SGX-based LibOS 中系统调用转发机制研究

刘西蒙[1]，黄应康[1]，刘维杰[2,3]，范倍汐[4]，章恬[1]，张杰[4]
（1. 福州大学计算机与大数据学院/软件学院，福建 福州 350108；2. 南开大学密码网络空间安全学院，天津 300350；
3. 数据与智能系统安全教育部重点实验室，天津 300350；4. 山西师范大学数学与计算机科学学院，山西 太原 030031）

摘　要：SGX-based LibOS 允许现有的未经修改的应用程序在 SGX Enclave 中运行。然而，不同的 SGX-based LibOS 在架构设计、系统调用模拟以及系统调用转发机制上存在差异，增加了用户使用门槛，并使得调试程序错误变得棘手。为了应对这些问题，提出了系统调用动态测试框架，对各种 SGX-based LibOS 进行了测试，追踪了系统调用在 LibOS 中的执行状况，并比较了其在 Linux 宿主机上的运行差异。同时，分析了实验结果，深入探讨了不同基于 SGX 的 LibOS 中系统调用转发机制的差异，总结了它们对 Linux 功能的模拟情况以及编程语言运行时的支持状态，并指出了该领域的不足和待改进之处。

关键词：IntelSGX；系统安全；库操作系统；系统调用
中图分类号：N92
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024214

Research on system call forwarding mechanism of SGX-based LibOS

LIU Ximeng[1], HUANG Yingkang[1], LIU Weijie[2,3], FAN Beixi[4], ZHANG Tian[1], ZHANG Jie[4]
1. College of Computer and Data Science /College of Software, Fuzhou University, Fuzhou 350108, China
2. College of Cryptology Cyber Science, Nankai University, Tianjin 300350, China
3. Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tianjin 300350, China
4. School of Mathematics and Computer Science, Shanxi Normal University, Taiyuan 030031, China

Abstract: SGX-based LibOS are designed to run unmodified applications within SGX Enclave, but differences in their architecture, system call simulation, and system call forwarding can make them difficult to use and debug. To overcome these challenges, a dynamic testing framework was introduced that traced system calls and verified their behaviors in various SGX-based LibOS. This framework compared the execution of system calls within the LibOS to their execution on regular Linux hosts, analyzing the differences in call forwarding mechanisms, Linux feature replication, and runtime support for programming languages. The study aims to highlight where improvements are needed and hopefully provides guidance for future research in this area.

Keywords: Intel SGX, system security, LibOS, system call

0　引言

在信息时代的背景下，数据保护和数据的合规计算成为大众重点的关注对象。对于数据安全以及隐私信息的合法保护，目前主流的解决方案是隐私计算[1]，隐私计算主要由机密计算、差分隐私、多方安全计算、零知识证明、联邦学习等技术所构

收稿日期：2024–10–06
通信作者：刘维杰，weijieliu@nankai.edu.cn
基金项目：国家自然科学基金资助项目（No.62072109）；福建省自然科学基金资助项目（No.2021J06013）
Foundation Items： The National Natural Science Foundation of China (No.62072109), The Natural Science Foundation of Fujian Province (No.2021J06013)

成。其中机密计算指的是由硬件提供的可信执行环境（TEE）作为技术支持，可以在数据处理的过程中将敏感的数据隔离在受保护的CPU区域内。近10年来，TEE技术发展迅速，其重点是基于CPU的隔离执行。其中突出的CPU隔离方案包括Intel SGX[2]、AMD SEV[3]、ARM TrustZone[4]等。

本文讨论体系成熟的Intel SGX，SGX使用硬件隔离作为安全保障，不依赖固件和软件安全状态，为用户空间提供可信的执行环境。目前SGX技术被广泛应用到各个领域[5]，包括机器学习隐私保护、区块链数据认证[6]、TEE联邦学习解决医疗“数据孤岛”问题[7]。由于SGX的强隔离性，未经修改的应用程序不能直接运行在SGX Enclave内。为解决这一问题，现存的方案有Intel SGX SDK、SGX-based LibOS、SGX-based Sandbox（如Enarx[8]、Chancel[9-10]），其中被广泛使用的方案是在内核态与用户态之间增加一个LibOS[11]层。LibOS被用作拦截应用程序发出的系统调用，并通过模拟Linux内核系统调用，让应用程序运行在LibOS上。

LibOS层的宗旨在于在确保安全的前提下，努力实现与Linux内核的100%兼容性。尽管目前广泛应用且功能完备的SGX-based LibOS（如Gramine、Occlum、Mystikos及SGX-LKL[12]等）已使得大部分应用程序能够直接在Enclave内运行，Github项目社区中仍频繁出现用户及开发者关于此的咨询：在应用程序运行过程中，遇到LibOS不支持的系统调用、系统调用标志无效，或者关于未来是否计划加入新的系统调用等问题，以及系统调用的转发机制如何实现及其安全性如何，用户和开发者在程序执行出错后应如何进行调试，均需用户和开发者投入大量时间研究SGX-based LibOS中的系统调用。因此，本文旨在为用户和开发者梳理这些SGX-based LibOS中的系统调用，并指出该领域的不足。

为解决上述问题，本文主要的研究工作如下。

1) 分析SGX-based LibOS中系统调用目前所面临的挑战以及缓解方案。

2) 通过追踪LibOS中的系统调用，总结出系统调用（包括库函数和通过汇编指令SYSCALL执行的系统调用）的转发机制。

3) 利用系统调用动态测试，比较SGX-based LibOS与Linux宿主机之间系统调用执行结果的差异，从而归纳出LibOS中实现的Linux特性。

4) 对那些存在较大差异且可能引发安全问题的系统调用进行详细描述和归纳。

1 背景介绍

1.1 SGX-based LibOS和ECALL/OCALL

Intel SGX是Intel在2013年推出的一种硬件级内存隔离技术。它允许应用程序在一个被称为飞地（Enclave）的加密且隔离的内存空间中运行，其中只有特定的可信线程可以访问。SGX的设计将操作系统和其他软件视为潜在的威胁，因此在飞地内部限制了系统调用的执行。

为了支持应用程序在飞地内运行而无须重构代码，研究人员提出SGX-based LibOS。该方案通过引入更轻量级的LibOS到飞地中，但仍然需要依赖Intel SGX的ECALL和OCALL机制来处理数据。ECALL用于将外部数据安全地复制到飞地中，而OCALL允许从飞地将数据传出以执行必要的系统调用，如write()、read()等。OCALL的执行涉及退出飞地（使用EEXIT指令）、在主机上处理数据，然后通过EENTER指令重新进入飞地。

本文将探讨目前仍在积极维护的SGX-based LibOS项目，包括Gramine[13-14]、Occlum[15-16]、Mystikos[17]。这些LibOS具有以下特点：LibOS层仅存在于用户态，并在用户态为应用程序提供基本的内核功能；通过LibOS层拦截和模拟系统调用，为应用程序提供与Linux内核一致的接口，使应用程序能够轻松运行在Enclave内部。

1.2 系统调用

系统调用（SYSCALL）是操作系统提供给应用程序的一种接口，允许应用程序请求操作系统执行特定的低级操作，通常这些操作涉及硬件资源的管理和控制。作为用户空间（应用程序执行的环境）与内核空间（操作系统核心组件执行的环境）之间的桥梁，其不仅提供了一个安全的机制，让应用程序能够执行需要操作系统介入的功能；还对硬件资源进行保护，确保系统的稳定性和安全性。

1.3 SGX-based LibOS框架

SGX-based LibOS框架旨在为SGX Enclave内提供一个隔离且安全的执行环境，同时模拟操作系统的基本功能，以便运行未经修改的应用程序。这种设计面临的挑战包括如何在Enclave内部提供丰富的操作系统功能，以及如何安全地与外部世界进行交互。

图 1 展示了常规 SGX-based LibOS 的框架，其中 SGX Enclave 内部包括应用程序、标准库、LibOS 以及屏蔽层。SGX-based LibOS 提供标准 C 库（如 Glibc、Musl libc）及一些常用库，以确保应用程序可以在无须修改的情况下直接在 SGX-based LibOS 上运行。LibOS 为应用程序提供了大多数常用的系统调用，这些模拟的系统调用支持内存申请与管理、网络通信、进程管理等功能，允许应用程序在不直接与外部操作系统交互的情况下运行。屏蔽层负责在 Enclave 的边界处验证或拒绝来自不受信任操作系统的输入，以降低被攻击的风险。此外，平台适配层（PAL）和外部的操作系统属于不受信任的部分，Enclave 内的应用程序需通过 OCALL 和 ECALL 机制与外部世界安全地进行交互。

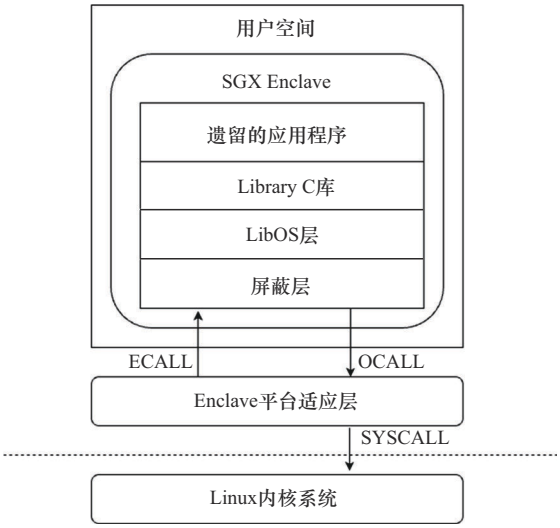

图 1　SGX-based LibOS 框架

2　挑战和解决方案

2.1　实现系统调用所面临的挑战

为了在 SGX-based LibOS 中实现足够多的系统调用，开发者需要应对诸多问题。在接口限制和兼容性方面，Gramine、Occlum 等 SGX-based LibOS 都是 ABI 级别兼容性的库操作系统（这是一种让系统调用在转发时无须重新编译源代码即可在外部操作系统上运行应用程序的技术）。使用这一技术让应用程序中大部分系统调用能够正常工作，但 ABI 级别的兼容性是不够的。即使应用程序在 LibOS 上加载和运行没有错误，也可能会因其功能缺失或不完整导致行为和预期不同。例如，在 Gramine 和 Occlum 的设计中，系统调用通过 shim 层和 PAL 转发从应用层到主机内核的系统调用，可以达到和 Linux 上的行为一致（如 setsockopt(2)在 LibOS 中没有实现，需转发到 Linux 内核执行）。另一种情况是应用程序调用 LibOS 内部实现的系统调用，这种情况会相对麻烦，由于 LibOS 本身作为应用程序，无法获取内核的全局视角，其功能无法达到内核级别的兼容，将导致在 SGX-based LibOS 上实现的行为始终无法与在 Linux 主机的行为完全相同。例如，SGX-based LibOS 内部实现的 epoll 相关的系统调用（具体参照 5.3.4 节的 I/O 多路复用部分），在 Gramine Github 页面上有关的问题数量有 57 个，而 Occlum GitHub 页面上有 27 个。

在性能方面，在 SGX-based LibOS 中，使用系统调用（从 Enclave 到 non-Enclave 切换）将造成大量的性能开销。SGX-based LibOS 中系统调用需要 OCALL 和 ECALL 机制的支持，使用 EEXIT 和 EENTER 指令。每个 EEXIT 都会刷新 CPU 缓存、EENTER 需要进行许多检查，并要求硬件内部同步核心。研究表明，每个 EEXIT 和 EENTER 的成本在 8 000~12 000 个 CPU 时钟周期（正常的系统调用成本大约在 100 个 CPU 时钟周期）。在 SGX-based LibOS 上，对普通的 Hello World 程序进行测试时发现：执行了 224 个 EENTER、192 个 EEXIT 和 201 个 AEX 指令，需要 30 亿个 CPU 时钟周期。

2.2　解决方案

为了缓解接口限制和兼容性，开发者需要投入大量的人力来补充系统调用接口和完善系统调用的实现，目前还不存在令人满意的解决方案。

在有效缓解性能开销的方面，目前工业界和学术界提出了 3 种方案。Linux SGX SDK 中最新的补丁包含了 Switchless Calls 技术[18]，通过使用工作线程、CPU 核心异步执行函数调用来避免 Enclave 切换；Eleos 等[19]提出 Exitless 分页机制和 Exitless 系统调用来减少 SGX 开销主要的 2 个因素：EPC 换页和 Enclave 上下文切换；SGX-based LibOS 在 EEXIT 上使用 XSAVE[20]指令（用于处理器状态保存），在 EENTER 上使用 XRSTOR 指令（用于处理器状态恢复）。XSAVE 指令用于每个 EEXIT 退出后，而每个 OCALL 机制都会导致一个 EEXIT 退出 Enclave，以及每次中断处理程序也将产生 EEXIT。因此使用 XSAVE 指令保存当前状态的开销会很大，使用优化

指令XSAVEC和XSAVEOPT可以减少这种开销，从而提高OCALL繁重工作负载的整体性能。

3　SGX-based LibOS系统调用机制分类

在宿主机的Linux内核中，系统调用的执行过程相当复杂，历经了3种执行系统调用的机制（如int 0X80软件中断、汇编指令SYSCALL/SYSENTER、vSDO执行SYSCALL）。SGX-based LibOS的系统调用的执行流程与宿主机执行Linux系统调用存在差异。本节介绍了系统调用（使用库函数API和直接使用汇编指令SYSCALL）在SGX-based LibOS中的执行流程。由于Gramine和Occlum的设计理念不同，在使用汇编SYSCALL指令执行时存在差异，因此将两者分开叙述。

3.1　库函数API

SGX-based LibOS中应用程序执行时，大多是直接链接库函数API，这样的好处是能够进行参数检查且执行速度更快。图2是应用程序使用库函数API在LibOS中的执行流程。首先应用程序使用标准库API，如malloc函数，在Glibc或Musl Libc标准C库被解析为系统调用brk()，而后brk()被LibOS层拦截并模拟。屏蔽层的作用是进行参数检验、地址转化、数据消毒等操作。随后LibOS通过OCALL机制请求外部的PAL（platform adaption layer）来请求宿主机上的系统调用，执行完后通过ECALL机制将结果返回至应用程序。

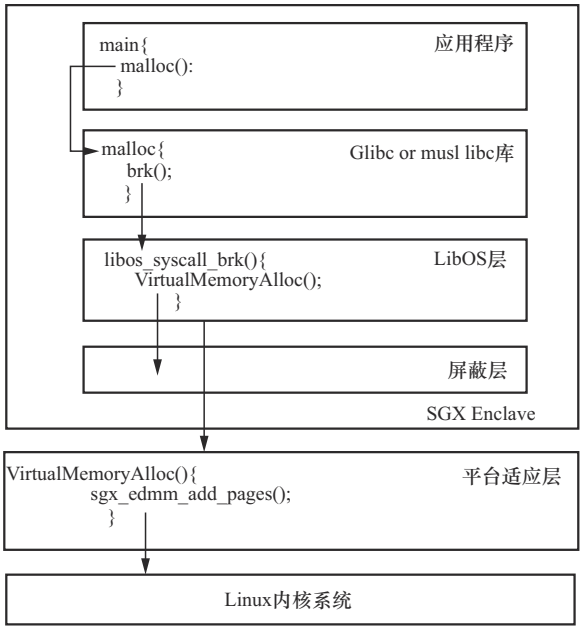

图2　库函数API执行流程

3.2　汇编指令SYSCALL

SYSCALL汇编指令指的是应用程序绕过Glibc/Musl libc标准C库直接发起的系统调用（如Go语言中静态编译的二进制文件）。Intel SGX硬件本身禁止SYSCALL指令，因此LibOS中会生成一个#UD（非法指令）异常，该异常被传递到进程中进行处理。

图3为Gramine在使用SYSCALL指令时触发的执行流程。应用程序使用SYSCALL指令来执行系统调用（还是以brk()系统调用为例），由于SGX Enclave内部不允许使用SYSCALL指令，会触发硬件中断，给出SIGILL信号表示指令非法。因此Enclave内部将产生中断异常，执行AEX（asynchronous enclave exit）操作，也就是退出Enclave到外部世界，并且将当前CPU信息和enclave线程执行的上下文存储在Enclave中安全的区域，也就是SSA（state save area）中，用来保护Enclave中的数据不被其他部分所影响。离开Enclave之后，线程进入PAL层进行异常的处理，Gramine会给出警告“尽量避免使用汇编SYSCALL指令，建议使用标准库提供的API”。然后从PAL层到内核中进行系统调用的处理。执行完系统调用后，使用ERESUME指令恢复上下文信息。

图4为Occlum在使用SYSCALL指令时触发的执行流程。Enclave内部不允许使用SYSCALL指令（由于Libc版本的ppoll()系统调用超时参数不被更新，Occlum允许使用SYSCALL指令执行），会发生中断异常且产生AEX操作，退出Enclave到SGX SDK进行异常处理，SDK会遍历到对应的处理程序，并且调用Occlum LibOS的异常处理程序。值得注意的是，Occlum将异常处理程序放置到LibOS内部，且将其作为一个Occlum独有的系统调用来处理，这点和Gramine的处理有所不同。Occlum LibOS将此异常处理后，最终将陷入Linux内核中执行该系统调用。

4　SGX-based LibOS系统调用测试框架

4.1　基于LTP的SGX-based LibOS系统调用测试框架概述

LTP（Linux test project）[21-22]是一个通用的、集成的、系统的测试套件，目的是帮助用户和Linux开发者更好地了解到可以正常使用的功能，

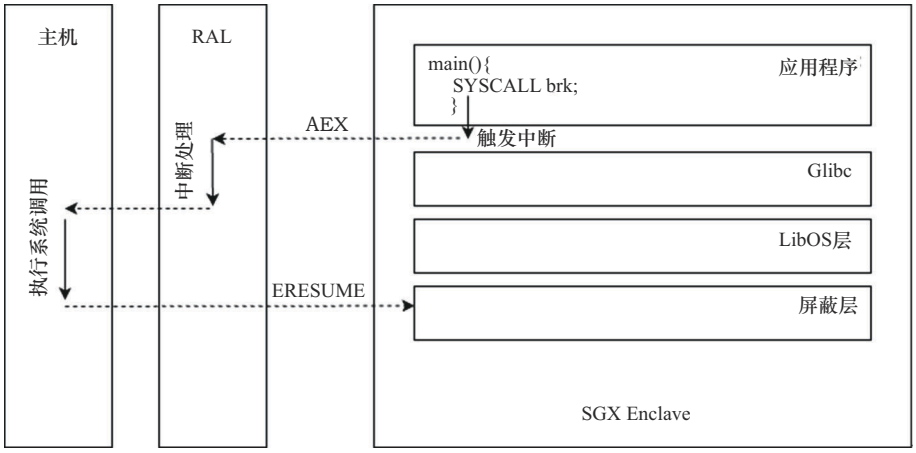

图3　Gramine 使用 SYSCALL 指令时触发的执行流程

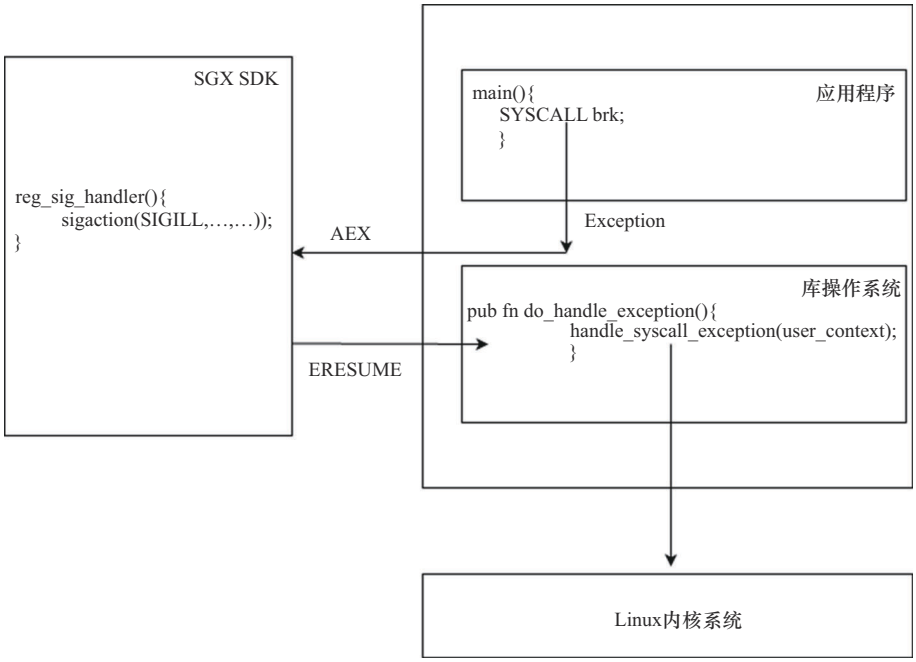

图4　Occlum 使用 SYSCALL 指令时触发的执行流程

是否存在有待改进的 Linux 特性。LTP 的目标是通过将自动化测试引入内核设计来改进 Linux 内核。例如，各家商业化定制的 Linux 内核（如 Red Hat Enterprise Linux、SUSE Linux Enterprise Server 等）都有使用到 LTP 测试套件来测试，以改进 Linux 内核和系统库。尽管大多数的 Linux 开发人员对自己新增的功能和修改的补丁漏洞都进行了单元测试，但没有一个系统的集成的测试工具。本文利用 LTP 工具开发了适用于 SGX-based LibOS 的测试框架，为开源社区提供一个测试套件，帮助验证 SGX-based LibOS 的可靠性、稳健性和稳定性。

4.2　测试环境

软件环境：LTP 版本为 20230929，3 个 SGX-LibOS 的版本分别是 Gramine V1.5、Occlum 0.29.3、Mystikos v0.11.0。其中 Occlum 使用 Docker 镜像，且 0.29.3 之后的版本强制需要 SGX2 支持。

硬件环境： Intel(R) Core(TM) i5-10505 CPU @ 3.20 GHz、支持 SGX1、SGX_LC（SGX Launch config）。

4.3　测试流程

如图 5 的整体测试流程所示，LTP 自动化测试系统的核心入口点是 runltp 脚本，该脚本提供了一套全面的参数配置选项。这些选项使得用户能够精

确定制测试环境、选择特定的测试集合以及灵活控制测试结果的输出格式和存储位置。当runltp脚本被触发时，它会根据用户的配置生成一个定制化的测试清单，随后调用测试执行引擎PAN来启动实际的测试流程。测试完成后，runltp会基于PAN返回的数据生成详细的测试报告。

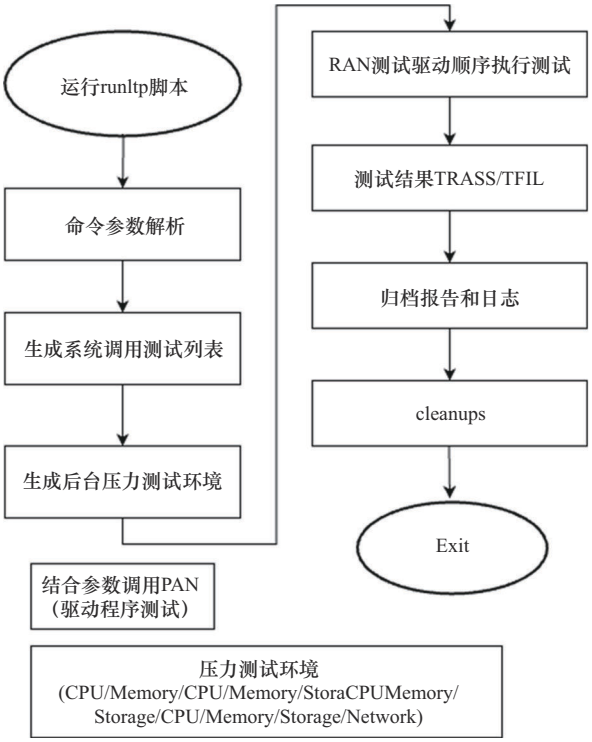

图5　整体测试流程

PAN作为LTP框架中的关键组件，是一组专门设计的测试驱动程序集合。它的主要职责是按照runltp传递的参数指令和测试清单，有序地执行每一个测试用例。在测试过程中，PAN不仅负责记录和输出详尽的执行信息，还会对每个测试用例的结果进行实时统计和分析。完成全部测试后，PAN会将汇总的测试结果反馈给runltp，为最终报告的生成提供必要的数据支持。

4.4　测试框架适配

LibOS的核心理念是通过实现操作系统，为应用程序提供操作系统虚拟化。鉴于Linux内核的复杂性和烦琐性，LibOS无法完全一比一复刻其全部功能，目前仅模拟了应用程序运行所需的最基本资源，包括进程、线程、进程间通信、内存管理、信号、文件系统、网络等。这些资源主要通过系统调用和一系列的文件实现。开发者的目标是确保大多数常规应用程序能够在SGX-based LibOS上运行，因此在LibOS中实现的系统调用数量是有限的。此外，由于Linux内核中系统调用的参数众多（其中标志参数可以根据需要的功能进行组合），并且实现某些功能可能需要增加可信计算基（TCB）或无法安全地实施，因此这些被模拟的系统调用通常只实现了基本功能。

在上述背景下，测试框架对LTP做了适配，做到运行在3个LibOS上的LTP的版本一致。对于Occlum，修改了LTP获取的内核版本为Linux kernel 4.19，以及将fork()系统调用修改为Occlum支持的vfork()系统调用。对这3个SGX-based LibOS的各个模块进行分组测试，还增加了一些Linux特性的测试样例（先前LTP未测试的部分）。

5　测试结果分析

本文深入探讨了SGX-based LibOS中目前实现系统调用所面临的挑战、系统调用的转发机制，以及用实验结果分析由兼容性不够和性能带来问题。分析显示，系统调用不完全支持主要源于SGX Enclave隔离特性和有限的资源访问能力，例如，epoll调用的复杂性在于其需要广泛访问和修改内核的事件通知结构，这在SGX的隔离模型中是不可行的。此外，涉及硬件操作或特定内核模块的系统调用调用，如直接内存访问（DMA）操作，也因SGX的安全隔离而难以实施。

5.1　测试用例及其特性

在进行SGX-based LibOS的测试之前，本文设计了一系列测试用例，涵盖了系统的各个模块和Linux特性。这些测试用例旨在全面评估LibOS的功能、性能和兼容性。

将测试用例按照SGX-based LibOS的主要模块进行分组，包括内存管理（内存的分配与释放、地址空间隔离等）、文件系统（基本的文件操作、目录操作等）、进程管理（进程创建与终止、进程间通信、线程管理等）、网络以及系统调用等测试模块。

为了确保SGX-based LibOS与标准Linux系统的兼容性，在其中增加了以下Linux特性的测试样例：POSIX兼容性测试、进程间通信（IPC）测试、网络协议栈测试、系统资源管理测试（资源限制、Cgroup功能、内存超限制测试）等，并在后

面进行阐述：SGX-based LibOS 仍存在与 Linux 系统不兼容或功能上不完整等问题。

5.2　功能性分析

本节从 SGX-based LibOS 实现的功能性（包括 Linux 特性、支持的编程语言、对于多进程的支持）和对系统调用的模拟程度进行了分析和对比。

1)语言支持。手动检查了 SGX-based LibOS 工具链规范，以确认每个 SGX-based LibOS 的语言支持。表 1 展示了 SGX-based LibOS 编程语言支持情况。例如，Golang 在 Gramine 中没有被支持，Golang 是目前开发应用程序流行的语言之一，然而其独特的设计，即不通过 Libc 来发起系统调用，对 SGX-based LibOS 而言是极大的挑战。可行的方案是对 Go 的工具链进行修改，但因工作烦琐且不易维护而不被支持[14]。

表 1　　语言支持

语言	Gramine	Occlum	Mystikos
C	√	√	√
Java	√	√	√
.Net	√	×	√
Golang	×	√	×
Rust	√	√	√
Bash	√	√	×
Python	√	√	√

2) Linux 特性。Linux 操作系统以其功能强大、稳定性和灵活性著称，支持全球数以百万计的服务器、桌面和嵌入式设备。目前的 SGX-based LibOS 做到了 ABI 级别的兼容性，未能达到内核级别的兼容，难以达到与 Linux 内核相媲美的程度。由于实现的复杂度高，SGX-based LibOS 通常在确保 LibOS 安全的情况下，实现目前开发者频繁使用且 Linux 内核社区仍在维护的 Linux 特性[23]。使用 Linux 测试项目（LTP）的自动化工具对主流的 SGX-based LibOS 进行系统调用测试，将 Linux 特性的支持情况整理至表 2。其中，○表示没有实现，□表示部分特性实现，●表示所有功能基本实现，dummy 是指这部分的系统调用返回的是原先设定好的固定值，待完善是指在 LibOS 中常用的基本功能使用出现问题，受限制是指这部分的系统调用实现的功能只能在特定的情形下使用，例如，Gramine 中的文件锁不能被中断且只能在常规文件中使用文件锁（不能对 pipes、sockets 等使用）。高级功能包括 BPF、内核模块、命名空间、内存保护密钥（pkey）、属性拓展（xttr）、零拷贝传输、进程间数据传输、文件系统配置上下文、内核密钥管理等均属于 Linux 中的高级功能，这些功能是 Linux 内核的新增功能，应用程序很少或根本不使用，因此在 SGX-based LibOS 中通常不被模拟或转发。

表 2　　Linux 特性实现情况

Linux 特性	Gramine	Occlum	Mystikos
进程/线程	□(受限制)	□	□
文件 I/O	●	●	●
文件锁	□(受限制)	○	○
网络(socket)	●	□	□
信号	●	□	□
用户(组)管理	□(dummy)	□(dummy)	□(dummy)
I/O 多路复用	□(受限制)	□	□
管道	●	●	●
内存管理	□(受限制)	□(待完善)	□(待完善)
异步 I/O	○	○	○
共享内存	○	□	○
消息队列/信号量	○	○	○
系统/资源信息	□(dummy)	□(dummy)	○
日期时间类	●	□	○
调度决策	□(dummy)	□	○
高级功能	○	○	○

5.3　安全性分析

本节探讨了 SGX-based LibOS 用于 Linux 宿主机系统调用行为不一致的现象，并指出 SGX-based LibOS 中存在不完善或安全隐患的系统调用和机制。

5.3.1　文件系统

Gramine、Occlum 和 Mystikos 提供基本的文件系统机制[24]：支持文件系统操作 write()、read()、lseek()、open()、close()等系统调用；文件系统事件的监管 fanotify()、inotify()等系统调用不被支持；文件锁 fcntl()和文件链接 link()相关的系统调用受到限制。不同的是 Occlum 依靠 JinDisk[25-26]还实现了异步的文件系统 Async-SFS。在安全的基础上，实

现了更加高效的I/O效率。由于进程设计的差异，Gramine和Mystikos的每个进程都开辟一个Enclave，文件系统的元数据将分布在各个Enclave。导致文件系统有如下限制：不支持动态的挂载文件系统；进程之间文件不能同步；文件的时间戳不能被设置和更新；实现的文件锁机制不能被中断等。

目前Mystikos支持3种文件系统：ramfs、ext2fs和hostfs。其中，ramfs和hostfs都是未加密的文件系统且没有进行完整性检查。hostfs文件系统被用于LibOS和host主机间数据的交互，因此无须保护和完整性验证，但ramfs没有被保护和完整性检查在SGX框架中是相当危险的，并且ramfs文件系统会在Enclave终止时造成数据丢失等问题。ext2fs托管在rootfs的非TEE文件系统中可以进行完整性保护和加密。因此，对于Mystikos而言，还需要加强文件系统机制的安全性，以确保数据的保密性和完整性。

5.3.2 共享内存、信号量和消息队列

内存共享、信号量和消息队列都是Linux操作系统中常见进程间通信（IPC）的机制，内存共享是允许不同进程共享同一块物理内存；信号量是用来实现进程同步的机制，主要用来实现对共享资源的同步和互斥；消息队列允许不同进程之间通过一个共享消息队列来交换信息。

共享内存、信号量以及消息队列都有POSIX和System V这2种实现，其概念是类似的。Gramine、Mystikos底层设计因素，导致数据将分散在各个Enclave中，难以安全地实现System V的内存共享、信号量；对POSIX内存共享、信号量也只是针对特殊样例进行实现（如在与GPU通信时），且不支持消息队列机制。Occlum独有的SIP设计，使数据文件和多个进程都在同一个Enclave内。Occlum实现了内存共享机制，没有对信号量以及消息队列机制进行支持。

测试发现，Occlum在内存共享机制方面的系统调用仍然不够完善，涉及shmat()、shmctl()、shmat()等system V IPC相关的系统调用。例如，测试结果显示shmget()系统调用申请的共享内存块大小和使用shmctl获取到的内存块大小不一致。此外，还观察到shmat()系统调用中的const void *_Nullable shmaddr参数在Occlum中只能被设置为0或共享内存段的起始地址，不能进行人为的指定地址。出于安全考虑，不允许指定内存地址是常规的操作。

5.3.3 内存管理

Linux内存管理是内核最复杂的任务之一，涉及许多与CPU相关的功能。在SGX-based LibOS中，通常只实现内存管理的最小子集，以确保大多数的应用程序能够运行。

Gramine实现了多个与内存管理相关的系统调用，如mmap()、munmap()、brk()、msync()、mprotect()等。这些系统调用提供了应用程序必需的基本功能，但一些特殊的标志和功能不被支持。例如，许多提高性能的系统调用（如set_mempolicy()、get_mempolicy()）、某些标志（如msyc系统调用的MS_INVALIDATE标志）和功能（如mincore()、mlock()、munlock()等返回虚拟值）都未在Gramine中实现。因此当应用程序在Gramine运行时，出现性能下降是一种常见现象。

尽管Occlum是使用Rust语言编写的SGX-based LibOS，其内存安全特性有助于避免许多低级的内存安全错误。然而，在实验过程中，仍然发现了一些内存相关系统调用的错误。特别是涉及mprotect()、mmap()、mremap()等系统调用的问题，例如，mprotect()能够错误地将通过mmap()设置为只读权限的文件，更改为写权限。

5.3.4 I/O多路复用

I/O多路复用是指一个进程可以处理多个I/O请求，Linux下提供了3种I/O多路复用的API，分别是select、poll、epoll。其中，select和poll本质相同，内部使用线性结构来存储进程关注的socket集合，当用户关注的socket越多监管起来就越困难；epoll解决了select、poll的问题，内部使用红黑树这类高效的数据结构关注socket集合，同时使用事件驱动机制将触发的socket集合发送给应用程序，而不像select、poll进行轮询。

本文所调研的SGX-based LibOS均支持Linux中这3种I/O多路复用API。它们通过模拟Linux主机系统调用ppoll()，实现了select()、pselect()、poll()以及epoll系列的系统调用epoll_*()。但在测试时发现，Gramine的epoll()系统调用忽略了EPOLLWAKEUP这个标志（因为Gramine不支持自动休眠）。同时Gramine模拟实现的epoll系列系统调用还有如下限制：进程之间不共享epoll实例

（这是由于进程被 Enclave 所隔离，参考 4.1 节）；不允许将 epoll 添加到另一个 epoll 实例中。测试也检测出 Occlum 中 epoll 的问题：Occlum 中 2 个 epoll 文件可以互相监听，出现嵌套循环，造成死锁问题；Occlum 中对 epoll 的嵌套没有设限制，能够一直嵌套下去。对上述提及 SGX-based LibOS 中待修复和有局限性的系统调用 API 进行整理，如表 3 所示。

表 3　　相关系统调用

系统调用	Gramine	Occlum	Mystikos
shmat	×	√(有限制)	×
shmctl	×	√(待修复)	×
shmget	×	√	×
mmap	√(有限制)	√(待修复)	√(有限制)
mprotect	√(有限制)	√(待修复)	√(有限制)
mremap	×	√(待修复)	×
fork	√(待修复)	×	√(有限制)
epoll_*	√(有限制)	√(待修复)	√(有限制)

针对以上限制，未来研究可以探索如何通过软件桥接层或修改 LibOS 的设计来增强对这些系统调用的支持。此外，研究如何在保持 SGX 安全性的前提下，优化系统调用性能和兼容性，是进一步提升 SGX-based LibOS 实用性的关键。

6　结束语

本文首先介绍了 SGX 的相关背景以及目前流行的 SGX-based LibOS。随后通过对系统调用的研究，深入探讨了以下几个方面的内容：系统调用在 SGX-based LibOS 上面临的挑战与解决方案；系统调用的转发机制；使用动态测试框架分析系统调用，比较其与 Linux 宿主机在行为上的差异性。通过对系统调用的深入分析和归纳总结，旨在为用户及 LibOS 的开发者提供有效的帮助，帮助他们更好地理解和优化 LibOS 的设计和实现。

参考文献：

[1] 李凤华, 李晖, 贾焰, 等. 隐私计算研究范畴及发展趋势[J]. 通信学报, 2016, 37(4): 1-11.
LI F H, LI H, JIA Y, et al. Privacy computing: concept, connotation and its research trend[J]. Journal on Communications, 2016, 37(4): 1-11.

[2] Intel Corporation. Intel software guard extensions (Intel SGX)[R]. 2019.

[3] AMD. AMD SEV-SNP: strengthening VM isolation with integrity protection and more white paper[R]. 2020.

[4] PINTO S, SANTOS N. Demystifying arm TrustZone[J]. ACM Computing Surveys, 2019, 51(6): 1-36.

[5] 董春涛, 沈晴霓, 罗武, 等. SGX 应用支持技术研究进展[J]. 软件学报, 2021, 32(1): 137-166.
DONG C T, SHEN Q N, LUO W, et al. Research progress of SGX application supporting techniques[J]. Journal of Software, 2021, 32(1): 137-166.

[6] ZHANG F, CECCHETTI E, CROMAN K, et al. Town crier: an authenticated data feed for smart contracts[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2016: 270-282.

[7] 汤凌韬, 陈左宁, 张鲁飞, 等. 联邦学习中的隐私问题研究进展[J]. 软件学报, 2023, 34(1): 197-229.
TANG L T, CHEN Z N, ZHANG L F, et al. Research progress of privacy issues in federated learning[J]. Journal of Software, 2023, 34(1): 197-229.

[8] Enarx. Confidential computing with Webassembly[R]. 2021.

[9] AHMAD A, KIM J, SEO J, et al. CHANCEL: efficient multi-client isolation under adversarial programs[C]//Proceedings 2021 Network and Distributed System Security Symposium. Reston: Internet Society, 2021: 1-18.

[10] LIU W J, CHEN H B, WANG X F, et al. Understanding TEE containers, easy to use? hard to trust[J]. arXivPreprint, arXiv: 2109.01923, 2021.

[11] PORTER D E, BOYD-WICKIZER S, HOWELL J, et al. Rethinking the library OS from the top down[C]//Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems. New York: ACM Press, 2011: 291-304.

[12] PRIEBE C, MUTHUKUMARAN D, LIND J, et al. SGX-LKL: securing the host OS interface for trusted execution[J]. arXiv Preprint, arXiv: 1908.11143, 2019.

[13] TSAI C C, PORTER D E, VIJ M. Graphene-SGX: a practical library OS for unmodified applications on SGX[C]//2017 USENIX Annual Technical Conference (USENIX ATC 17). Berkeley: USENIX Association, 2017: 645-658.

[14] Gramine. The gramine project[R]. 2024.

[15] SHEN Y R, TIAN H L, CHEN Y, et al. Occlum: secure and efficient multitasking inside a single enclave of intel SGX[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 955-970.

[16] Occlum. Occlum team[R]. 2024

[17] Mystikos. Deis Labs[R]. 2024

[18] TIAN H L, ZHANG Q, YAN S M, et al. Switchless calls made practical in intel SGX[C]//Proceedings of the 3rd Workshop on System Software for Trusted Execution. New York: ACM Press, 2018: 22-27.

[19] ORENBACH M, LIFSHITS P, MINKIN M, et al. Eleos: ExitLess OS services for SGX enclaves[C]//Proceedings of the Twelfth European Conference on Computer Systems. New York: ACM Press, 2017: 238-253.

[20] SONG K T, OU S Q. A client-server architecture for object volume measurement on a conveyor belt[C]//2019 12th Asian Control Conference (ASCC). Piscataway: IEEE Press, 2019: 901-906.

[21] LARSON P. Testing linux with the linux test project[C]//Proceedings of the Ottawa Linux Symposium. Piscataway: IEEE Press, 2002: 265-273.
[22] Linux. Linux test project[R]. 2024.
[23] TSAI C C, JAIN B, ABDUL N A, et al. A study of modern linux API usage and compatibility: what to support when you’re supporting[C]// Proceedings of the Eleventh European Conference on Computer Systems. New York: ACM Press, 2016: 1-16.
[24] AHMAD A, KIM K, SARFARAZ M I, et al. OBLIVIATE: a data oblivious filesystem for intel SGX[C]//Proceedings 2018 Network and Distributed System Security Symposium. Reston: Internet Society, 2018: 1-15.
[25] Github. Jinzhao-disk[R]. 2024.
[26] Github. Mlsdisk[R]. 2024.

[作者简介]

刘西蒙（1988– ），男，陕西西安人，博士，福州大学教授、博士生导师，主要研究方向为密码学、人工智能安全。

黄应康（2000– ），男，江西上饶人，福州大学硕士生，主要研究方向为系统安全、可信计算。

刘维杰（1991– ），男，湖北武汉人，博士，南开大学副教授、硕士生导师，主要研究方向为系统安全、可信计算环境和二进制程序分析。

范倍汐（2000– ），女，四川眉山人，山西师范大学硕士生，主要研究方向为容器安全、漏洞挖掘。

章恬（2001– ），女，江西上饶人，福州大学硕士生，主要研究方向为云原生安全、漏洞挖掘。

张杰（1996– ），女，山东临沂人，山西师范大学硕士生，主要研究方向为系统安全、容器安全。