高技术通讯 2024 年 第 34 卷 第 11 期:1127-1141

doi:10.3772/j.issn.1002-0470.2024.11.001

ARM 架构下硬件辅助的内存隔离机制综述①

许佳丽② 武成岗 王 喆③

(处理器芯片全国重点实验室,中国科学院计算技术研究所 北京 100190)

(中国科学院大学 北京 100090)

摘 要 内存隔离是一项缓解软件潜在安全危害、提高软件安全性和鲁棒性的重要防御机制。内存隔离机制保护不同组件中的敏感数据,强制敏感数据仅由组件信任的代码访问。虽然软件漏洞无法避免,但内存隔离机制可以在代码存在漏洞时依旧提供有效的敏感数据隔离保护,使得攻击者无法直接窃取或篡改被隔离的数据。出于性能考虑,依靠硬件辅助的内存隔离机制得到了广泛重视,该方向力求在提供有效安全保证的前提下,利用硬件大幅减少内存隔离机制给软件带来的额外开销。因此大量研究都着力于挖掘硬件潜力,从而支持高效、安全和完备的内存隔离机制。本文针对主流架构之一的 ARM,系统性地总结了近年 ARM 架构硬件辅助的内存隔离机制的相关研究,并进一步归纳出它们的研究现状和适用场景。最后,针对现有硬件辅助内存隔离机制面临的安全现状和新场景,展望了该领域未来的发展方向。

关键词 硬件辅助的内存隔离; ARM 架构; 域隔离; 地址隔离

最小特权原则是系统安全最基本的原则之一,其要求将软件划分成互相隔离的组件,每个组件只以完成其任务所需的最低权限运行,有效增强计算机安全性[1]。内存隔离机制遵循最小特权原则,将敏感数据隔离在单独的内存空间中,限制不同组件对隔离区域的访问权限,以阻止攻击者利用代码漏洞影响整体软件的安全。

软件无法避免漏洞的产生,究其原因,主要是:(1)软件功能随实际需求的变化而日趋复杂,更迭频繁、代码量日益增多的复杂逻辑中包含漏洞的可能性更高;(2)针对复杂软件的完全形式化验证面临种种挑战,漏洞和缺陷隐藏在代码中难以杜绝。开发者往往只能将部分被充分验证的代码视为可信代码,而其他代码都是可能存在漏洞风险的不可信代码。对于应用软件,不可信代码可能来自于第三方库或即时生成的本地码;对于系统软件,不可信代码可能是应用软件、驱动、扩展的伯克利包过滤器(extended Berkeley packet filter, eBPF)程序或没有充分验证的补丁代码。长久以来,攻击者都在挖掘和利用不可信代码中的漏洞以窃取和篡改敏感数据,危害软件安全。

为避免攻击者窃取和篡改软件中的敏感数据,内存隔离机制作为漏洞危害的缓解手段应运而生。代码和数据被划分成独立的组件后,内存隔离机制需要保证组件之间不能直接读写对方的内存空间,以此来保护组件自身数据的机密性和完整性[2-4]。内存隔离机制与内存管理系统紧密相关,该机制需要在虚拟内存或物理内存上划分组件区域,同时利用权限映射或地址范围检查的方式,确保组件只能访问合法的内存区域。内存隔离机制可以有效避免漏洞带来的危害。例如,著名漏洞 HeartBleed (CVE-2014-0160)[5-6]可被攻击者利用读取 OpenSSL(open

① 国家自然科学基金青年基金(61902374)和国家自然科学基金联合重点基金(U1736208)资助项目。

② 女,1994 年生,博士生;研究方向:系统安全,内存隔离;E-mail: xujiali@ict.ac.cn。

③ 通信作者,E-mail: wangzhe12@ict.ac.cn。

(收稿日期:2023-12-04)

secure sockets layer)中的隐私数据(如会话密钥)。ERIM 提供的内存隔离机制就可以限制仅 OpenSSL 部分可信代码能访问隐私数据,以防攻击者通过不可信代码的漏洞窃取隐私数据[4]。

内存隔离机制按照实现途径划分,可分为纯软件的隔离方式和硬件辅助的隔离方式。纯软件的内存隔离方案,如软件错误隔离(software fault isolation,SFI)[7-9],在访问指令前插桩,检查指令中指针范围的合法性,进而拒绝非法的内存访问。它的优势是不依赖任何底层硬件,通用性好。但是,插桩带来的运行开销和代码膨胀问题是无法忽视的。经过测试,即便是硬件 MPX(memory protection extensions)加速的 SFI,给 Nginx(engine x)的吞吐量带来的开销也高达 15% ~30%[4],而插桩代码让基准测试集 SPEC(standard performance evaluation corporation CPU benchmark package)的代码平均膨胀了 41.76%[10]。因此,目前主流的内存隔离方案是采用硬件辅助的方式,让内存检查内化在底层硬件中,减少显式的软件介入。一种常见的硬件辅助隔离为权级隔离,利用权级保证内核和用户进程之间的双向隔离。权级隔离通过在页表上标注虚拟页的特权和非特权属性,将用户空间和内核空间分开,使得用户代码在直接访问内核空间时会触发硬件异常;同时利用其他硬件(如 PAN(privileged access never)[11])使得内核代码也无法随意访问用户空间,防止内核的数据流被攻击者劫持到用户空间。

本文针对 ARM(advanced RISC machine)架构硬件辅助内存的隔离机制相关工作进行了系统性的整理。硬件辅助的内存隔离机制与其实现的架构关系密切:ARM 精简指令集架构因其性能强、可定制性好在手机和平板市场已经成为绝对主流;同时,依托苹果 M 系列芯片和国产芯片的崛起,ARM 架构在私人电脑端的市场份额增长迅猛,从 2020 年的 2% 增长到了 2022 年的 12%[12]。针对 ARM 平台的内存隔离机制因此成为研究热点之一。然而,目前暂时还没有针对 ARM 架构的内存隔离机制的完整分析与介绍,因此本文对 ARM 架构的相关研究进行了梳理和归纳,以填补这一空白。本文主要贡献如下。

(1)梳理了 ARM 架构硬件辅助内存隔离机制的相关工作以及目前业界中使用的内存隔离机制,归纳了它们的研究现状。

(2)对比不同内存隔离方案的各项参数,进而讨论它们各自的适用场景。

(3)总结了目前硬件辅助内存隔离机制的安全现状以及面临的挑战,分析了该领域未来的发展趋势。

1 背景知识

1.1 异常等级和权级切换

在 ARM 架构中,权级被称为异常等级(exception level,EL)。64 位的 ARM 架构 AArch64 下,权级范围从 EL0 到 EL3,数字越高代表权级越高。最低权级 EL0 被称为非特权级,运行用户进程;其余等级被称为特权级,分别运行着内核、虚拟机管理软件和固件。从低权级切换到高权级必须通过异常事件,常见的异常事件有缺页异常、系统调用、中断等。从高权级回到低权级,则需要执行异常返回指令 ERET、ERETAA 或 ERETAB。

1.2 虚拟内存系统架构及访问控制

AArch64 进程拥有 64 位的虚拟地址空间。最新版的 Linux 内核设置高 256 TB 为内核空间,低 256 TB 为用户空间,虚拟地址的有效位是 48 位。通常情况下,如果没有开启 MTE(memory tagging extension)[13] 和 PAC(pointer authentication)[14] 机制,虚拟地址高 16 位必须是全 0 或者全 1。页表结构记录从虚拟空间到物理空间的映射关系,及虚拟空间中每个页的属性和权限设置。页表以树结构组织,通常一次完整的虚拟地址翻译需要内存管理单元(memory management unit,MMU)经过四级页表查找。查找页表非常耗时,因此主流中央处理器(central processing unit,CPU)中都为 MMU 配备了 TLB(translation lookaside buffer)以缓存查找结果,加速虚实地址翻译过程。但因为 CPU 在多个进程之间共享,所以切换进程需要刷新 TLB 以确保映射关系的正确性。频繁刷新 TLB 会引入高开销,因此引入了地址空间标识符(address space identifier,ASID)

机制。TLB 的每一项都标记了 ASID,ASID 会与一个运行的进程绑定,这样使得进程在运行时仅能访问匹配自身 ASID 的 TLB 项。然而,因为内核页被所有进程共享,且被频繁访问,所以内核页的 TLB 项都带有 Global 标记,不受 ASID 限制。

ARM 的虚拟页访问权限包括读、写、执行以及特权级 4 种,由每个页表项的 AP(access permission bits)、UXN(unprivileged execute never)和 PXN(privileged execute never) 4 个域编码。两位宽的 AP 域有 4 种可能的编码,Linux 内核将 AP[2] 位作为只读/读写的标识位,将 AP[1] 作为特权/非权级的标识位。例如,当 AP 等于“11”时,代表该虚拟页为只读的非特权页。AP 域位于最后一级页表上,而剩余层级页表项上存在 APTable 域,可被用于覆盖 AP 的权限。UXN 和 PXN 位处于最后一级页表上,分别控制该页在非特权级下和特权级下是否具有可执行权限。类似地,剩余层级页表项上存在 UXNTable 和 PXNTable 域。

ARM 的 64 位指令集 AArch64 为内核空间和用户空间各自提供一个页表基地址寄存器(translation table base registers,TTBR),独立的页表机制使得内核空间和用户空间可被分开管控,互不干扰。地址翻译控制寄存器 (translation control register,TCR) 包含了多个控制域以控制 2 个空间的虚实地址翻译过程和最终结果。

1.3 可信执行环境的物理内存隔离

早期的计算机设计中,物理空间是全系统都共享的资源。但当可信的概念出现以后,运行在芯片之上的软件就有了可信和不可信之分。不可信部分不可访问可信部分的物理内存成为了可信执行环境概念下的关键之一。ARM 架构提供的隔离执行环境 TrustZone 会将物理内存分配给 2 个世界:NormalWorld 和 SecureWorld。Normal 模式下运行的 CPU 不能访问 SecureWorld 的物理内存,而 Secure 模式下运行的 CPU 能够访问所有的物理内存。ARMv9 新推出的可信计算架构(confidential compute architecture,CCA)提供了一个新的世界 RealmWorld,用于提供系统级可信执行环境,保护数据的机密性和完整性[15-16]。RealmWorld 与 NormalWorld、SecureWorld 是物理内存隔离的,需要依靠最高权级的 RootWorld 中的代码才可以修改上述 3 个 World 的物理内存划分布局。

1.4 ARM 虚拟化

虚拟化是让多个操作系统能共享同一套硬件的技术。在虚拟化技术中,每个虚拟机(virtual machine,VM)会包含一个完整的操作系统,而 Hypervisor 是用来管理这些虚拟机的软件。出于性能考虑,硬件辅助的虚拟化技术占据了主流,各大架构为 Hypervisor 专门提供了单独的高权级(如 ARM 的 EL2)运行,让 VM 运行在低特权级(如 ARM 的 EL1)中。VM 的代码可以直接在 CPU 上执行,而 Hypervisor 可以设置当 VM 执行特殊指令或者调用时自动下陷到 Hypervisor 中处理,以此监控和管理 VM。VM 中的虚拟地址访问需要映射到实际的物理地址上。这个过程在 ARM 架构中被称为两阶段内存翻译(two-stage memory translation),第 1 阶段需要先将虚拟地址(virtual address,VA)翻译成 VM 的中间物理地址 IPA(intermediate physical address),第 2 阶段再从 IPA 翻译成真正的物理地址(physical address,PA)。这两阶段会索引各自的页表基址寄存器。

2 内存隔离概述

2.1 内存隔离机制面临的攻击威胁模型

软件可以分为可信和不可信两部分代码。通常假设,可信代码已经被充分验证,不包含漏洞,且可访问软件中的敏感数据;而不可信代码包含可能被攻击者利用的漏洞。攻击通过发现和利用漏洞,获得访问任意地址数据的能力(任意地址读写原语)。攻击者还可以通过篡改代码指针、构造 ROP(return-oriented programming)链的方式进一步获得任意代码执行的能力(任意代码执行原语)。这 2 个原语被攻击者利用以试图访问、篡改和泄漏软件中的敏感数据。

内存隔离机制通过各式隔离手段,保证只有可信代码能够访问敏感数据,不可信代码或是不可设置指向敏感数据所在内存的指针,或是访问敏感数

据将会触发硬件异常,被内存隔离机制发现。不可信代码试图访问敏感数据则必须通过指定的接口跳转到可信代码中完成。内存隔离机制还需保证接口调用过程不会出现接口绕过或者接口滥用问题,因此部分研究中的内存隔离机制会与粗粒度控制流完整性机制配合,阻止攻击者使用任意代码执行原语绕过接口的攻击。

上文考虑的是软件层面的攻击,而硬件层面的攻击也能够窃取和篡改敏感数据,例如侧信道攻击[17-18]、Rowhammer[19]等。面向这些物理攻击的防御有充分的相关研究讨论[20-23],它们与内存隔离机制是正交的,可以与内存隔离机制配合完成敏感数据保护,但不在本文考虑的攻击威胁之内。

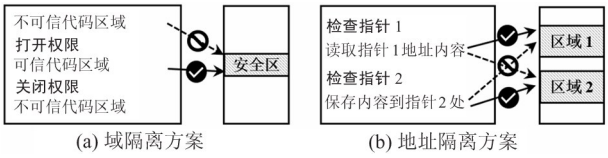
2.2 内存隔离机制及其分类

内存隔离的本质是使用硬/软件方法构造指令的访问差异。内存隔离机制方案可以大致分为 2 类:域隔离和地址隔离。二者方案不同,也有着各自的优劣势。

域隔离方案将内存划分为权限不同的域,并且将代码也进行了划分,不同代码能只能访问其对应域的内存。

图 1(a) 展示了域隔离方案的简要架构,可信代码区域执行前需要打开安全区域的访问权限,而可信代码执行完以后需要将该权限关闭。域隔离方案中通常需要考虑以下 3 部分内容:(1)内存划分设置。域的划分体现在代码和数据内存的划分上,不同区块的代码能够访问的内存空间域不同,而这通常需要利用权限差异控制代码对内存空间域的访问。(2)访问接口设计。为满足程序中正常的跨域访问请求,让一个域的代码能够将控制流合法地转移到另一个域中以访问数据,需要经过精心设计的访问进出接口。接口中包含域权限切换指令,以控制对应域的内存访问权限。域隔离方案必须保证域切换指令仅在接口中出现,这使得所有的域切换必须经过访问接口。(3)对接口的保护。当保证了仅有接口处存在域切换指令后,域隔离方案就要防止有任意地址执行原语的攻击者滥用访问接口让内存隔离机制失效。

地址隔离方案限制指针可访问的地址范围,方式是通过将范围信息记录下来,在指针访问内存时将访问地址与范围信息进行比对。图 1(b) 展示了地址隔离方案的简要工作示意图,在指针访问内存区域前一定有一条或多条地址范围检查指令,非法地址范围的指针访问会被拒绝。地址隔离方案需要考虑以下两点。(1)指针保护范围。程序中一般都会包含数据指针和代码指针,代码指针则又可细分为函数指针和栈返回地址指针。保护的指针越多,给程序带来的开销越高;不同的指针有着各自的特点,所需要的保护方案一般不同。现有研究会依据其具体方案的可适用性和性能开销综合考虑选择不同的范围。(2)地址范围记录方法。指针可访问的地址范围至少包含 2 个值(起始位置/大小或者上界/下界),而范围区间的记录方式也影响着最终指针的比对方式,进而影响最终的程序运行效率。通常的做法是将记录放置在指针未使用的高位区域,或者开辟一块专门的区间(bitmap)用于记录信息。



(a) 域隔离方案　　(b) 地址隔离方案

图 1　内存隔离机制的 2 类方案

两类隔离方案有其各自的优劣势。域隔离方案将代码做了划分,只需在进出域的时候进行权限切换,因此给程序添加的插桩代码少、代码膨胀率低,带来的开销相对于地址隔离方案要低;相对地,地址隔离方案需要在每个被保护的指针使用点插桩比对代码。但是,地址隔离方案可以使用字节粒度来细粒度管控每个指针的可访问区域;而域隔离方案需要按页大小设置权限和隔离域,因此保护粒度一般为页粒度。

3 现有基于硬件辅助的内存隔离研究

本节介绍 ARM 架构下的基于硬件辅助的内存隔离相关研究。研究中使用到的 ARM 硬件特性各不相同。为了便于阅读和理解,本节首先将使用到的硬件隔离机制按照“域隔离”和“地址隔离”方案

进行分类。而域隔离方案是 ARM 架构中的主流方案,大部分研究可归为此类。因此本文将域隔离方案进一步按照隔离机制的作用范围细分成 3 类,并总结了 4 类隔离方案,其中域隔离方案包括了“基于 MMU”、“基于物理空间”、“基于调试机制”3 类隔离方案。地址隔离则有“基于 MTE”的隔离方案。本节从这 4 类隔离方案展开叙述,并为复杂方案中绘制了隔离机制原理图表,以帮助读者理解该类方案的核心思想。

3.1 基于 MMU 的硬件辅助隔离方案

虚拟空间是进程的内存抽象模型。为了系统安全运行,虚拟空间管理机制需要保证进程和内核之间、进程之间、可信和不可信代码之间的隔离。MMU 转换虚实地址过程中包含了大量检查逻辑,如地址可翻译范围检查、权限检查,以保证这 3 类隔离。图 2 展示了 MMU 中从输入地址到输出地址需要经过的流程和检查。相当一部分的研究利用了图 2中阴影部分的硬件机制设计了内存隔离方案。下面将分段讲述如何依据这些硬件和 MMU 检查构建内存隔离机制。

3.1.1 利用页表控制属性构建隔离机制

可翻译内存映射。页表翻译控制寄存器(TCR)包含虚实地址翻译过程的控制域。TCR. TxSZ 用来控制虚拟空间的可访问的区间,任何访问未开放区域的操作均会触发翻译错误(translation fault)。开放空间大小计算公式为 $Memory\ Size = 2^{64-TxSZ}$。从公式中可以得到,TCR. TxSZ 每增加 1,可访问的内存大小就减少一半,如图 3所示。经过内存布局,可

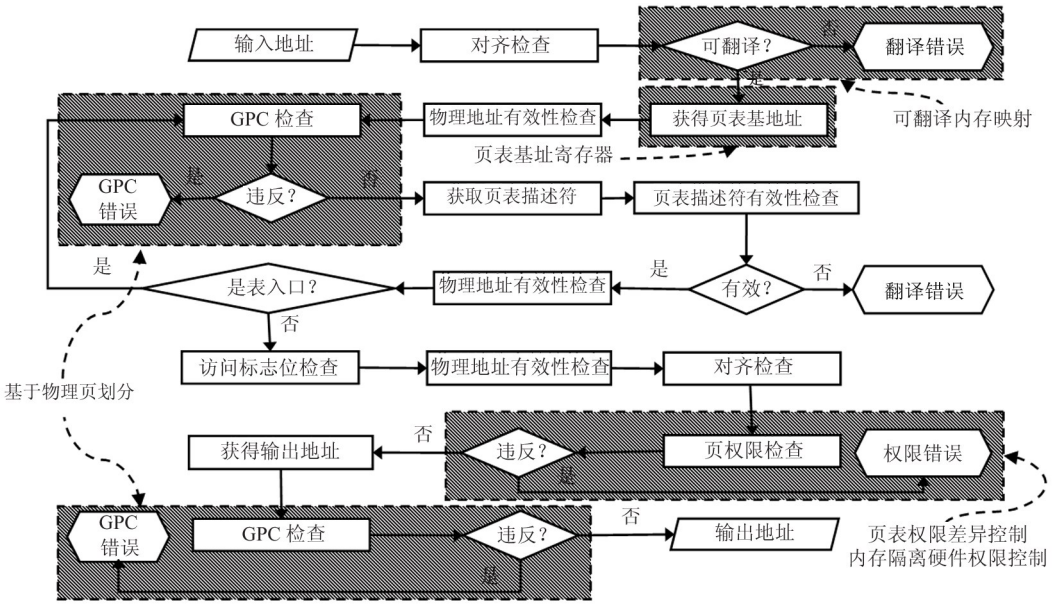


图 2 MMU 错误检查流程及隔离硬件作用位置

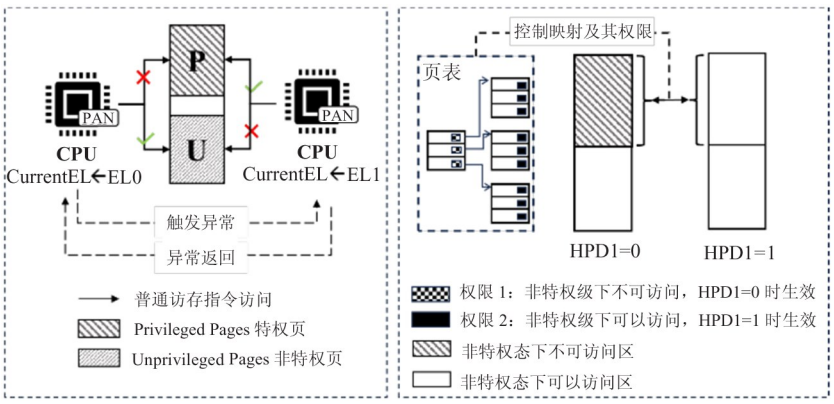


图 3 利用页表控制寄存器构建隔离机制

以将 TCR. TxSZ 作为内存隔离机制用来保护代码和数据。Hilps[24]、XNU(X is not unix)[25-26]布局内存数据和代码,将需要保护的内容放入被 TxSZ 隐藏的半部区间内,在隐私数据访问接口处放置 TxSZ 设置指令开关被隐藏的空间。TxSZ 隔离理论上可以用于各个权级。然而,TxSZ 对半切割空间划分隔离区域的方式不够灵活,而调整代码和数据布局会增加开发者的负担。

页表基地址寄存器。TTBR 指向根页表的物理起始地址,切换页表基地址寄存器代表切换整个虚实映射关系和页权限设置。如图 3 所示,只需切换基地址就可以抹去隔离区域的虚实映射关系,从而达到隐藏内存区域的效果。该方法通常需要 2 套以上的页表。仅当需要访问被隔离域时,可以通过包含页表基地址切换指令的域切换接口。这类方法需要重点保护页表切换指令,因为 ARM 架构规定页表基地址寄存器赋值必须通过通用寄存器,不支持常量赋值,所以攻击者可以借由通用寄存器篡改页表基地址寄存器。另外,修改页表基地址寄存器将立即生效,无法在指令执行后检查赋值合法性,因此必须阻止攻击者执行页表切换指令。SKEE(secure kernel level execution environment)[27]提出了基于页表基地址寄存器隔离保护内核空间的隐私数据的方案,并构建了不同 ARM 版本下的页表切换指令保护措施。在 ARMv7 中,SKEE 让被隔离区域独占空闲页表基地址寄存器 TTBR1,并给控制寄存器 TTBCR. N 赋值常量 0 或 1 来控制 TTBR1 是否启用。ARMv8 下 2 个页表基地址寄存器都被使用,因此无法复用 ARMv7 的方案。SKEE 直接采用了切换页表基地址寄存器的方式,并使用虚拟化和隔离区域隐藏的方式保护入口和出口的页表切换指令,但增加了域切换开销。

页表权限差异控制。利用内存权限的差异来隔离不同组件是最常见的内存隔离方式。权级隔离便是通过配置页面权限,使得低权级代码不可访问高权级的代码和数据。但权限设置无法禁止特权级对非特权页的访问,这存在安全隐患。特权代码(如内核)的数据流可能会被攻击者劫持到用户空间中,例如 ret2user 攻击[28]。PAN 被用来阻止这种内核去访问非特权页的情况。在该机制下,特权代码在执行过程中的普通访存指令 LS(load/store)访问非特权页均会触发权限异常,如图 4 所示。为了满足高权级代码访问非特权页面的正常需求,ARM 提供不受 PAN 约束的 LSU(load/store unprivileged)专用指令以访问非特权的页面。利用 PAN 隔离和 LSU/LS 访存指令差异,部分研究[29-31]将被保护的隐私数据放在非特权页中,并且只能由可信代码内嵌 LSU 指令访问隐私数据;还有研究逆向使用 LSU 硬件特性实现了特权态下沙箱机制,即将不可信的代码中的访存指令变为 LSU 指令,仅可访问放在非特权页中的数据,而无法访问其之外的可信代区域数据[9]。TCR. HPDs 则是通过控制高层页表的页表项权限设置来翻转所管控的内存访问权限,达到一键切换页面权限的效果。当 TCR. HPDs 设置为 0 时,遍历多级页表时发现的第 1 个非零的权限位作为页的最终权限;当 TCR. HPDs 设置为 1 时,由最低级的权限位决定权限。图 4 以两级页表为例,展示了 HPD1 的原理。InversOS[32]利用 TCR. HPDs 特性隔离保护变为非特权的内核页,阻止普通进程访问这些内核页。微内核[33-34]的设计则更为彻底,它的理念是只将页表机制和 IPC(inter-process communication)机制保留在内核中,其他的系统服务则全部被降级到非特权的服务进程中。利用权级隔离和进程隔离,微内核中的非核心系统服务中的缺陷和漏洞无法直接危害到内核关键部分。然而微内核会引入高 IPC 性能开销问题。

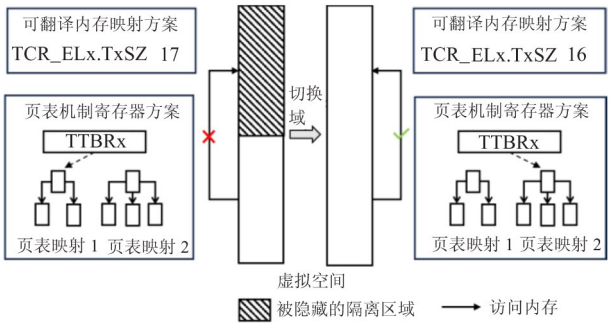


图 4　基于页表权限构建隔离机制

3.1.2　基于内存隔离专用硬件完成隔离

域隔离硬件权限控制。上述利用页表控制属性

方案可以很好地构建内存隔离机制,但是所用到的硬件是为 MMU 服务,而 MMU 的加速硬件 TLB 会对隔离机制产生干扰。每次切换隔离域都需要清除 TLB 相应缓存项或使用 ASID 分离不同组件的 TLB 缓存项,以避免因 TLB 缓存映射导致组件隔离失效。使用专门的域隔离硬件则从源头上避免了 TLB 的干扰,会有更高的切换效率。但域隔离硬件一般有域数量上限,不能划分无限多个域。域隔离硬件内存域(memory domain)与域访问控制寄存器(domain access control register, DACR)是一类专门用于内存划分与隔离的硬件。DACR 寄存器有 16 个 2 比特宽的域,用于标识每个域的访问权限。每个页表项上有 4 比特宽的 Domain 域,用来标记一个虚拟页从属于哪个域。如图 5 所示,DACR 中的域权限标识着当前代码的钥匙,而页表上的域 ID 就像锁一般,阻止非正常的跨域访问。Shred[35] 在 DACR 的基础上,通过更新页表项的 Domain 域(而非 DACR 寄存器)将内存区域划分给不同的组件,通过灵活地安排 Domain 域分配以扩充隔离域的数量上限。ARMLock[36] 则利用 MemoryDomain 和 DACR 构造硬件辅助的错误隔离系统:它给主区域和沙箱赋予不同的域 ID,使得主区域和沙箱都只能访问各自的内存区域,并且进出沙箱需要更新 DACR 寄存器的权限,以防止跨域访问。然而 MemoryDomain 和 DACR 只存在 32 位 ARMAProfile 芯片中,64 位的 ARMv8 芯片中并没有该机制。ARMv9[37] 则又引入了可被称为域隔离的硬件机制——遮蔽权限机制。该机制将页表权限从页表项上转移到了权限寄存器中,页表项上的权限位变为了权限寄存器的域偏移索引[37]。每个页表项包含 2 个索引域:PIIndex 用于索引基础权限寄存器(PIRXX _ ELx),而 POIndex 用于索引遮蔽权限寄存器(POR _ ELx)。基础权限是一个块或页能够拥有的最大权限集合,会被缓存在 TLB 中;而遮蔽权限则用于收紧基础权限,收紧后的权限不会缓存在 TLB 中。每次访存,CPU 依据当前运行权级,将索引对应权级的基础权限寄存器和遮蔽权限寄存器,综合决定最终的页访问权限。非特权态下的代码可以读写遮蔽权限寄存器 POR _ EL0,因此非特权代码获得了无需下陷到特权级中就可以更改页权限的能力,这一点与 IntelMPK 机制类似。虽然目前还没有相关的研究讨论如何使用这一机制完成内存隔离,但容易联想到一些可被迁移到 ARM 架构上的 IntelMPK 和 IntelPKS 的研究[4,38-39],即将修改遮蔽权限寄存器作为切换隔离域的方法,使得只有可信代码能够访问隔离域。

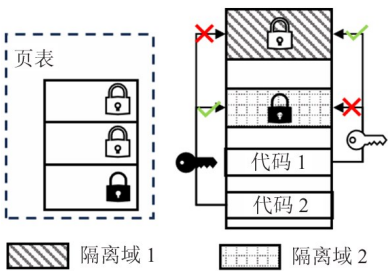


图 5　基于内存隔离专用硬件构建隔离机制

特殊权限控制域隔离硬件。ARM 公司授权了多家公司可以定制其 ARM 架构。苹果公司就在其自研的 ARM 芯片中加入了特有硬件机制 Apple APRR(access permission remapping registers)[40]、SPRR(secure permission remapping registers)[41]技术,用于保护苹果产品的安全。APRR 机制将页表上的权限位置变为标识位,提供了一种可以通过寄存器控制有特定编码的虚拟页的实际访问权限的方法。SPRR 是 APRR 功能的改进,SPRR 收紧了可以设置的权限集合,页面权限只能被改为指定 4 个值中的一个。对于经验丰富的黑客来说,可以利用 APRR 在用户空间创建可写可执行页。不过,该方法在 SPRR 机制中无法实现,因为 SPRR 机制下无法编码此类权限[41]。APRR 和 SPRR 在内核中被用来保护关键的数据,如页表;而在用户态下则用在了 JIT(just-in-time)引擎机制中,一方面作快速的页权限切换,另一方面可以阻止 JITed 代码被篡改[40-41]。KTRR(kernel text readonly region)[25] 是苹果公司专门用来保护内核页表代码段和只读数据段的机制。该机制提供的 KTRR 寄存器和 RoRgn 区域标注了代码段、只读数据段和页表的区间并锁定它们的修改,任何企图直接修改页表权限的方式或者篡改内核代码和只读数据的攻击均失效。

3.2　基于物理空间划分的隔离方案

物理空间上划分内存隔离域,相较于基于虚拟

空间的方案,可以提供更强的隔离保护,甚至当 MMU 被关闭时都能保证被隔离区域的安全性不会被破坏。但基于物理空间的隔离方案的运行效率不如基于虚拟空间的隔离方案,其一是因为物理空间资源相对于虚拟空间更加昂贵,当物理空间被划分为专用内存时,就无法参与同其他区域的统一调配,这会造成整体系统的性能损失;此外,也因为切换物理空间的隔离域依赖于更高特权级的协助,往往比虚拟空间的隔离域切换开销更高。

物理空间划分。TrustZone 是目前应用广泛的可信执行环境,可信软件运行在 SecureWorld 中,其余则运行在 NormalWorld 中,其底层依赖位于系统总线和内存之间的 TZASC(TrustZone address space controller)将物理内存划分成不同世界可访问的区域。TZASC 还可以利用总线事务的附加特性——基于身份过滤——来分离受保护的内存区域,创造一块仅由某个特定身份的设备(如 CPU)可访问的物理内存。SANCTUARY[42] 通过 TZASC 构建了类 Intel SGX(software guard extensions)的环境,让需要隐私内存空间的进程运行在指定的核心上,并给该 CPU 核心划分了仅该核心可访问的物理空间。但创建一个这样的进程需要操作系统下陷被独占的核心,以防操作系统将不可信进程调度到该核心上。SHELTER[43] 使用了 ARMv9 中支持的另一套可信执行环境 CCA 中的 GPT 硬件,避免了下陷核心带来的开销。GPT(granule protection tables)是 TZASC 的增强版,它也将物理内存划分成了不同 World 可访问的区域,如图 6“物理空间划分方案”所示,而 GPTBR _ EL3 是每核心一个,因此可以让运行在不同核心上的进程看到不同的 GPT 划分结果,并且在切换进程的同时切换 GPT 基地址寄存器,避免下陷核心。图 2 仅标识了 MMU 开启时的 GPC 检查,但实际上当 MMU 关闭时 GPC(granule protection checks)检查依旧生效,因此 GPT 和 GPC 的配合能够提供完备的隔离防护。

虚拟化管控物理空间。使用虚拟化方法让 VM 物理内存呈现不同的布局以此来构建隔离的物理内存是一种可行的做法,如图 6 所示“虚拟化管控方案”虚线中所示。xMP[44]、SeCage[45]、MemSentry[46]、DBox[47] 可让 VM 在物理空间上划分成不同的域,每个域有专门的第 2 阶段页表基地址,而通过第 2 阶段页表的权限设置,可以让不同域对不同物理页有不同的访问结果。这种方法在支持虚拟化的架构上都可以实现,并不限于某种架构。但虚拟化方法并不是一种高效的做法,每次的虚拟机退出和进入都需要保存和恢复客户机上下文,这是比较耗费时间的操作,而切换域的操作在 ARM 架构中必然导致虚拟机退出,反观在 x86 上使用 VMFUNC 机制[48],虚拟机无需下陷即可修改第 2 阶段页表基地址,以加速物理页映射的切换过程。

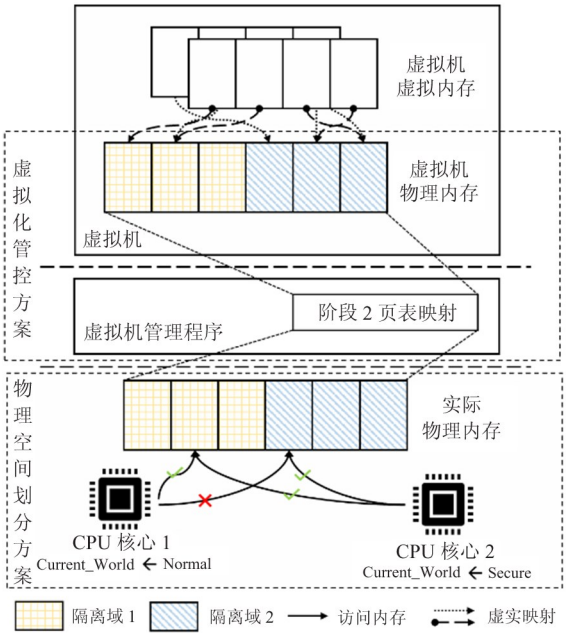


图 6 基于物理空间划分的隔离方案

3.3 基于调试机制的隔离方案

ARM 架构提供了自托管式调试机制(self-hosted debug),该机制和前述的页表权限管理机制是相互独立的,当该机制开启时,只要满足了预设的调试条件,就会触发调试异常,被高权级异常处理函数捕获。这其中包括了一类调试方法——监视点(Watchpoint)。Watchpoint 可以监控一段连续虚拟内存的访问,当存在指令访问被监视区域时则触发异常。因此,一些研究[49-50] 使用了 Watchpoint 在不可信代码执行时监视被隔离区域,防止被隔离区域被非法访问。但 Watchpoint 的数量有限,常见配置是每核心仅有 4 套 Watchpoint 相关寄存器。此外每

套 Watchpoint 可以监控的内存范围上限仅为 2 GB,如果被隔离区域超过这个上限,则需要配合其他的手段(如切换页表基地址寄存器的方式)完成隔离区域的保护[49-50]。

3.4 基于内存标记拓展的地址隔离方案

ARMv8.5-A 中引入了新的硬件特性——内存标记拓展(memory tagging extension,MTE),开启该机制,可以为每 16 个字节的内存打上“颜色”,由此将内存划分成不同颜色的区域[13]。颜色信息被存在只有专用指令可访问的标记内存中。指针在访问有“颜色”的区域时需要在指针的高位加上对应“颜色”的标识,否则触发标识检查错误,如图 7 所示。MTE 严格意义上来说是一种基于地址的隔离方案。然而,只用 MTE 硬件标记内存颜色并给指针打上 Tag 位并不足够安全。拥有任意地址读写原语的攻击者可以篡改指针的 Tag,使该指针能够访问任意颜色的内存;而拥有任意代码执行的攻击者甚至可以执行 MTE 访存指令修改内存的颜色。因此,要使用 MTE 作内存隔离,必须要结合其他的机制保护内存颜色和 Tag 位的一致性和有效性。HAKC(hardware-assisted kernel compartment)[51] 提出了将 MTE 和 PAC 配合完成两级划分,让每个可加载内核驱动作为一个组件,而组件内部代码再划分不同颜色的 Clique,利用 PAC 解决组件之间的指针传递和保护问题,实现了可加载内核驱动的隔离。ColorMyWorld[52] 使用了软件方法保护了 Tag 位不被篡改,它在每条访存指令之前插桩,将指针高位的强制覆盖为编译时预设的颜色。Sfitag(software fault isolation with memory tagging)[53] 则使用了 MTE 作为沙箱机制的重要隔离手段,它利用专用寄存器作为沙箱的访存寄存器,每次访存前会将地址加载到寄存器中;此外,它还使用了编译和验证器来确保沙箱内部不存在 Tag 内存的写指令 stg。

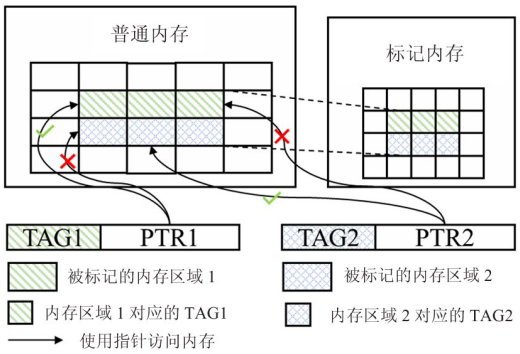


图 7 MTE 功能原理示意图

4 相关研究对比与实际场景应用

表 1 总结了上一节提到的内存隔离机制方案,列举了其所用到的硬件、作用的空间、隔离域数量和域切换方法。然而这些硬件之间并非可以相互替换,它们各自有优缺点。表 2 从作用权级(非特权态、特权态)、域数量(有限域和无限域)、有无 TLB 干扰、隔离粒度(字节粒度)、硬件存在的 ARM 版本(ARMv7、ARMv8、ARMv9)、切换开销和是否需要陷入特权态中切换域(支持非特权态切换)来横向对比各个研究中所用的硬件特性。从表 2 中可以看出,MTE 复合 PAC 的硬件方案优势最明显,适用场景广且隔离粒度最新,但是因为 MTE 的硬件只在高版本 ARM 中支持,且因为 MTE 复合 PAC 方案属于地址隔离方案,所以面临着代码膨胀和更高的性能开销问题。Watchpoint 硬件适用性也很广,因此应用在了许多实际场景中,但因为 Watchpoint 有隔离范围上限(8 GB),因此需要精心布局被隔离的数据和代码。PAN 复合 LSU 方案虽然局限性很多,但是因为分离的访存指令使得该方案可以不作域切换,因此有着最快的隔离域访问速度。其他硬件的优缺点如表 2 所示。

针对不同场景,依照表 2 所列出的各个硬件的特性选择硬件辅助隔离方案,其中需要着重从实际场景、权级、保护对象数量、性能这几个方面考虑。以几个常用场景为例进行展示。

(1)场景 1:特权态下敏感数据保护

宏内核追求极致的效率同时也带来了风险:任何一个模块漏洞都会影响整个内核的安全运行,由此产生了内核敏感数据的保护需求,如 Credential 和页表。表 2 中可以支持特权态的隔离硬件都可以在内核中使用。但需要注意的是,如果隔离机制建立在页表设置正确前提下,那一定要将页表纳入隔离保护的范围内,否则同权级的攻击者可以通过修改页表轻松地突破隔离机制。

表 1 硬件辅助的内存隔离机制总结

方案	硬件	空间	域数量	切换方法/校验方法
基于页表控制属性	TCR. TxSZ	虚拟	2	设置 TCR _ ELx
	TTBR/TTBCR	虚拟	≥2	设置 TTBR 或 TTBCR
	PAN + LSU	虚拟	2	设置 PAN 或不切换使用 LSU
	TCR. HPDs	虚拟	2	设置 TCR _ ELx
基于内存隔离专用硬件	DACR	虚拟	≥2	设置权限寄存器
	ARM Overlay	虚拟	≥2	设置遮蔽寄存器
	APRR/SPRR	虚拟	≥2	设置控制寄存器
基于物理空间划分	TZASC	物理	≥2&&≤9	切换到指定 CPU
	GPT + GPC	物理	≥2	切换 GPT 基址寄存器
	硬件虚拟化	物理	≥2	切换 Stage 2
基于 MTE 的地址隔离	MTE + PAC	虚拟	≥2	MTE Tag 检查和 PAC 校验
基于调试机制	Watchpoint	虚拟	≥2&&≤4	设置控制寄存器

表 2 对比各类内存隔离硬件方案中使用的硬件

硬件	非特权态	特权态	有限域(≥3)	无限域	无 TLB 干扰	字节粒度	ARMv7	ARMv8	ARMv9	切换时间(Cycles)	支持非特权态切换
TCR. TxSZ	√	√	×	×	×	×	×	√	√	≥29	×
TTBR/TTBCR	√	√	√	√	×	×	√	√	√	≥29	×
PAN + LSU	○	√	×	×	×	×	○	○	√	~15 或 ~0	×
TCR. HPDs	√	○	×	×	×	×	×	○	√	≥29	×
DACR	√	√	√	○	√	×	√	×	×	≥29	×
ARM Overlay	√	√	√	○	√	×	×	○	○	-	√
APRR/SPRR	√	√	√	○	√	×	×	○	-	-	√
TZASC	√	√	√	○	√	×	○	○	○	98 ~ 310 μs	×
GPT + GPC	√	√	√	√	√	×	×	×	○	-	×
硬件虚拟化	√	√	√	√	×	×	√	√	√	≥290	×
MTE + PAC	√	√	√	√	√	√	×	○	√	1 ~ 2	√
Watchpoint	√	√	√	×	√	√	√	√	√	~56	×

√：支持 ○：不完全支持 ×：不支持

(2)场景 2:特权态下内核驱动隔离

考虑到驱动的数量繁多,驱动可能访问的数据粒度到了字节级别,并且需要高效的域隔离切换,因此可以使用 MTE 复合 PAC 的方式作隔离。切换页表基址寄存器 TTBR 也是可行方案之一,但是使用页表隔离就需要考虑内核和驱动之间的数据传输和同步问题。这是因为驱动与内核之间有共享数据的情况存在,但数据只能被置于一个域中,因此可能的解决方案是在不同域之间拷贝数据并同步,或是考虑别名映射的手段。

(3)场景 3:非特权态下单一敏感数据保护

在非特权态下的进程中经常面临用户隐私数据保护的需求,例如对密钥的隔离保护。32 位 ARM 架构,例如 ARMv7-A 或者 ARMv8-A-AArch32 中可以使用 DACR 机制,将隐私数据放在单独域中,只能由可信代码访问。AArch64 架构可以使用 TxSZ、HPDs 和 Watchpoint。但无论上述的哪个硬件,都需要依赖特权级下切换域。陷入特权级再返回非特权级的时间开销不低,因此在域切换频繁的场景下,切换域的开销便不可忽视。PANIC[54]将进程落在了内核态下执行,使得进程可以使用特权硬件(如 PAN 和 LSU),提供高效的进程内的内存隔离机制,

极大减少了域切换的开销。ARMv9 的 Overlay 机制可以很好解决陷入特权带来的开销问题,但同样地,Overlay 所需要的 ARM 版本太高,通用性较差。

(4)场景 4:非特权态下多域隔离

这种场景多见于进程内部几个组件模块互不信任的情况,同场景 2 有诸多相似之处,MTE 加上 PAC 的机制、Watchpoint 机制以及 ARMOverlay 寄存器都可以提供隔离机制。另外还有多个进程之间互相不信任或者进程和内核互不信任的情况,这些情况下可以应用物理空间隔离 TZASC 和 GPT,强制进程使用仅本进程可访问的物理页。

4 硬件辅助内存隔离机制的安全性分析

虽然内存隔离机制提供了隐私数据的安全防护,但这些方案在目前看来并不是绝对安全。

域隔离方案面临的安全威胁。对于域隔离方法来说,可能遇到的威胁有 2 个,分别是接口滥用攻击和混淆代理攻击。

接口滥用攻击产生的原因是域切换指令仅在接口中出现,因此攻击者会想方设法通过控制流劫持方式,绕过接口入口处的检查代码直接跳转到关键切换指令,以达成随意切换任意域并访问敏感数据的目的。为了防止攻击者利用这种接口滥用攻击绕过隔离机制,相关研究中提出了以下几类阻止方法。

(1)循环检查法。那些需要使用通用寄存器才能设置关键系统寄存器的同级域隔离方案,会在设置完系统寄存器后立刻检查设置值是否正确,如果不正确则跳转回设置指令处,再次设置。

(2)断点法。同样是针对使用通用寄存器才能设置关键系统寄存器的同级域隔离方案,断点法在关键域切换指令处设置指令断点,只有在域切换接口入口处才可以清除断点,直接跳转到关键域切换指令会触发调试异常。

(3)分页法。SKEE[27] 提到的一种有趣的保护方式,将页表基地址寄存器的设置指令放在了虚拟页的末尾,而该页和下一页被分在了不同域中,一旦页表基地址寄存器的切换指令生效了,该页的指令就无法被访问了,而下一页的值指令还可以正常取指、译码和执行。

(4)虚拟机下陷法。当上面的方法失效时,可以寻求更高权限的帮助,如虚拟化,但是这样的开销相对来说更高。虚拟机管理软件可以配置域切换指令下陷到虚拟机中完成,这样低权级的攻击者就无法滥用切换指令。

另一类攻击混淆代理的基本想法是:攻击者虽然无法滥用接口绕过隔离机制,但可以伪装成接口调用者,修改域切换接口参数,通过正常的访问接口来读写隔离数据。这样的攻击方式同样可以完成篡改隔离数据的目的。此类攻击依据特定的场景和接口实现,没有统一的攻击方式,因此防御此类攻击也需要加入探针收集调用者的信息,有针对性地识别调用者实际调用意图。目前还没有统一、高效的防御方法能够抵抗此类攻击。

地址隔离方案面临的安全威胁。前文中提到的地址隔离方法需要重点考虑如何保护地址范围信息元数据不被篡改。虽然 HAKC 方法中使用了 PAC 保护指针 Tag,但是 PAC 自身有无法杜绝的重放攻击问题,即 PAC 计算认证码使用的上下文信息可能重复,导致了虽然是不同的指针却都可以通过 PAC 验证的问题。同样地,不同区域之间需要数据传递,因此地址隔离方法也存在混淆代理攻击问题。

5 研究展望

近年来,硬件辅助的内存隔离机制从各个角度迅速发展,在硬件高效使用、域上限提升、安全切换等多个方面都有了成熟的技术。然而现有研究还存在局限性,这也是未来探索的方向。

(1)利用指令级隔离减少隔离机制开销

目前内存隔离方案大部分都依赖于域切换或者地址检查方式,这面临着代码膨胀问题,也带来了一定的开销。指令级隔离方式是更加高效、安全且代码膨胀率更低的方案。指令级隔离可以让隔离域拥有自己的专用指令,这使得攻击难度增加,攻击者无法利用有漏洞组件的指令发起对其他组件的攻击。另外,因为不同域的关键指令并不相同,检查和权限被内化在了指令中,从而避免了域切换。ARM 的

LS 和 LSU 访存指令就是典型示例,通过让不同域使用不同的访存指令和页面,并配合 UAO(user access override)硬件,可以达到双向隔离的效果[54]。这类属于访存指令隔离,相关工作[55-56]也采用了类似的想法。更进一步,指令级隔离可以扩展到任意指令,依据信任等级给不同组件划分可执行指令集合,阻止攻击者在低可信组件中执行高危指令(如切换寄存器指令)[57]。

(2)构建多机隔离场景下的隔离机制

本文提到的相关研究聚焦在单机隔离场景,即在同一物理机器下如何完成内存隔离。但“多机场景下如何构建隔离机制”是目前新兴的一个课题。CXL(compute express link)等技术的推出,促进了内存池化发展。多机的物理内存被联合起来交由进程统一访问,给内存密集型的进程带来了可观性能提升的同时,也给内存隔离机制带来了以下挑战。1)内存庞大,如何管理驻留在内存中庞大的域元数据、维持组件隔离的高效运转是一大挑战。部分工作试图通过复用域标识来实现无限域的做法就不再适用。2)如何高效管理并划分被多机共享的物理内存是内存隔离的另一大挑战。

(3)攻克混淆代理攻击

针对接口的攻击,尤其是混淆代理攻击,一直是内存隔离机制中悬而未决的问题。有研究指出,在其测试的 25 个应用和 36 个组件 API 中找到了 629 个接口相关漏洞[58]。接口问题严峻,而如何为接口攻击问题提供完备且低开销的防御方案是未来需要考虑的问题。

6 结 论

内存隔离机制是软件漏洞缓解措施中不可缺失的一环,通过这一机制,可以有效遏制攻击者对敏感数据窃取攻击。硬件辅助的内存隔离机制则能够兼顾安全和效率,给软件带来强有力的保护。本文梳理了主流架构 ARM 下的硬件辅助的内存隔离相关研究和业界方案,并归纳了它们的研究现状和适用场景。最后,整理了目前硬件辅助内存隔离机制所面临的安全现状,展望了该领域未来的发展方向,可为后续研究提供重要的参考价值。

参考文献

[1] SALTZER J H, SCHROEDER M D. The protection of information in computer systems[J]. Proceedings of the IEEE, 1975,63(9):1278-1308.

[2] WANG Z, WU C, XIE M, et al. SEIMI: efficient and secure SMAP-enabled intra-process memory isolation[C]//2020 IEEE Symposium on Security and Privacy (SP). San Francisco, USA :IEEE, 2020:592-607.

[3] WITCHEL E, RHEE J, ASANOVI Ć K. Mondrix: memory isolation for Linux using mondriaan memory protection [C]//Proceedings of the 20th ACM Symposium on Operating Systems Principles. New York, USA: ACM, 2005: 31-44.

[4] VAHLDIEK-OBERWAGNER A, ELNIKETY E, DUARTE N O, et al. ERIM: secure, efficient in-process isolation with protection keys (MPK)[C]// The 28th USENIX Security Symposium. Santa Clara, USA: USENIX Association, 2019:1221-1238.

[5] Anonymous. Heartbleed bug[EB/OL]. (2020-06-03)[2023-11-13]. https://heartbleed.com/.

[6] DURUMERIC Z, LI F, KASTEN J, et al. The matter of heartbleed[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. New York, USA: ACM, 2014:475-488.

[7] 白家驹,刘虎球,王瑀屏. 一种内核模块的指针错误隔离方法[J]. 小型微型计算机系统, 2015,36(5): 1127-1132.

[8] 甄凯,白家驹,刘虎球,等. FPSFI:一种基于 llvm 架构的驱动错误隔离机制[J]. 小型微型计算机系统, 2013,34(10):2246-2250.

[9] AWEKE Z B, AUSTIN T. USFI: ultra-lightweight software fault isolation for IoT-class devices[C]//2018 Design, Automation and Test in Europe Conference and Exhibition (DATE). Dresden, Germany: IEEE, 2018: 1015-1020.

[10] BUROW N, ZHANG X, PAYER M. SoK: shining light on shadow stacks[C]//2019 IEEE Symposium on Security and Privacy (SP). San Francisco, USA: IEEE, 2019:985-999.

[11] ARM. ARM ARMv8-a architecture registers[EB/OL].

(2021-12-01)[2023-11-13]. https://developer.arm.com/documentation/ddi0595/2021-12/AArch64-Registers/PAN-Privileged-Access-Never.

[12] Counterpoint. ARM-based PCS to nearly double market share by 2027[EB/OL]. (2023-04-10)[2023-11-13]. https://www.counterpointresearch.com/insights/arm-based-pcs-to-nearly-double-market-share-by-2027/.

[13] ARM. ARMv8.5-a memory tagging extension white paper [EB/OL]. (2021-12-01)[2023-11-13]. https://developer.arm.com/documentation/102925/latest/.

[14] VERMA P. Understand ARM pointer authentication[EB/OL]. (2021-12-13)[2023-11-13]. https://learn.arm.com/learning-paths/servers-and-cloud-computing/pac/.

[15] ARM. The realm management extension (RME) for ARMv9-a[EB/OL]. (2022-06-01)[2023-11-13]. https://developer.arm.com/documentation/ddi0615/latest/.

[16] LI X, LI X, DALL C, et al. Design and verification of the ARM confidential compute architecture[C]// The 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). Carlsbad, USA: USENIX Association, 2022:465-484.

[17] KOCHER P, HORN J, FOGH A, et al. Spectre attacks: exploiting speculative execution[C]//2019 IEEE Symposium on Security and Privacy (SP). San Francisco, USA: IEEE, 2019:1-19.

[18] LIPP M, SCHWARZ M, GRUSS D, et al. Meltdown [EB/OL]. (2018-01-03)[2023-11-14]. http://arxiv.org/pdf/1801.01207.

[19] MUTLU O, KIM J S. Rowhammer: a retrospective[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019,39(8):1555-1571.

[20] FUSTOS J, FARSHCHI F, YUN H. SpectreGuard: an efficient data-centric defense mechanism against spectre attacks[C]//Proceedings of the 56th Annual Design Automation Conference 2019. New York, USA: ACM, 2019:1-6.

[21] Anonymous. SoK: practical foundations for software spectre defenses[EB/OL]. (2021-05-12)[2023-11-24]. https://arxiv.org/pdf/2105.05801.

[22] 石培涛,刘宇涛,陈海波. 基于虚拟化内存隔离的 Rowhammer 攻击防护机制[J]. 信息安全学报, 2017, 2(4):1-12.

[23] 王崇,魏帅,张帆,等. 缓存侧信道防御研究综述[J]. 计算机研究与发展, 2021,58(4):794-810.

[24] CHO Y, KWON D, YI H, et al. Dynamic virtual address range adjustment for intra-level privilege separation on ARM[C]// The 24th Annual Network and Distributed System Security Symposium. San Diego, USA: The Internet Society, 2017:17321266.

[25] SIGUZ A. KTRR[EB/OL]. (2018-08-17)[2023-06-14]. https://blog.siguza.net/KTRR/.

[26] Apple. Proc _ reg-t1sz-details[EB/OL]. [2023-11-14]. https://opensource.apple.com/source/xnu/xnu-6153.61.1/osfmk/arm64/proc _ reg.h.auto.html.

[27] AZAB A, SWIDOWSKI K, BHUTKAR R, et al. SKEE: a lightweight secure kernel-level execution environment for ARM[C]//Proceedings of the 2016 Network and Distributed System Security Symposium. San Diego, USA: Internet Society, 2016:23009.

[28] KEMERLIS V P, PORTOKALIDIS G, KEROMYTIS A D. kGuard: lightweight kernel protection against return-to-user attacks[C]// The 21st USENIX Security Symposium. Bellevue, USA: USENIX Association, 2012:459-474.

[29] ZHOU J, DU Y, SHEN Z, et al. Silhouette: efficient protected shadow stacks for embedded systems[C]// The 29th USENIX Security Symposium. Virtual:USENIX Association, 2020:1219-1236。

[30] KWON D, SHIN J, KIM G, et al. uXOM: efficient execute-only memory on ARM cortex-M[C]// The 28th USENIX Security Symposium. Santa Clara, USA: USENIX Association, 2019:231-247

[31] CHO Y, KWON D, PAEK Y. Instruction-level data isolation for the kernel on ARM[C]//Proceedings of the 54th Annual Design Automation Conference. New York, USA: ACM, 2017:1-6.

[32] SHEN Z, CRISWELL J. InversOS: efficient control-flow protection for aarch64 applications with privilege inversion [EB/OL]. (2023-04-17)[2023-04-22]. http://arxiv.org/pdf/2304.08717.

[33] LIEDTKE J. On micro-kernel construction[J]. ACM SIGOPS Operating Systems Review, 1995,29(5):237-250.

[34] KLEIN G, ELPHINSTONE K, HEISER G, et al. SeL4: formal verification of an OS kernel[C]//Proceedings of

the ACM SIGOPS 22nd Symposium on Operating Systems Principles. New York, USA: ACM, 2009:207-220.

[35] CHEN Y, REYMONDJOHNSON S, SUN Z, et al. Shreds: fine-grained execution units with private memory[C]//The 2016 IEEE Symposium on Security and Privacy (SP). San Jose, USA: IEEE, 2016:56-71.

[36] ZHOU Y, WANG X, CHEN Y, et al. ARMlock: hardware-based fault isolation for ARM[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. Scottsdale, USA: ACM, 2014: 558-569.

[37] ARM. Learn the architecture-aarch64 memory attributes and properties[EB/OL]. (2021-12-01)[2023-10-15]. https://developer.arm.com/documentation/102376/0200/Permission-indirection-and-permission-overlay-extensions/An-example-of-using-permission-indirection-and-permission-overlay-features.

[38] GU J, LI H, LI W, et al. EPK: scalable and efficient memory protection keys[C]//2022 USENIX Annual Technical Conference. Carlsbad, USA: USENIX Association, 2022:609-624.

[39] 李浩,古金宇,夏虞斌,等. 基于 PKS 硬件特性的 eBPF 内存隔离机制[J]. 软件学报, 2023, 34(12):5921-5939.

[40] Siguza. APRR[EB/OL]. (2019-08-08)[2023-10-15]. https://blog.siguza.net/APRR/.

[41] PETER S. Apple silicon hardware secrets: SPRR and guarded exception levels (GXF)[EB/OL](2021-05-06)[2023-11-13]. https://blog.svenpeter.dev/posts/m1_sprr_gxf/.

[42] BRASSER F, GENS D, JAUERNIG P, et al. SANCTUARY: arming TrustZone with user-space enclaves[C]//The 26th Annual Network and Distributed System Security Symposium. San Diego, USA: The Internet Society, 2019: 1-15.

[43] ZHANG Y, HU Y, NING Z, et al. SHELTER: extending ARM CCA with isolation in user space[C]//The 32nd USENIX Security Symposium. Anaheim, USA: USENIX Association, 2023:1-18.

[44] PROSKURIN S, MOMEU M, GHAVAMNIA S, et al. XMP: selective memory protection for kernel and user space[C]//2020 IEEE Symposium on Security and Privacy (SP). San Francisco, USA: IEEE, 2020:563-577.

[45] LIU Y, ZHOU T, CHEN K, et al. Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. Denver, USA: ACM, 2015:1607-1619.

[46] KONING K, CHEN X, BOS H, et al. No need to hide: protecting safe regions on commodity hardware[C]//Proceedings of the 12th European Conference on Computer Systems. Belgrade, Serbia: ACM, 2017:437-452.

[47] 余劲,黄皓,诸渝,等. DBox:宏内核下各种设备驱动程序的高性能安全盒[J]. 计算机学报, 2020,43(4): 724-739.

[48] 钟炳南,邓良,曾庆凯. 基于硬件虚拟化的内核同层多域隔离模型[J]. 软件学报, 2022, 33(2): 473-497.

[49] WU H, CHEN Y, ZHOU Y, et al. DriverJar: lightweight device driver isolation for ARM[C]//2023 60th ACM/IEEE Design Automation Conference (DAC). San Francisco, USA: IEEE, 2023:1-6.

[50] JANG J, KANG B B. In-process memory isolation using hardware watchpoint[C]//2019 56th ACM/IEEE Design Automation Conference(DAC). Las Vegas, USA: IEEE, 2019:1-6.

[51] MCKEE D, GIANNARIS Y, PEREZ C O, et al. Preventing kernel hacks with HAKC[C]//Proceedings of the 2022 Network and Distributed System Security Symposium. San Diego, USA: The Internet Society, 2022:1-17.

[52] LILJESTRAND H, CHINEA C, DENIS-COURMONT R, et al. Color my world: deterministic tagging for memory safety[EB/OL](2022-04-08)[2023-10-30]. https://arxiv.org/pdf/2204.03781v2.

[53] SEO J, YOU J, CHO Y, et al. Sfitag: efficient software fault isolation with memory tagging for ARM kernel extensions[C]//Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. New York, USA: ACM, 2023:469-480.

[54] XU J, XIE M, WU C, et al. PANIC: PAN-assisted intra-process memory isolation on ARM[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. Copenhagen, Denmark: ACM, 2023:919-933.

[55] FRASSETTO T, JAUERNIG P, LIEBCHEN C, et al. IMIX: in-process memory isolation extension[C]//The 27th USENIX Security Symposium. Baltimore, USA:

USENIX Association, 2018:83-97.

[56] NARAYAN S, GARFINKEL T, TARAM M, et al. Going beyond the limits of SFI: flexible and secure hardware-assisted in-process isolation with HFI[C]// Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2023:266-28.

[57] FAN S, HUA Z, XIA Y, et al. ISA-grid: architecture of fine-grained privilege control for instructions and registers [C]// Proceedings of the 50th Annual International Symposium on Computer Architecture. New York, USA: ACM, 2023:1-15.

[58] LEFEUVRE H, BADOIU V A, CHEN Y, et al. Assessing the impact of interface vulnerabilities in compartmentalized software[C]// Proceedings of the 2023 Network and Distributed System Security Symposium. San Diego, USA: The Internet Society, 2023:1-15.

A survey of hardware-assisted memory isolation mechanisms for ARM

XU Jiali, WU Chenggang, WANG Zhe

(State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(University of Chinese Academy of Sciences, Beijing 100090)

Abstract

Memory isolation is an important defense mechanism used to mitigate potential security hazards and improve the security and robustness of software. The memory isolation mechanism protects data in different components and restricts the data to be accessed only by code trusted by the components. Although software vulnerabilities cannot be avoided, memory isolation can effectively isolate and protect sensitive data so that even if there are vulnerabilities in the code, attackers cannot directly steal or tamper with the isolated data. For performance reasons, hardware-assisted memory isolation mechanisms have been widely emphasized, aiming to effectively reduce the extra overhead of isolation mechanisms to software while providing the same security guarantees. Therefore, the research focuses on exploiting the potential of hardware to provide efficient, secure and complete support for memory isolation mechanisms. This paper systematically summarizes the research on hardware-assisted memory isolation mechanisms and the actual memory isolation mechanisms used in ARM architecture, and further summarizes their current research status and application scenarios. Finally, in view of the current security situation and new scenarios faced by the existing hardware-assisted memory isolation mechanisms, the conception of the future development trend in this field is proposed.

Key words: hardware-assisted memory isolation, ARM architecture, domain-based isolation, address-based isolation