OAMAC：面向实用型失陷后攻击面缩减的来源感知强制访问控制

奥马尔·阿卜杜勒马吉德·伊德里斯·穆罕默德·伊尔哈米·M·奥拉克

arXiv:2601.14021v1 [cs.CR] 20 Jan 2026

摘要

现代操作系统提供了强大的强制访问控制机制，但它们主要关注的是谁在执行代码，而不是执行是如何发起的。因此，一旦获得权限，远程、本地或由后台服务启动的进程往往会被一视同仁，这使安全分析更加复杂，也为入侵后的敏感系统接口滥用创造了机会。

我们提出**来源感知强制访问控制（OAMAC）**，这是一种内核级强制执行模型，将执行来源（例如用户物理在场、远程访问或服务执行）视为一等安全属性。OAMAC 根据执行来源而非仅凭身份来调控对安全关键子系统的访问，从而实现对多个攻击面的集中式治理，同时显著降低策略复杂度。

我们展示了一种可部署的原型，它完全基于 Linux eBPF LSM 框架实现，无需修改内核。OAMAC 利用内核可见元数据对执行来源进行分类，在进程创建时传递来源，并在敏感文件系统接口和内核 BPF 控制平面上强制执行来源感知策略。策略保存在内核中的 eBPF 映射里，并可通过一个精简的用户态工具在运行时重新配置。

我们的评估表明，OAMAC 能有效限制远程攻击者在入侵后可实施的常见操作，同时保留正常的本地管理能力并维持系统稳定性。我们认为，执行来源是当代操作系统安全模型中缺失的一种抽象；将其提升为一等概念，无需具备特定子系统的专业知识或采用重量级安全框架，即可实现实用的攻击面缩减。

关键词： 来源感知的强制访问控制；内核加固；eBPF LSM；操作系统安全；基于来源的访问控制。

1 引言

远程失陷仍是现代系统面临的主要威胁途径之一。攻击者一旦获得远程 shell（通常通过 SSH），通常便能访问与本地管理员相同的内核接口。诸如 `/sys`、 `/proc/sys`和内核调试设施等接口能够强力控制系统行为，且常被滥用于实现持久化、权限提升和内核操纵。

传统 Linux 安全机制——包括 DAC、能力、SELinux 和 AppArmor——主要关注进程是谁，而不关注如何或从何处启动。这使得执行来源（例如物理在场与远程访问）无法通过现有访问控制模型明确表达。因此，远程获得的 root shell 与由实际在场的管理员打开的 root shell，实际上无法区分。

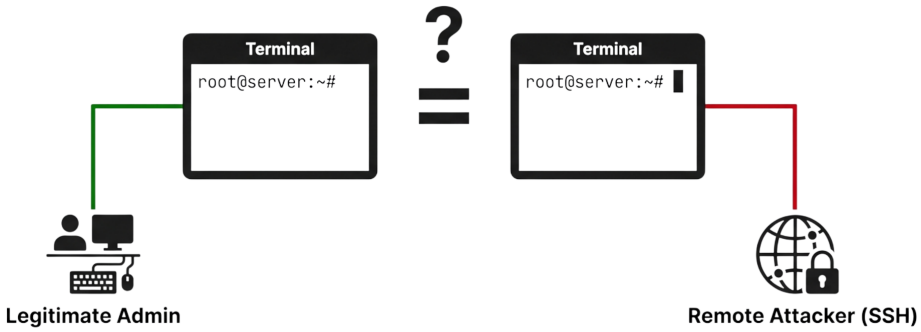


图1：威胁模型：远程 root 与物理访问的 root。

我们认为，**执行来源**是一种有意义但尚未得到充分利用的安全信号。在许多现实的威胁模型中，即使物理在场和远程访问使用相同的用户身份，前者也意味着更高程度的信任。这一观察结果引出了以下研究问题：

Linux 能否根据执行来源强制实施强制访问控制，而无需修改内核或使用重量级安全框架？

本文借助 eBPF LSM 对这一问题给出了肯定的回答。

本文并未提出新的访问控制框架或策略语言，而是指出，**执行来源** 是当代操作系统安全模型中缺失的一种抽象。现有机制虽然表达能力丰富，但要求管理员通过标签、域或子系统专用规则间接编码来源。我们的方法将执行来源提升为一等安全属性，从而支持集中式的推理与强制执行，同时不增加策略复杂度，也不改变现有的信任假设。

2 威胁模型与目标

本节阐明 OAMAC 设计所依据的对抗性假设，并说明其旨在实现的安全目标。我们的目标并非提出一种能够抵御所有形式系统失陷的通用防御方案，而是准确刻画一种现实的威胁环境，并评估在该环境下，来源感知强制访问控制如何缩小攻击者可利用的攻击面。我们明确说明所考虑和排除的攻击者能力，为理解本文后续部分介绍的设计、实现和评估结果奠定清晰的基础。

2.1 威胁模型

我们考虑这样一种对手：通过暴露的服务或被盗用的凭据获得系统的**远程访问**，例如利用 SSH 等方式进行交互式登录。攻击者随后可能通过配置错误、利用应用程序漏洞或利用权限提升漏洞获得**根权限** 。但是，我们假设攻击者**没有** 对机器的物理访问权限。图~1 展示了这一场景：在传统 Linux 访问控制下，通过 SSH 获得的远程 root shell，与现场管理员打开的 Shell 实际上无法区分。

这一威胁模型反映了现代服务器和工作站环境中常见且与实际密切相关的失陷后场景：远程访问经常暴露在外，权限提升也仍然是现实风险。因此，我们关注的是在远程攻击者攻陷系统之后 限制其可执行的操作，而不是阻止系统本身遭到攻陷。

我们明确排除具有物理访问权限的攻击者、恶意内核或遭入侵的内核，以及硬件级攻击。与现有强制访问控制框架一样，OAMAC 假设内核和执行层可信。防御内核遭入侵或物理篡改与我们的目标无关，仍是系统安全领域尚未解决的问题。

2.2 安全目标

在此威胁模型下，OAMAC 旨在实现以下目标 .

首先，系统必须**在内核内部区分执行来源**，辨别进程是由用户物理在场、远程访问还是后台服务发起的。这种区分必须基于内核可见的执行来源，而非特定于应用程序或协议的语义。

其次，OAMAC 旨在**限制源自远程上下文的进程对安全敏感的内核接口的访问** 。其目的并非阻止所有恶意行为，而是在权限提升后，减少远程攻击者可实施的高影响操作。

与此同时，系统必须**保留正常的本地管理工作流**，包括对 `sudo` 的交互式使用，以及用户物理在场时对系统接口的直接访问。来源感知强制执行不应给合法的本地管理任务带来不便。

从部署角度来看，OAMAC 旨在**避免使用内核补丁或自定义模块**，完全依赖 Linux 上游机制。这使其能够在现有系统上实际部署，并最大限度地减轻维护负担。

在运行层面，系统必须**支持实时策略重新配置** ，无需系统重启或进程重启，并且必须**能够安全启动**，包括在初始化关键服务的系统启动早期阶段。

这些目标共同使 OAMAC 成为一种实用的入侵后攻击面缩减机制，可与现有访问控制系统互为补充，而非取而代之。

3 相关工作

在 Linux 上限制特权执行的理念，已从多个角度得到探索，包括传统的强制访问控制（MAC）框架、基于 eBPF 的安全与隔离系统，以及限制入侵后行为的机制。本节将 OAMAC 置于这一研究格局中。我们首先回顾现有 Linux MAC 机制如何依据身份、标签和客体进行判断，以及近期的系统化研究如何仍未将执行来源视为明确的授权属性。随后，我们讨论基于 eBPF 的安全监控、内核加固和可编程策略执行方法，并指出这些系统主要将 eBPF 本身或特定子系统视为受保护客体。最后，我们介绍有关入侵后遏制以及滥用合法接口进行“就地取材”的研究。在这些领域中，我们表明，先前的研究并未将执行来源（例如物理会话与远程会话，或服务执行）提升为一等安全抽象，而这正是 OAMAC 明确针对的空白。

3.1 Linux 中的强制访问控制

Linux 提供了多种成熟的强制访问控制（MAC）框架，其中最著名的包括 SELinux、AppArmor 和 SMACK [2, 16, 21]。这些系统为主体和客体关联标签、域或配置文件，并针对主体—客体关系强制执行策略。Linux 安全模块领域的大量研究强调了这种以身份和标签为中心的授权视角 [25]。近期对 Linux 强制访问控制执行机制（包括基于 eBPF 的 KRSI 扩展）的比较分析，也采用策略表达能力和性能等维度评估系统，但仍将决策建模为进程身份、标签、能力和客体属性的函数，并未将执行来源或进程溯源纳入明确的输入项 [4]。

在实践中，这意味着，除非管理员针对每个入口点手动设计独立的域或配置文件，否则从本地控制台启动的 Shell 与通过远程 SSH 会话获得的 Shell 会被等同对待。我们的工作并不提出另一种标记框架；相反，我们引入了**执行来源**，将其作为一个额外的授权维度，在概念上与这些现有的强制访问控制模型正交。

3.2 现有强制访问控制关于执行来源的局限性

现有的强制访问控制系统从根本上关注进程是什么（其身份、角色或标签）以及它执行哪个程序，而不是进程是如何产生的。在 SELinux 和 AppArmor 部署中，一旦进程进入某个域或配置文件，后续的访问决策便不再反映它是从物理在场的控制台、远程登录、后台服务还是控制平面代理启动的 [2, 16]。原则上，管理员可以间接编码来源——例如，为远程 Shell 和本地 Shell 复制域，并仔细管理转换——但除定制的加固工作外，这类配置并不常见。

造成这一差距的因素有三个。
首先，**策略复杂度**：编码来源需要在各个应用和入口点重复编写规则，导致组合式增长。

其次，**脆弱性**：随着系统演进，要在 `fork()`、 `exec()`和服务管理器之间维护来源特定的转换十分脆弱。

第三，**缺乏一等信号**：经典强制访问控制框架和基于扩展伯克利包过滤器的强制访问控制扩展，都无法将执行来源或会话来源作为原生策略属性公开，因此只能通过间接机制近似表示来源 [4, 25]。

因此，即使是近期对 Linux 强制访问控制机制的系统化研究，也没有发现任何一种设计会将执行来源——例如物理会话与远程会话，或服务执行——视为一等授权属性 [4]。OAMAC 明确针对这一缺失维度：在内核中对来源进行分类并传播，并使其可直接供授权逻辑使用。图~2通过对比身份、能力与作为正交授权维度的来源，总结了这一概念上的差距。

3.3 基于扩展伯克利包过滤器的安全、加固与遏制

eBPF 已成为实现安全内核可扩展性和内核内策略强制执行的标准机制 [18, 22]。Cilium 和 Tetragon 等生产系统分别使用 eBPF 来执行网络策略和提供运行时安全可观测性 [5, 6], ；此前的研究也已展示了基于 eBPF 的内核完整性监控和系统调用跟踪在安全分析中的应用 [9]。这些系统大多将 eBPF 用作灵活的机制 ，用于实现监控或钩子专用过滤；它们并未在现有强制访问控制模型所提供的基础上引入新的授权语义。

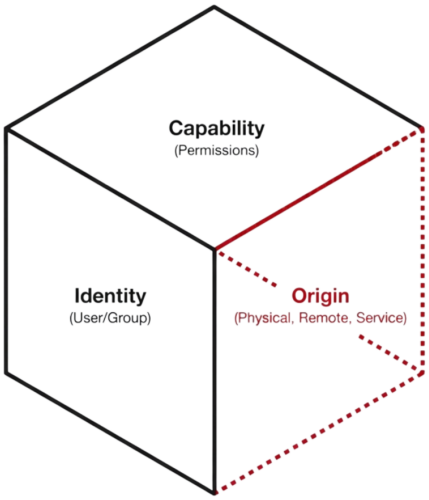


图2：Linux MAC模型中的概念差距。

另一类互补研究着重于**强化 eBPF 本身，将其视为攻击面**。近期研究（包括一篇全面探讨eBPF安全及防御措施的论文）对验证器缺陷、辅助函数滥用和 JIT漏洞进行了系统梳理，并提出了多种缓解措施，例如专用的 BPF LSM程序、运行时完整性检查和增强监控 [8]。其他研究，例如 SafeBPF，则通过软件故障隔离或硬件辅助标记等方式，对 eBPF 程序实施动态的内核内沙箱，即使存在验证器缺陷，也能保证空间内存安全 [14]。这些工作将 eBPF 视为受保护对象：它们限制 eBPF 程序可以执行的操作，以及访问内核内存的方式。相比之下，OAMAC 仅将 eBPF LSM 钩子用作**实现手段**；受保护资源是内核控制接口和系统路径，而策略的决定因素是执行来源，而非 eBPF 程序的内部行为。

有几个系统使用扩展伯克利包过滤器来构建**可编程MAC和隔离框架**。进程隔离框架将扩展伯克利包过滤器程序附加到多个钩子点，并查询各子系统的映射，以便针对文件系统和网络操作实施灵活的进程级策略[17]。SNAPPY 通过引入安全命名空间，将这一理念加以推广：容器和其他主体可以将自己的强制性eBPF策略加载到内核中，并借助动态辅助函数和可堆叠强制执行来支持丰富的状态化检查 [3]。这些设计最大限度地提升了**策略表达能力**：策略是按命名空间或应用程序编写的任意扩展伯克利包过滤器代码。相比之下，OAMAC 刻意避免采用可编程策略语言。它使策略**保持声明式且集中管理**，仅将一组规模小、语义稳定的来源（例如物理、远程、服务、引导/未知）作为授权的额外输入。隔离框架和 SNAPPY 关注的是“我们在这里应该运行哪些检查？”，而 OAMAC 关注的是“哪些来源应当能够访问此接口？”。

其他基于扩展伯克利包过滤器的系统则以**内核隔离和有状态行为强制执行**为目标。即时内核隔离方面的工作（例如 O2C）使用扩展伯克利包过滤器来强制实施软件故障隔离，并在内核中嵌入机器学习模型，以便在转换期间对对象类型进行分类，从而遏制易受攻击的内核组件 [24]。诸如 FG-RCA 这样的细粒度运行时遏制代理，会在初始阶段学习正常行为，生成最小权限允许列表，然后使用 eBPF LSM 钩子在运行时阻止此前未见过的操作 [1]。基于扩展有限状态机的有状态处理框架会跟踪

时间事件序列（例如 `memfd_create` → `execve`），并在观察到关键模式后主动阻止或终止进程 [20]。所有这些系统都以**行为为中心**：它们根据代码或进程随时间执行的操作来做出判断。

OAMAC 是**以来源为中心**的。它不跟踪时序行为，不学习允许列表，也不对内核组件实施沙箱隔离。相反，它利用内核可见信号，依据执行来源（物理控制台、远程登录、服务执行、控制平面辅助程序、引导）对进程进行分类，并将该来源跨 `fork()` 和 `exec()` 进行传播。随后，授权依赖于这一**因果谱系**，而非详细的行为历史。因此，SafeBPF、O2C、FG-RCA、基于有状态 XFSM 的强制执行，以及 SNAPPY 和 eBPF 隔离等可编程框架与 OAMAC **互补**：它们强化内核机制，或检测/排除危险行为，而 OAMAC 则从一开始就管控哪些执行上下文可以触及敏感接口 [1, 3, 14, 17, 20, 24]。

除这些基于内核和扩展伯克利包过滤器的框架之外，Optimus 专注于通过基于关联的动态系统调用过滤 [26]缩减容器化工作负载的攻击面。Optimus 跟踪程序执行阶段，并根据推断的权限要求启用或禁用系统调用，从而缩小敏感操作可用的时间窗口。与以系统为中心、基于执行来源约束访问的 OAMAC 不同，Optimus 以程序为中心，在进程的生命周期内细化权限。这两种方法互为补充：Optimus 限制何时 可以行使特权系统调用，而 OAMAC 则从一开始就约束此类特权操作从何处 才是可接受的。

3.4 入侵后与就地取材攻击

大量研究记录了攻击者在获得远程访问权限后如何滥用合法系统接口，这种做法通常被称为就地取材 [15]。常见的入侵后操作包括通过 `/proc/sys` 修改内核参数、写入 `/sys` 以改变内核或设备行为，以及利用 BPF 接口进行隐蔽持久化或权限提升 [10]。由于特权用户可以合法访问这些接口，传统访问控制和异常检测难以区分正常管理操作与恶意的失陷后行为。

现有的强制访问控制和基于扩展伯克利包过滤器的监控系统可以通过收紧策略或检测可疑访问模式来降低这种风险，但它们通常缺乏执行来源的概念 [2, 9, 16]。因此，获得特权 Shell 的远程攻击者往往能够使用与身处现场的管理员相同的高影响力控制路径。通过依据**执行来源**而非仅依据身份来强制实施访问控制，OAMAC 可直接应对这类攻击：即使在权限提升之后，也可以一概禁止远程来源进程访问内核控制接口和 BPF 控制平面操作，同时不影响现场管理操作。

3.5 正交与互补方法

还有其他机制旨在限制攻陷造成的影响，或从不同维度构建安全防护，例如 Linux 能力、命名空间与容器隔离，以及基于硬件的证明和可信执行环境 [7, 12, 13, 23]。这些机制主要约束进程拥有何种权限 或代码在哪里执行 （例如，在容器或飞地中）。来源感知的强制访问控制则从另一个维度发挥作用：它限制哪些执行上下文 可以使用某些强大的接口。结合这些维度——身份/标签、遏制、硬件信任根和执行来源——能够提供比单一机制更强大、更直观的防护。

在更高层，远程桌面系统和用户界面级隔离机制（例如向远程查看者隐藏敏感窗口的工具）强制执行的是关于**可见性和交互**的策略，而非内核可见的执行来源。从内核的角度来看，由此类工具驱动的进程仍在本地会话中执行，因此会被视为物理来源。OAMAC 有意关注执行权限 而非用户界面语义，使内核级强制执行不依赖于具体应用，并允许用户界面层控制作为独立防御机制并行发挥作用。

总而言之，先前的工作显著提升了强制访问控制表达能力、扩展伯克利包过滤器安全性、基于行为的遏制以及有状态检测，但尚未将**执行来源** 提升为一等授权属性。OAMAC 弥补了这一概念空缺：它使来源在内核中变得明确、持久且可强制执行，并展示了如何利用这一单一抽象实现实用的攻击面缩减，而无需引入新的策略语言或进行内核修改。

4 设计概述

现代操作系统已经能够提供丰富的执行来源信号，例如终端关联、会话上下文和进程祖先关系，但这些信号在授权模型中仍基本缺席。因此，现有访问控制机制主要判断进程是谁 （身份、标签或能力），而非它是如何 得以执行的。OAMAC（来源感知强制访问控制）将**执行来源** 提升为一等安全属性，明确区分用户物理在场、远程访问和后台服务执行等执行上下文，并将这种区分直接作为授权决策的输入。这样便能实现覆盖多个子系统的集中式来源感知治理，同时避免授权逻辑与特定应用或协议的语义纠缠在一起。

从较高层次来看，OAMAC 分为三个概念阶段运行。首先，在进程创建时，基于可观测的来源信号对执行来源进行**分类**。这些信号反映的是执行上下文，而非用户身份或网络状态，因此无需依赖特定协议假设即可推断来源。其次，来源一旦分配，便被视为**持久的执行元数据**，并在进程创建过程中传播，从而确保后代进程保留其祖先进程的来源，除非明确重新分类。第三，在访问仲裁期间查询来源元数据，以便在安全敏感的系统接口上强制执行**来源感知授权策略**。

图3从系统层面说明了这一模型。进程通过多个执行上下文进入操作系统，包括交互式本地会话、远程登录和服务执行。无论入口点如何，所有执行都关联有一个单一的来源标签，并在进程生命周期内保持稳定。强制执行集中在选定的中介点，从而使单一的来源感知策略能够一致地管理多个子系统。

OAMAC 的一项关键设计原则是**正交性**。来源并非旨在取代基于身份或基于标签的访问控制，而是通过引入额外的授权维度来补充现有机制。因此，策略可以表达诸如“仅当操作在本地发起时才允许执行”之类的约束，而无需枚举用户、角色或域。这种分离使 OAMAC 能够降低策略复杂度，同时保持与现有自主访问控制和强制访问控制框架的兼容性。

重要的是，该设计刻意避免对分类、传播与强制执行之间的执行顺序或控制流假设进行编码。这些机制作用于不同的系统事件，仅通过共享的来源状态在逻辑上相连。这种抽象使 OAMAC 能够独立于特定操作系统子系统或实现策略来推理执行来源。

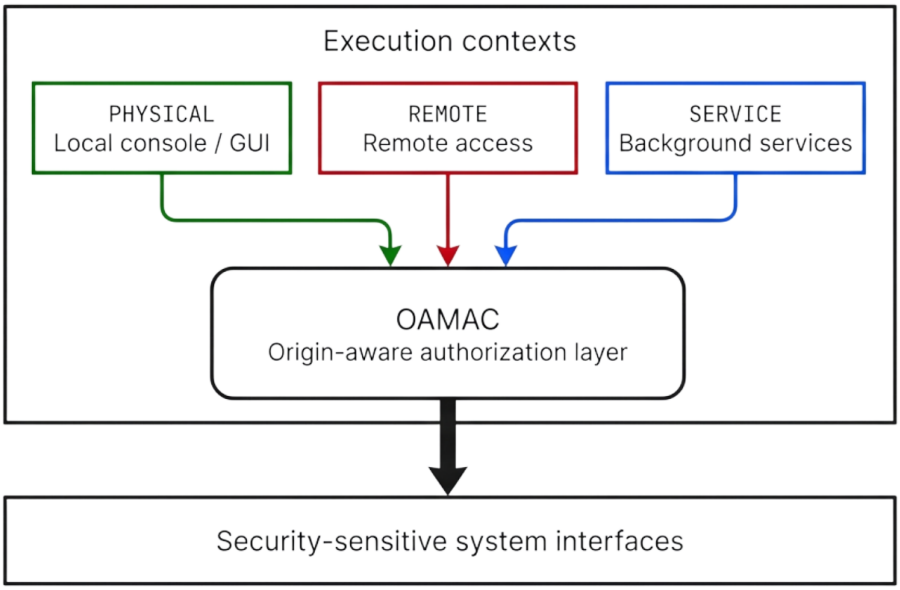


图3：来源感知的强制访问控制架构。

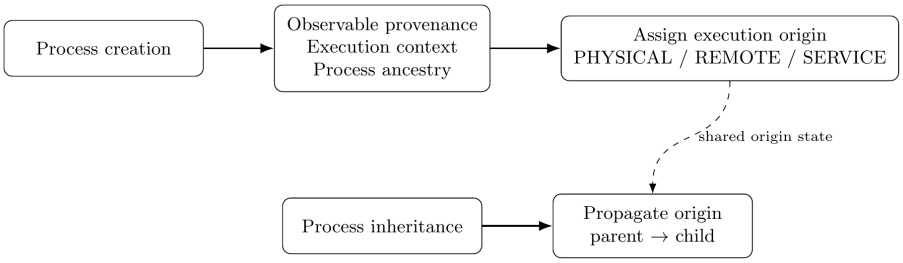


图4：来源分类与传播。

5 方法

本节将说明如何将 OAMAC 实现为一个具体的 Linux 原型，以及其核心机制在实际运行中的交互方式。我们首先介绍如何根据内核可见的来源信号推断执行来源，并在进程创建过程中传播该信息。随后，我们详细说明如何在文件系统和 BPF 控制平面中介点实现基于来源的强制执行，以及如何在不使用内核补丁或长期运行的特权守护进程的情况下，将这些检查集成到运行中的系统。最后，我们概述如何在真实的管理场景和失陷后场景中配置并应用策略，为后续评估奠定基础。

5.1 来源分类与传播

图4详细说明了用于对执行来源进行分类和传播的方法。来源分类发生在进程执行边界处，此时会检查内核可见的来源信号，以推断执行上下文。这些信号包括终端关联和进程祖先关系，两者结合后，系统无需依赖网络层或特定于应用程序的信息，即可区分交互式本地执行、远程会话和服务发起的进程。

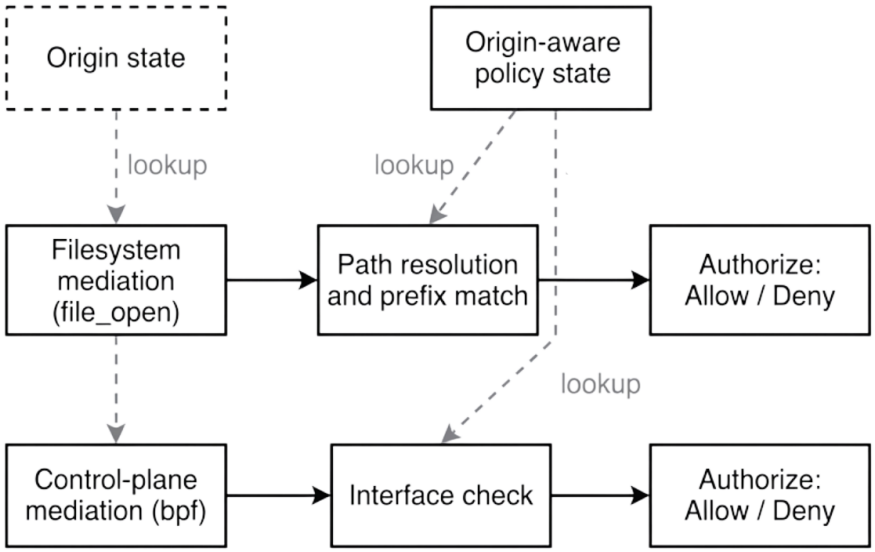


图5：集中式来源感知强制执行。

分类完成后，来源元数据会作为共享内核状态存储，并与进程标识符关联。传播在进程创建事件发生时独立处理，子进程会继承其父进程的来源。图4中的虚线依赖关系表示通过共享来源状态建立的逻辑关系，而非严格的执行顺序；这反映出分类和传播发生在不同的内核路径上，且可能由彼此独立的事件触发。

该方法确保每个执行上下文仅执行一次来源分类，并在整个进程树的生命周期内保持一致地复用该分类。因此，在攻陷或权限提升后生成的后代进程仍保留原始执行来源，从而防止通过进程创建进行来源洗白。

5.2 来源感知强制执行

图5展示了应用于选定内核中介点的强制执行方法。OAMAC 采用**集中式强制执行策略**，重点关注暴露高影响系统控制功能的接口，而不是试图拦截所有可能的访问路径。

在当前原型中，强制执行沿两个互补平面展开。首先，通过解析访问路径，并针对受保护集合执行基于前缀的匹配，对文件系统中介接口进行控制。这使 OAMAC 能够根据执行来源，限制对内核暴露的控制接口（例如配置路径和调试路径）的访问。其次，通过在特权系统接口上执行来源感知授权，对内核控制平面操作进行中介，从而实现对修改内核行为或全局系统状态的操作进行细粒度控制。

在这两种情况下，强制执行决策均由简单的按来源索引的策略查找得出。来源元数据和策略状态完全由内核驻留数据结构维护，因此无需用户空间参与即可作出授权决策。每次作出决策时都会更新审计计数器，从而能够查看获准和遭拒的操作，同时保持强制执行确定性。

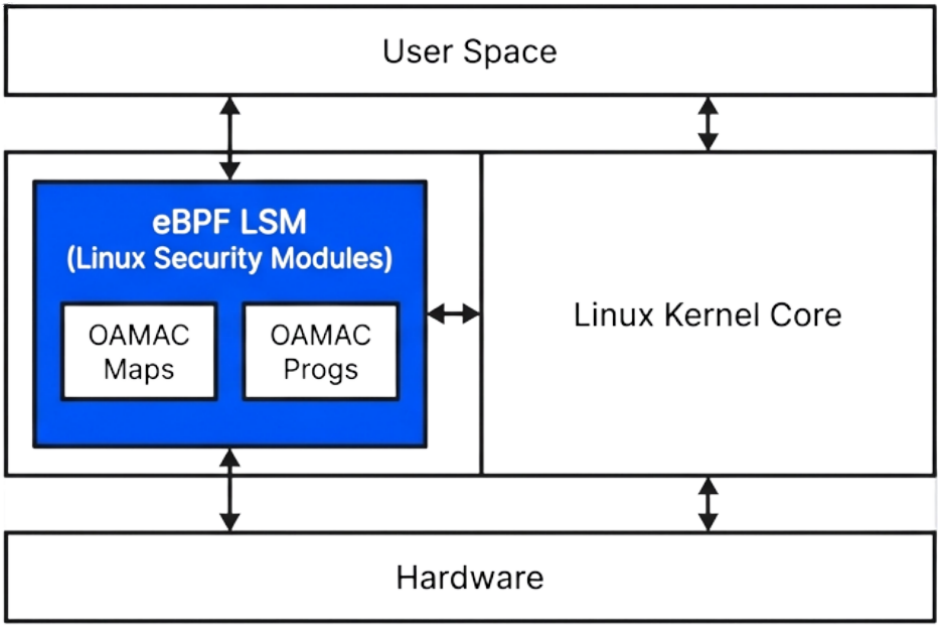


图 6：OAMAC 在 Linux 内核中的集成。

5.3 系统集成

如图~6所示，OAMAC 被集成为内核内强制执行层，在来源分类之后、执行敏感操作之前对执行过程进行中介。精简的用户空间接口仅用于策略配置和检查，不参与授权决策。这种设计确保强制执行能够抵御用户空间攻陷，并避免引入长期运行的特权守护进程。

通过将分类、传播和强制执行分离为彼此独立但相互协调的机制，OAMAC 实现了对多个攻击面的来源感知控制，同时保持了可信计算基精简且易于审计。

6 评估

本节评估 OAMAC 是否能在真实的 Linux 系统上正确、安全地实施来源感知强制访问控制，同时不干扰正常的管理工作或系统运行。我们不侧重于合成基准测试或微观性能测量，而是采用**行为和功能评估**，并使其与真实的入侵后和管理场景相契合。

我们的评估围绕三个问题展开：

1. OAMAC 能否正确区分执行来源并据此强制执行策略？2. 来源感知强制执行是否会干扰正常的本地管理或系统服务？3. 强制执行机制是否会带来可观测的开销或运行不稳定性？

6.1 实验设置

所有实验均在一台运行近期 Ubuntu 发行版且使用支持 eBPF LSM 的上游内核的 Linux 虚拟机上进行。系统配置如下：

- 在启动时启用 eBPF LSM
- `bpffs`挂载于 `/sys/fs/bpf`
- 无需内核补丁或自定义模块
- 无需修改 SELinux 或 AppArmor 策略

OAMAC 在启动时通过 `systemd` oneshot 服务自动加载。所有强制执行决策均完全通过 eBPF LSM 钩子在内核中执行。

我们评估了三种执行来源：

- **物理**：从本地控制台或图形界面终端启动的进程
- **远程**：通过 SSH 启动的进程
- **SERVICE**：没有关联 TTY 的系统服务

所有实验均在多次重启后重复进行，以确认启动时的稳定性和可复现性。在所有实验中，除非另有说明，我们都采用最小化的默认策略：禁止非物理执行环境进行高影响的内核控制，同时保留安全内省和本地管理。具体而言，允许远程和服务源通过 `/sys/kernel/btf` 只读访问内核 BTF 元数据，以支持可观测性，但禁止远程源访问范围更广的 sysfs 控制接口。OAMAC 配置目录（`/etc/oamac`）受到保护，远程源和服务源均无法访问，以防止在物理在场会话之外篡改运行时策略。在接口层面，凡是会改变内核行为的 BPF 控制平面操作——加载程序以及创建或更新映射——均禁止远程源和服务源执行。因此，启动后的策略更改只能通过用户态工具在物理在场会话中进行，而远程会话则仅保留只读的内核类型内省权限。

6.2 评估场景与结果

6.2.1 远程攻陷后场景

为模拟真实的失陷后环境，我们与系统建立了一个 SSH 会话，并以特权用户身份执行命令。在此远程环境中，我们尝试访问安全敏感的内核接口，包括：

- `/sys`
- `/sys/kernel`
- 选定的内核配置路径

观察结果。 启用拒绝远程访问的来源感知策略后，从 SSH 会话访问受保护路径的所有尝试均被内核以 `-EPERM` 拒绝。拒绝操作始终发生在强制执行边界处，早于任何子系统特定逻辑的执行。

结果。 即使远程来源进程以根权限运行，OAMAC 仍成功阻止其访问敏感内核接口。这证实了来源感知强制执行在权限提升后依然有效，并符合预期的失陷后威胁模型。

6.2.2 通过声明式例外实现策略灵活性

我们评估了 OAMAC 表达灵活策略的能力：广泛限制敏感接口，同时有选择地允许特定来源的安全、高价值例外。

- 设置。我们配置了广泛限制（例如， `path /sys deny remote`和 `path /sys denyservice`），并添加了有针对性的例外规则（例如， `path /sys/kernel/btf allow remote`和`path /sys/kernel/btf allow service`），以保留只读可观测性。
- 过程。从远程 SSH 会话和服务进程中尝试：
  - 读取 `/sys/kernel/btf/vmlinux`– 读取 `/sys` 下的另一路径（例如 `/sys/kernel/debug`）
- 观察结果。允许访问 `/sys/kernel/btf/*`，而其他 `/sys/*` 路径则被拒绝，返回 `-EPERM`。
- 结果。OAMAC 支持最小权限策略组合：管理员可以定义广泛的拒绝规则，并叠加范围狭窄且安全的例外规则，以满足运维需求（例如可观测性），同时避免重新开放影响重大的控制面。实现细节：由于策略按首匹配顺序进行评估，将例外规则置于广泛规则之前可确保达到预期结果。

6.2.3 本地管理场景

我们从本地图形界面终端执行了相同的命令，使用相同的用户身份和权限级别，其中包括使用 `sudo`。

观察结果。 所有操作均未受限制，且均成功完成。权限提升过程中，来源分类保持稳定；交互使用期间未观察到意外拒绝或行为变化。

结果。 OAMAC 未干扰合法的本地管理。这证实，来源感知策略能够限制远程攻击面，同时保留正常的本地工作流，包括特权操作。

6.2.4 服务稳定性与启动安全

我们评估了 来源感知强制执行是否影响系统服务或启动时执行 n.

观察结果。 启用 OAMAC 后，系统成功启动。标准系统服务继续正常运行，未观察到服务故障或启动回归。强制执行未影响非交互式工作负载。

结果。 OAMAC 可安全启动，且不会干扰服务执行，满足强制访问控制机制的一项核心部署要求。

6.2.5 Load锁定模式下的服务重启（SERVICE 源强制执行)

我们评估了重 启 systemd 加载器时，对 SERVICE 源应用严格接口拒绝规则 .

- 设置。默认策略包含 `iface bpf-prog-load deny service`, `iface bpf-map-creat e deny service`，以及 `iface bpf-map-update deny service`。加载器运行时没有控制 TTY，因此被归类为 SERVICE。
- 观察结果。在这些拒绝规则生效的情况下重新启动加载器导致单元失败，这符合预期：加载器的 BPF 操作在 lsm/bpf 强制执行点被阻止。
- 结果。这证实了分类（SERVICE）正确，并且针对源自服务的控制平面操作正确执行了强制执行。实际运维中，可以通过 PHYSICAL 会话进行维护：使用 `oamacctl` 临时移除 SER VICE BPF 拒绝规则，重启加载器，然后重新应用这些拒绝规则。

6.2.6 策略重新配置与控制平面保护

我们使用 `oamacctl`controltool 评估了动态策略更新。评估内容包括：

1. 为文件系统路径和 BPF 控制平面操作启用基于来源的限制 2. 验证限制立即强制执行，无需重启进程 3. 禁用限制并确认访问已恢复

此外，我们还安装了接口策略，禁止远程来源加载 BPF 程序和更新映射。

观察结果。 策略变更立即生效。远程来源进程无法加载新的 BPF 程序或修改策略映射，而物理现场会话仍保有完全控制权。

结果。 OAMAC 支持安全的实时重新配置，并可防止远程攻击者在入侵后削弱强制执行，即使他们暂时获得了根权限。

6.3 正确性验证

在所有场景中，均根据以下标准评估正确性：

- **来源分类准确性**：进程根据其执行上下文被持续标记为物理、远程或 SERVICE
- **强制执行正确性**：访问决策符合配置的来源感知策略
- **非干扰性**：允许的工作流继续正常运行
- **启动安全**：启用强制执行后，系统仍可正常启动

使用内核驻留审计计数器确认强制执行钩子按预期触发，并且允许和拒绝的操作均被正确记录。

6.4 开销与性能考量

尽管性能基准测试并非首要目标，我们仍在交互式使用过程中对开销进行了定性评估。

观察结果。未观察到明显的性能下降。强制执行仅在 LSM 钩子中增加了常数时间检查，且不会引入常驻的用户空间守护进程。
结果。OAMAC 带来的开销极小，在典型工作负载下不会影响系统响应性。

6.5 可复现性

OAMAC 的所有组件——包括驻留内核的 eBPF LSM 程序、用户空间控制工具和启动时集成脚本——都是确定且可复现的。本次评估所用的完整原型实现已作为开源软件发布，并可在网上公开获取 [19]。

只要启用了上游 eBPF LSM 支持，任何 Linux 系统都可以复现本评估，无需修改内核、安装自定义模块，也无需标准 eBPF 工具链之外的外部依赖。本文所述的所有策略配置、强制执行逻辑和控制平面操作，均与已发布的实现直接对应。

7 讨论

本节将讨论来源感知强制访问控制在本文评估的具体原型之外所具有的更广泛影响。我们将探讨 OAMAC 与现有访问控制机制的关系，分析设计权衡和部署考量，并思考如何将其底层抽象更深入地集成到操作系统内核中，以及如何将其扩展到其他环境。这些讨论有助于理解本文所作的设计选择，并阐明来源感知强制执行作为一种系统安全原语的优势与边界。

7.1 将内核集成的来源作为设计扩展

当前 OAMAC 原型利用上游 Linux 功能实现，无需修改内核，而是依靠 eBPF LSM 钩子和内核可见的执行来源在运行时对进程进行分类。这一设计选择优先考虑可部署性，使 OAMAC 能够在未经修改的系统上进行评估。不过，这也带来了固有的局限：在 OAMAC 初始化之前就已存在的进程（例如早期系统任务和启动时服务）无法被追溯分类，因此会被保守地分配为 `UNKNOWN` 来源。

为确保启动安全并避免干扰现有系统行为，当前设计默认不将来源感知策略应用于 `UNKNOWN` 来源。虽然这是一种安全且实用的选择，但它混淆了两种不同情况：必须保持不受限制的早期引导执行，以及来源确实模糊不清的后续执行上下文。这种混用限制了对未知或可疑执行上下文应用限制性策略的能力，因为这样做可能会干扰核心系统进程。

将 OAMAC 原生集成到内核中，可以妥善解决这一局限。在这种设计中，执行来源将作为内核进程元数据中的一等字段（例如位于 `task_struct` 中）表示，并在创建任务时分配。内核可以使用专用来源（例如 `BOOTSTRAP`）明确区分早期引导执行，该来源仅在系统初始化期间分配。此来源将不受来源感知强制执行的约束，从而从设计上确保启动安全。

系统达到稳定运行状态且来源感知策略生效后，将不再允许任何新任务进入 `BOOTSTRAP` 来源。此后创建的进程将根据可用的内核可见来源信息，被归类为 `PHYSICAL`、 `REMOTE`、 `SERVICE`或 `UNKNOWN` 。重要的是，这种隔离使 `UNKNOWN` 来源能够受到策略控制并可在必要时加以限制，同时不会影响早期系统任务，也不会破坏系统启动语义。

除了消除引导阶段的分类歧义之外，内核集成式设计还带来了其他好处。来源传播变得不变且无竞争，授权检查不再依赖辅助数据结构，并且来源信息可以统一提供给所有内核授权机制。更广泛地说，将执行来源提升为一等内核属性，能够明确其语义，并使跨子系统对来源感知策略进行有原则的推理成为可能。

尽管本工作并未实现此类内核修改，但我们认为，将执行来源引入内核的核心进程表示，是对现有安全模型自然而有价值的扩展。所提出的设计保留了当前原型在可部署性方面的优势，同时为更强的语义、更细粒度的策略控制，以及与现有强制访问控制框架更清晰的集成，勾勒出明确的发展路径。

7.2 与现有强制访问控制系统的关系

OAMAC 在适用范围和抽象层级方面都不同于 SELinux 和 AppArmor 等传统强制访问控制系统。现有强制访问控制框架主要通过为主体和客体关联标签、域或配置文件，并制定用于判断进程是谁或执行哪个程序的策略来规范访问。尽管这些机制功能强大且表达能力丰富，但它们需要进行显式标记，并且通常需要复杂的、针对特定子系统的策略工程。

相比之下，OAMAC 不会为文件或进程引入新标签，也不会定义新的策略语言。相反，它将**执行来源**作为额外的授权维度引入，与基于身份和标签的控制相互独立。来源感知策略关注执行是如何发起的——例如，进程是否源自用户物理在场、远程访问或后台服务执行——而非仅关注应用程序身份。

重要的是，OAMAC 并非旨在取代现有的强制访问控制框架。其设计明确与之正交：来源感知强制执行可以与 SELinux 或 AppArmor 共存，并且可以独立于它们的策略应用。由于 OAMAC 无需重新标记客体或重构现有策略，因此可以与成熟的强制访问控制系统并行部署，在无需策略重构或内核修改的情况下约束执行上下文。

7.3 输入设备与物理在场

当前的来源分类不会直接区分不同类型的物理输入设备（例如，内置键盘、USB 键盘、虚拟键盘），而是依赖内核可见的执行上下文——具体而言，是 TTY 关联和进程祖先关系。

从安全角度来看，这是经过审慎考虑的设计选择。Linux 内核不会可靠地向安全钩子公开输入事件的来源（物理或虚拟），而试图根据输入硬件推断信任，则会引入脆弱且依赖特定平台的假设。

重要的是，我们的威胁模型将本地内核执行上下文 视为信任边界，而非输入设备本身。无论命令是通过物理键盘、USB 设备还是无障碍接口输入，最终产生的进程仍由同一本地 TTY 和进程层级进行管理。

我们认为，来源感知的强制访问控制应依据执行来源进行判断，而非依据人机交互方式。这样可以确保设计稳健、可审计，并与现有内核抽象兼容。

7.4 推广至其他操作系统

尽管 OAMAC 的当前实现面向 Linux，但来源感知访问控制这一基本概念并不局限于特定操作系统。大多数现代操作系统已经隐式维护执行来源信息。例如，Windows 会区分交互式登录、服务执行和远程会话；macOS 会跟踪控制台会话和远程登录上下文；移动操作系统则会区分前台用户操作和后台服务。这些区别通常用于会话管理和审计，但很少作为一等输入提供给内核级授权决策。

通过将执行来源定义为进程的一项显式且持久的属性，可以将来源感知设计推广到各种平台。这需要定义一组规模小且固定的来源类别，例如引导、物理、远程、SERVICE 和未知；在进程创建时自动传递来源元数据；并允许安全策略除了依赖传统的基于身份的属性外，还依赖来源。重要的是，这种设计不需要

重新思考现有的访问控制模型，而是在其基础上增加一个独立的授权维度。

我们认为，授权层缺少执行来源，是当代操作系统安全模型中的结构性缺口。将执行来源提升为一等安全属性，可以制定表达力更强、更直观的策略，并提供一种统一抽象，用于跨子系统和平台治理执行上下文。

7.5 以用户为中心的治理与集中式安全推理

系统安全领域反复面临的一项挑战是，有效防护往往需要深入了解特定子系统。要保护文件系统、内核接口、设备节点、网络协议栈和运行时服务，通常需要详细掌握各组件的威胁模型和配置语义。这种复杂程度超出了普通用户，甚至许多高级管理员所能合理承担的范围。

我们的设计有意将安全推理从各个子系统转向基于执行来源的*以用户为中心的治理层*。用户无需理解每个受保护接口的安全影响，而是可以表达更简单、更直观的意图：**哪些子系统应允许哪些执行来源访问**。

对许多用户而言，尤其是在服务器和工作站环境中，这种意图十分明确。用户通常不需要远程访问敏感内核接口、设备控制路径或系统配置子系统。通过允许用户声明此类子系统仅可由物理在场会话访问，系统便能以极低的策略复杂度，大幅限制一大类潜在攻击面。

这种以来源为中心的抽象带来了多项优势：

1. **降低认知负荷**
   用户不再需要推理每个独立子系统的安全属性，而是转而考量对执行上下文（本地、远程或服务）的信任，这更符合人类直觉和运维实践。

2. **大幅缩减攻击面**
   由于许多影响重大的入侵后攻击都依赖于访问可通过 `/sys`、`/proc/sys`、设备节点或类似控制平面触达的内核和系统接口，按来源限制这些路径，便能显著减少远程攻击者可采取的行动。

3. **集中监控点**
   所有受保护子系统均由一个**逻辑上集中式的**来源感知强制执行层进行调控。因此，监控该层的健康状态和行为，可提供一个集中式可见性点。任何试图禁用、绕过或修改基于来源的策略的行为，都是针对受保护的系统内部组件的持续进行的攻击的强烈迹象。

4. **组合式安全保证**
   尽管没有任何单一机制能够保证绝对系统安全，来源感知强制执行仍能带来一种强大的组合性质：**只要来源层保持正确且未被攻陷，所有受其保护的子系统在执行来源方面都会继承其安全保证**。这使用户和管理员能够在更高层次的抽象上对系统安全进行推理。

5. **面向真实用户的实用安全**
   该设计认识到，安全机制只有在易于使用和理解时才有效。通过提供简单、集中的控制平面，该系统使用户无需具备专门的领域知识，也能实施有意义的安全限制。

我们认为，这种从子系统特定加固转向基于来源的治理的变化，是一种切实可行且可扩展的方法，可减少现代操作系统中的攻击面。

7.6 区分 CloudControl 平面访问

在云环境中，并非所有远程访问都相同。云服务提供商通常通过经过身份验证的控制平面接口提供管理访问，这与用户发起的直接 SSH 访问有着本质区别。

我们的设计将控制平面访问视为独立的执行来源，以此纳入这种区别。系统不试图根据网络级属性推断信任，而是依赖可信组件提供的明确信号，例如 PAM 模块、系统服务或提供商专用代理。

例如，云服务提供商可以通过其控制面板对用户进行身份验证，并启动一个会话，明确标记由此产生的进程源自控制平面。该标记可通过可信系统组件（例如经过身份验证的服务或提供商专用代理）传递给内核，使来源感知强制执行能够区分控制平面会话与任意远程登录。

这种方法使云运营商能够出于管理目的，授予对敏感子系统有限且可审计的访问权限，同时严格限制一般远程访问。重要的是，它保留了集中式治理模型：用户和提供商从来源层面而非单个子系统层面来判断信任。

7.7 处理非 SSH 远程访问

我们的设计不依赖 SSH 特有的语义，而是根据内核可见的来源信息（包括终端关联和进程祖先关系）对执行来源进行分类。

因此，各种远程访问机制——包括反向 shell、基于 Web 的管理界面、容器 exec 操作以及代理介导的访问——都可以自然地映射为非物理来源。在许多情况下，这些机制的分类比交互式 SSH 会话更严格，因为它们缺少物理终端上下文。

这种抽象避免了特定协议假设，并确保即使远程访问技术不断演进，来源感知强制执行依然有效。

8 局限性

尽管来源感知访问控制提供了一种切实有效的机制来减少攻击面，但它也存在若干需要明确认识的局限性。

8.1 对内核和执行层的信任

我们的设计假设内核完整性得到保障，且来源感知执行机制正确无误。与所有强制访问控制系统一样，成功的内核级漏洞利用、Rootkit 或任意内核内存损坏都可能绕过或禁用强制执行逻辑。

这一局限并非我们的方法所独有，现有的强制访问控制框架（如 SELinux 和 AppArmor）也同样存在。防御内核遭入侵与我们的目标无关，仍然是系统安全领域的一项开放挑战。

8.2 来源分类的粒度

当前原型将执行来源分类为一组数量较少且固定的类别（物理、远程、SERVICE、控制平面）。虽然这种粗粒度设计是有意为之，也足以表达许多实际策略，但它可能无法涵盖复杂环境中所有细微的信任区分。

例如，远程图形会话、嵌套虚拟化或高度定制的访问路径，可能需要进一步细化或明确标记。我们认为，这并非根本性限制，而是策略和部署方面的考量。

8.3 对正确控制平面信令的依赖

要区分特权控制平面访问与任意远程访问，需要受信任的系统组件配合，例如 PAM 模块、系统服务或提供商专用代理。如果此类组件配置错误或遭到攻陷，来源分类可能会受到影响。

不过，这项要求符合受管环境中现有的信任假设：在这类环境里，控制平面软件本就属于可信计算基的一部分。

8.4 保护范围

来源感知强制执行根据执行来源限制对受保护子系统的访问。它无法阻止远程攻击者实施的所有形式的恶意行为。例如，完全在未受保护子系统内运行的攻击，或利用应用层漏洞实施的攻击，均不在其范围之内。

我们的方法最好被理解为一种攻击面缩减机制，而不是全面的入侵防御系统。

8.5 硬件与输入信任假设

该系统不试图区分不同类型的物理输入设备（例如，内置键盘与 USB 设备）。相反，它依赖内核可见的执行上下文作为信任边界。虽然这简化了设计并提高了稳健性，但它假设本地执行上下文比远程执行上下文更值得信任；这一假设与许多现有的安全模型一致。

8.6 跨操作系统的可移植性

尽管来源感知访问控制的概念与操作系统无关，我们的实现以 Linux 上的 eBPF LSM 为目标。要将该机制移植到其他操作系统，需要与各自的内核授权框架和进程元数据模型集成。

尽管如此 ，我们认为其底层抽象具有通用性，也适用于 Linux 以外的平台 .

8.7 未来工作

尽管我们的原型展示了基于来源的访问控制的可行性和实际价值，但仍有若干扩展方向有待未来探索。

内核控制接口的来源感知中介。 我们的原型已将来源感知中介应用于部分内核控制接口，即通过 bpfLSM 钩子中介 BPF 系统调用，**展示了控制平面中介的可行性**。一个自然的扩展方向是将这一方法推广至其他特权系统调用（例如 `perf_event_open` ）以及其他控制平面入口点。在这些接口上实施基于来源的限制，将进一步约束入侵后的攻击面，尤其是那些用于隐蔽内核内省或操纵的攻击面。

网络层来源传播。 在分布式和容器化环境中，执行来源可能跨越多个主机或编排层。未来工作可以探索如何在网络边界之间传播来源元数据，例如从云控制平面传播到受管理节点，从而使来源感知强制执行能够区分通过可信管理通道发起的操作与源自不可信远程会话的操作。

与现有强制访问控制框架集成。 尽管我们的设计作为一个正交的治理层运行，但与 SELinux 或 AppArmor 等现有强制访问控制框架进行更紧密的集成，仍是一个有待探索的方向。将执行来源作为一等安全属性提供给这些系统，可以实现混合策略，将细粒度标记与粗粒度来源约束相结合，同时保留来源感知强制执行带来的集中式推理优势。

来源感知策略语义的形式化。 我们当前的策略模型有意保持简单且命令式。未来工作可以为来源感知规则定义一种形式化策略语言，以支持静态分析、冲突检测以及对基于来源的安全不变量进行验证。此类语言还有助于推理策略与现有访问控制机制的组合。

通过硬件支持增强来源信号。尽管当前原型是根据内核维护的软件信号（例如终端关联和进程祖先关系）推断执行来源，但未来的系统可以受益于**硬件辅助支持，以保护来源元数据**。关于硬件辅助保护的既有研究指出，安全关键元数据应受到完整性保护、能够被高效访问，并与执行状态紧密耦合，从而抵御篡改并降低强制执行开销 [11]。OAMAC 中的执行来源自然符合这一模型。

硬件支持可以通过两种互补方式增强来源分类。首先，硬件机制可以保护来源元数据本身，确保来源在任务创建时一经指定，就不会被受入侵的软件组件伪造或篡改。其次，硬件辅助信号可以提供更可靠的溯源输入，例如可信的物理在场指示器、安全控制台设备，或用于云控制平面访问的经证明的管理通道。如果通过可信接口向内核开放，这类机制便能实现无法伪造的来源归属，同时与本文提出的来源感知强制执行模型完全兼容。

9 结论

我们提出了一种来源感知强制访问控制机制，将**执行来源**提升为 Linux 内核中的一等安全属性。通过区分用户物理在场、远程访问和服务执行，我们的方法能够对安全关键子系统实施集中式治理，并减少仅凭身份或基于标签的机制难以控制的入侵后攻击面。

我们的设计有意与现有强制访问控制框架保持正交。来源感知强制执行并非要取代 SELinux 或 AppArmor 等系统，而是与它们互为补充

它通过限制特权操作 的来源，不受特定于应用程序的策略影响。这样的分离使管理员能够表达高层次的安全意图，而无需深入了解子系统或投入大量策略工程工作。

我们证明了，利用 eBPF LSM 框架和 Linux 上游功能，如今即可实现来源感知访问控制，无需修改内核，且仅引入极低的开销。除了强制执行之外，来源层还提供了一个统一且可监控的控制点；其完整性与受保护子系统的安全性高度相关，因此既能提供切实保护，也能增强运行可见性。

综上，我们的结果表明，执行来源是当代操作系统安全模型中缺失的一种抽象。提升这一维度，有助于切实缩减攻击面，并为未来系统中将来源感知强制机制与现有访问控制机制相结合开辟新的方向。

参考文献

[1] Fouad Ailabouni、Jesús-Ángel Román-Gallego 和 María-Luisa Pérez-Delgado。Fg-rca：基于内核的物联网事后利用遏制，结合策略综合与零日攻击缓解。物联网，7(1)，2026年。ISSN 2624-831X。doi：10.3390/iot7010003。URL https://www.mdpi.com/2624-831X/7/1/3。

[2] AppArmor Project。Apparmor security project。https://www.kernel.org/doc/Documentation/security/apparmo 2018年。

[3] Maxime Bélair、Sylvie Laniepce 和 Jean-Marc Menaud。《SNAPPY：面向容器的可编程内核级策略》。载于第 *36* 届 *ACM* 应用计算年会论文集，SAC '21，第 1636–1645 页，美国纽约州纽约市，2021 年。计算机协会。ISBN 9781450381048。doi：10.1145/3412841.3442037。URL https://doi.org/10.1145/3412841.3442037。

[4] Brennon Brimhall、Justin Garrard、Christopher De La Garza 和 Joel Coffman。《Linux 强制访问控制策略强制机制的比较分析》。载于第 *16* 届欧洲系统安全研讨会论文集，EUROSEC '23，第 1–7 页，美国纽约州纽约市，2023 年。计算机协会。ISBN 9798400700859。doi：10.1145/3578357。3589454。URL https://doi.org/10.1145/3578357.3589454.[5] Cilium Project。《Cilium：基于扩展伯克利包过滤器的网络与安全》。https://cilium.io，未注明日期。[6] Cilium Project。《Tetragon：基于扩展伯克利包过滤器的运行时安全》。GitHub 代码库，未注明日期。URLhttps://github.com/cilium/tetragon。访问日期：2026年-01-20。[7] Victor Costan 和 Srinivas Devadas。《英特尔 SGX 详解》。技术报告，Cryptology ePrint Archive，报告 2016/086，2016 年。URL https://eprint.iacr.org/2016/086.pdf。

[8] Vincenzo Costanzo。 *Linux* 内核中扩展伯克利包过滤器安全风险的检测与缓解。博士论文，都灵理工大学，2025年。

[9] Guillaume Fournier、Sylvain Afchain 和 Sylvain Baubeau。使用扩展伯克利包过滤器进行运行时安全监控。载于 *SSTIC2021*，2021。URL https://www.sstic.org/media/SSTIC2021/SSTIC-actes/runtime_security_with_ebpf/SSTIC2021-Article-runtime_security_with_ebpf-fournier_afchain_baubeau.pdf。

[10] Zhibin He 等。跨容器攻击：云上的扩展伯克利包过滤器令人困惑。载于 第 *32* 届 *USENIX* 安全研讨会论文集，2023。URL https://www.usenix.org/system/files/.[11] Yue Jin、Tian-Yi Huang、Si-Yuan Zeng、Yi-Bin Xu、Han Wang、Tian-Yue Lu 和 Ming-Yu Chen。硬件辅助的地址空间内保护综述。计算机科学与技术杂志，40(5):1347–1367，2025 年 9 月。ISSN 1860-4749。doi：10.1007/s11390-025-4827-z。URL https://doi.org/10.1007/s11390-025-4827-z.[12] Michael Kerrisk 和贡献者。capabilities(7) — Linux 能力。man7.org 手册页，日期不详。URL https://man7.org/linux/man-pages/man7/capabilities.7.html。访问日期：2026-01-20。[13] Michael Kerrisk 和贡献者。namespaces(7) — Linux 命名空间概述。man7.org 手册页，日期不详。URL https://man7.org/linux/man-pages/man7/namespaces.7.html。访问日期：2026-01-20。[14] Soo Yee Lim、Tanya Prasad、Xueyuan Han 和 Thomas Pasquier。SafeBPF：为扩展伯克利包过滤器内核扩展提供硬件辅助的纵深防御。载于*2024* 年云计算安全研讨会论文集，CCSW '24，第 80–94 页，纽约，NY，美国，2024。计算机协会。ISBN 9798400712340。doi：10.1145/3689938.3694781。URL https://doi.org/10.1145/3689938.3694781.[15] LOLBAS 项目。Lolbas：借助系统自带的二进制文件、脚本和库，日期不详。URLhttps://lolbas-project.github.io/。访问日期：2026-01-20。[16] Peter Loscocco 和 Stephen Smalley。安全增强型 Linux。NSA 技术报告，2001。[17] Carlo Mazzocca、Andrea Garbugli、Michele Armillotta、Rebecca Montanari 和 Paolo Bellavista。使用扩展伯克利包过滤器实现灵活且安全的进程隔离。载于 Fabio Martinelli 和 Ruben Rios 编，安全与信任管理，第 97–109 页。施普林格自然瑞士，Cham，2025。ISBN 978-3-031-76371-7。[18] Samantha Miller、Kaiyuan Zhang、Danyang Zhuo、Shibin Xu、Arvind Krishnamurthy 和 Thomas E. Anderson。实用的安全 Linux 内核可扩展性。载于第 *17* 届操作系统热点研讨会论文集（*HotOS '19*）。计算机协会，2019。doi：10.1145/3317550.3321429。URL https://dl.acm.org/doi/10.1145/3317550.3321429。[19] Omer Abdelmajeed Idris Mohammed。OAMAC：面向 Linux 的来源感知强制访问控制。https://github.com/omeroooor/oamac，2026年。[20] Arianna Quinci、Giacomo Belocchi、Francesco Quaglia 和 Giuseppe Bianchi。关键事件的有状态处理：利用扩展伯克利包过滤器实现扩展有限状态机抽象，2025。[21]Casey Schaufler。嵌入式计算中的 SMACK。载于渥太华 *Linux* 研讨会（*OLS*）论文集，第 179–186 页，2008。URL https://landley.net/kdocs/ols/2008/ols2008v2-pages-179-186.pdf。[22] Alexei Starovoitov。LLVM 和内核中的 BPF。载于*Linux Plumbers* 大会，2015。URL https://blog.linuxplumbersconf.org/2015/ocw/sessions/3249.html。

[23]可信计算组。TPM 2.0 库规范，第 1 部分：架构。规范，2019 年。URL https://trustedcomputinggroup.org/resource/tpm-library-specification/。访问日期：2026-01-20。[24] Zicheng Wang、Tiejin Chen、Qinrun Dai、Yueqi Chen、Hua Wei 和 Qingkai Zeng。扩展伯克利包过滤器与机器学习相遇：实时操作系统内核隔离，2024 年。URL https://arxiv.org/abs/2401.05641。[25] Chris Wright、Crispin Cowan、Stephen Smalley、James Morris 和 Greg Kroah-Hartman。Linux 安全模块：Linux 内核的通用安全支持。载于第 *11* 届 *USENIX* 安全研讨会（*USENIX Security 02*），2002 年。[26] S. Yang、B. B. Kang 和 J. Nam。Optimus：用于容器攻击面缩减的基于关联的动态系统调用过滤。《云计算期刊》，13:71，2024 年。doi：10.1186/ s13677-024-00639-3。URL https://doi.org/10.1186/s13677-024-00639-3。