-505

Research on AnC and Xlate attack defense

Li Xiaoxin[*,**], Hou Rui[*], Meng Dan[*]

([*]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

([**]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049)

Abstract

The side channel attack caused by the memory management unit (MMU) and the central processing unit (CPU) core shared cache is analyzed. The key of defense is to isolate the page table entries and normal data in the cache. At the operating system level, the basic mechanism is to uncache all page table entries utilizing page attribute table (PAT). The average performance overhead is reduced from 82.35% to 26.95% by further assisted with the transparent huge pages. At the chip level, uncaching all page table entries is the basic scheme. Futher the page table cache (PTC) are improved to also cache the L1 page table entries. When the PTC is increased to 256 entries, the average performance overhead is 1.59%. Way-partitioned caches cache page table entries and normal data in different way(page table entries occupy one way), the average performance overhead is 6.61%. Further the hybrid isolation mechanism adapting to the locality divergence of page table entries at different levels is explored: the high level page table entries are stored in the PTC, and the lowest-level page table entries are stored in the way-partitioned cache. When the PTC size is 64, the average performance is increased by 0.81%.

Key words: cache, memory management unit (MMU), side channel attack, address randomization, encryption algorithm, page attribute table (PAT), partition