# 面向物联网设备安全的多层次内核访问控制方法

詹东阳[1], 俞兆丰[1], 叶 麟[1], 张宏莉[1]

[1]哈尔滨工业大学网络空间安全学院 哈尔滨 中国 150001

**摘要** 物联网设备受能耗、计算能力等因素限制, 通常采用轻量化的操作系统以及精简化的安全保护机制, 导致物联网设备的操作系统安全保护能力不足, 更容易被用户态程序攻破。为了增强操作系统的隔离能力, 现有的安全保护方法通常限制应用程序可访问的系统调用种类, 使其仅能访问运行所必须的系统调用, 从而缩小操作系统的攻击面。然而, 现有的动态或者静态程序分析方法无法准确获取目标程序运行所依赖的系统调用。动态跟踪方法通过跟踪程序执行过程中触发的系统调用, 仅能获取程序依赖系统调用的子集, 以此作为依据的访问控制可能会影响程序的正常执行。而静态分析方法通常构造程序及其依赖库的控制流图并分析其可达的系统调用, 然而由于静态分析无法精准构建控制流图, 仅能获取目标程序依赖系统调用的超集, 会在访问控制中引入多余的系统调用, 造成操作系统攻击面依然较大。针对现有系统调用访问控制面临的可用性以及精准度问题, 研究多层次的内核访问控制方法, 在现有系统调用访问控制的基础上, 引入了动态链接库的访问控制, 并提出了多层联动的动态安全分析机制, 以动态分析的方法排除由于静态分析不准确引入的额外系统调用, 从而进一步缩小物联网系统的攻击面, 提升物联网设备的隔离能力与安全性。实验结果表明, 相比于现有内核访问控制方法, 本文提出的方法能够抵御更多漏洞而且引入的实时负载更低。

**关键词** 物联网系统安全; 攻击面缩小; 静态程序分析; 动态访问控制; 多层访问控制

**中图法分类号** TP309 DOI 号 10.19363/J.cnki.cn10-1380/tn.2022.11.08

# Multi-layer Kernel Access Control Method for Internet of Things Device Security

ZHAN Dongyang[1], YU Zhaofeng[1], YE Lin[1], ZHANG Hongli[1]

[1] School of Cyberspace Science, Harbin Institute of Technology, Harbin 150001, China

**Abstract** IoT devices are limited by factors such as energy consumption and computing power, and usually use lightweight operating systems and simplified security protection mechanisms. As a result, IoT devices have insufficient operating system security protection capabilities and are more likely to be compromised by user-mode programs. In order to enhance the isolation capability of the operating system, the existing security protection methods usually limit the numbers of system calls accessible to applications, so that they can only access system calls necessary for running, thereby reducing the attack surface of the operating system. However, the existing dynamic or static program analysis methods cannot accurately obtain the necessary system call set of the target program. The dynamic tracking approaches can only obtain a subset of the program-dependent system calls by tracking the system calls triggered during the execution of the program, and the access control based on this may affect the normal execution of the program. The static analysis approaches usually construct the control flow graph of the program and their dependent libraries and analyze their reachable system calls. However, because static analysis cannot accurately construct the control flow graph, it can only obtain the over-approximated set of the target program's dependent system calls, which will introduce extra system calls in access control and result in a large attack surface of the operating system. Aiming at the usability and accuracy problems faced by the existing system call access control, a multi-level kernel access control method is studied. Based on the existing system call access control, the dynamically-linked library access control is introduced, and the multi-level correlation dynamic security analysis is proposed. The dynamic verification mechanism uses dynamic analysis to eliminate additional system calls introduced by inaccurate static analysis, thereby further reducing the attack surface of the IoT systems and improving the isolation capability and security of IoT devices. The experimental results show that, compared with the existing kernel access control methods, our approach can mitigate more vulnerabilities and introduce lower overhead.

**Key words** IoT system security; attack surface reduction; static program analysis; dynamic access control; multi-layer access control

**通讯作者:** 叶麟, 博士, 副教授, Email: hityelin@hit.edu.cn。

本课题得到国家重点研发计划资助项目(No. 2021YFB2012402), 国家自然科学基金资助项目(No. 61872111)资助。

收稿日期: 2022-06-29; 修改日期: 2022-08-24; 定稿日期: 2022-09-05

## 1 引言

近年来, 物联网快速发展, 已经成为最重要的计算平台之一, 大量的物联网设备被部署在多种应用场景下, 如工业互联网、无线传感器网络、智能家居等[1]。然而, 物联网设备的计算能力通常较弱, 而且对能耗有较高的要求, 导致了物联网系统通常以较轻量化的配置运行, 使其安全能力较弱[1-2]。此外, 物联网系统内运行的安全保护程序同样受到计算能力和能耗等因素影响, 安全保护能力受到限制[3]。

在安全保护能力较弱的环境下, 保护物联网设备的操作系统十分重要[4]。系统调用是用户空间程序与操作系统之间最大的接触面, 用户空间程序能够利用大量系统调用中的漏洞(如 CVE-2017-7308、CVE-2017-5123 等)攻击操作系统, 以实现进一步的权限提升等攻击[5]。因此, 隔离物联网设备中的应用程序与操作系统是增强物联网系统安全的重要部分。

Linux Seccomp 能够用于增强应用与操作系统之间的隔离性, 它是 Linux 内核的一个安全模块, 能够阻止应用程序访问的系统调用类别。基于 Seccomp 限制用户态程序仅能访问其运行依赖的必要系统调用, 可以防止其他系统调用中包含的漏洞被利用, 从而提升内核的隔离能力。Eclipse IoT 开发者调查报告表明 Linux 是目前最流行的物联网设备操作系统[6], 因此, Seccomp 经常被用于增强物联网设备中操作系统的隔离能力[7]。此外, Seccomp 是 Linux 内核中的内嵌模块, 通常引入的开销较低, 能够满足物联网设备的性能需求[5]。

现有的分析程序运行所依赖系统调用的方法通常分为动态跟踪以及静态分析两类。动态跟踪方法通常跟踪并收集目标程序在执行过程中触发的系统调用, 基于已触发的系统集合构建目标程序运行依赖的系统调用库[8-10]。然而, 动态执行无法遍历目标程序所有的执行路径, 因此会导致动态收集的依赖系统调用集合不全, 以此为依据生成的 Seccomp 访问规则由于可能会缺乏必要的系统调用而影响目标程序的正常运行。绝大多数程序是通过动态链接库(如: glibc 等)触发系统调用的, 基于静态分析的方法通常分析目标程序的代码或者二进制文件, 识别其依赖的动态链接库 API, 之后建立动态链接库 API 到系统调用的映射, 从而分析程序依赖的系统调用集合[11-14]。然而, 受到控制流分析过程中间接跳转分析精度较差的影响, 静态分析方法仅能构建动态链接库中 API 到系统调用的超集, 导致得到的依赖系统调用集合中存在无关的系统调用, 使得访问控制不够精准, 无法最大化隔离操作系统。综上, 现有的系统调用依赖分析方法面临着可用性以及精准度的问题。针对这个问题, 文献[5]在静态分析的基础上动态验证动态链接库中涉及间接跳转的执行路径的安全性, 在一定程度上解决了间接跳转分析不准确引入无关系统调用问题。然而, 该方法依然面临着性能开销以及全面性的问题。

为了实现物联网环境下高效精准的内核访问控制, 本文提出一种多层次的内核访问控制方法, 通过分别在动态链接库以及系统调用两个层次构建关联化的访问控制机制, 不仅能对系统调用进行访问控制, 而且能够限制程序访问的动态链接库的 API 以及函数, 提升对内核系统调用的攻击难度。首先, 研究面向 glibc 等动态链接库的高效的控制流分析技术, 分析动态链接库中的控制流图, 并构建 API 到系统调用的映射。之后, 分析目标程序依赖的动态链接库 API 集合, 从而基于 API 到系统调用映射分析出程序运行所依赖的系统调用集合, 并生成 Seccomp 配置, 在系统调用层面实现访问控制。此外, 基于依赖的 API 集合以及控制流图删除静态链接库中无用的 API 以及函数, 在动态链接库层面实现访问控制, 使攻击者无法利用无用的代码实现内核攻击。最后, 构建高效的动态分析机制, 实现关联化的访问控制。对于允许执行的系统调用, 在内核系统调用入口基于用户栈还原对应的动态链接库 API 和栈深度, 并匹配系统调用以及 API 的关联性, 从而识别并阻断非程序逻辑触发的系统调用。

本方法从应用的动态链接层以及系统调用层两个层次对应用进行内核访问控制, 并且引入了动态分析方法进行两个层次的关联化访问控制。相较于现有的仅包含系统调用的内核访问控制方法, 本方法能够在不同的层次上抵御更多面向操作系统内核的安全威胁。首先, 通过引入动态链接库层次的访问控制, 攻击者无法利用动态链接库中无用代码触发系统调用, 增加了实现攻击的难度, 并且缓解了动态链接库中无用代码引发的漏洞。此外, 通过基于动态分析的多层关联访问控制, 能够排除由于静态分析不准确引入的额外系统调用。静态分析引入的额外系统调用在实际执行中是不存在相应路径的, 因此通过关联匹配 API 和系统调用, 能够识别程序实际执行过程中 API 与系统调用与分析得到的安全规则不相符的情况, 从而识别恶意调用路径。而且, 相比于文献[5]分析执行路径中的全部函数, 本方法仅动态分析 API 以及调用点的安全性, 因此引入的负

载更低。

本文的主要贡献包括 4 个方面:

1) 提出了一种多层次的内核访问控制方法, 通过分别在动态链接库以及系统调用两个层次构建关联化的访问控制机制, 提升用户态程序对内核的攻击难度;

2) 提出了一种适用于物联网应用的动态链接库无用 API 和代码缩减方法, 在系统调用访问控制的基础上, 使攻击者无法利用无用的代码实现攻击;

3) 通过动态分析系统调用与 API 的对应关系以及栈深度, 高效地验证了系统调用触发路径的安全性, 阻止了静态分析引入的额外系统调用以及由无用动态链接库代码触发的系统调用;

4) 对原型系统进行了有效性和性能测试, 实验结果表明系统能够在保证程序正常运行的前提下, 有效阻止无用系统调用的访问, 并且能够阻止物联网设备中大量潜在漏洞。

## 2 相关工作

随着物联网的快速发展, 物联网安全被越来越多的研究者关注[15-18]。物联网设备有限的硬件资源是物联网设备更容易受到攻击的主要原因之一, 因此许多研究关注于保护物联网系统安全[19-20]。物联网设备中的程序通常是由低层次编程语言(如: C/C++语言)编写, 由于这类语言不具备完善的安全检查、异常处理等机制, 编写者可能由于疏忽引入一些漏洞(如: 缓冲区溢出、内存泄露等)[21]。由于硬件资源有限, 物联网设备通常缺乏必要的动态系统防御措施(如控制流完整性检测等), 这使得攻击者更容易利用漏洞。大量研究指出代码注入攻击导致的漏洞常见于物联网固件[19-20], 通过篡改函数返回地址劫持物联网应用的控制流仍然是物联网应用面临的主要威胁[22-23]。在这种背景下, 攻击者更容易触发操作系统中系统调用包含的漏洞, 从而攻击操作系统。

为了保护操作系统, 大量研究基于 Linux Seccomp 阻止应用程序访问与其运行无关的系统调用, 从而防止其他系统调用中包含的漏洞被利用, 提升内核的隔离能力[8-14]。Linux Seccomp 是 Linux 操作系统内嵌的用于限制系统调用的安全模块, 通常引入的实时开销较低, 能够满足物联网设备的性能需求[5,7]。

为了获取目标程序运行依赖的必要系统调用, 动态跟踪方法通常跟踪并分析目标程序的执行过程[8-10]。文献[24]将程序的执行过程分为初始化阶段和服务阶段, 通常这两个阶段依赖的系统调用是不同的, 需在程序运行的不同阶段采用不同的系统调用访问控制策略。然而, 动态跟踪通常无法覆盖目标程序的所有执行路径, 导致收集到的系统调用不全。静态分析的方法通常构建动态链接库(如: glibc 等)的控制流图, 并基于控制流图生成 API 到系统调用的映射, 从而根据目标程序调用的 API 分析其依赖的系统调用集合[11-14]。然而, 静态分析由于缺乏动态执行信息无法精准地获取间接跳转的目标函数, 仅能获取其超集, 导致控制流图会包含实际不存在的函数调用关系, 进而导致 API 到系统调用的映射中存在无关的系统调用, 存在精准度不足的问题。综上, 现有的系统调用依赖分析方法面临着可用性以及精准度的问题。

针对这个问题, 文献[5]在静态分析的基础上动态验证动态链接库中涉及间接跳转的执行路径的安全性。静态分析中间接跳转引入的额外路径在真实执行环境中是不存在的, 通过动态验证能够还原系统调用的触发路径, 如果该路径与静态分析中路径相符, 则说明该系统调用是程序真实依赖的。相反, 当攻击者调用间接跳转引入的额外系统调用时, 其在动态链接库中的执行路径是静态分析中不存在的。然而, 该方法依然面临着性能开销以及分析全面性方面的问题。首先, 该方法在系统调用内核入口处恢复触发系统调用的动态链接库中全部的函数执行路径并验证其安全性, 这对于物联网设备来说性能开销较大。其次, 该方法未对动态链接库 API 进行限制, 导致攻击者能够利用整个链接库中的代码实现各种攻击, 并且可以不按照程序的执行逻辑触发系统调用。例如, 当动态链接库中存在一条真实可行的执行路径可访问由间接跳转引入的额外系统调用时, 攻击者能够利用缓冲区溢出、ROP 等漏洞执行该路径从而触发相应系统调用。而该调用已经由间接跳转分析引入了允许的系统调用集合, 并且动态链接库中触发该系统调用的执行路径也是真实存在的, 因此, 系统无法识别这类攻击。

限制程序可访问的动态链接库代码也是提高程序安全性的重要方法之一, 通过重新设计或编写动态链接库, 能够防止攻击者利用动态链接库中的无用代码。Piece-wise Debloating[25]只加载必要的动态链接库并用空代码替换运行无关的部分。Nibbler[26]通过分析目标应用程序的函数调用图删除未使用的代码。LibFilter[27]删除动态链接库中未使用的函数。但是, 这类方法没有限制系统调用的类别, 导致攻击者可能通过引入新的动态链接库或者直接利用汇编代码对操作系统进行攻击。

综上, 现有的内核访问控制方法普遍面临着可用性、精准度、性能等方面的问题, 因此研究精准高效的内核访问控制方法是十分必要的。

## 3 多层次的内核访问控制方法

### 3.1 系统概述

本文研究一种多层次的内核访问控制方法, 分别在目标程序的动态链接库层和系统调用层进行内核访问控制, 以提升物联网操作系统的安全性。系统架构如图 1 所示。首先, 对目标程序进行静态程序分析, 从而获取其依赖的动态链接库种类、API 以及通过汇编指令直接触发的系统调用集合。第二步, 构建动态链接库的控制流图并分析其中 API 和系统调用的映射关系, 从而生成目标程序依赖的系统调用集合。之后, 基于动态链接库的控制流图以及目标程序依赖的动态链接库 API, 消除动态链接库中的无用代码, 从而实现动态链接库层次的内核访问控制。最后, 基于目标程序依赖的系统调用集合生成相应的系统调用访问控制策略。引入动态分析机制, 在系统调用访问时, 动态分析触发系统调用的 API 与系统调用的对应关系, 从而分析动态链接库执行路径的安全性, 发现动态链接库中的代码滥用问题。

### 3.2 目标程序分析

为了获取目标程序依赖的动态链接库以及 API 的集合, 分析目标程序的二进制可执行文件。首先, 通过读取并分析可行性程序的 ELF 文件, 能够获取其依赖的动态链接库列表。之后分析二进制文件识别用于触发系统调用的汇编指令, 从而发现利用内联汇编直接触发的系统调用。

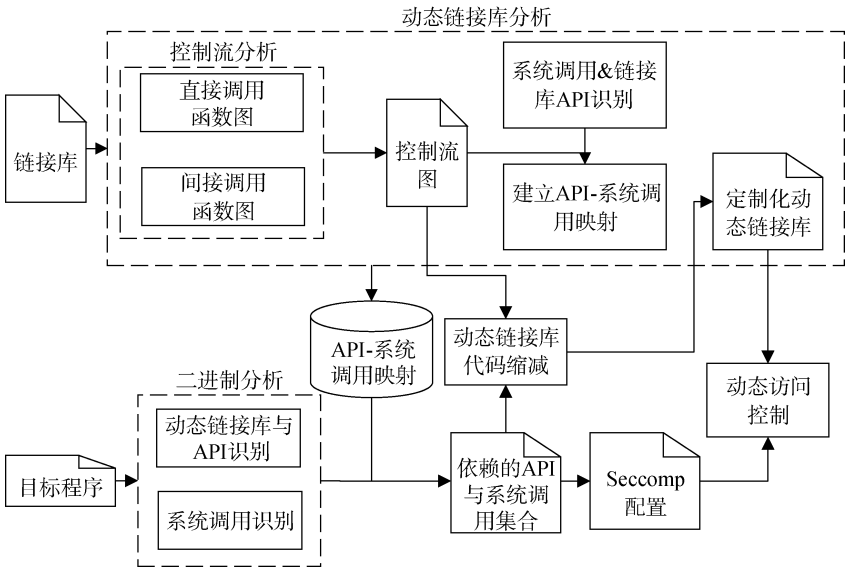


**图 1 系统架构图**

**Figure 1 System architecture**

大部分程序通过动态链接库(如, glibc)触发系统调用, 本文在获取了动态链接库种类的基础上识别二进制文件中各类动态链接库的 API。通常, 对动态链接库 API 函数的访问可以通过二进制文件中的注释来识别, 例如 glibc 的 API 通常在目标程序中被标记为"<API_name@@GLIBC__version>"。因此, 在反汇编二进制文件后可以通过注释以及函数名识别各 API。

当二进制文件通过内嵌汇编代码直接访问系统调用时, 本文分析其汇编代码中系统调用部分识别其访问的系统调类别。具体的, ARM 处理器采用 SVC 指令实现系统调用的访问机制, 系统调用号采用 R7 寄存器传递。为了通过静态分析识别系统调用号, 本文从系统调用指令开始, 采用文献[28]中静态污点分析的思想向前查找 R7 寄存器的数值来源。即将 R7 寄存器标记为污点数据, 当有寄存器或内存向其传递数据时, 该寄存器或内存也被标记为污点数据, 当数据来源是一个常量时污点分析结束, 该常量为系统调用号。最后, 将系统调用号与系统调用名相对应, 得到程序触发的系统调用类别。

### 3.3 构建 API 到系统调用的映射

本文基于源代码分析利用编译器辅助构建动态链接库的控制流图。由于 glibc 不支持 LLVM 编译器, 本文利用 gcc 的编译中间文件(如, cgraph 文件以及 cfg 文件)分析控制流图。cgraph 文件记录了源代码中的符号信息, 包括: 类型(如变量、外部函数、函数定

义、函数别名等)、函数间调用关系等信息, 可以据此建立函数别名和函数调用关系。而且 cgraph 文件同样定义了函数别名的对应关系, 基于函数间调用关系以及函数别名关系, 能够构建函数之间的直接调用图。

本文采用文献[29]的多层间接调用分析思想分析函数间的间接调用关系。首先, 识别源代码中的 address-taken 函数, 即函数的地址曾被保存于某些结构体的函数, 这些函数可能是间接调用的目标函数。之后, 分析间接调用的参数数量、类型以及返回值的类型, 并与 address-taken 函数进行比对, 从而识别与间接调用点相匹配的 address-taken 函数, 作为可能的目标函数集合。最后, 基于指针的数据传递关系进一步排除数据无关的指针, 从而筛选出相关的目标函数。为此, 本文分析编译过程生成的 cfg 文件, 该文件保存了各函数的控制流图信息, 通过 cfg 文件中可以获取函数的定义、函数调用等信息, 从而识别函数调用的参数类型、数量以及返回值类型。通过构建并合并直接调用图以及间接调用图, 能够生成动态链接库的完整函数调用图。

相比于文献[5]采用二进制分析构建直接控制流图, 本文采用的基于编译器辅助的源代码分析能够获得更丰富的语义, 而且分析结果可以直接用于后续的动态链接库代码缩减。

在生成完整函数调用图后, 识别调用图中的 API 函数以及触发系统调用的函数。本文基于动态链接库介绍文档构建了 API 函数名列表, 基于该列表能够在函数调用图中识别全部 API。本文以 glibc 为例分析该动态链接库中的系统调用触发函数。glibc 通常采用汇编指令、编译过程中嵌入汇编等方式触发系统调用[11]。为此, 本文分析编译生成的 expand 文件, 该文件由寄存器传输语言(register transfer language, RTL)编写, 包含了程序的内嵌汇编语言。通过分析汇编语言能够获取调用的系统调用类型。对于在编译过程中动态生成的汇编语言, 本文分析此类函数列表从而获取函数与系统调用的对应关系。在识别 API 以及系统调用之后, 能够基于函数调用图分析 API 可达的系统调用, 从而构建 API 与系统调用的对应关系。

### 3.4 动态链接库代码缩减

本文基于动态链接库的函数调用图以及目标程序所依赖的动态链接库 API, 查找该程序运行所依赖的动态链接库函数, 并将无用函数删除, 从而形成定制化的动态链接库。定制化的动态链接库只能服务于特定的应用程序, 因此需修改系统中动态链接库加载器使其能够为特定的程序加载特定的动态链接库。

文献[25]提出了一个动态链接库定制化缩减方法, 它在编译时分析代码, 并在生成的二进制文件中嵌入函数依赖的元数据。之后修改加载程序, 在动态链接库加载时根据元数据覆盖动态链接库中未使用的代码。该方法适合通用的库函数应用场景, 而物联网设备的功能通常较为单一, 其内部的服务程序通常是固定的且数量是有限的。因此, 本文采用的生成定制化动态链接库的方法在加载器工作过程中不会因分析函数级的依赖关系产生实时负载。

### 3.5 关联化的动态分析机制

动态分析模块在获取程序的依赖系统调用后, 首先利用 Seccomp 阻止程序访问无关的系统调用。对于静态分析得到的程序运行依赖的必要系统调用, 在 Seccomp 允许其执行的基础上提出一种动态分析机制, 分析该系统调用的安全性。

基于静态分析获得的系统调用控制策略面临两方面的问题。首先, 间接调用分析不准确扩大了动态链接库中 API 到系统调用的映射, 引入了无关系统调用。其次, 动态链接库存在多个 API 对应同一系统调用的情况, 程序可能仅依赖某一个 API, 而仅限制系统调用的种类, 无法识别该系统调用是由程序逻辑触发的还是由其他非法路径(如 ROP 攻击等)触发的。

为了解决这个问题, 本文通过在内核入口处分析触发系统调用的用户态内存栈重构出动态链接库中触发该系统调用的执行路径, 进而分析其安全性, 架构图如图 2 所示。与文献[5]不同, 本文仅分析触发系统调用的动态链接库 API 的种类是否符合 API 与系统调用映射以及 API 是否存在于目标程序的依赖 API 集合中, 来分析路径的安全性。首先, 仅分析 API 的完整性降低了动态分析的负载, 从而更适用于物联网设备内。其次, 本文在文献[5]的基础上进一步验证了 API 的安全性。

为了动态分析路径的安全性, 需从栈中提取动态链接库 API 种类。为此, 需构造 API 函数与其内存地址之间的映射。由于静态分析无法获得动态内存位置, 因此本文在重新编译的动态链接库中加入元数据从而标记 API, 之后使用自定义的 ELF loader 在运行时加载对应的动态链接库并获得各 API 的内存地址。

动态分析模块作为内核模块在各系统调用入口处分析 Seccomp 允许的系统调用的安全性。相比于在总的内核入口点处拦截, 在系统调用入口拦截分

析的触发频率更低, 性能开销更小。当特定系统调用被触发时, 分析模块检查当前进程是否属于被监控的目标进程, 如果是则进行安全分析。

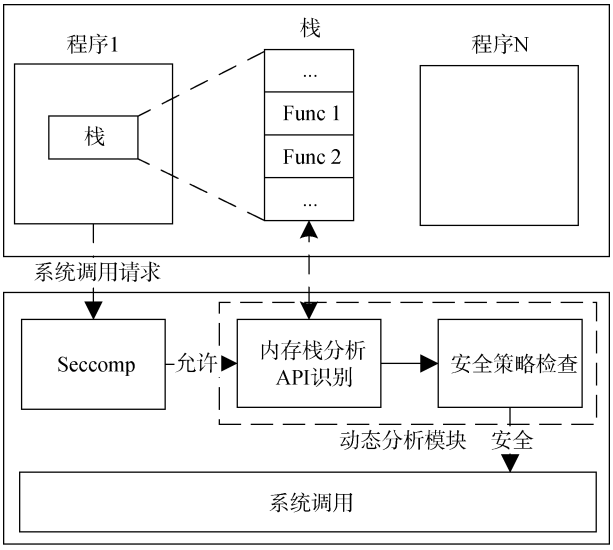


**图 2 动态分析架构图**

**Figure 2 System architecture of dynamic analysis**

### 3.6 讨论与分析

本文通过分析执行过程中动态链接库 API 以及栈深度来分析系统调用路径的安全性。与仅验证系统调用类别的方法相比, 本文提出的方法提升了攻击的难度。但是攻击者可能会篡改整个用户栈来躲避动态分析的检查。

然而, 这种攻击是不易实现的。首先, 内存栈的 canary 机制在栈中插入校验位来检测栈溢出攻击。当标记位被覆盖时, 程序将被终止。虽然 canary 机制并不完全安全, 但是研究者已经提出了大量保护方法[30], 提升了栈篡改的难度。其次, 动态链接库是经过修改的, 其内存分布与通用的动态链接库完全不同, 因此攻击者难以基于公开的动态链接库函数偏移来伪造正确的返回地址以欺骗动态分析模块。此外, 缩减的动态链接库中仅有必要的代码, 基于这些代码构造 ROP 攻击是更加困难的, 而且动态分析模块能够发现程序引入的新动态链接库。最后, 伪造全部内存栈会影响攻击控制流。攻击者通常需要按照一定逻辑调用一些高权限的系统调用或利用一些系统调用中的漏洞进行提权攻击, 如果栈被安全的执行路径覆盖, 控制流将无法返回到相应的攻击路径, 这可能导致程序崩溃, 攻击者无法继续攻击。

因此, 动态分析模块增强了访问控制机制的安全性, 使攻击更加困难。

本方法面向物联网常用的ARM Linux平台设计, 因此在设计和实现上与面向 x86 架构的内核访问控制方法[5,11]有一些差异。首先, 在静态分析目标程序与动态链接库代码以识别触发的系统调用类别时, 由于 ARM 平台的系统调用机制与 x86 不同, 需要单独构建 ARM 环境中系统调用触发代码的分析过程以及系统调用名与调用号的对应关系。其次, ARM 架构下的 glibc 调用系统调用的汇编代码与传统 x86 不同, 需要识别并分析 SVC 指令并且识别相应的寄存器数值。此外, 由于汇编指令不同, 面向 x86 平台的污点分析、控制流构建方法不适用于 ARM 平台, 需要面向 ARM 指令集重新设计。

本文通过分析应用程序从而生成定制化的动态链接库以及访问控制策略, 随着物联网设备种类的不断增加, 本方法面临着易用性和通用性方面的挑战。首先, 随着设备种类的多样化, 高度定制化程序可能采用不同的动态链接库, 这会增加本方法的分析复杂性, 降低易用性; 其次, 很多定制化的传感器不具有独立的操作系统, 使得本方法无法直接适用于该类系统, 降低了方法的通用性。本文将在未来工作中使本方法适配更多种类的物联网设备和系统, 提升方法的易用性和通用性。

## 4 系统实现

本章介绍原型系统实现中的一些关键细节。

### 4.1 编译动态链接库

由于 glibc 不支持 Clang/LLVM 编译器, 本文采用 gcc 编译 glibc。为了生成编译过程文件, 在编译的 CFLAGS 参数中分别加入-fdump-ipa-cgraph、-fdump-rtl-expand、-fdump-tree-cfg 等参数, 从而分别生成 cgraph、expand 和 cfg 文件。

### 4.2 控制流构建

在获取动态链接库(如 glibc)的编译中间文件后, 需分析其中的程序语义以获取函数调用图。cgraph 文件中每一个符号都有两个名称, 本文分别称为 firstname 以及 secondname。这两种名称在构建函数调用关系时会交替使用。firstname 用于 References 字段和 Referring 字段, 而 secondname 用于 Calls 和 Called by 字段。使用 cgraph 文件建立调用关系和别名关系时, 调用关系可以直接从 Calls 字段读出, 而别名则需要进一步处理。对于一个符号, 如果它是别名, 需要根据他的 References 字段的值寻找它指向的符号, 由于指向的符号也可能是别名, 所以要递归式的查找, 直到遇到一个函数定义, 此时可以将最开始的别名映射到这个函数名。

## 5 实验

在实现了原型系统后, 本文在 ARM Linux 平台分别测试了系统的有效性、性能以及漏洞防御能力, 并且与现有方法进行对比。本方法的有效性主要体现在通过禁用目标程序的内核与动态链接库的代码访问, 缓解了禁用代码中包含的 CVE 漏洞, 分别在 5.1 节和 5.3 节进行了测试。本文选择 50 个常用的 ARM Linux 程序作为分析的目标。为此, 本文从文献[31]的物联网固件数据集中收集了 1000 个 ARM 固件镜像, 并利用 binwalk 和 Firmwalker 解析镜像。从解压后的文件镜像中, 本文选择 50 个通用的基于 glibc 的程序进行测试。

### 5.1 有效性测试

为了测试系统的有效性, 首先利用静态分析获取的程序依赖系统调用列表生成相应的定制化动态链接库以及 Seccomp 配置。分析结果如图 3 所示, 该图展示了通过静态分析为每个程序禁止的系统调用的数量。图中横坐标是程序的序号, 纵坐标是禁止的系统调用数量, 平均数为 248.08 个。图 4 展示了各程序禁用动态链接库函数的数量, 平均数为 708.64 个。结果表明, 本方法能够禁止大量的系统调用以及动态链接库的代码访问, 缩小系统攻击面。为了验证静态分析的正确性, 实验将目标程序与相应的 Seccomp 配置一一执行。实验发现所有程序都可以正常执行, 这证明了本方法的健壮性。在执行过程中, 动态分析模块分析允许系统调用的安全性, 平均每个程序执行的系统调用为 14.18 个, 说明实际执行的系统调用数量远小于允许的数量, 因此通过动态分析进一步分析系统调用的安全性是必要的。

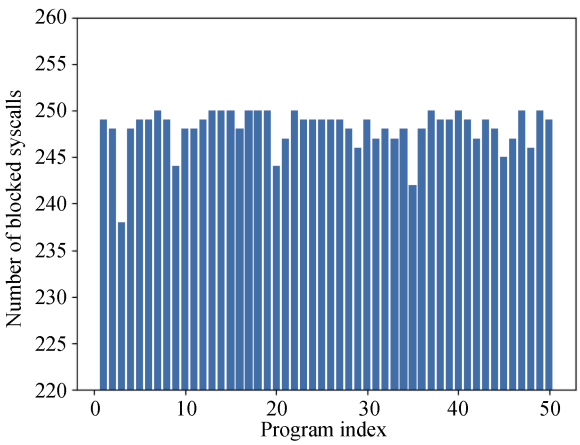


**图 3 各程序禁止系统调用数量**

**Figure 3 The number of syscall limitation of each program**

### 5.2 性能测试

动态访问控制会给目标程序带来实时性能开销, 为了测量开销, 实验首先在没有访问控制的情况下运行目标程序 100 次, 并记录执行结果。然后, 在访问控制下使用相同的输入对它们进行测试。实验选用了 LMBench 作为测试的工具, 该工具是一种常用的性能测试工具, 能够测试内存、文件等方面的性能。实验选择 2 个测试指标(内存读写 bw_mem 和文件块 I/O bw_file_rd)来测量开销, 测试得到的各指标的分数如表 1 所示。从表中可以看出, 访问控制对 bw_mem 测试引入的负载小于 1%, bw_file_rd 测试引入的负载约为 3.2%, 说明了本系统引入的实时负载较低。

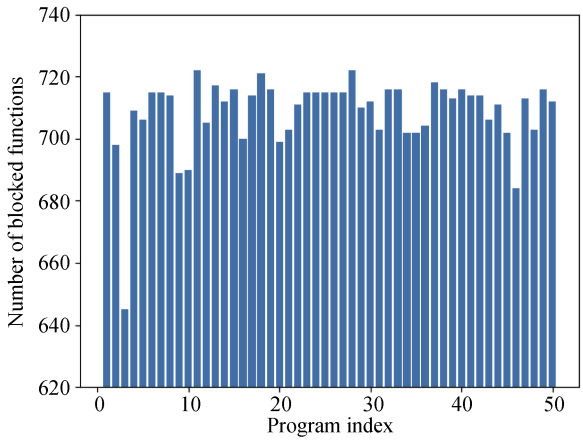


**图 4 各程序禁止动态链接库函数的数量**

**Figure 4 The number of function limitation of each program**

sysverify[5]同样采用了动态验证的方法, 该方法分析了内存栈每个地址的安全性并且重构完整的系统调用触发路线。因此, 本文与完整路径分析的方法进行性能对比。为此, 本文首先实现了 sysverify 提出的路径分析方法。测试采用了相同的性能测试工具以及参数。测试结果如表 1 所示, 完整路径分析的方法在两个测试中分别引入了约 2.8%以及 6.7%的实时负载。通过结果可以发现本文提出的方法性能优于完整路径分析的性能。原因是完整路径分析会对每个栈中地址进行分析, 并且需要重构执行路径, 再

**表 1 不同访问控制的性能对比**

**Table 1 Performance comparison of different access controls**

| 测试指标 | 未控制 | 访问控制 | 对比完整路径分析 |
|---|---|---|---|
| bw_mem | 46786 | 46107 | 45469 |
| bw_file_rd | 14992.483 | 14509.728 | 13986.421 |

与已有路径库进行比对; 而本文提出方法仅需要分析 API 的安全性。此外, 物联网设备由于性能有限, 也会放大性能差异。因此, 本文提出的方法更适用于硬件资源更为有限的物联网设备。

静态分析主要包含两部分, 即分析动态链接库生成 API-系统调用的映射, 以及分析各目标应用程序依赖的 API, 从而生成对应的动态链接库以及系统调用访问控制策略等安全配置。其中, 动态链接库 API-系统调用映射仅需生成一次, 安全配置需要根据目标程序按需生成。本文在配备 Intel i5 2.7 GHz CPU、8G 内存的计算机上分别测试了静态分析 2 个阶段所需的时间, 其中生成 glibc 动态链接库 API-系统调用映射的时间 10 次平均为 27min 30s, 生成 50 个目标程序所需动态链接库以及安全配置所需时间平均为 16min 28s。由于静态分析是离线分析, 不会对应用程序产生实时负载, 且每个应用仅需分析一次, 因此静态分析的性能是可以接受的。

### 5.3 漏洞防御能力测试

本方法静态分析的目标是生成程序依赖的动态链接库 API 集合、系统调用集合以及 API 和系统调用的映射。本方法能够基于静态分析生成的结果, 通过对动态链接库 API 和系统调用的访问控制阻止包含漏洞的代码或系统调用的执行, 从而阻止动态链接库或系统该调用包含的漏洞(如: CVE-2016-8655, CVE-2017-5123, CVE-2017-7308 等)被攻击者利用。因此, 实验测量可以通过删除未使用的代码或系统调用来缓解的 CVE 漏洞数量。为此, 本文首先爬取 CVE 漏洞网站并收集与动态链接库(glibc)和系统调用相关的 CVE 漏洞, 同时收集其涉及的的具体函数名、动态链接库 API 以及系统调用名。当漏洞涉及的函数被阻止访问时, 对应的漏洞就无法被攻击者利用。

根据程序依赖的动态链接库 API 列表, 并结合控制流图, 能够分析得出各程序需删减的动态链接库函数。基于删减库函数与 CVE 漏洞的对应关系, 能够计算出针对各程序所阻止的 CVE 漏洞数量。通过计算, 本系统共抵御了 15 个动态链接库中包含的 CVE 漏洞, 部分代表性的 CVE 漏洞如表 2 所示。

与动态链接库分析相似, 当包含漏洞的系统调用被阻止访问时, 该漏洞被缓解。基于系统调用与 CVE 漏洞的映射以及各程序禁用的系统调用列表, 本系统共阻止了目标程序涉及的 95 个 CVE 漏洞, 平均为每个程序阻止了 89 个 CVE 漏洞。通过禁用系统调用缓解的部分代表性的 CVE 漏洞展示在表 3 中。

**表 2　由动态链接库缩减缓解的部分 CVE 漏洞列表**

**Table 2　Partial list of CVEs mitigated by dynamically-linked library debloating**

| 序号 | API | CVE 漏洞 |
|---|---|---|
| 1 | iconv | CVE-2021-3326 |
| 2 | wordexp | CVE-2021-35942 |
| 3 | iconv | CVE-2020-29562 |
| 4 | iconv | CVE-2020-27618 |
| 5 | iconv | CVE-2019-25013 |

**表 3　禁用系统调用缓解的部分 CVE 漏洞列表**

**Table 3　Partial list of CVEs mitigated by syscall limitation**

| 序号 | 系统调用 | 代表性 CVE 漏洞 |
|---|---|---|
| 1 | inotify_add_watch | CVE-2019-9857 |
| 2 | ptrace | CVE-2019-13272 |
| 3 | clock_nanosleep | CVE-2018-13053 |
| 4 | get_mempolicy | CVE-2018-10675 |
| 5 | add_key | CVE-2017-7472 |
| 6 | ioctl | CVE-2016-4569 |
| 7 | perf_event_open | CVE-2017-6001 |
| 8 | waitid | CVE-2017-5123 |
| 9 | reboot | CVE-2017-18079 |
| 10 | rt_sigreturn | CVE-2017-15537 |

由于动态链接库的复杂性, 多个 API 可能会触发相同的系统调用。动态分析模块分析动态链接库 API 和系统调用的对应关系, 能够识别由非法 API 触发的系统调用。例如, 在静态分析中, openat 系统调用能够被 open 和 fopen 这两个动态链接库 API 访问, 而目标程序仅依赖 fopen, 动态分析模块能够阻止从 open 执行路径访问该系统调用的情况。经统计, 动态分析模块能为平均 12.3 个涉及漏洞的系统调用识别非法的触发路径。

Confine[11]是一个基于系统调用限制的内核攻击面缩小工具, 它首先分析目标应用依赖的 API 列表, 之后通过动态链接库 API 和系统调用的映射关系建立应用运行依赖的系统调用结合, 从而在目标应用运行过程中通过 Seccomp 禁用无关系统调用。本文与 Confine 的不同之处主要在于, 在限制无关系统调用访问的基础上, 额外限制了应用对动态链接库中无关代码的访问并且检查应用在触发系统调用时执行路径的安全性。因此, 本文主要测试了本方法与 Confine 不同之处提供的额外安全能力。首先, 如前文所述, 本方法通过禁用无关的动态链接库代码能够额外缓解 15 个动态链接库中包含的 CVE 漏洞。其次, 通过动态分析触发系统调用路径的安全性,

本方法能够为测试应用识别平均 12.3 个涉及漏洞的系统调用的非法触发路径。因此, 相较于 Confine 本方法能够进一步增强系统安全性。

综上, 通过与现有方法的对比, 本文提出的方法能够缓解更多漏洞, 提升攻击者对内核攻击的难度, 此外本文提出的方法引入的实时负载更低。

## 6 结论

本文提出了一种多层次的物联网系统内核访问控制方法, 通过分别在动态链接库以及系统调用两个层次构建关联化的访问控制机制, 提升用户态程序对内核的攻击难度, 增强物联网操作系统的隔离能力。实验结果表明, 本方法能够有效缩减物联网设备内程序可访问的动态链接库代码以及系统调用数量, 阻止攻击者利用无用代码中的漏洞。未来的工作将关注于把本系统扩展到更多的动态链接库以及更多的应用场景下, 提升更多种类的操作系统的安全性。

## 参考文献


[1] Liao B, Ali Y, Nazir S, et al. Security Analysis of IoT Devices by Using Mobile Computing: A Systematic Literature Review[J]. *IEEE Access*, 8: 120331-120350.

[2] Li C T, Lee C C, Weng C Y, et al. Towards Secure Authenticating of Cache in the Reader for RFID-Based IoT Systems[J]. *Peer-to-Peer Networking and Applications*, 2018, 11(1): 198-208.

[3] Yaqoob I, Hashem I A T, Ahmed A, et al. Internet of Things Forensics: Recent Advances, Taxonomy, Requirements, and Open Challenges[J]. *Future Generation Computer Systems*, 2019, 92: 265-275.

[4] Zikria Y B, Kim S W, Hahm O, et al. Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution[J]. *Sensors* (*Basel*, *Switzerland*), 2019, 19(8): 1793.

[5] Zhan D Y, Yu Z F, Yu X Z, et al. Shrinking the Kernel Attack Surface through Static and Dynamic Syscall Limitation[J]. *IEEE Transactions on Services Computing*, 2022(99): 1.

[6] Eclipse Foundation. 2020 IoT Developer Survey [OL]. [2022-05-20] https://iot.eclipse.org/community/resources/iot- surveys/assets/iot-developer-survey-2020.pdf.

[7] Xu M, Song C Y, Ji Y, et al. Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques[J]. *ACM Computing Surveys*, 2017, 49(2): 38.

[8] Lei L G, Sun J H, Sun K, et al. SPEAKER: Split-Phase Execution of Application Containers[M]. Detection of Intrusions and Malware, and Vulnerability Assessment. Cham: Springer International Publishing, 2017: 230-251.

[9] Wan Z Y, Lo D, Xia X, et al. Mining sandboxes for linux containers[C]. *2017 IEEE International Conference on Software Testing, Verification and Validation*, 2017: 92-102.

[10] Barlev S, Basil Z, Kohanim S, et al. Secure yet Usable: Protecting Servers and Linux Containers[J]. *IBM Journal of Research and Development*, 2016, 60(4): 12: 1-12: 10.

[11] Ghavamnia S, Palit T, Benameur A, et al. Confine: Automated system call policy generation for container attack surface reduction[C]. *23rd International Symposium on Research in Attacks, Intrusions and Defenses*, 2020: 443-458.

[12] DeMarinis N, Williams-King K, Jin D, et al. Sysfilter: Automated system call filtering for commodity software[C]. *23rd International Symposium on Research in Attacks, Intrusions and Defenses*, 2020: 459-474.

[13] Zeng Qiang, Xin Zhi, Wu Dinghao, et al. Tailored application-specific system call tables[J]. *The Pennsylvania State University, Tech. Rep*, 2014.

[14] Canella C, Werner M, Gruss D, et al. Automating Seccomp Filter Generation for Linux Applications[C]. *The 2021 on Cloud Computing Security Workshop*, 2021: 139-151.

[15] Khan M A, Salah K. IoT Security: Review, Blockchain Solutions, and Open Challenges[J]. *Future Generation Computer Systems*, 2018, 82: 395-411.

[16] Hassija V, Chamola V, Saxena V, et al. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures[J]. *IEEE Access*, 7: 82721-82743.

[17] Yang Y Y, Zhou W, Zhao S R, et al. Survey of IoT Security Research: Threats, Detection and Defense[J]. *Journal on Communications*, 2021(8):188-205.
(杨毅宇, 周威, 赵尚儒, 等. 物联网安全研究综述:威胁、检测与防御[J]. *通信学报*, 2021(8):188-205.)

[18] Chen Z, Zeng F P, Chen G Z, et al. A Survey for IoT Security Assessment Technologies[J]. *Journal of Cyber Security*, 2019, 4(3): 2-16.
(陈钊, 曾凡平, 陈国柱, 等. 物联网安全测评技术综述[J]. *信息安全学报*, 2019, 4(3): 2-16.)

[19] Clements A A, Almakhdhub N S, Saab K S, et al. Protecting bare-metal embedded systems with privilege overlays[C]. *2017 IEEE Symposium on Security and Privacy*, 2017: 289-303.

[20] Pewny J, Garmany B, Gawlik R, et al. Cross-architecture bug search in binary executables[C]. *2015 IEEE Symposium on Security and Privacy*, 2015: 709-724.

[21] Zhan D Y, Yu X Z, Zhang H L, et al. ErrHunter: Detecting Error-Handling Bugs in the Linux Kernel through Systematic Static Analysis[J]. *IEEE Transactions on Software Engineering*, 2022, (99): 1.

[22] Almakhdhub N S, Clements A A, Bagchi S, et al. $\mu$RAI: securing embedded systems with return address integrity[C]. *Proceedings 2020 Network and Distributed System Security Symposium*, 2020.

[23] Zhou J, Du Y F, Shen Z J, et al. Silhouette: Efficient Protected Shadow Stacks for Embedded Systems[C]. *The 29th USENIX Conference on Security Symposium*, 2020: 1219-1236.

[24] Ghavamnia S, Palit T, Mishra S, et al. Temporal System Call Specialization for Attack Surface Reduction[C]. *The 29th USENIX Conference on Security Symposium*, 2020: 1749-1766.

[25] Quach A, Prakash A, Yan L. Debloating Software through

Piece-Wise Compilation and Loading[C]. *The 27th USENIX Conference on Security Symposium*, 2018: 869-886.

[26] Agadakos I, Jin D, Williams-King D, et al. Nibbler: Debloating Binary Shared Libraries[C]. *The 35th Annual Computer Security Applications Conference*, 2019: 70-83.

[27] Shteinfeld B. Libfilter: Debloating dynamically-linked libraries through binary recompilation[J]. *Undergraduate Honors Thesis. Brown University*, 2019.

[28] Mishra S, Polychronakis M. Shredder: breaking exploits through API specialization[C]. *The 34th Annual Computer Security Applications Conference*, 2018: 1-16.

[29] Lu K J, Hu H. Where does it Go? : Refining Indirect-Call Targets with Multi-Layer Type Analysis[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 1867-1881.

[30] Lopez T, Tun T, Bandara A, et al. An anatomy of security conversations in stack overflow[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society*, 2019: 31-40.

[31] Chen D D, Egele M, Woo M, et al. Towards automated dynamic analysis for linux-based embedded firmware[C]. *Proceedings 2016 Network and Distributed System Security Symposium*, 2016: 1: 1-1.

**詹东阳** 于 2019 年在哈尔滨工业大学网络空间安全专业获得博士学位。现任哈尔滨工业大学讲师。研究领域为云计算安全、系统安全。研究兴趣包括: 容器安全、漏洞挖掘等。Email: zhandy@hit.edu.cn


**俞兆丰** 哈尔滨工业大学网络空间安全专业博士研究生。研究领域为物联网安全、云安全。Email: 20S003135@stu.hit.edu.cn

**叶麟** 于 2011 年在哈尔滨工业大学信息安全专业获得博士学位。现任哈尔滨工业大学副教授。研究领域为网络安全、云安全。研究兴趣包括: 逆向分析、流量安全等。Email: hityelin@hit.edu.cn

**张宏莉** 哈尔滨工业大学教授, 博士生导师。研究领域为网络空间安全、网络内容安全。研究兴趣包括: 社交网络分析、流量分析等。Email: zhanghongli@hit.edu.cn