

**硕士学位论文**

**针对Intel SGX SDK的安全增强框架技术研究与实现**

**作者姓名： 陈力恒**

**指导教师： 马恒太 副研究员**

**中国科学院软件研究所**

**学位类别： 工学硕士**

**学科专业： 计算机软件与理论**

**培养单位： 中国科学院软件研究所**

**2021年6月**

**Research and Implementation of Security Enhancement Framework Technology for Intel SGX SDK**

**A thesis submitted to**

**University of Chinese Academy of Sciences**

**in partial fulfillment of the requirement**

**for the degree of**

**Master of Science in Engineering**

**in Computer Software and Theory**

**By**

**CHEN Liheng**

|  |  |  |
| --- | --- | --- |
| **Supervisor:** |  | **Associate Professor MA Hengtai** |
|  |  |  |

**Institute of Software Chinese Academy of Sciences**

**June 2021**

**中国科学院大学**

**研究生学位论文原创性声明**

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：

日 期：

**中国科学院大学**

**学位论文授权使用声明**

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名： 导师签名：

日 期： 日 期：

**摘　要**

云计算技术不断发展，吸引越来越多的用户将应用程序部署到云平台。云平台上大量应用程序对安全敏感，如何防止攻击者窃取应用程序中敏感内容是云平台的热难点问题。

Intel SGX所提供的用户态可信执行环境Enclave能有效解决云计算等场景中的远程计算安全问题，保护云平台上应用程序敏感内容。Intel SGX主要通过密码学方法和访问控制保障应用程序代码数据的机密性和完整性，抵御Ring0权限攻击。但最近研究表明攻击者能通过Enclave接口和共享资源等攻击面开展攻击，破坏SGX安全性。SGX Enclave的安全不仅依赖于强隔离机制，还依赖于安全可靠的软件架构以确保安全的接口调用，开发者如果忽视接口安全，SGX Enclave将丧失安全保障。

本文对SGX安全增强技术展开研究，旨在抵御SGX攻击并提升SGX安全性。本文主要贡献包括：

1. 现有SGX攻击向量能够破坏SGX安全性，但已有防御方案针对性地防御特定SGX攻击向量或攻击面，各个方案遵循的安全准则不统一，至今没有一个方案对SGX安全关键点进行全面分析，导致现有SGX架构难以有效兼容这些防御方案。针对上述问题，本文以SGX软件栈为主刻画了SGX架构及其关键执行路径，总结了其中的安全关键点。
2. 针对利用SGX SDK无法管控Enclave线程执行序及无法审计安全/非安全上下文切换事件等安全问题透过Enclave接口及共享资源等攻击面的多种SGX攻击向量，本文设计实现了针对Intel SGX SDK的安全增强框架（SGX-SEF），审计SGX关键执行路径，兼容已有安全准则防御多种攻击向量。
3. 针对恶意篡改Enclave代码调用顺序的调用排序和并发调用攻击向量，本文提出了“调用顺序白名单”安全策略，基于此策略审计检查关键执行路径，判断调用顺序是否符合安全策略。
4. 实验验证了SGX-SEF能有效审计SGX关键执行路径，评估了SGX-SEF审计功能引起的性能开销，分析了引起性能开销的原因。审计功能开销随日志容量增大而增大，开销包括日志使用维护开销及缓存未命中开销。当日志大小100项时，审计功能对各种调用方式引起的开销增幅不明显，100项大小的日志足够记录最近的安全事件用于审计检查功能。日志容量为100项时，开销增幅在最优情况（Ordinary调用方式）下为13%~18%。实验验证了防御者可以在SGX-SEF中灵活部署安全策略以抵御多类攻击向量。

**关键词**：Intel SGX，安全增强框架，关键执行路径，安全关键点

**Abstract**

The continuous development of cloud computing technology attracts more and more users to deploy applications on cloud platforms. A large number of applications on cloud platforms are sensitive to security. How to prevent attackers from stealing sensitive content in applications is a hot and difficult problem for cloud platforms.

The user-mode trusted execution environment "Enclave" provided by Intel SGX can effectively solve remote computing security issues in cloud computing and other scenarios, and protect sensitive content of applications on the cloud platform. Intel SGX mainly protects the confidentiality and integrity of application code data through cryptographic methods and access control, and resists Ring0 permission attacks. However, recent studies have shown that attackers can attack through the Enclave interface, shared resources and other attack surfaces to undermine the security of SGX. The security of SGX Enclave not only relies on a strong isolation mechanism, but also relies on a secure and reliable software architecture to ensure secure interface calls. If developers ignore interface security, SGX Enclave will lose security guarantees.

This paper conducts research on SGX security enhancement technology, which aims to resist SGX attacks and improve SGX security. The main contributions of this paper include:

1. Existing SGX attack vectors can undermine the security of SGX, but there are existing defense schemes to specifically defend against specific SGX attack vectors or attack surfaces. The security principles followed by each scheme are not uniform. So far, there is no scheme to conduct a comprehensive analysis of SGX security critical points, which makes it difficult for the existing SGX architecture to effectively compatible with these defense schemes. In response to the above problems, this paper analyses the SGX architecture and its critical execution paths mainly about SGX software stack, and summarizes the security critical points.
2. In response to security issues such as SGX SDK's inability to control Enclave thread execution order and the inability to audit secure/non-secure context switching events, which are leveraged by various SGX attack vectors to attack through the attack surfaces of Enclave interfaces and shared resources, this paper designs and implements Security Enhancement Framework for Intel SGX SDK (SGX-SEF), audits the SGX critical execution path, and is compatible with existing security principles to defend against multiple attack vectors.
3. Aiming at the Call Permutation and Concurrent Calls attack vectors that maliciously tamper with the calling sequence/timing of Enclave code, this paper proposes the "calling sequence/timing white-list" security strategy, auditing and checking the critical execution paths, and judging whether the calling sequence/timing complies with this security strategy.
4. The experiments verify that SGX-SEF can effectively audit the critical execution path of SGX, evaluates the performance overhead caused by the SGX-SEF audit function, and analyzes the reasons for the performance overhead. The audit function overhead increases as the log capacity increases, and the overhead includes log maintenance overhead and cache miss overhead. When the log size is 100 items, the audit function will not cause significant overhead for various calling methods. A log size of 100 items is enough to record the most recent security events for the audit check function. When the log capacity is 100 items, the overhead increase is 13%-18% under the optimal situation (Ordinary call method). The experiments also verify that the defender can flexibly deploy security strategies in SGX-SEF to resist multiple attack vectors.

**Key Words**: Intel SGX, Security Enhancement Framework, Critical Execution Path, Security Critical Point

**目　录**

[第一章　引言 1](#_Toc3721)

[1.1　选题背景和意义 1](#_Toc1393)

[1.2　相关研究概述 2](#_Toc1039)

[1.3　主要研究内容和贡献 3](#_Toc9831)

[1.4　论文结构 5](#_Toc19510)

[第二章　国内外相关研究现状 7](#_Toc15862)

[2.1　Intel SGX 7](#_Toc25121)

[2.1.1　Intel SGX编程模型 8](#_Toc24717)

[2.1.2　Intel SGX软硬件架构 8](#_Toc24310)

[2.1.3　SGX Enclave安全机制 10](#_Toc1893)

[2.2　SGX攻击现状 11](#_Toc12550)

[2.2.1　源于Enclave开发中的安全问题 12](#_Toc24963)

[2.2.2　源于SGX设计存在安全缺陷 13](#_Toc17755)

[2.2.3　代表性攻击 14](#_Toc409)

[2.3　传统系统防御方案 19](#_Toc32383)

[2.4　SGX防御现状 20](#_Toc30751)

[2.4.1　Enclave接口消毒 20](#_Toc27323)

[2.4.2　Enclave内外资源隔离、痕迹防泄露及攻击痕迹检测 23](#_Toc8995)

[2.4.3　其他防御方案 24](#_Toc32038)

[2.5　本章小结 24](#_Toc2459)

[第三章　Intel SGX关键执行路径及安全关键点分析 25](#_Toc26673)

[3.1　Enclave接口关键执行路径 25](#_Toc1598)

[3.1.1　ECALL进入Enclave 25](#_Toc28057)

[3.1.2　ECALL结束退出Enclave 27](#_Toc26161)

[3.1.3　OCALL离开Enclave及OCALL结束返回Enclave 27](#_Toc22245)

[3.1.4　AEX离开Enclave及异常处理完成返回Enclave 28](#_Toc518)

[3.2　其他关键执行路径 29](#_Toc8922)

[3.3　SGX线程模型 32](#_Toc19431)

[3.4　本章小结 35](#_Toc6029)

[第四章　针对Intel SGX SDK的安全增强框架设计 37](#_Toc25572)

[4.1　针对Intel SGX SDK的安全增强框架总体设计 37](#_Toc9718)

[4.1.1　审计检查点插桩方式 37](#_Toc2989)

[4.1.2　审计检查点插桩位置 38](#_Toc23885)

[4.1.3　安全事件审计 39](#_Toc15589)

[4.1.4　安全事件检查 39](#_Toc1189)

[4.2　对SGX原生软件栈的修改 41](#_Toc14650)

[4.3　本章小结 42](#_Toc31148)

[第五章　针对Intel SGX SDK的安全增强框架验证 43](#_Toc22706)

[5.1　实验环境 43](#_Toc27074)

[5.2　审计功能效果验证及性能分析 43](#_Toc16029)

[5.2.1　审计功能效果验证 44](#_Toc12444)

[5.2.2　审计功能性能开销分析 44](#_Toc11996)

[5.3　防御（检查功能）效果验证 46](#_Toc1428)

[5.3.1　针对“调用排序”攻击向量的防御效果验证 46](#_Toc25206)

[5.3.2　针对“并发调用”攻击向量的防御效果验证 49](#_Toc26715)

[5.3.3　针对“恶意Enclave线程调度”攻击向量的防御效果验证 52](#_Toc507)

[5.3.4　针对“时间侧信道”的防御效果验证 56](#_Toc28770)

[5.4　本章小结 57](#_Toc31687)

[第六章　总结与展望 59](#_Toc31867)

[6.1　总结 59](#_Toc23974)

[6.2　展望 60](#_Toc27770)

[参考文献 61](#_Toc21619)

[致　谢 65](#_Toc7387)

[作者简历及攻读学位期间发表的学术论文与研究成果 67](#_Toc18837)

**图目录**

[图1.1　云平台上部署的服务 1](#_Toc14320)

[图1.2　Intel SGX保护云平台上服务程序安全 2](#_Toc18509)

[图2.1　Enclave应用生命周期](#_Toc12186)[[30]](#_Toc12186) [8](#_Toc12186)

[图2.2　SGX软硬件架构 9](#_Toc31745)

[图2.3　SGX Enclave安全问题、攻击面和攻击向量 12](#_Toc5776)

[图2.4　调用排序攻击示意图 15](#_Toc1623)

[图2.5　AsyncShock攻击示意图 17](#_Toc11703)

[图2.6　Haven组件和接口](#_Toc23285)[[26]](#_Toc23285) [21](#_Toc23285)

[图2.7　Graphene-SGX架构图](#_Toc3720)[[27]](#_Toc3720) [22](#_Toc3720)

[图2.8　SCONE架构图](#_Toc11456)[[28]](#_Toc11456) [22](#_Toc11456)

[图3.1　Ordinary ECALL代码流程图 25](#_Toc15798)

[图3.2　Switchless ECALL的代码流程图 26](#_Toc13730)

[图3.3　AEX及异常处理流程图 29](#_Toc20752)

[图3.4　受保护文件系统布局](#_Toc11147)[[4]](#_Toc11147) [30](#_Toc11147)

[图3.5　本地认证（使用DH密钥交换库）](#_Toc11434)[[4]](#_Toc11434) [31](#_Toc11434)

[图3.6　安全信道的建立（使用DH密钥交换库）](#_Toc19518)[[4]](#_Toc19518) [32](#_Toc19518)

[图3.7　SGX应用中Ordinary/Switchless ECALL/OCALL线程模型 33](#_Toc12737)

[图4.1　插桩方式对比 37](#_Toc24959)

[图4.2　OCALL Stub函数示意图 38](#_Toc30371)

[图4.3　审计检查点内部架构 40](#_Toc29830)

[图4.4　SGX-SEF对SGX软件栈的修改 41](#_Toc7885)

[图5.1　审计功能的效果 44](#_Toc18243)

[图5.2　调用排序攻击之前程序执行结果图 46](#_Toc26580)

[图5.3　调用排序攻击之后程序执行结果图 47](#_Toc20756)

[图5.4　调用排序攻击检测结果 47](#_Toc14895)

[图5.5　调用排序攻击前SGX\_SQLite执行情况图 48](#_Toc9531)

[图5.6　调用排序攻击后SGX\_SQLite执行情况图 48](#_Toc25414)

[图5.7　SGX\_SQLite中调用排序攻击检测结果 49](#_Toc7567)

[图5.8　并发调用攻击之前程序执行结果 49](#_Toc13772)

[图5.9　恶意篡改并发调用 50](#_Toc7537)

[图5.10　并发调用攻击之后程序执行结果 50](#_Toc1103)

[图5.11　并发调用攻击检测结果 51](#_Toc258)

[图5.12　并发调用攻击前SGX\_SQLite执行情况图 51](#_Toc10765)

[图5.13　并发调用攻击后SGX\_SQLite执行情况图 52](#_Toc3220)

[图5.14　SGX\_SQLite中并发调用攻击检测结果 52](#_Toc5642)

[图5.15　恶意线程调度攻击前程序执行结果 53](#_Toc28766)

[图5.16　恶意中断ecall\_free\_obj1函数 54](#_Toc3167)

[图5.17　恶意线程调度攻击后程序执行结果 54](#_Toc13427)

[图5.18　使用Kprobe的页错误型Enclave线程暂停原语示意图 55](#_Toc4565)

[图5.19　使用SIGSEGV的页错误型Enclave线程暂停原语示意图 55](#_Toc8437)

[图5.20　恶意线程调度攻击检测结果 56](#_Toc13746)

[图5.21　SGX-Step/Nemesis源码分析流程图 56](#_Toc30341)

**表目录**

[表2.1　“Enclave接口”攻击表 18](#_Toc32403)

[表2.2　“微架构共享资源”攻击表 18](#_Toc22557)

[表2.3　“部分功能依赖内核”攻击表 19](#_Toc9500)

[表2.4　“可观测的物理信息”攻击表 19](#_Toc1219)

[表2.5　“密封文件”攻击表 19](#_Toc149)

[表5.1　基于Switchless的审计功能性能开销测量表 45](#_Toc992)

[表5.2　基于SampleEnclave的审计功能性能开销测量表 46](#_Toc14416)

# 第一章　引言

## 1.1　选题背景和意义

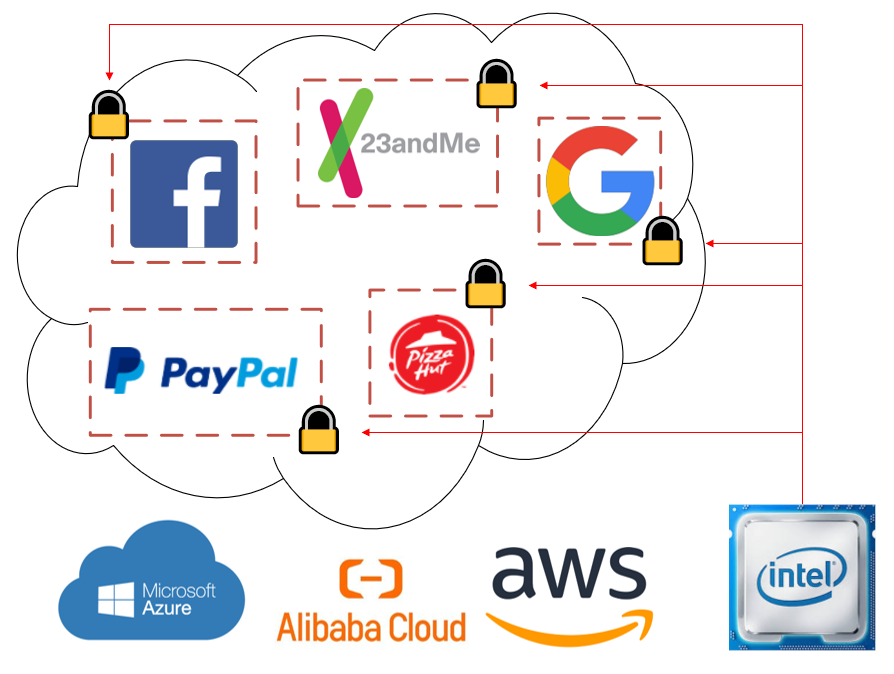
云计算不断发展，吸引越来越多的用户将应用程序部署到云平台。根据Gartner 2019年的报告，全球云计算市场（IaaS部分）份额达到445亿美元，市场份额前三名的云平台分别为亚马逊AWS、微软Azure和阿里云。部署在云平台上的应用程序（如图1.1所示）包括Facebook（即时通信服务程序）、23andMe（基因技术服务程序）及PayPal（网络支付服务程序）等。



**图1.1**　云平台上部署的服务

**Figure1.1**　Services deployed on the cloud platform

云平台上的大量应用程序对安全敏感。尽管云平台已经部署了Web应用防火墙和DDOS防御系统等，针对云平台上应用程序的攻击事件依然频发。黑客能够构建提权攻击，云平台管理员能够直接访问云用户内容，使得云用户无法完全信任云平台。如何保护远程计算（如云平台上程序的计算过程及计算内容）安全性的问题被论文[1]总结为远程计算安全问题。Intel Software Guard Extensions（简称SGX）能够有效解决远程计算安全问题，保护远程计算安全性。SGX构建了用户态可信执行环境Enclave，通过密码学方法和访问控制保护Enclave免受Ring0权限攻击。云平台上的应用程序可以使用SGX保护其安全（图1.2）。



**图1.2**Intel SGX保护云平台上服务程序安全

**Figure1.2**Intel SGX protects the security of service programs on the cloud platform

阿里云研发了“基于SGX2.0和TPM的虚拟化实例”和“系统可信解决方案”，标志着阿里云具备提供完整云可信产品的能力。其它云平台（如亚马逊AWS和微软Azure）也应用Intel SGX增强云安全能力，达到企业及国家法规要求（如中国国标等保2.0）。

2013年SGX概念被提出，2015年SGX在Intel Skylake架构CPU中得到部署。此后，SGX在云安全、网络通信和本地应用中被大量应用，多方计算、机器学习、区块链、人工智能和生物识别技术保护等场景逐渐使用SGX解决其问题[2]。但最近研究表明，Enclave开发过程存在诸多安全问题，SGX设计存在安全缺陷，攻击者针对这些安全问题透过SGX攻击面利用各种攻击向量破坏SGX安全性。对此，本文旨在提出一种框架性的防御方案全面加固SGX。

## 1.2　相关研究概述

Intel SGX[3-5]旨在提供用户态的可信执行环境（Trusted Execution Environment，缩写为TEE），保护并证明计算过程及计算内容可信。TEE技术是可信计算技术的发展，基于隔离技术保护程序敏感内容。TEE技术实现上可分为基于软件的和基于硬件的，基于CPU硬件的除Intel SGX外还包括ARM TrustZone、RISC-V Keystone、AMD SEV，基于GPU硬件的包括论文[6,7]等，基于软件的包括Virtual Ghost[8]等。基于软件的TEE能快速形成原型系统，基于硬件的TEE性能得到提升。TEE技术根据架构可分为全栈式TEE和用户态TEE（常称为Enclave），前者将计算资源划分为安全世界和非安全世界，两者均包含用户态和内核态架构但彼此隔离，SGX等提供的用户态可信执行环境更着重于保护用户态应用程序的敏感内容。

本文关注的SGX主要通过密码学方法和访问控制保护用户态可信执行环境Enclave免受Ring0权限攻击[1,9,10]。但最近研究表明，开发者开发Enclave过程中未严格遵循SGX开发规范滋生安全问题，SGX设计中对安全场景考虑不充分存在安全缺陷，攻击者针对安全问题利用“缓存侧信道”、“页表攻击”、“影子分支攻击”及“调用排序攻击”等攻击向量开展攻击[11-25]，严重破坏SGX Enclave的安全性。这些攻击向量主要透过Enclave接口和共享资源攻击面。Enclave接口是指Enclave向外暴露的接口，攻击者能够轻易操控Enclave接口攻击面展开攻击。而共享资源是指Enclave内外软硬件层面的共享资源，攻击者观察共享资源上的信息变化推测Enclave内的执行流或数据流，但共享资源攻击面难以被发现且难以被有效利用。

针对SGX攻击，有若干防御方案被提出[26-38]，它们在Enclave代码中对Enclave接口进行消毒，将Enclave内外共享资源用更彻底的隔离机制进行隔离避免Enclave内将状态等信息泄露到Enclave内外共享资源，检测攻击代码在Enclave内留下的痕迹。但分析发现各个防御方案所遵循的安全准则不统一，难以被SGX架构同时兼容。至今没有防御方案深入刻画SGX架构，全面加固SGX。

## 1.3　主要研究内容和贡献

SGX安全问题深刻影响SGX实际应用，本文为了系统分析SGX架构，增强SGX安全性，展开了如下研究：

1. Intel SGX关键执行路径及安全关键点分析。归纳现有SGX攻击所针对的安全问题、所利用的攻击向量及所涉及的攻击面，分析SGX架构关键执行路径及安全关键点位，加固SGX安全关键点以提升SGX自身安全性。
2. 针对Intel SGX SDK的安全增强框架设计。本文基于SGX原生架构设计针对Intel SGX SDK的安全增强框架（简称SGX-SEF），以审计检查关键执行路径及安全关键点，记录安全事件，兼容多种安全策略防御多种攻击向量。
3. 针对Intel SGX SDK的安全增强框架验证。本文实验验证SGX-SEF对关键执行路径和安全关键点的审计效果，评估SGX-SEF审计功能引起的性能开销，分析引起性能开销的原因，实验验证SGX-SEF兼容多种安全策略防御多种攻击向量的能力。

本文主要贡献包括：

1. 现有SGX攻击向量能够破坏SGX安全性，但已有防御方案针对性地防御特定SGX攻击向量或攻击面，各个方案遵循的安全准则不统一，至今没有一个方案对SGX安全关键点进行全面分析，导致现有SGX架构难以有效兼容这些防御方案。针对上述问题，本文以SGX软件栈为主刻画了SGX架构及其关键执行路径，总结了其中的安全关键点。
2. 针对利用SGX SDK无法管控Enclave线程执行序及无法审计安全/非安全上下文切换事件等安全问题透过Enclave接口及共享资源等攻击面的多种SGX攻击向量，本文设计实现了针对Intel SGX SDK的安全增强框架（SGX-SEF），审计SGX关键执行路径，兼容已有安全准则防御多种攻击向量。
3. 针对恶意篡改Enclave代码调用顺序的调用排序和并发调用攻击向量，本文提出了“调用顺序白名单”安全策略，基于此策略审计检查关键执行路径，判断调用顺序是否符合安全策略。
4. 实验验证了SGX-SEF能有效审计SGX关键执行路径，评估了SGX-SEF审计功能引起的性能开销，分析了引起性能开销的原因。审计功能开销随日志容量增大而增大，开销包括日志使用维护开销及缓存未命中开销。当日志大小100项时，审计功能对各种调用方式引起的开销增幅不明显，100项大小的日志足够记录最近的安全事件用于审计检查功能。日志容量为100项时，开销增幅在最优情况（Ordinary调用方式）下为13%~18%。实验验证了防御者可以在SGX-SEF中灵活部署安全策略以抵御多类攻击向量。

## 1.4　论文结构

本论文共分为六章，各章内容组织如下：

第一章是引言。该章阐述了选题的出发点及意义，总结了相关研究现状，提出了本研究课题的主要研究内容及贡献。

第二章是国内外相关研究现状。该章分析了SGX的编程模型、软硬件架构及安全机制，总结了SGX的安全问题、攻击向量及攻击面，借鉴传统防御方案思路，分析SGX防御的不足。

第三章是Intel SGX关键执行路径及安全关键点分析。该章分析了SGX架构、关键执行路径，总结了SGX安全关键点位。

第四章是针对Intel SGX SDK的安全增强框架设计。该章设计了针对安全关键点的审计检查代码插桩方式及插桩位置，设计实现了安全事件审计检查功能，全面提升SGX架构安全性。

第五章是针对Intel SGX SDK的安全增强框架验证。该章实验验证了SGX-SEF审计关键执行路径及安全关键点的效果，通过实验评估了SGX-SEF审计功能的性能开销，分析了引起性能开销的原因，验证了SGX-SEF兼容多种安全策略防御多种攻击向量的能力。

第六章是总结与展望。该章总结了本文的主要内容及成果，指出了后续可研究的方向。

# 第二章　国内外相关研究现状

本研究课题围绕SGX安全性展开研究，本章针对SGX阐述其编程模型、软硬件架构和安全机制并总结SGX特点，分析现有SGX攻击并指出SGX安全问题、攻击面和攻击向量，分析传统防御方案及现有SGX防御方案，对比传统防御方案思路指出现有SGX防御方案不足。

## 2.1　Intel SGX

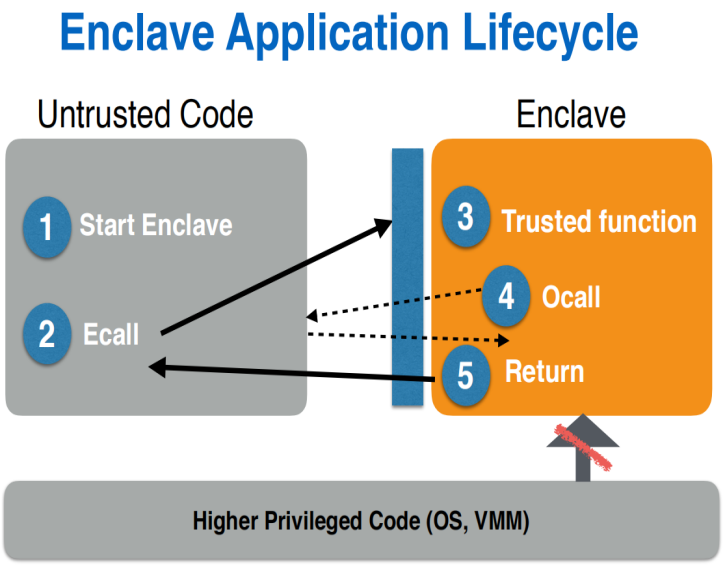
Intel SGX[3-5]全称为Intel Software Guard Extension，SGX提供了用户态的可信执行环境（TEE），通过密码学方法和访问控制保护并证明计算过程及计算内容可信。基于CPU实现的可信执行环境技术除Intel SGX外还包括ARM TrustZone、RISC-V Keystone、AMD SEV（Secure Encrypted Virtualization），基于GPU实现的可信执行环境技术包括论文[6,7]等，基于软件实现的可信执行环境技术包括Virtual Ghost[8]等。研究人员最早使用高权限或运行时检查技术基于软件构建可信执行环境，软件TEE的优势在于能够快速生成原型系统，但与部分硬件架构不兼容且性能较差。随后CPU厂商基于硬件实现可信执行环境，TEE性能得到提升，但仅能保护CPU中的计算过程及计算内容。人工智能运用GPU及硬件加速器加速模型训练过程，研究人员为了保护GPU中的计算过程及计算内容，提出了若干种GPU TEE技术，并结合CPU TEE技术保护CPU和GPU中的计算过程及计算内容。

可信执行环境可分为全栈式可信执行环境和用户态可信执行环境。全栈式可信执行环境将计算资源划分为安全世界和非安全世界，安全世界及非安全世界各自均包含用户态和内核态架构等，但两者彼此隔离。而SGX等技术提供的用户态可信执行环境只在用户态构建可信执行环境保护用户态应用程序的敏感内容。可信执行环境技术是可信计算技术的发展，可信计算技术（如TPM）中通过单独的硬件模块保护BIOS、操作系统等上层架构可信，而可信执行环境技术保护程序执行过程中的敏感代码数据，更加适合保护应用程序级敏感内容。

### 2.1.1　Intel SGX编程模型

Intel SGX建议仅将用户态应用程序中敏感内容放到用户态可信执行环境Enclave中，以减少Enclave内的可信计算基（Trusted Computing Base，缩写为TCB）大小。Enclave中过大的TCB会导致Enclave漏洞多，安全性差。

SGX编程模型中，SGX应用被分为不可信部分和Enclave。Enclave通过密码学方式和访问控制抵御Ring0权限攻击（包括来自操作系统和虚拟机监控程序等的攻击，如图2.1下方所示）。简化的SGX应用执行流程及Enclave生命周期：（1）不可信部分显式创建（及销毁）Enclave，（2）不可信部分调用ECALL（全称为Enclave Call，即可信函数的入口）进入Enclave，（3）执行可信函数，（4）可信函数调用OCALL（全称为Outside Call）暂退到不可信部分并执行Enclave内无法实现的功能，（5）可信函数执行完成返回到不可信部分。

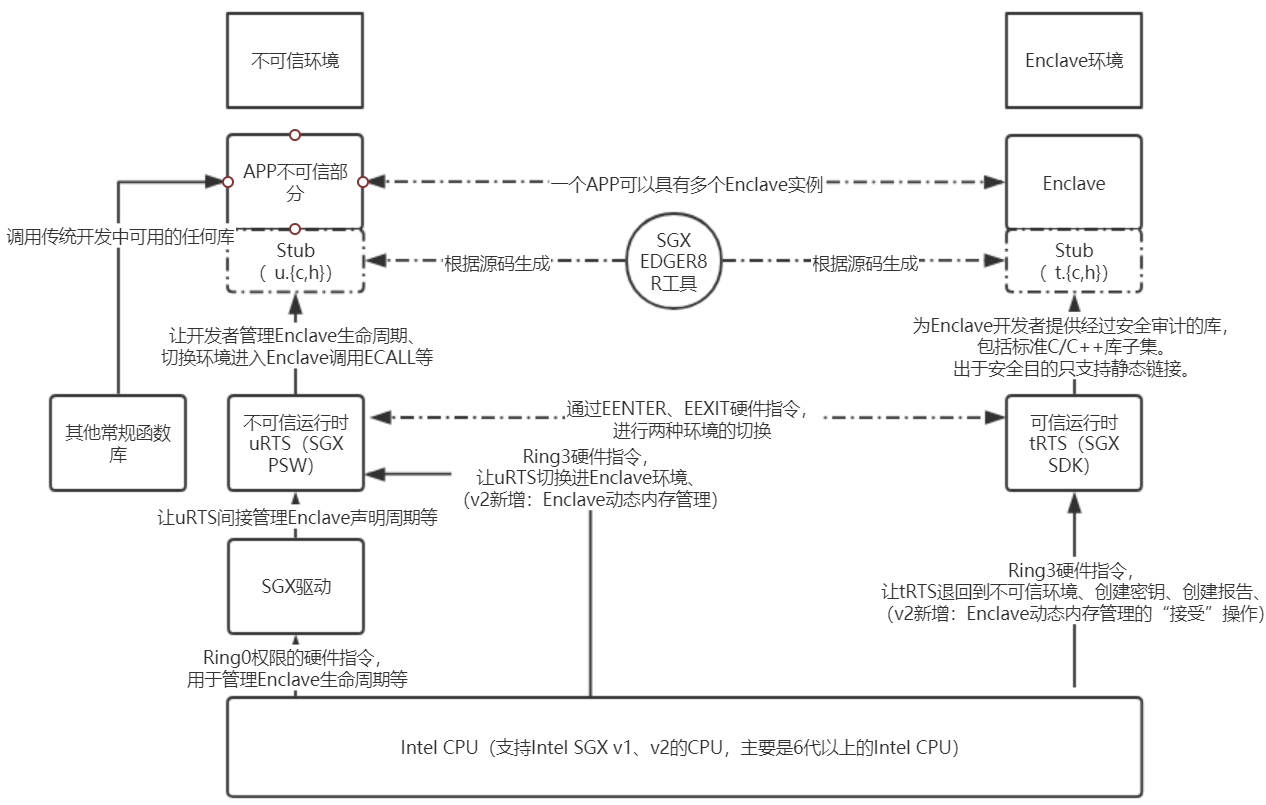


**图2.1**　Enclave应用生命周期[30]

**Figure2.1**　Enclave application lifecycle[30]

### 2.1.2　Intel SGX软硬件架构

Intel SGX狭义上是Intel CPU提供的一套扩展指令集，包括Ring0和Ring3权限指令。广义上还包括SGX软件栈，方便开发者使用硬件功能，SGX软件栈包括Intel SGX PSW（Platform Software）、Intel SGX SDK（Software Development Kit）和Intel SGX Driver。Intel SGX软硬件架构（包括SGX硬件及SGX软件栈）的架构如图2.2所示。



**图2.2**SGX软硬件架构

**Figure2.2**SGX software and hardware architecture

Intel SGX提供了Ring0指令ENCLS（S指Supervisor模式），对ENCLS指令指定不同的EAX寄存器值调用不同的叶功能（Leaf Function，SGX手册中也将叶功能直接称为指令）。ENCLS指令被内核态的（不可信）SGX驱动调用，以提供Enclave实例生命周期管理等功能。如ECREATE指令（ENCLS.00H，EAX值为00H）主要用于创建Enclave实例的SGX Enclave控制结构体（SGX Enclave Control Structure，缩写为SECS，每个SECS唯一标识一个Enclave实例）。

Intel SGX还提供了Ring3指令ENCLU（U指User模式），提供CPU模式（普通模式或Enclave模式）切换等功能。ENCLU的叶功能可划分为两部分：

1. 用户态的SGX应用不可信代码和uRTS（untrusted Run-Time System）调用的叶功能。如EENTER指令（ENCLU.02H）将CPU模式从普通模式切换成Enclave模式（CR\_ENCLAVE\_MODE寄存器置1），标识CPU能访问指定Enclave实例的资源（CR\_ACTIVE\_SECS置为目标Enclave实例的SECS）。
2. 用户态的Enclave代码和tRTS（trusted Run-Time System）调用的叶功能。如EEXIT指令（ENCLU.04H）将CPU模式从Enclave模式切换成普通模式（CR\_ENCLAVE\_MODE寄存器置0），CPU不能再访问任何Enclave实例的资源。

SGX SDK（也称为tRTS）向Enclave开发人员提供经过安全审计的静态链接库（如ENCLU部分叶功能的封装函数、经过安全审计的C/C++库子集等），方便开发人员编写Enclave代码。静态链接的目的是确保Enclave代码均存储于安全内存中。

通过SGX PSW（也称为uRTS），SGX应用不可信部分可以调用部分ENCLU指令，并利用IOCTL通告SGX驱动调用ENCLS指令。SGX应用不可信部分还可以调用其他常规函数库。

SGX应用开发方式比较特殊。开发过程中，开发人员需要将敏感代码数据统一编写成Enclave。SGX应用不可信部分与Enclave边界处的代码（简称边界代码，也称为Stub）由SGX EDGER8R工具依据开发者编写的EDL（Enclave Definition Language）文件自动生成。

SGX假设除Enclave外均是不可信的。SGX应用不可信部分可以被攻击者攻占，并以任意顺序调用Enclave接口。

### 2.1.3　SGX Enclave安全机制

SGX通过密码学方法和访问控制等安全机制保护Enclave的机密性、完整性、新鲜度（Freshness）和真实性（Authenticity）等，抵御Ring0攻击[1,9,10]。SGX 1代提供的安全机制包括EPC（全称为Enclave Page Cache）访问控制机制、EPC机密性保护机制、EPC完整性保护机制、EPC新鲜度保障机制、Enclave真实性保障机制和Enclave实例可信建立机制。EPC是处理器保留内存（Processor Reserved Memory，缩写为PRM，位于DRAM起始位置，无法被系统软件及外设DMA直接访问）中的64M/128M/256M大小的物理内存。Enclave实例的虚拟内存（即ELRANGE）以动态库形式位于SGX应用的进程地址空间中，ELRANGE的虚拟页映射EPC中的物理页。

EPC访问控制机制确保CPU在普通模式下无法直接访问EPC，被EENTER指令切换为Enclave模式后才能访问特定Enclave实例的资源，包括：

1. 内核段页机制中的访问控制。
2. EPC无法被系统软件及外设DMA直接访问。
3. Page Miss Handler硬件查询SGX独自维护的EPCM（EPC Map），在代码非法访问EPC时触发中止页面语义。

EPC机密性保护机制确保Enclave内容在CPU中明文存在，在物理内存（包括EPC部分和非EPC部分）和硬盘中密文存在，包括：

1. 内存加密引擎（Memory Encryption Engine，缩写为MEE）使用SGX在计算机启动时生成的密钥将CPU中的明文加密存储于EPC。
2. 将EPC内容改用Enclave实例特定密钥加密存储到普通内存。
3. 文件密封：Enclave内操作的文件加密存储于不可信外存中。

EPC完整性保护机制中，MEE构建哈希树保护EPC完整性。哈希树维护开销大，只能保护较小尺寸（64M/128M/256M）的EPC。

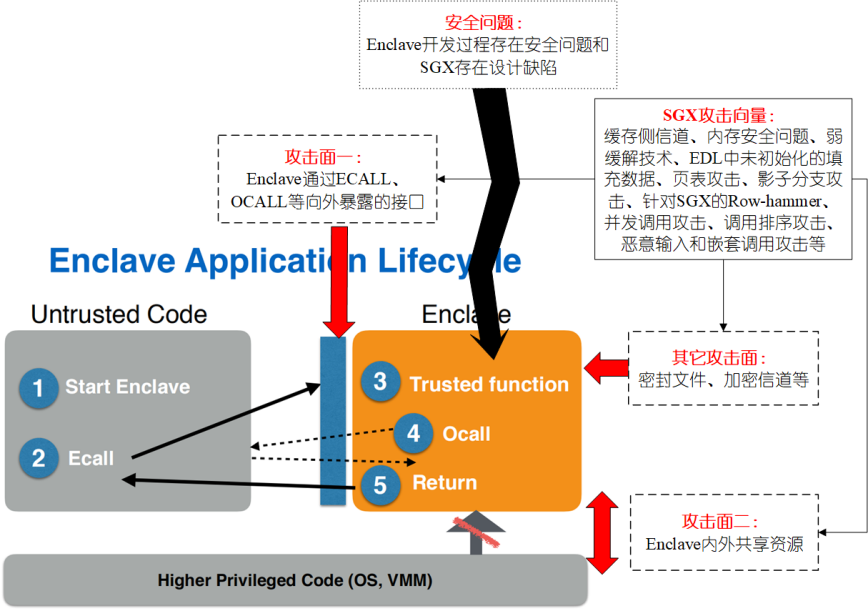
EPC新鲜度保障机制。为了抵御针对EPC的重放攻击（Replay Attack），MEE在哈希树中加入Nonce，在加密数据时使用包含地址时间信息的Version Nonce。

Enclave真实性保障机制包括本地认证和远程认证等。

Enclave实例可信建立机制。Enclave实例建立过程中，EINIT指令验证Enclave文件签名及Enclave实例建立过程度量值的有效性。

## 2.2　SGX攻击现状

现有攻击表明，由于Enclave开发过程存在安全问题，SGX设计存在安全缺陷，2.1.3节中的安全保护机制不足以全面保护SGX。攻击者针对安全问题利用各类攻击向量（指攻击者为了触发攻击所具体采用的方法）如缓存侧信道、内存安全问题、弱缓解技术、EDL中未初始化的填充数据、页表攻击、影子分支攻击、针对SGX的Row-hammer、并发调用攻击、调用排序攻击、恶意输入和嵌套调用攻击[11,15]，透过Enclave接口、Enclave内外共享资源等攻击面（指系统中可以被攻击者输入或提取数据造成攻击的点位）破坏Enclave安全性。（如图2.3所示）



**图2.3**　SGX Enclave安全问题、攻击面和攻击向量

**Figure2.3**　SGX Enclave security issues, attack surfaces and attack vectors

### 2.2.1　源于Enclave开发中的安全问题

SGX假设除Enclave外均是不可信的，SGX应用不可信部分可以被攻击者攻占，并以任意顺序调用Enclave可信函数。但开发者在开发过程中会忽视这个情况，产生的安全问题被攻击者频繁利用，SGX未对此提供安全服务功能。

Coin Attack[15]归纳了并发调用攻击、调用排序攻击和嵌套调用攻击三种攻击向量。这些攻击向量主要源于开发者忽视SGX开发特点，未限制不可信代码对Enclave的调用逻辑。它们主要透过Enclave接口攻击面展开攻击，具体描述如下：

1. 调用排序攻击（Call Permutation）。SGX编程模型中，不可信代码可以任意顺序调用Enclave函数，但是Enclave开发人员未做出限制，使得攻击者能够恶意顺序调用Enclave函数绕过特定接口检查。
2. 并发调用攻击（Concurrent Calls）。不可信代码可以并发调用多个Enclave函数，触发Enclave并发错误，如Enclave内的Race Condition。
3. 嵌套调用攻击（Nested Calls）。嵌套ECALL和OCALL构建非预期的控制流，进而形成攻击。

SGX要求开发者基于先验知识针对Enclave接口进行安全检查，SGX只向开发者提供基础性的接口安全检查功能。开发者编写Enclave代码时，对Enclave接口检查不充分，产生安全问题被多类攻击向量利用。针对此安全问题的攻击向量包括：

1. 恶意篡改输入（Input Manipulation）。攻击者在Enclave接口处输入非预期值，影响Enclave内状态，形成攻击。SGX tRTS（trusted Run-Time System）无法先验地知道输入数据的处理方法，只能提供安全能力有限的接口消毒功能（如EDL关键字）。
2. EDL中未重新初始化的填充数据（Uninitialized Padding In EDL）。传入传出Enclave的数据结构内存在用于保证数据结构正确对齐的填充数据，填充数据可能携带残留的敏感信息，使得Enclave敏感信息泄露，Enclave外非法输入恶意数据影响Enclave状态。

### 2.2.2　源于SGX设计存在安全缺陷

研究表明SGX提供的安全机制存在缺陷。CPU资源隔离机制中对资源隔离不充分，形成Enclave内外共享资源攻击面，被大量侧信道攻击利用[1,10]。Enclave内外共享资源包括：

1. 微架构共享资源。Enclave内外存在微架构层面的共享资源，如缓存、分支预测器和TLB。
2. 部分功能依赖不可信内核。用户态Enclave部分功能依赖于不可信内核，如页错误处理和线程调度。
3. 可观测的物理信息。Enclave内的物理信息如执行时间可被Enclave外感知。

针对此安全问题的攻击向量包括：

1. 接口侧信道。攻击者观察Enclave在接口处所泄露的侧信道信息构建攻击。
2. 缓存侧信道。攻击者观察Enclave在Enclave内外共享缓存（L1、L2和LLC）中泄露的信息，推测Enclave内容。
3. 分支预测器侧信道。攻击者观察Enclave在Enclave内外共享分支预测器（Branch Predictor）中泄露的信息，推测Enclave内容。
4. 影子分支攻击（Branch Shadowing Attack）。该攻击向量类似于分支预测器侧信道，但它使用影子分支提取被泄露的信息，传统分支预测器侧信道使用缓存侧信道提取被泄露的信息。
5. 页表攻击。攻击者观察Enclave在内核页表中泄露的信息（如页错误时会泄露页地址），推测Enclave内容。
6. 恶意线程调度。攻击者利用Ring0权限恶意调度（暂停及恢复）Enclave线程执行，触发非预期并发状态并形成攻击。
7. 时间侧信道。攻击者观察Enclave执行过程中不同粒度的时延信息，推测Enclave内容。

密封文件等安全机制对安全场景考虑不充分，存在安全缺陷，向攻击者暴露攻击面。相关攻击向量如文件回滚攻击：当Enclave操作密封文件时，不可信环境向Enclave内提供旧文件，将Enclave状态回滚到旧状态，形成攻击。

### 2.2.3　代表性攻击

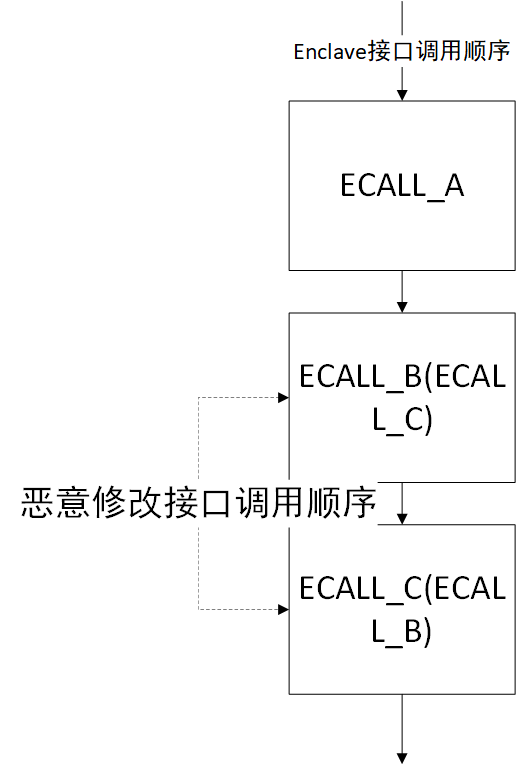
本小节阐述代表性攻击，指出每个代表性攻击所针对的安全问题、所利用的攻击向量及所涉及的攻击面。

Iago attacks[12]中，攻击者恶意构造系统调用OCALL（Enclave无法提供系统调用）返回值影响Enclave状态，形成攻击。针对“Enclave开发中的安全问题”，用到“恶意篡改输入”攻击向量，涉及“Enclave接口”攻击面。

Lee S等人的工作[13]中，攻击者观察Enclave内传出的数据结构中未初始化的填充数据，获得残留的敏感数据。针对“Enclave开发中的安全问题”，用到“EDL中未初始化填充数据”攻击向量，涉及“Enclave接口”攻击面。

A Tale of Two Worlds[14]总结了针对SGX软件栈的ABI和API攻击，实现了新型的ABI攻击（通过修改标志寄存器）及API攻击（对SGX帮助性函数进行时间侧信道）。针对“Enclave开发中的安全问题”和“SGX设计存在安全缺陷”，用到“恶意篡改输入”和“时间侧信道”等攻击向量，涉及“Enclave接口”和“可观测的物理信息”等攻击面。

COIN Attacks[15]归纳了并发调用攻击、调用排序攻击、恶意篡改输入和嵌套调用攻击这四种透过Enclave接口攻击面的攻击向量，基于此开发了SGX应用漏洞自动挖掘工具。这些攻击向量针对“Enclave开发中的安全问题”，涉及“Enclave接口”攻击面。“调用排序”攻击向量如图2.4所示，攻击者攻占SGX应用不可信部分，恶意顺序调用ECALL以绕过特定接口检查，触发非预期的控制流。如攻击者利用ECALL C中的状态设置操作绕过ECALL B的状态检查。



**图2.4**　调用排序攻击示意图

**Figure2.4**　Diagram of Call Permutation Attack

Wang J等人的工作中[16]中，攻击者通过侧信道手段观察受害者Enclave的接口调用顺序、参数及时延，推测受害者Enclave的控制流。针对“SGX设计存在安全缺陷”，用到“接口侧信道”攻击向量，涉及“Enclave接口”和“可观测的物理信息”等攻击面。

Foreshadow[17]中，攻击者手动构造页错误绕过SGX中止页面语义，瞬态执行未被授权执行的代码，将Enclave敏感信息泄露到L1缓存，窃取泄露信息。针对“SGX设计存在安全缺陷”，用到“缓存侧信道”攻击向量，涉及“微架构共享资源”攻击面。

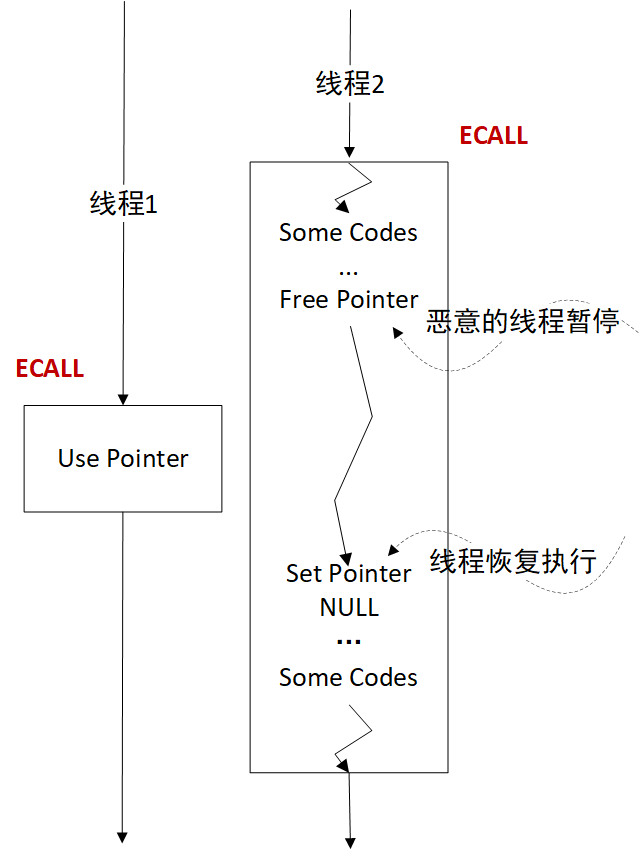
SgxPectre[18]是Spectre攻击在SGX下的复现，攻击者恶意训练分支预测器，受害者Enclave在执行过程中分支预测错误，瞬态执行非预期分支，将敏感信息泄露到缓存，攻击者窃取泄露信息。针对“SGX设计存在安全缺陷”，用到“分支预测器侧信道”和“缓存侧信道”等攻击向量，涉及“微架构共享资源”攻击面。

Branch Shadowing[19]中，由于Enclave内外共享分支预测器及Enclave在退出时不刷新分支预测器，攻击者构建地址碰撞的影子分支（Shadow Branch），感知受害Enclave分支选择情况。针对“SGX设计存在安全缺陷”，用到“影子分支攻击”攻击向量，涉及“微架构共享资源”攻击面。

Controlled-channel attacks[20]中，攻击者清除受害Enclave的“页存在（Present）位”，使受害Enclave执行过程中触发页错误，泄露错误页地址，攻击者观察错误页地址推测Enclave内页粒度执行流。针对“SGX设计存在安全缺陷”，用到“页表攻击”攻击向量，涉及“部分功能依赖不可信内核”攻击面。

SGX-PTE[21]中，攻击者利用A/D通道【利用页访问/脏（Access/Dirty）位构建侧信道】及针对PTE的缓存侧信道，观察页地址访问情况，推测Enclave内页粒度控制流。针对“SGX设计存在安全缺陷”，用到“页表攻击”和“缓存侧信道”攻击向量，涉及“部分功能依赖不可信内核”和“微架构共享资源”攻击面。

AsyncShock[22]中，攻击者利用Ring0权限恶意调度受害Enclave多个线程，触发非预期并发控制流，构建UAF（Use after Free）和TOCTOU（Time to Check to Time to Use）攻击。AsyncShock如图2.5所示，线程2释放一个指针后，需要将其置NULL避免悬空指针（Dangling Pointer）存在，但攻击者可通过“Enclave线程暂停原语”（采用页错误、APIC时钟中断和Linux Signal等方法）将线程2暂停在释放指针后及置NULL前，调度线程1执行ECALL使用悬空指针，构建UAF攻击。主要针对“SGX设计存在安全缺陷”，用到了“恶意线程调度”攻击向量，涉及“部分功能依赖不可信内核”攻击面。



**图2.5**　AsyncShock攻击示意图

**Figure2.5**　Diagram of AsyncShock Attack

Game of Threads[23]中，攻击者利用“页错误型线程暂停原语”恶意调度受害Enclave多个线程，对ASGD（异步随机梯度下降）场景进行攻击。针对“SGX设计存在安全缺陷”，用到“恶意线程调度”攻击向量，涉及“部分功能依赖不可信内核”攻击面。

Nemesis[24]中，攻击者利用APIC时钟中断按一定的时间间隔单步中断Enclave汇编代码执行，测量每条汇编代码执行时长构建时延序列，将时延序列与先验的Enclave各分支时延序列对比，推测Enclave分支选择情况。针对“SGX设计存在安全缺陷”，用到“时间侧信道”攻击向量，涉及“可观测的物理信息”攻击面。

RIDL[25]中，攻击者利用受害者在微架构LFB（Line Fill Buffer）泄露的信息实现侧信道攻击。针对“SGX设计存在安全缺陷”，用到“LFB侧信道”攻击向量，涉及“微架构共享资源”攻击面。

本文将Enclave“恶意线程调度”攻击向量中恶意暂停线程执行的方法命名为“Enclave线程暂停原语”，包括：

1. 清除目标Enclave页存在位/访问位等触发页错误，暂停Enclave线程执行。
2. 设置APIC时钟中断，暂停Enclave线程执行。
3. 利用Linux Signal（如SIGSEGV、SIGUSR1和SIGUSR2）暂停Enclave线程执行。

本文将代表性SGX攻击归纳成表2.1至2.5。

**表2.1**“Enclave接口”攻击表

**Table2.1**　Table of attacks on "Enclave interface"

|  |  |
| --- | --- |
| 攻击向量（安全问题） | “Enclave接口”攻击 |
| 并发调用（Enclave开发中的安全问题） | COIN Attacks |
| 调用排序（Enclave开发中的安全问题） | COIN Attacks |
| 恶意篡改输入（Enclave开发中的安全问题） | Iago attacks、A Tale of Two Worlds、COIN Attacks |
| 嵌套调用（Enclave开发中的安全问题） | COIN Attacks |
| EDL中未初始化填充数据（Enclave开发中的安全问题） | Leaking Uninitialized Secure Enclave Memory via Structure Padding |
| 接口侧信道（SGX设计存在安全缺陷） | Interface-based side channel attack against intel SGX |

**表2.2**　“微架构共享资源”攻击表

**Table2.2**　Table of attacks on "shared resources of micro-architecture"

|  |  |
| --- | --- |
| 攻击向量（安全问题） | “微架构共享资源”攻击 |
| 缓存侧信道（SGX设计存在安全缺陷） | Foreshadow、SgxPectre、SGX-PTE |
| 分支预测器侧信道和影子分支攻击（SGX设计存在安全缺陷） | SgxPectre、Branch Shadowing |
| LFB侧信道（SGX设计存在安全缺陷） | RIDL |

**表2.3**　“部分功能依赖内核”攻击表

**Table2.3**　Table of attacks on "partial functions depend on kernel"

|  |  |
| --- | --- |
| 攻击向量（安全问题） | “部分功能依赖内核”攻击 |
| 页表攻击（SGX设计存在安全缺陷） | Controlled-channel attacks、SGX-PTE |
| 恶意线程调度（SGX设计存在安全缺陷） | AsyncShock、Game of Threads |

**表2.4**“可观测的物理信息”攻击表

**Table2.4**　Table of attacks on "observable physical information"

|  |  |
| --- | --- |
| 攻击向量（安全问题） | “可观测的物理信息”攻击 |
| 接口侧信道（SGX设计存在安全缺陷） | Interface-based side channel attack against intel SGX |
| 时间侧信道（SGX设计存在安全缺陷） | A Tale of Two Worlds、Nemesis |

**表2.5**　“密封文件”攻击表

**Table2.5**　Table of attacks on "sealed file"

|  |  |
| --- | --- |
| 攻击向量（安全问题） | “密封文件”攻击 |
| 文件回滚攻击（SGX设计存在安全缺陷） | 文件回滚攻击[1] |

## 2.3　传统系统防御方案

针对SGX安全问题、攻击向量及攻击面，可以借鉴传统系统防御方案的思路。传统系统防御方案中，有针对特定攻击类型以补丁形式防御的方案。如Dangling Pointer Tagging和Red-zone Inserting缓解UAF和栈溢出等内存安全问题，CFI和DFI增强目标应用程序控制流/数据流的完整性缓解控制流/数据流劫持，ASLR和KASLR将程序和内核加载地址随机化增加攻击者定位漏洞难度，SMEP、SMAP和KPTI单向或双向隔绝用户态和内核态代码数据抵御针对内核的攻击。但这些防御方案安全准则不统一，难以被系统同时支持。

还有防御方案深入分析系统，加固系统自身安全。如访问控制领域中，Matetic S等人对Linux系统的访问控制模型及关键访问控制点位深入分析，提出Linux Security Module[39]（缩写为LSM）访问控制框架，将安全准则与实施框架分离，高效兼容多种安全准则。SELinux[40]（Security-Enhanced Linux）基于LSM进行了具体实现。LSM使Linux同时支持了Domain and Type Enforcement和Linux capabilities等多种访问控制模型，启发了AppArmor、TOMOYO和YAMA等安全增强模块产生。实现上，LSM在Linux关键执行路径中安插了许多Hook点。基于LSM实现的访问控制模型选用Hook点并自定义安全策略来实施主客体访问控制。Hook点上获取的主客体访问控制信息会交给策略引擎（Policy Engine）决断，确定当前访问是否符合安全策略。

框架性的防御方案深入分析系统，加固系统自身安全，相较于针对性的防御方案更加原生，兼容多种安全准则的能力更强，提供更高安全性，启发新的安全准则。

## 2.4　SGX防御现状

针对SGX攻击，学术界及工业界均提出了防御方案，用到的防御方法包括Enclave接口消毒、Enclave内外资源隔离、痕迹防泄露和攻击痕迹检测等。尚无防御方案系统分析SGX架构及安全关键点，加固SGX架构自身安全。

### 2.4.1　Enclave接口消毒

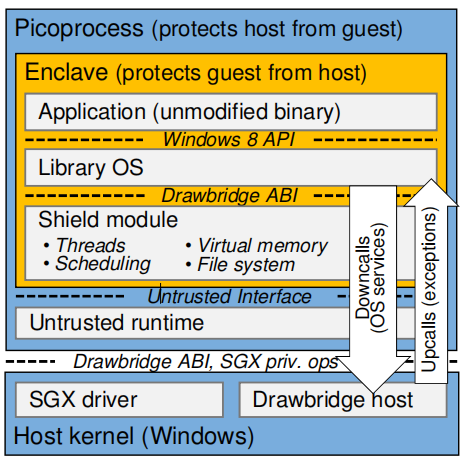
Enclave接口消毒指对Enclave接口进行安全检查，避免攻击者从Enclave接口输入恶意数据危害Enclave。代表性防御方案包括SGX原生软件栈提供的和学术界提出的防御方案。SGX原生软件栈提供了EDL关键字及SGX帮助性函数，由于无法先验地知道Enclave接口具体用法，因此只能够简单消毒Enclave接口。

EDL关键字。描述Enclave边界的EDL文件中，Enclave开发者可以设置EDL关键字，对跨越Enclave内外的函数和传参进行简单限定。如ECALL中“In”关键字可以修饰一个原指向Enclave外缓冲区的指针，SGX软件栈会将指针指向的缓冲区内容拷贝进Enclave，Enclave内使用指向Enclave内缓冲区的指针。“In”关键字防止Enclave执行时Enclave外通过修改缓冲区直接影响Enclave内部状态。但EDL关键字缺乏针对函数和传参的先验场景知识，接口消毒能力有限，复杂接口消毒功能需要Enclave开发者实现。

SGX帮助性函数（主要指sgx\_is\_within\_enclave和sgx\_is\_outside\_enclave函数）是SGX软件栈提供的一类函数，帮助Enclave判断变量位于Enclave内部（sgx\_is\_within\_enclave）还是Enclave外部（sgx\_is\_outside\_enclave），方便开发者进行接口检查。

学术界代表防御方案包括Haven[26]、Graphene-SGX[27]、SCONE[28]、Ryoan[29]和Glamdaring[30]等。

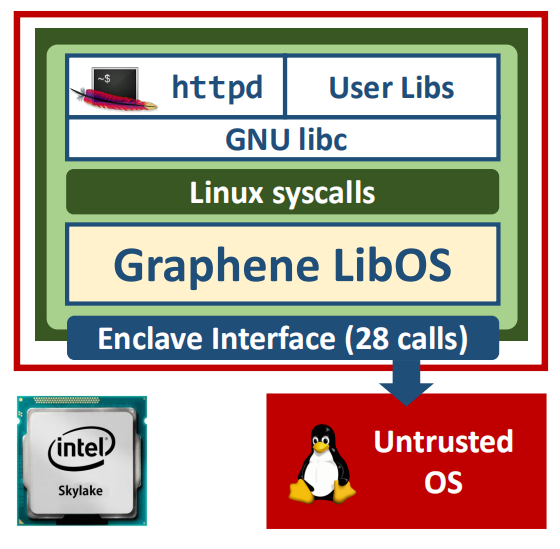
Haven[26]（如图2.6所示）基于Drawbridge在Enclave中内嵌LibOS，使Enclave内能二进制级别兼容非SGX应用，Shield模块通过20多个接口向Host内核请求Enclave内无法实现的功能。针对Iago Attack等攻击透过Enclave接口攻击面展开攻击，Haven在Shield模块中基于先验LibOS场景检查接口传参安全。



**图2.6**　Haven组件和接口[26]

**Figure2.6**　Components and interfaces of Haven[26]

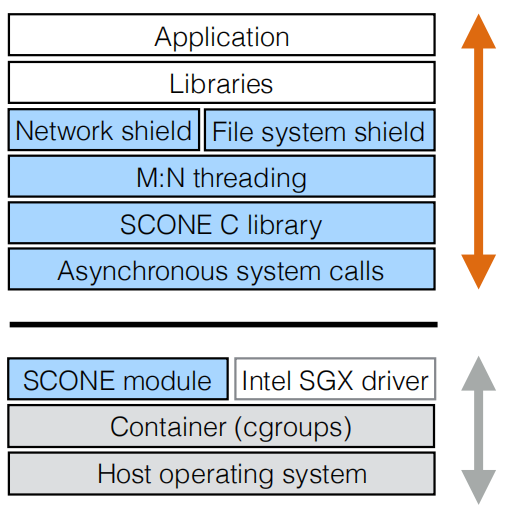
Graphene-SGX[27]（如图2.7所示）类似于Haven，在Linux系统下的SGX Enclave中内嵌LibOS，二进制级别兼容非SGX应用和Docker，基于先验LibOS场景检查28个接口安全。文件读取（通过文件校验码）和进程间协作（通过认证和加密）等18个接口被全面检查。



**图2.7**　Graphene-SGX架构图[27]

**Figure2.7**　Graphene-SGX architecture diagram[27]

SCONE[28]（如图2.8所示）源码级别兼容非SGX应用（需要重新编译）。将Graphene-SGX和Haven中LibOS及C库移出Enclave，通过Shim层（SCONE C Library）间接调用Enclave外C库，使Enclave保持较小TCB。Shield层向下调用HostOS功能，向上提供文件系统等功能，基于先验文件系统等功能设置接口消毒。



**图2.8**　SCONE架构图[28]

**Figure2.8**　SCONE architecture diagram[28]

Ryoan[29]构建Enclave内沙箱并检查Enclave接口，防止服务提供商的Enclave恶意向Enclave外传递用户隐私数据。

Glamdaring[30]自动划分非SGX应用的可信部分和非可信部分，组装成SGX应用，它通过减少Enclave接口数量缓解调用排序攻击向量。

### 2.4.2　Enclave内外资源隔离、痕迹防泄露及攻击痕迹检测

为避免侧信道等攻击利用“Enclave内外共享资源”攻击面，有防御方案基于资源隔离、执行痕迹防泄露和攻击痕迹检测等思路展开防御。代表性防御方案包括学术界提出的及Intel提出的防御方案。学术界的防御方案包括T-SGX[31]和Racing in Hyperspace[32]等。

T-SGX[31]利用Intel TSX事务代码出错回滚特性，将页错误泄露的页地址信息限制到跳板页地址。针对会触发大量AEX的Controlled-channel Attack，它在跳板页记录AEX频率并预警高频AEX（即检测攻击痕迹）。

Racing in Hyperspace[32]针对超线程侧信道攻击，度量受害者两个线程间通讯时长，判断是否在兄弟核上（同一个物理核的两个逻辑核），若通讯时长超出阈值，那么受害者两个线程不在兄弟核上，即可能有攻击者抢占受害者的兄弟核并发起超线程攻击。

OBFUSCURO[33]将代码数据访问模式、分支模式及代码块执行时长固定化，只留下相同的访问痕迹，避免攻击者观察访问模式差别推测Enclave控制流。

Eleos[34]实验证明进出Enclave时切换上下文开销巨大，通过类RPC机制及SUVM（Secure User Virtual Memory，即Enclave内页表机制）避免频繁进出Enclave。SUVM机制还能避免页错误地址信息泄露到Enclave外，缓解页表攻击。

CoSMIX[35]使用LLVM Pass插桩Enclave源码，实现了多层Memory Store（基于底层物理内存、内核虚拟内存或自定义虚拟内存构建新一层自定义虚拟内存）栈式堆叠，满足多种（安全）需求，如ORAM和Enclave内页表机制的双重堆叠。

Autarky[36]针对Controlled-Channel Attack，修改软硬件使Enclave内页错误处理优先于OS页错误处理，Enclave内页错误处理中，Enclave结合内部记录判断当前异常是攻击者伪造的还是正常触发的。Autarky中还可以编写策略检测攻击。

Gruss D等人的工作[37]针对TAA（TSX Asynchronous Abort利用TSX回滚事务时不回滚微架构的缺陷）攻击向量，在封装敏感代码的事务入口处加载冗余缓存，增加攻击噪声。

Intel公司通过微码补丁和硅上改（Silicon-based Change）对危害严重的攻击进行防御。

Intel修改SGX硬件指令微码流（SGX指令是由若干微码组成的微码流，执行时被翻译成微码并被发送到执行单元执行）实施防御措施。如针对缓存侧信道，在进出Enclave所用指令微码流中增加L1D缓存刷新操作。

Intel通过硅上改重新设计Intel CPU微架构，缓解某些侧信道。如修改乱序执行中异常提交和状态回滚相关机制。

### 2.4.3　其他防御方案

有研究人员提出防御方案，缓解透过“密封文件”攻击面的文件回滚攻击。

ROTE[38]基于拜占庭容错系统思路，将标识文件新旧的版本号存储于分布式主机群中，当Enclave读取密封文件时，比对密封文件的版本号和主机群中读出的版本号，避免使用旧版本文件引起文件回滚攻击。

SGX Counter。Intel公司在支持SGX的CPU中添加了SGX Counter（单调递增计数器），记录每个密封文件的最新状态。当不可信环境提供旧文件时，Enclave能检测到旧文件标识号与SGX Counter值不一致。

## 2.5　本章小结

本章阐述了SGX编程模型、软硬件架构和原生安全机制，指出SGX的特殊之处。归纳了SGX攻击所针对的安全问题包括“Enclave开发中的安全问题”和“SGX设计存在安全缺陷”，所透过的攻击面包括“Enclave接口”和“共享资源”等，所利用的攻击向量包括调用排序、并发调用等。阐述了代表性攻击及其相关的安全问题、攻击面和攻击向量。分析了传统系统防御方案的思路，指出框架性防御方案的优势。归纳现有SGX防御方法包括接口消毒、资源隔离、痕迹防泄露和攻击痕迹检测等。指出尚无防御方案系统分析SGX架构及安全关键点，加固SGX架构自身安全。

# 第三章　Intel SGX关键执行路径及安全关键点分析

本章刻画SGX架构，分析SGX关键执行路径及安全关键点，总结SGX线程模型，为SGX架构安全增强提供着力点。主要围绕“Enclave开发中的安全问题”分析SGX软件栈关键执行路径及安全关键点，分析方法也适用于其他类型安全问题。本文将安全关键点中记录的安全信息称为安全事件或事件。

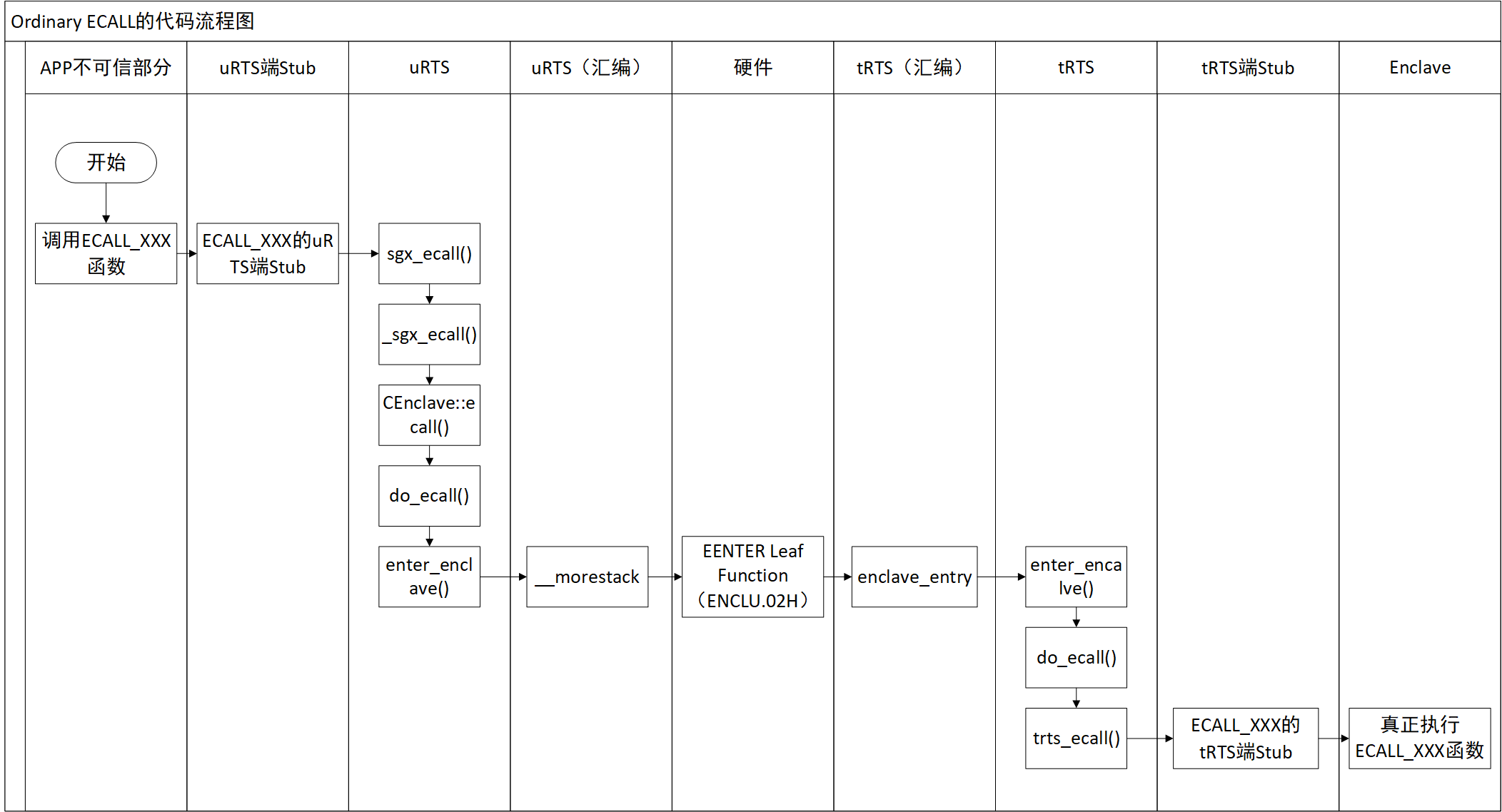
## 3.1　Enclave接口关键执行路径

分析Enclave接口关键执行路径，能够有效定位SGX架构中不可信环境与可信环境交界面上的安全关键点，为SGX架构安全增强提供着力点。

### 3.1.1　ECALL进入Enclave

不可信环境线程通过ECALL进入Enclave访问Enclave内容，ECALL实现方式包括Ordinary方式和Switchless方式。

Ordinary ECALL代码流程如图3.1所示。位于SGX应用不可信部分的普通线程调用ECALL Stub函数进入uRTS，调用EENTER指令（ENCLU.02H）切换上下文进入tRTS，变成Enclave线程，最终真正执行开发者编写的ECALL代码。



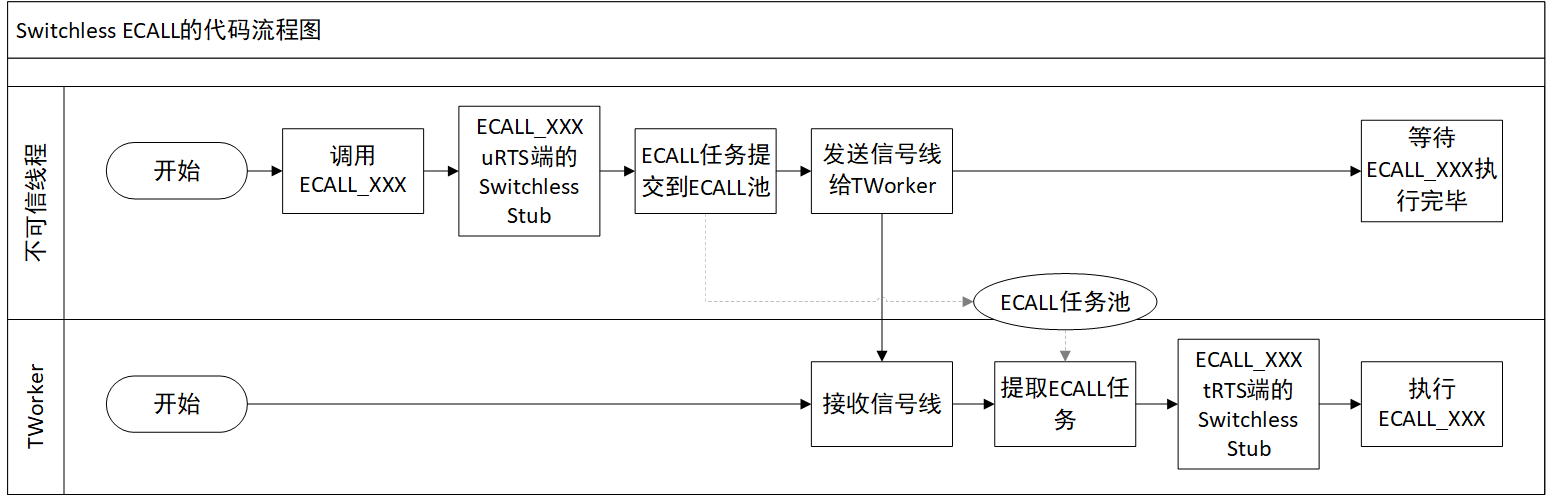
**图3.1**　Ordinary ECALL代码流程图

**Figure3.1**　Flowchart of Ordinary ECALL

Ordinary ECALL的关键在于线程调用EENTER指令切换上下文进入Enclave。在切换上下文时，线程（物理上对应一个CPU核）通过元数据寄存器标记它从普通模式转变为Enclave模式，将不可信上下文的部分内容备份，并将线程上下文完全替换成Enclave内的上下文。

Ordinary方式是最早提出的ECALL方式。为避免Ordinary方式切换上下文带来巨大开销，SGX软件栈新增Switchless方式[41,42]。

Switchless ECALL代码流程如图3.2所示。APP不可信部分的普通线程执行Switchless ECALL时，将ECALL封装成任务放到ECALL任务池中，通过信号线（Signal Line）通知tRTS端的TWorker（可信工人线程）提取并代理执行ECALL任务，等待TWorker执行完ECALL任务。如果Switchless ECALL调用者不等待TWorker执行完ECALL任务并继续执行剩余代码，那么程序执行流将不符合开发者预期。



**图3.2**　Switchless ECALL的代码流程图

**Figure3.2**　Flowchart of Switchless ECALL

Switchless ECALL的关键在于有一些预先切换上下文进入Enclave内的TWorkers帮助Switchless ECALL调用者代理完成ECALL任务。TWorkers自身的一次上下文切换，可以避免ECALL调用者N次的上下文切换。根据论文[42]的描述，当ECALL/OCALL函数代码量较少且被频繁调用时，占用工人线程带来的性能开销低于上下文切换的性能开销，Switchless方式在性能上会优于Ordinary方式。如公式1所示，当可信代码执行时长与不可信代码执行时长之和等于上下文切换时长时，两种方式的性能相同。

 (1)

“Switchless/Ordinary ECALL”中攻击者能通过恶意传参等方法攻击Enclave。“Switchless/Ordinary ECALL”是一个安全关键点，对此在其tRTS端（确保环境位置可信）插桩代码审计检查“ECALL进入Enclave”相关事件，审计检查的安全事件信息包括寄存器、ECALL名、ECALL序号、ECALL方向、Ordinary/Switchless和传参等。

### 3.1.2　ECALL结束退出Enclave

ECALL执行完后，控制流会回到ECALL调用者的下一行代码。Ordinary ECALL和Switchless ECALL的结束过程描述如下。

Ordinary ECALL结束过程。ECALL执行完后，线程从tRTS端Stub逐级返回并清理上下文信息，调用EEXIT指令切换上下文返回uRTS端，将控制流归还给ECALL调用者。

Switchless ECALL结束过程。TWorker执行完ECALL后，将函数执行结果放回到ECALL任务池中，通知Switchless ECALL调用者ECALL任务已执行完。Switchless ECALL调用者停止等待并执行后续代码。TWorker空转等待Switchless ECALL调用者提交新任务，若空转次数超过（默认20000次）阈值（即一定时间内没有新的ECALL任务）就返回tRTS，睡眠等待新的ECALL任务。当有新的ECALL任务时，TWorker重新进入Enclave并代理执行ECALL任务。

“Switchless/Ordinary ECALL返回”中，攻击者能恶意泄露Enclave内敏感内容及构建非法调用顺序等。“Switchless/Ordinary ECALL返回”是一个安全关键点，对此在其tRTS端插桩代码审计检查“ECALL结束退出Enclave”相关事件，可审计检查的安全事件信息包括寄存器、ECALL名、ECALL序号、ECALL方向、Ordinary/Switchless和返回值等。

### 3.1.3　OCALL离开Enclave及OCALL结束返回Enclave

Enclave内部无法实现部分功能（系统调用和CPUID指令等），但能通过OCALL调用Enclave外提供的功能。

“OCALL离开Enclave”在形式上与“ECALL进入Enclave”类似，但方向和目的相反，OCALL方式包括Ordinary方式和Switchless方式。Ordinary OCALL中，Enclave内线程调用EEXIT指令切换上下文前往Enclave外执行不可信函数。Switchless OCALL中，Enclave内线程将OCALL任务交给Enclave外的不可信工人线程UWorker代理执行。

由于ECALL内可以调用OCALL及OCALL内可以调用ECALL，Enclave线程栈中会出现ECALL栈帧和OCALL栈帧堆叠的情况（即嵌套调用）。

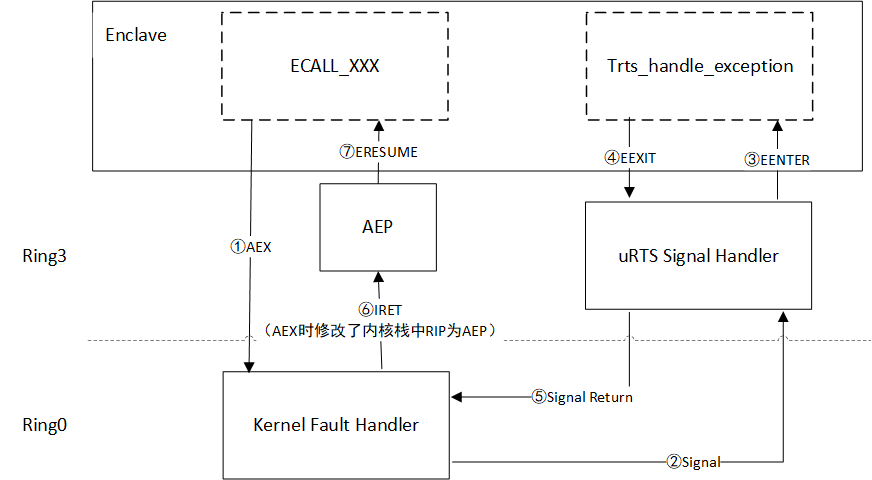
“OCALL结束返回Enclave”在形式上与“ECALL结束退出Enclave”类似，但方向和目的相反。由于OCALL返回值能被不可信环境篡改，Enclave需要基于先验知识检查不可信的OCALL返回值。

在“OCALL离开Enclave”及“OCALL结束返回Enclave”中，攻击者能恶意输入数据、恶意泄露Enclave内容及构建非法调用顺序等。“OCALL离开Enclave”及“OCALL结束返回Enclave”是安全关键点，对此在其tRTS端插桩代码审计检查“OCALL离开Enclave”及“OCALL结束返回Enclave”相关事件，可审计检查的安全事件信息包括寄存器、OCALL名、OCALL序号、OCALL方向、Ordinary/Switchless、传参和返回值等。

### 3.1.4　AEX离开Enclave及异常处理完成返回Enclave

Enclave异常处理流程与传统异常处理流程不同。非SGX应用中出现异常时，CPU立即让OS完成异常处理。而Enclave中出现异常时，CPU首先执行AEX（Asynchronous Enclave Exit，异步Enclave退出），AEX将Enclave上下文保存到SSA（State Save Area，状态保存区域，异常处理完成返回Enclave时从SSA中恢复上下文），切换上下文退出Enclave，进入内核态执行内核态异常处理句柄。AEX中包含Enclave内到Enclave外的切换及Ring3到Ring0的切换。AEX保护Enclave信息在中断时不被泄露。AEX及异常处理流程如下图3.3所示。

1. AEX及进入内核异常处理句柄。
2. 若内核无法处理异常，会发出Signal告知用户态。位于用户态的uRTS针对SIGSEGV、SIGFPE、SIGILL、SIGBUS和SIGTRAP五种信号注册了信号处理句柄。
3. 信号处理句柄中，线程以ECMD\_EXCEPT标记调用EENTER指令进入tRTS，线程栈包含两个ECALL栈帧，普通ECALL栈帧和ECMD\_EXCEPT ECALL栈帧。
4. 线程在tRTS端处理完异常后调用EEXIT指令退回到信号处理句柄。
5. 信号处理句柄完成相关操作后返回到内核错误句柄。
6. 内核错误句柄通过IRET（Interrupt Return）返回到AEP。返回到AEP的原因是AEX时将AEP赋值给内核栈中的RIP。
7. AEP调用ERESUME指令回到Enclave，从中断处继续执行代码。



**图3.3**　AEX及异常处理流程图

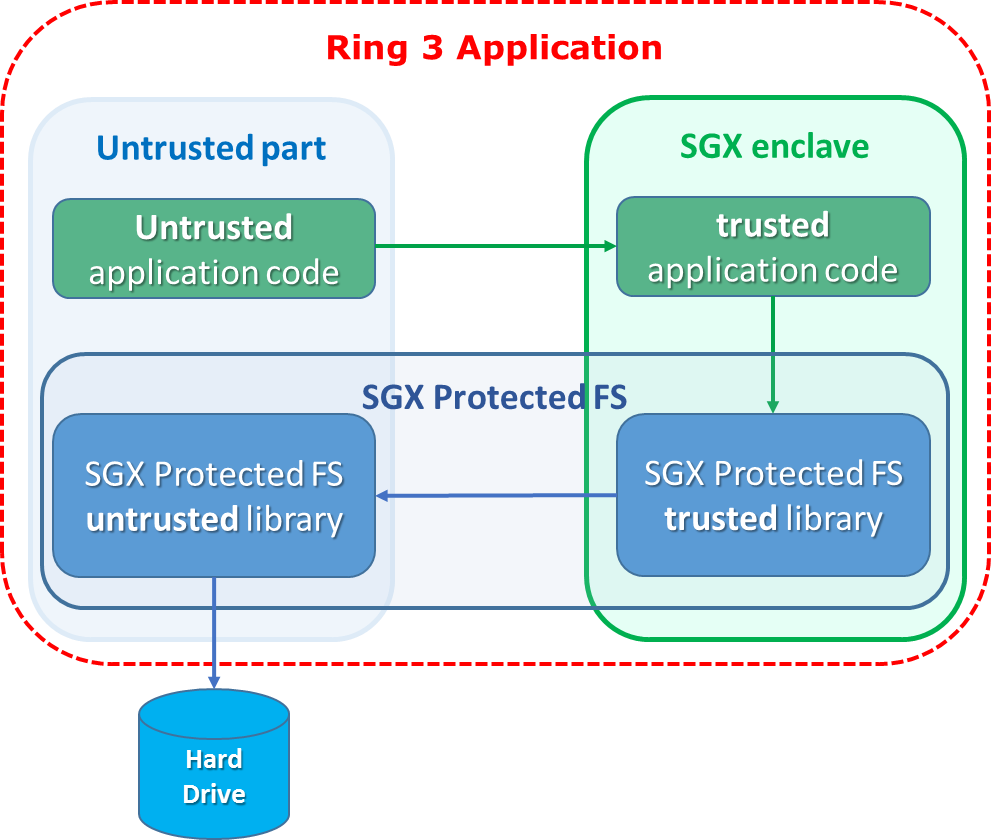
**Figure3.3**　Flowchart of AEX and exception handling

“AEX离开Enclave”及“异常处理完成返回Enclave”是重要的安全关键点，但现有防御方案并未深入分析。对此，在AEX触发处和tRTS异常处理句柄处插桩代码审计检查“AEX离开Enclave”及“异常处理完成返回Enclave”相关事件，可审计检查的安全事件信息包括寄存器、异常相关信息和AEX方向等。

## 3.2　其他关键执行路径

除Enclave接口外对其它关键执行路径进行分析，提取其中的安全关键点，进一步增强SGX架构安全。例如，密封文件、加密信道和认证过程等是tRTS向Enclave提供的安全服务机制，它们的关键执行路径中存在安全关键点，需要安全加固。

tRTS中受保护文件系统库向Enclave提供密封文件功能。如图3.4所示，不可信线程进入Enclave后，通过受保护文件系统库完成文件密封操作。tRTS将文件加密，通过OCALL让uRTS将加密文件存储到不可信外存中。

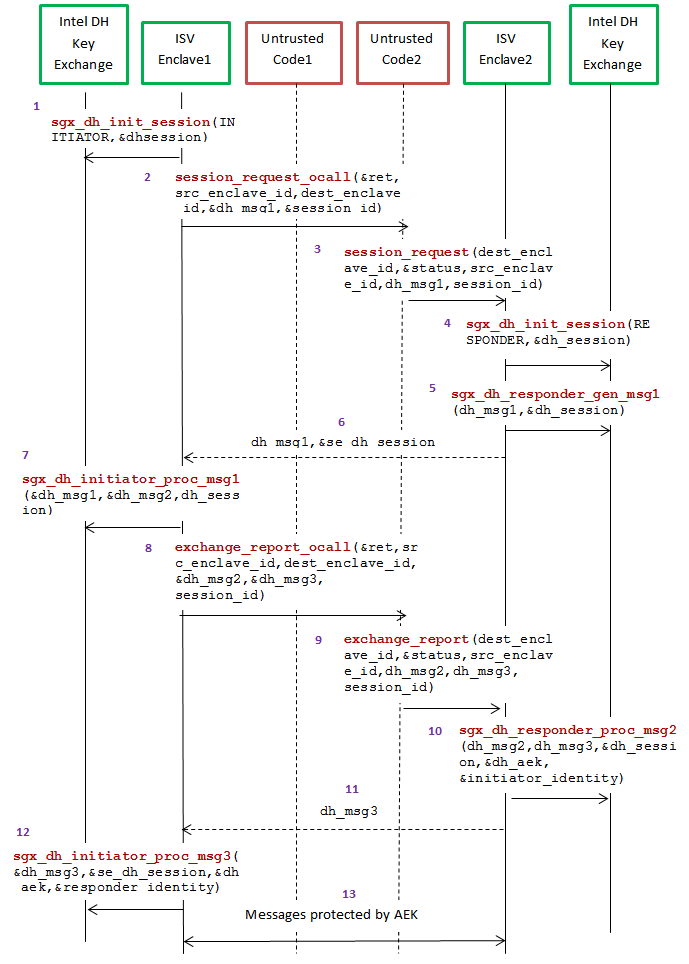


**图3.4**　受保护文件系统布局[4]

**Figure3.4**　Protected File System Layout[4]

文件密封机制中进出Enclave及加解密文件处是安全关键点，对此插桩代码审计检查相关安全事件，防范恶意密封文件等。可审计检查的安全事件信息包括加密文件、明文文件、文件元数据和操作方向等。

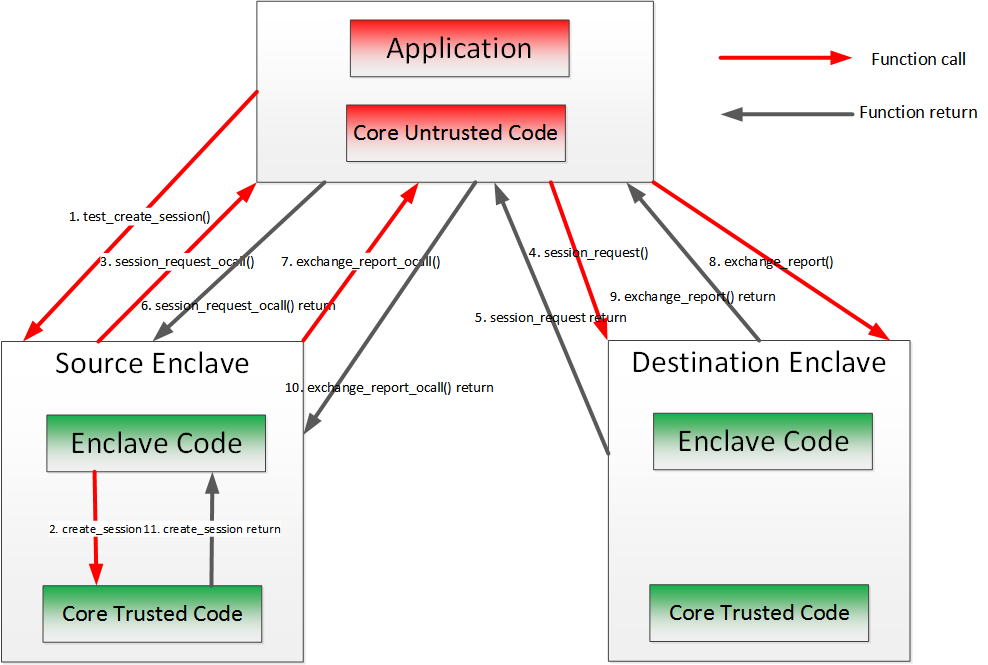
本地认证过程如图3.5所示，两个Enclave间通过uRTS不断交换信息，验证对方本地认证报告，完成Diffie-Hellman密钥交换。



**图3.5**　本地认证（使用DH密钥交换库）[4]

**Figure3.5**　Local Attestation Flow with the DH Key Exchange Library[4]

安全信道建立过程如图3.6所示，两个Enclave间通过uRTS不断交换信息，完成Diffie-Hellman密钥交换，使用协商的密钥构建安全信道。



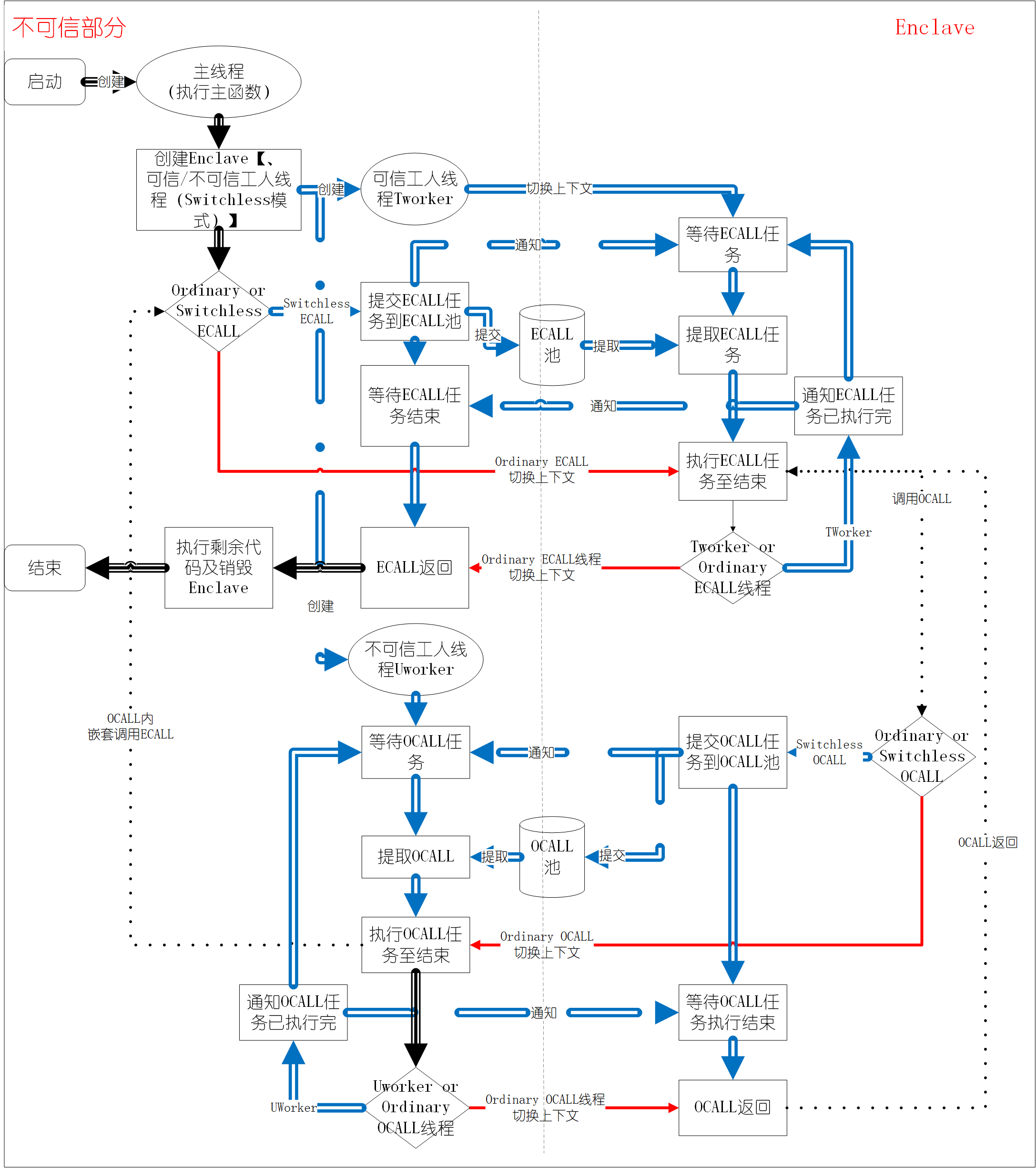
**图3.6**　安全信道的建立（使用DH密钥交换库）[4]

**Figure3.6**　Secure Channel Establishment Flow with the DH Key Exchange[4]

这些安全机制中进出Enclave及加解密数据等处是安全关键点，对此在其tRTS端插桩代码审计检查相关安全事件。

## 3.3　SGX线程模型

基于对SGX关键执行路径及安全关键点位的分析，归纳总结Ordinary/Switchless ECALL/OCALL中线程模型，如图3.7所示。



**图3.7**　SGX应用中Ordinary/Switchless ECALL/OCALL线程模型

**Figure3.7**　Thread model of Ordinary/Switchless ECALL/OCALL in SGX application

基于图3.7及SampleEnclave/Switchless示例代码（属于linux-sgx Github仓库）分析SGX线程模型。

Ordinary ECALL/OCALL线程模型：（基于SampleEnclave示例代码，图中红色单条细直线及黑色三条粗直线）

1. 主线程创建Enclave实例。
2. 主线程Ordinary ECALL切换上下文进入Enclave：主线程（物理上是CPU核）占用一个已注册的TCS（Thread Control Structure，线程控制结构体）调用EENTER指令进入tRTS【将CR\_ENCALVE\_MODE置1标识CPU处于Enclave模式，将不可信栈帧寄存器（u\_RSP和u\_RBP）等上下文存储到Enclave内的SSA（State Save Area，状态存储区域）】，tRTS清理通用寄存器，设置Enclave线程栈为Root ECALL栈帧（Enclave线程栈独立于不可信线程栈）。
3. 被切换上下文的主线程执行ECALL内容。
4. ECALL中，线程调用Ordinary OCALL切换上下文到Enclave外：线程在tRTS中将上下文封装成OCALL栈帧保存到Enclave线程栈中，未来OCALL返回时从Enclave线程栈中恢复上下文（Enclave线程栈变成“Root ECALL栈帧+OCALL栈帧”），清理通用寄存器，调用EEXIT指令（恢复不可信栈帧，将CR\_ENCLAVE\_MODE置0）退到Enclave外uRTS并执行OCALL。
5. 线程执行OCALL函数。如果OCALL中嵌套调用ECALL，Enclave线程栈变成“Root ECALL栈帧+OCALL栈帧+ECALL栈帧”。
6. OCALL执行结束，线程返回Enclave。线程调用EENTER指令返回到Enclave，从OCALL栈帧恢复上下文（Enclave线程栈变成“Root ECALL栈帧”）。
7. 线程执行ECALL剩下代码。当ECALL执行完毕时，线程清理通用寄存器，调用EEXIT指令（恢复不可信栈帧寄存器）退到Enclave外uRTS。
8. 线程执行主函数剩余代码，销毁Enclave。

Enclave外主线程可以创建若干线程并发进入Enclave，每个线程执行如上单线程场景。

Switchless ECALL/OCALL线程模型：（基于Switchless示例代码，图中蓝色双条粗直线加黑色三条粗直线。）

1. 主线程创建Enclave实例，初始化Switchless模式。主线程在Enclave外创建若干线程，一部分切换上下文进入Enclave成为可信工人线程TWorker，另一部分留在Enclave外成为不可信工人线程UWorker。
2. 主线程调用Switchless ECALL：主线程将ECALL任务提交到ECALL任务池并通知TWorker，然后等待ECALL执行完。TWorker收到通知，从ECALL任务池提取任务并执行。若无空闲TWorker，Switchless ECALL任务会退化，被主线程以Ordinary方式执行。
3. Switchless ECALL中执行Switchless OCALL：TWorker将OCALL任务提交到OCALL任务池并通知UWorker，然后等待OCALL执行完。UWorker收到通知，从OCALL任务池提取任务并执行。若无空闲UWorker，Switchless OCALL任务会退化，被当前TWorker以Ordinary方式执行。
4. UWorker通知TWorker Switchless OCALL任务已执行完，向TWorker传输函数返回值。TWorker解除等待。
5. TWorker执行ECALL剩余代码，直至Switchless ECALL任务执行完。TWorker通知主线程Switchless ECALL任务已执行完，向主线程传输函数返回值。主线程解除等待。
6. 主线程执行主函数剩余代码，销毁Enclave。

SGX线程模型中，线程或者切换上下文执行目标代码，或者委托预先切换上下文的工人线程代理执行目标代码。线程切换上下文过程中，备份切换前的上下文，设置CPU内元数据寄存器值并转变模式，清理通用寄存器，启用新的上下文。

“Switchless/Ordinary ECALL/OCALL及其结束”时，攻击者能够透过这些攻击面恶意输入输出信息。对此，在其tRTS端插桩代码审计检查相关安全事件（如图3.7斜纹竖长条柱所示）。

## 3.4　本章小结

本章分析了Enclave接口关键执行路径及其他关键执行路径，描述了其代码流程，提取其安全关键点。以SampleEnclave和Switchless示例代码描述SGX线程模型，归纳总结Ordinary/Switchless ECALL/OCALL中的线程模型。SGX关键执行路径及安全关键点分析能指导审计检查代码插桩方式及位置、审计检查功能实施和已有安全准则兼容，还能启发新的安全准则。

# 第四章　针对Intel SGX SDK的安全增强框架设计

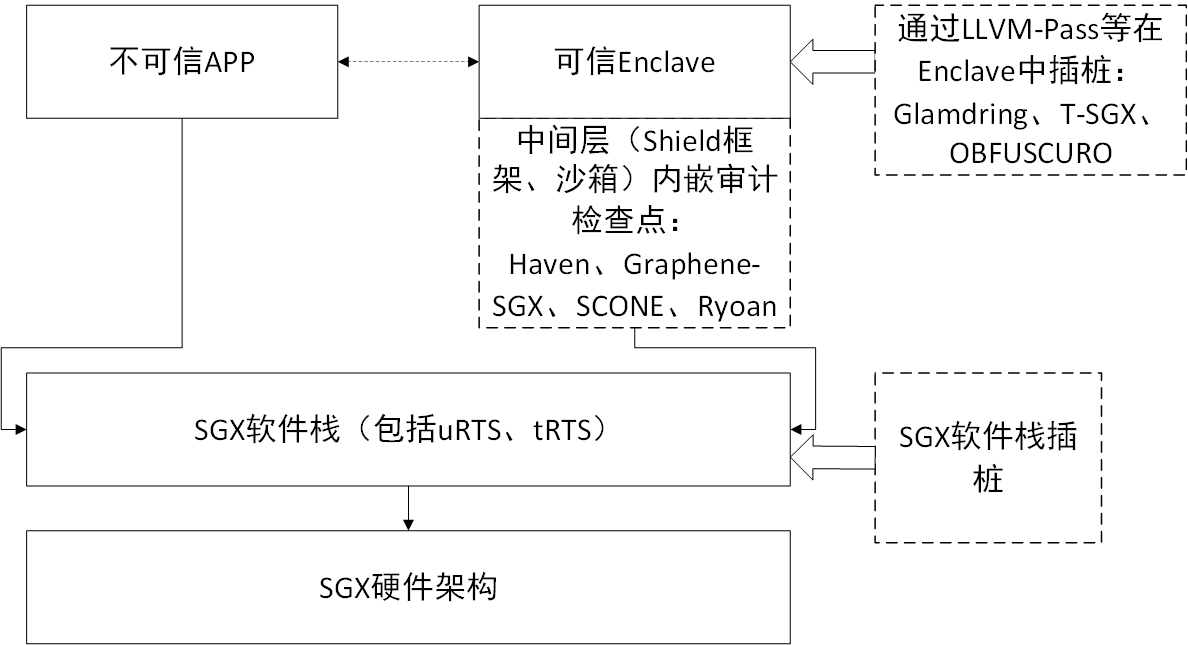
基于关键执行路径及安全关键点分析，选择针对安全关键点的审计检查代码插桩方式及插桩位置，设计针对安全事件的审计检查功能，提出针对Intel SGX SDK的安全增强框架（简称SGX-SEF），加固SGX架构自身安全。本文将安全关键点上部署的审计检查代码称为审计检查点。

## 4.1　针对Intel SGX SDK的安全增强框架总体设计

### 4.1.1　审计检查点插桩方式

为了审计检查SGX关键执行路径中的安全事件，需要对安全关键点插桩审计检查代码。本节对比分析了三种审计检查点插桩方式（如图4.1所示）：

1. 使用编译器插桩技术在Enclave代码插桩审计检查点。LLVM-Pass等编译器插桩技术适合插桩模式固定的场景，操作简单且自动化，但需要先验的插桩模式。本文旨在对关键执行路径插桩，没有先验的固定模式。Enclave代码不如tRTS能获取更多的安全事件信息。
2. 中间层（Shield框架和沙箱）内嵌审计检查点。中间层丰富扩充了Enclave功能，但也引入大量TCB。中间层不如tRTS能获取更多安全事件信息。
3. SGX软件栈插桩。tRTS中能获取更丰富的安全事件信息，实施更丰富的安全策略，但SGX软件栈插桩方式要求深入理解SGX软件栈。



**图4.1**　插桩方式对比

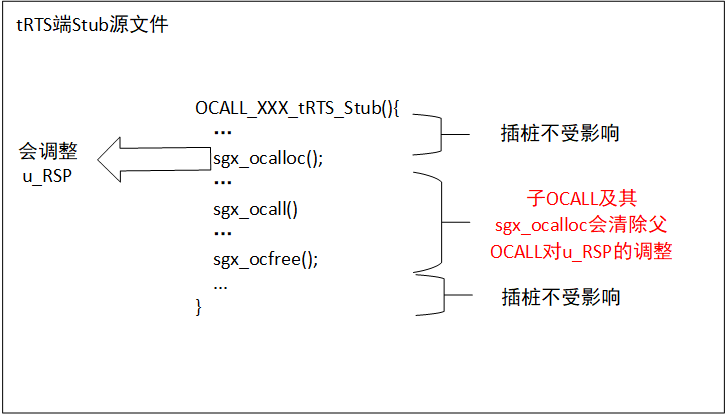
**Figure4.1**　Comparison of instrumentation methods

本文设计SGX-SEF时，针对关键执行路径上的安全关键点采用SGX软件栈插桩方式插桩审计检查代码。

### 4.1.2　审计检查点插桩位置

一个安全关键点中存在若干代码点位可供插桩审计检查代码，但有些代码点位存在相关冲突，不适合插桩审计检查点。例如针对OCALL，如果在tRTS端OCALL流程中插桩审计检查点，审计检查代码内嵌子OCALL（如用于记录日志的文件操作和维护并发的互斥量/读写锁等功能需要通过OCALL调用系统调用），那么就可能出现非预期的错误。错误原因的关键点在于（如图4.2所示）：

1. 带参数OCALL的tRTS端Stub函数中使用sgx\_ocalloc函数将存储于Enclave内的参数复制到Enclave外，供OCALL后续使用。
2. 由于当前环境为tRTS，不可信栈帧的u\_RBP和u\_RSP值（64位寄存器，u代表不可信）已被备份到SSA中，因此sgx\_ocalloc会调整SSA中的u\_RBP及u\_RSP值将拷贝到Enclave外的参数包裹到不可信栈帧中。
3. 但是如果父OCALL的sgx\_ocalloc与sgx\_ocfree之间内嵌了子OCALL（含sgx\_ocalloc），那么父OCALL对u\_RSP的调整操作将丢失，导致父OCALL代码出错。



**图4.2**　OCALL Stub函数示意图

**Figure4.2**　Diagram of OCALL Stub function

因此，sgx\_ocalloc和sgx\_ocfree之间不适合内嵌（包含sgx\_ocalloc的）子OCALL。SGX-SEF将针对OCALL的审计检查点位定在了tRTS端Stub函数中及sgx\_ocalloc-sgx\_ocfree外。

针对“Ordinary/Switchless ECALL/OCALL及其结束”，修改SGX Edger8r工具将审计检查点自动部署到tRTS端各个Stub函数内。针对AEX，在tRTS端异常处理函数（trts\_handle\_exception）中插桩审计检查点，但该审计检查点能被不可信内核绕过。针对tRTS提供的受保护文件系统库等，在其关键执行路径上插桩审计检查点，如在sgx\_fread插桩审计检查点避免读取的文件内容存在恶意信息。

### 4.1.3　安全事件审计

为了对SGX安全关键点进行安全加固，本节设计了针对安全事件的审计功能，旨在将安全关键点上的安全事件逐一记录，以便防御者度量安全事件并分析攻击痕迹。

审计功能：当代码执行到审计检查点时，将当前安全事件的环境信息【包括当前事件类型（ECALL/OCALL/AEX/其他）、函数名、函数索引值（索引值唯一对应了其地址）和参数信息等】传入审计检查点，审计检查点将安全事件记录到Enclave内的全局日志中。针对全局日志，目前实现了两种存储方式：Enclave内存方式和密封文件方式。

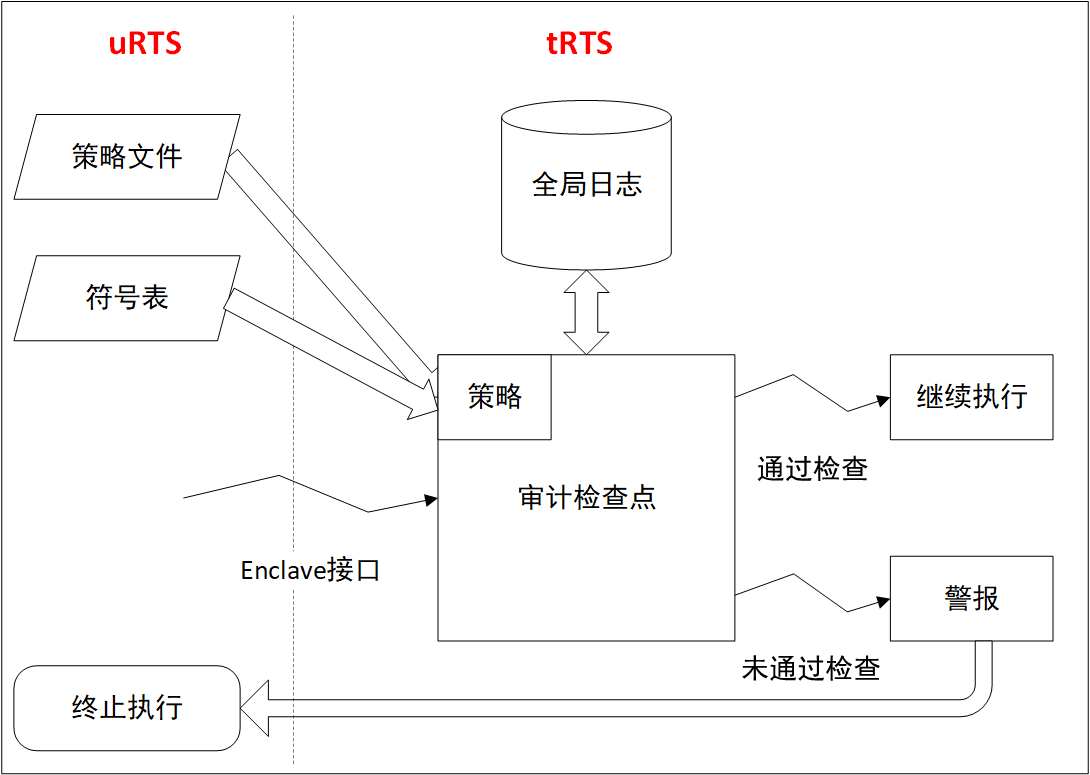
Enclave内存方式：全局日志以数据结构形式存储在Enclave内存中。本文采用有限长度的双端队列数据结构，删除最久远的日志记录避免占用过大的Enclave内存，避免程序崩溃。由于EPC物理内存大小有限（如64M/128M/256M），使得每个Enclave实例的内存大小有限，当内存使用超出限制时会导致程序崩溃。

密封文件方式：将日志以密封文件方式存储到外存。这能减少对Enclave内存的消耗，但由于密封文件能被攻击者恶意删除，使得全局日志的可用性无法得到保障。密封文件方式开销巨大（涉及加解密和OCALL等）使其在应用中导致程序执行时间过长。

SGX-SEF设计实现了这两种日志存储方式，默认使用Enclave内存方式记录全局日志，不建议使用开销巨大的密封文件方式。

### 4.1.4　安全事件检查

设计安全事件检查功能，分析审计检查点获取的安全事件信息根据安全策略检测攻击。审计检查点内部架构如图4.3所示。Enclave内从不可信环境（依靠安全建立等技术）可信读取策略文件及符号表。当代码执行到审计检查点时，审计检查点会记录安全事件信息到全局日志中（即审计功能），根据安全策略进行安全事件检查操作（能够查询全局日志实现联动检查），若符合策略，代码继续执行，若不符合策略，审计检查点发出警告或终止代码运行等。



**图4.3**　审计检查点内部架构

**Figure4.3**　Internal architecture of audit-checkpoints

审计检查点主要包括事件获取、日志记录、安全策略、符号表和策略匹配五个关键部分。

事件获取。当代码执行到审计检查点时，将安全事件环境信息传给审计检查点。安全事件信息包括事件类型（ECALL/OCALL/AEX/其他）、函数符号名、函数索引值（索引值唯一对应其地址）和参数信息等。

日志记录。本文维护了全局日志以记录所有安全事件，检查功能中可以查询全局日志实现联动检查。针对全局日志，设计了Enclave内存方式及密封文件方式，默认采用Enclave内存方式，不推荐开销巨大的密封文件方式。

安全策略。防御者基于SGX-SEF制定安全策略，达到所要求的安全准则。tRTS端的审计检查点需要可信读取存储在不可信硬盘上的安全策略文件。本文修改SGX Edger8r以读取安全策略文件，将安全策略重新组织成tRTS端Stub内的数据结构，使安全策略与Stub一起编译到Enclave镜像文件中（Enclave镜像文件受签名保护），利用Enclave实例安全建立机制检测安全策略是否被篡改。

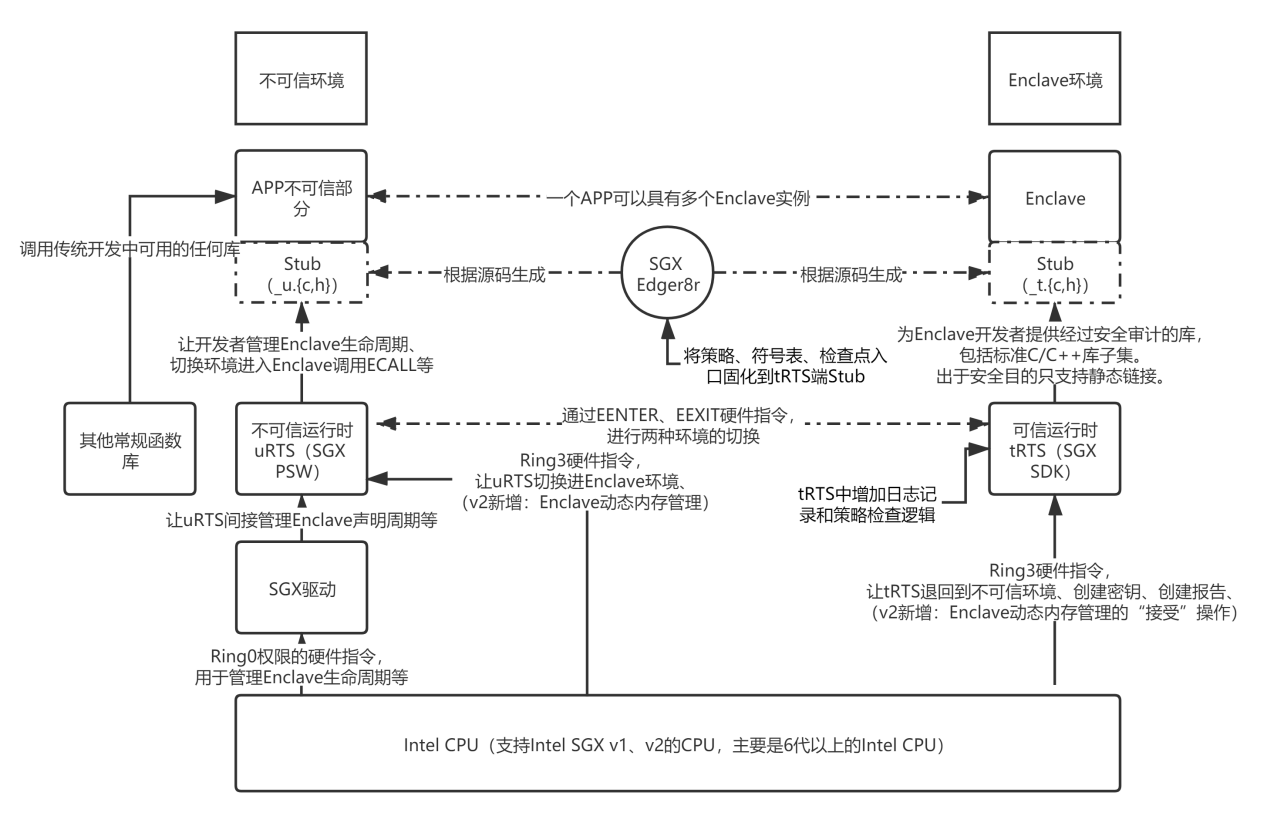
符号表。安全策略以字符形式存储于策略文件中，但实际检查时只能使用函数和参数的地址信息（非Debug模式下去除了大部分符号名），审计检查点无法直接比对字符名与地址。对此修改SGX Edger8r，在SGX Edger8r读取EDL文件时，在Stub文件中额外生成多个符号表（包含了地址与符号名的对应关系），帮助审计检查点完成检查操作。

策略匹配。本文针对调用排序、并发调用、恶意线程调度和时间侧信道攻击向量提供了安全策略文件及策略匹配算法，以检测当前程序运行是否符合安全策略，如果符合安全策略，程序继续执行，否则审计检查点返回错误码并终止程序执行。

## 4.2　对SGX原生软件栈的修改

SGX-SEF在SGX软件栈关键执行路径中插桩若干审计检查点，审计检查安全事件，加固安全关键点。SGX-SEF对SGX原生软件栈所做修改如图4.4所示（图中斜纹形状）：

1. 修改SGX Edger8r将安全策略、符号表和审计检查点入口等固化到tRTS端Stub，使新增内容受到Enclave实例安全建立机制的保护。
2. 在tRTS中增加了全局日志记录及策略检查等功能。



**图4.4**　SGX-SEF对SGX软件栈的修改

**Figure4.****4**　Modification of SGX software stack by SGX-SEF

## 4.3　本章小结

为了审计检查SGX关键执行路径中的安全事件，本章对比分析了多种审计检查点插桩方式，选择SGX软件栈插桩方式。分析了审计检查代码冲突点位及冲突原因，针对各个安全关键点选择审计检查点位。设计了审计功能，阐述了用于记录全局日志的Enclave内存方式和密封文件方式。设计了检查功能，阐述了检查功能中五个关键部分。形成了针对Intel SGX SDK的安全增强框架（SGX-SEF），指出了SGX-SEF对SGX原生软件栈所做的修改，对SGX原生Enclave实例安全建立机制的利用。

# 第五章　针对Intel SGX SDK的安全增强框架验证

本章描述实验过程中的实验环境。展示SGX-SEF的审计效果，评估SGX-SEF审计功能的性能开销，分析引起性能开销的原因。针对调用排序、并发调用、恶意线程调度和时间侧信道攻击向量编写安全策略验证SGX-SEF防御（检查功能）效果。

## 5.1　实验环境

本文的实验环境为笔记本，CPU型号为i5-9300H，内存大小为16GB，操作系统为Ubuntu20.04，Linux内核版本为5.4，linux-sgx（Intel SGX PSW及SDK）版本为2.11，linux-sgx-driver（Intel SGX驱动）版本为2.11，CPU及BIOS均支持SGX。

SkyLake及更新架构的CPU均支持Intel SGX特性，但部分架构（如TigerLake）不支持Intel SGX，具体内容可以前往Intel官网查询。除需要CPU支持SGX特性外，还需要确保BIOS中开启了SGX。有些旧版BIOS不支持SGX，但部分电脑厂商在后续BIOS更新中增加SGX开启选项，对此可以升级BIOS以获得SGX开启选项。

建议使用真机（Bare Metal）形式安装Ubuntu操作系统，VMWare Workstation等大部分虚拟机管理程序中安装的Ubuntu虚拟机不支持SGX特性（即使CPU及BIOS均支持SGX，但VMWare Workstation自身不能向虚拟机提供Intel SGX特性）。KVM-SGX及QEMU-SGX能够帮助SGX用户在QEMU虚拟机中开启SGX特性。

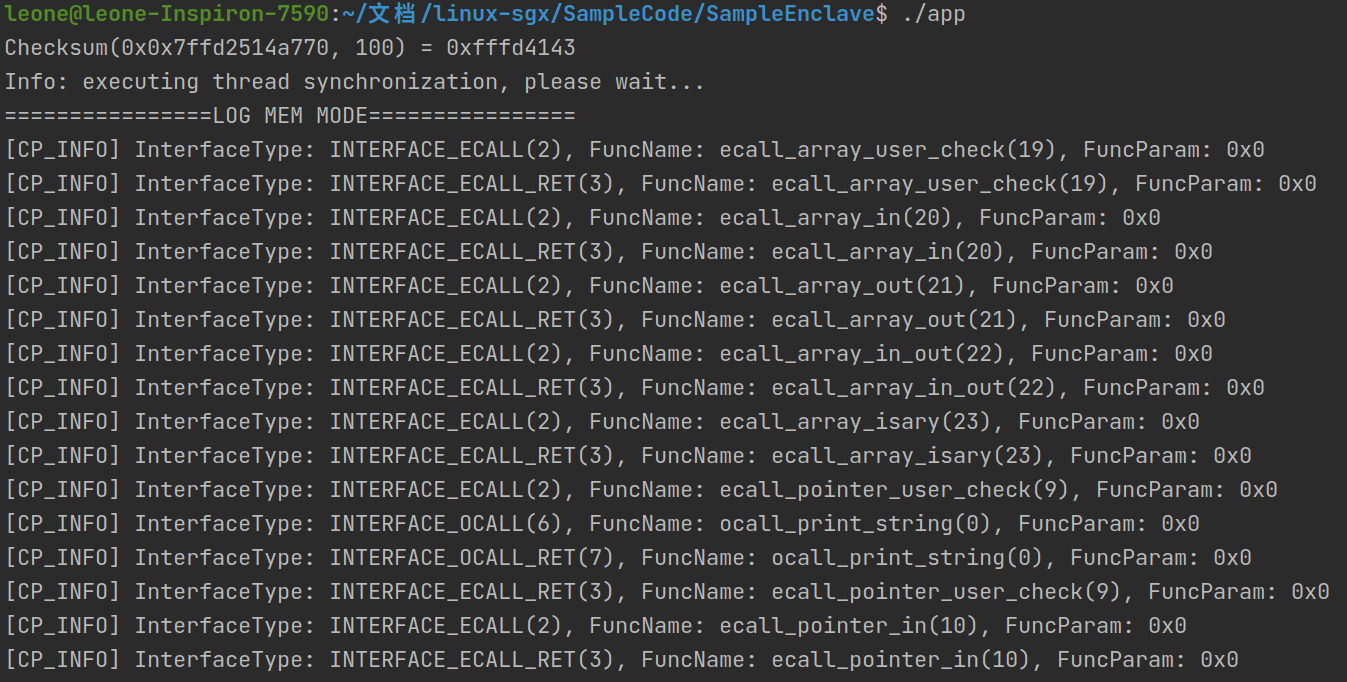
可以通过sgx-software-enable Github仓库或参照SGX手册使用CPUID硬件指令检测当前平台是否成功开启了SGX。

## 5.2　审计功能效果验证及性能分析

本节展示审计功能效果，评估审计功能开销，分析引起开销的原因。

### 5.2.1　审计功能效果验证

审计功能的实现效果如图5.1所示。对SampleEnclave中的安全事件进行审计，安全事件信息包含接口类型、接口名称、接口索引和参数（实现中可以捕获参数，但由于参数数量各异，先用0值填充）等。图中的演示效果是通过调用ECALL打印（printf）位于Enclave内的全局日志。由于printf需要通过OCALL实现（Enclave内不支持），因此要求审计检查点在审计时过滤printf所用的OCALL，避免无限次嵌套导致程序崩溃。



**图5.1**　审计功能的效果

**Figure5.1**　Effect of Audit function

### 5.2.2　审计功能性能开销分析

SGX-SEF审计功能实现中，采用密封文件方式记录全局日志会导致性能开销过大，原因在于加解密和OCALL等会造成巨大开销，建议采用Enclave内存方式记录全局日志。本小节将对采用Enclave内存方式记录全局日志的审计功能进行性能分析。基于linux-sgx Github仓库的SampleEnclave和Switchless示例代码，评估审计功能（在Switchless/Ordinary ECALL/OCALL上）的性能开销，并分析其原因。本文所测时间值以秒为单位，小数点精确到微秒，所有时间值均已被测量三次取平均。

基于Switchless示例代码评估SGX-SEF审计功能性能开销。Switchless代码中重复50000次Switchless和Ordinary调用，所调用函数中没有任何代码（即空负载），因此该评估工作旨在测量Switchless和Ordinary调用的固有开销。由于EPC物理内存大小有限，因此审计功能采用有限长度的（删除最久之前的记录）双端队列以内存形式记录全局日志。双端队列大小为10000、1000、100、10项以及不使用SGX-SEF时的时间开销如下表5.1所示。

**表5.1**　基于Switchless的审计功能性能开销测量表

**Table5.1**　Table of audit performance overhead measurement based on Switchless

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 日志记录项数 | 10000项 | 1000项 | 100项 | 10项 | No SGX-SEF |
| Switchless OCALL时长（秒） | 3.149677 | 0.346707 | 0.081700 | 0.057652 | 0.016433 |
| Ordinary OCALL时长（秒） | 3.494144 | 0.527837 | 0.263078 | 0.238200 | 0.174782 |
| Switchless ECALL时长（秒） | 3.254656 | 0.363362 | 0.096939 | 0.067975 | 0.021817 |
| Ordinary ECALL时长（秒） | 3.587716 | 0.722888 | 0.452293 | 0.429314 | 0.382114 |

当日志记录项数增大时，审计功能对各种调用方式产生的开销增大，原因在于日志使用维护开销增大，缓存命中率降低进一步导致开销增大。当日志项数过大时，占用内存超出Enclave实例可用EPC大小，Enclave实例崩溃。使用SGX-SEF审计功能相较于不使用SGX-SEF时，Switchless调用方式开销增幅明显，Ordinary调用方式开销增幅较小，原因在于Switchless调用方式中多个Enclave工人线程并发记录日志所用的同步机制及OCALL大幅增加性能开销，记录日志代码流程待优化。依据实验中接触的Enclave应用场景及反复实验得出的经验，日志大小为100项的审计功能对各种调用方式引起的开销增幅不明显，但又足够记录最近的安全事件用于审计检查功能。日志大小100项的SGX-SEF审计功能最优情况（Ordinary调用方式）下开销增幅仅为18%。

基于SampleEnclave示例代码评估SGX-SEF审计功能性能开销。SampleEnclave代码流程以Ordinary调用为主，调用函数并非空负载，该评估工作旨在评估审计功能在SGX实际应用中产生的性能开销。测量过程使用日志大小100项Enclave内存形式日志的SGX-SEF审计功能，结果如表5.2所示，使用SGX-SEF审计功能时，SampleEnclave总时间开销为0.072994秒，而不使用SGX-SEF时的时间开销为0.064461秒，开销增幅为13%。

**表5.2**　基于SampleEnclave的审计功能性能开销测量表

**Table5.2**　Table of audit performance overhead measurement based on SampleEnclave

|  |  |  |
| --- | --- | --- |
| 日志记录项数 | 100项 | No SGX-SEF |
| SampleEnclave执行时长（秒） | 0.072994 | 0.064461 |

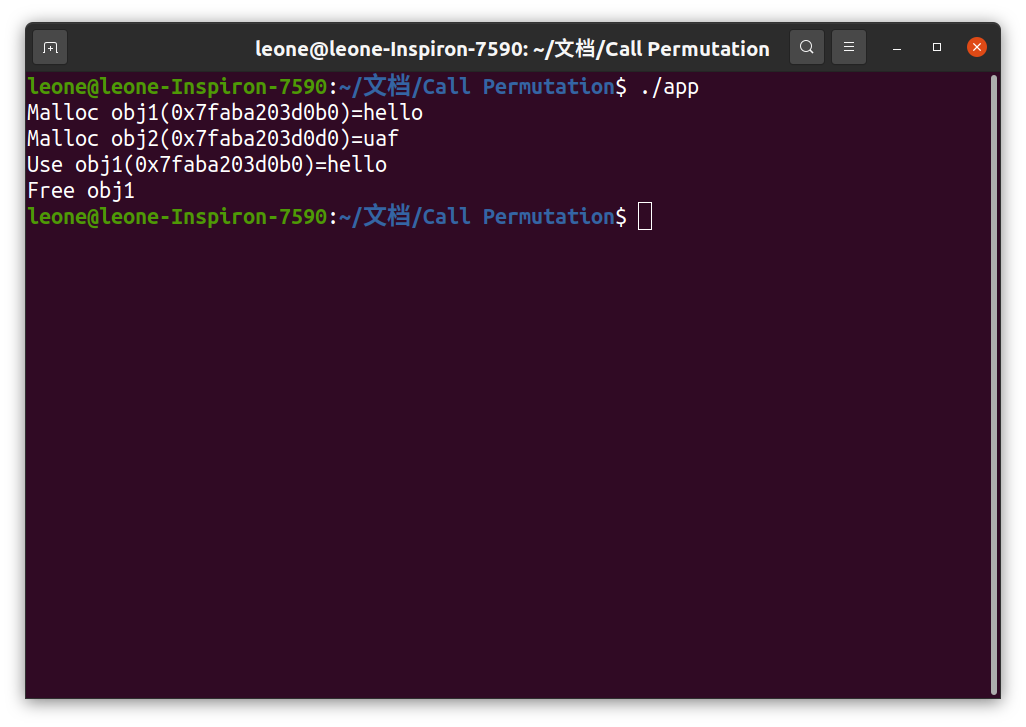
综上所述，SGX-SEF审计功能最优情况下产生13%~18%的开销增幅。

## 5.3　防御（检查功能）效果验证

本节针对调用排序、并发调用、恶意线程调度和时间侧信道攻击向量编写安全策略验证防御（检查功能）效果。

### 5.3.1　针对“调用排序”攻击向量的防御效果验证

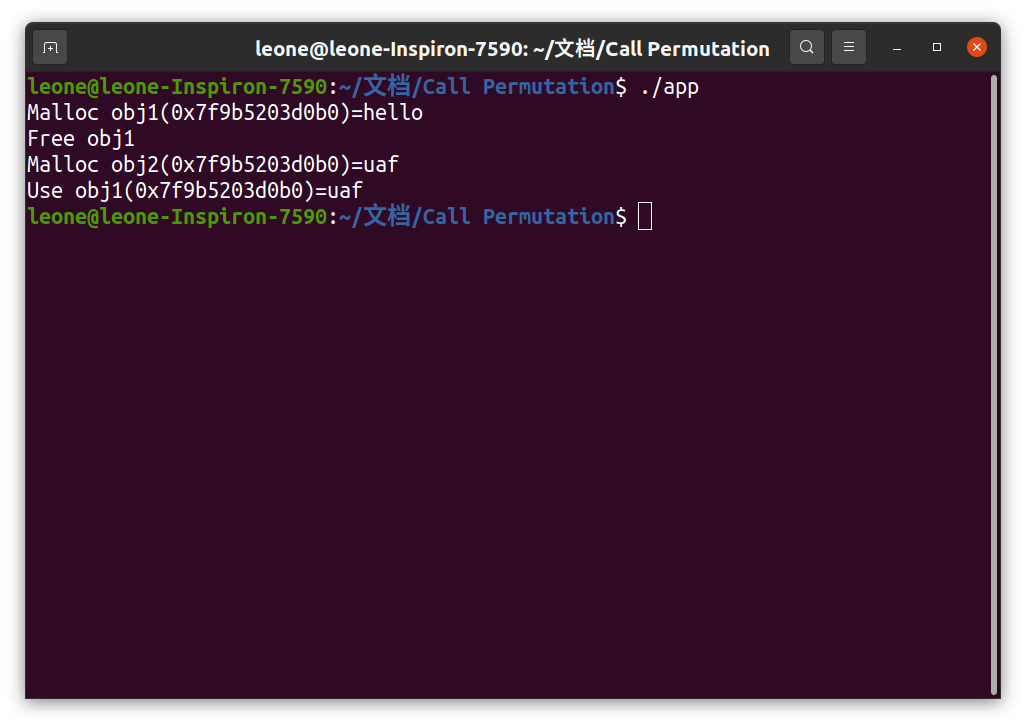
实验过程中构造实现了“调用排序”攻击向量的POC。攻击之前，ECALL调用顺序先后为ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数，ecall\_malloc\_obj1函数在Enclave堆上创建值为“hello”的对象1（即所指内容为“hello”的对象1指针，下文不再对此特别说明），ecall\_malloc\_obj2\_use\_obj1函数在Enclave堆上创建值为“uaf”的对象2并使用对象1的值“hello”，ecall\_free\_obj1函数释放对象1但未将对象1置NULL（对象1成为悬空指针，悬空指针即所指对象已被释放但自身未被置NULL的指针）。程序结果如图5.2所示



**图5.2**　调用排序攻击之前程序执行结果图

**Figure5.2**　Diagram of program execution result before Calling Permutation Attack

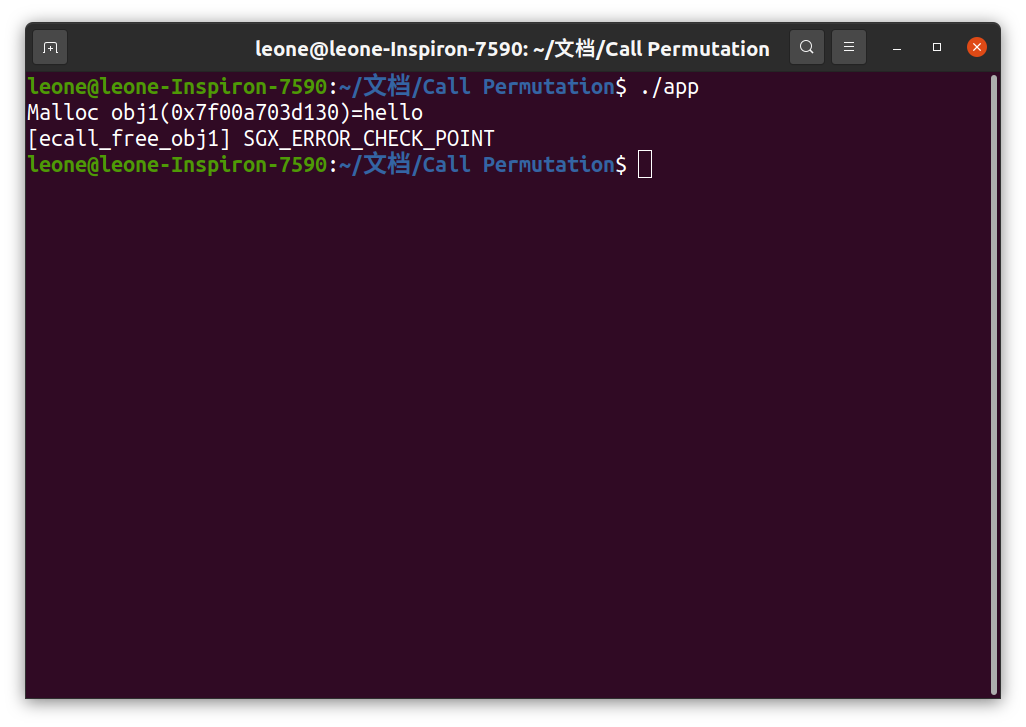
恶意将ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数的先后顺序修改为ecall\_malloc\_obj1函数、ecall\_free\_obj1函数和ecall\_malloc\_obj2\_use\_obj1函数，使得ecall\_free\_obj1中的悬空指针（即对象1指针）被ecall\_malloc\_obj2\_use\_obj1使用，触发UAF。程序执行结果如图5.3所示，ecall\_malloc\_obj2\_use\_obj1函数中，对象2指针被分配了在与悬空指针（对象1指针）相同的地址，使用悬空指针（对象1指针）所指内容等同于使用对象2的值“uaf”。



**图5.3**　调用排序攻击之后程序执行结果图

**Figure5.3**　Diagram of program execution result after Calling Permutation Attack

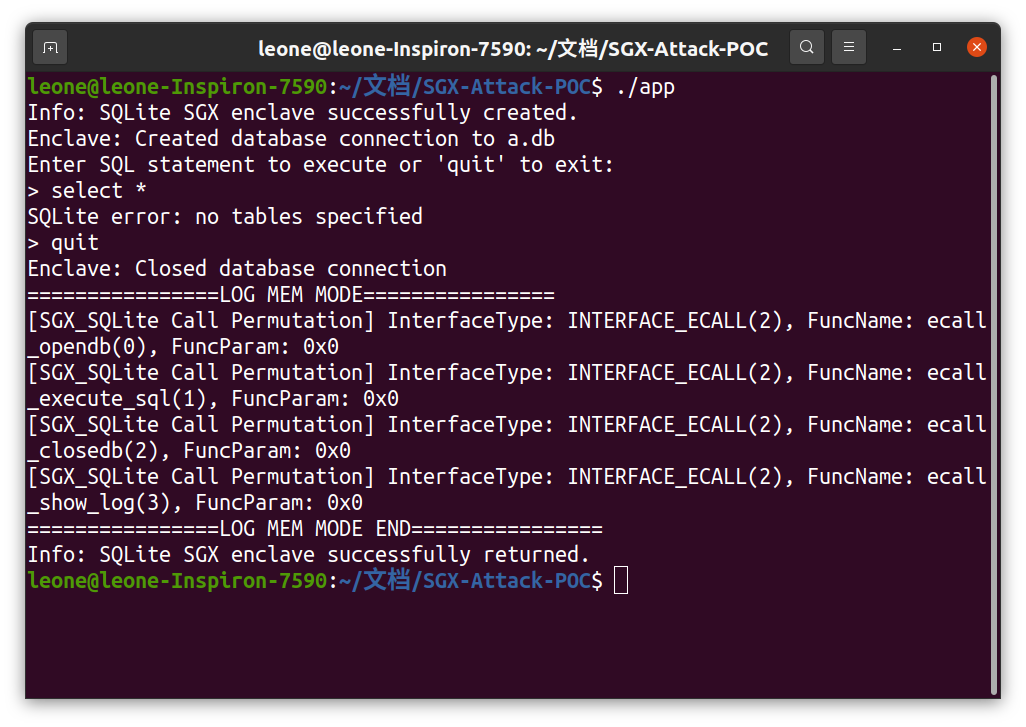
为了对抗“调用排序”攻击，Enclave开发者能够依据先验的预期执行情况编写策略文件，在SGX-SEF中设置“ECALL调用顺序白名单”安全策略，要求检测到的ECALL调用顺序先后分别为ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数，否则SGX-SEF将向Enclave代码返回错误码SGX\_ERROR\_CHECK\_POINT并终止代码运行。如图5.4所示，在SGX-SEF中设置“ECALL调用顺序白名单”策略能成功检测非预期的调用顺序。



**图5.4**　调用排序攻击检测结果

**Figure5.4**　Detection results of Call Permutation Attack

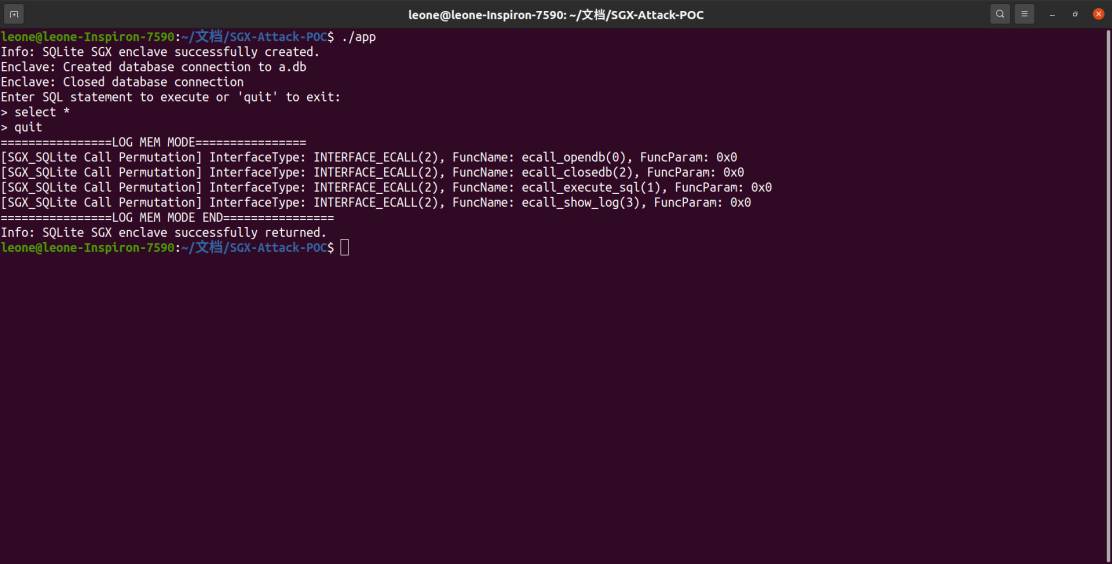
Github上开源的SGX\_SQLite中存在数据库句柄释放后未置NULL成为悬空指针的情况，能够被“调用排序”攻击。“调用排序”攻击前SGX\_SQLite程序执行情况如图5.5所示，即打开数据库句柄（ecall\_opendb），使用数据库句柄执行SQL语句（ecall\_execute\_sql），关闭数据库句柄（ecall\_closedb）。



**图5.5**　调用排序攻击前SGX\_SQLite执行情况图

**Figure5.5**　Diagram of SGX\_SQLite execution before Call Permutation attack

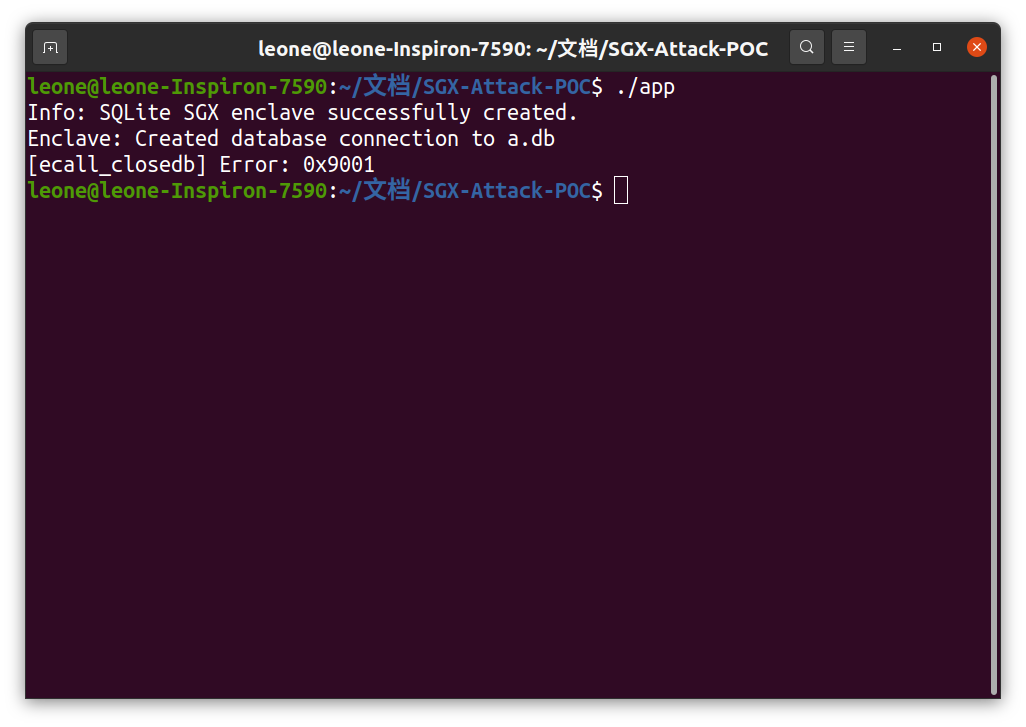
恶意篡改SGX\_SQLite中ECALL函数的调用顺序，使得数据库句柄关闭（ecall\_closedb）后未置NULL形成的悬空指针被继续使用（ecall\_execute\_sql），形成安全隐患。程序执行情况如图5.6所示。



**图5.6**　调用排序攻击后SGX\_SQLite执行情况图

**Figure5.6**　Diagram of SGX\_SQLite execution after Call Permutation attack

为了缓解“调用排序”攻击，Enclave开发者能够依据先验的预期执行情况编写策略文件，在SGX-SEF上设置“ECALL调用顺序白名单”策略，要求检测到的ECALL调用顺序先后分别为ecall\_open\_db函数、ecall\_execute\_sql函数和ecall\_close\_db函数。如果执行情况违反策略，SGX-SEF将返回错误码SGX\_ERROR\_CHECK\_POINT（0x9001）并终止程序运行。如图5.7所示，在SGX-SEF中设置“ECALL调用顺序白名单”策略能成功检测非预期的调用顺序。



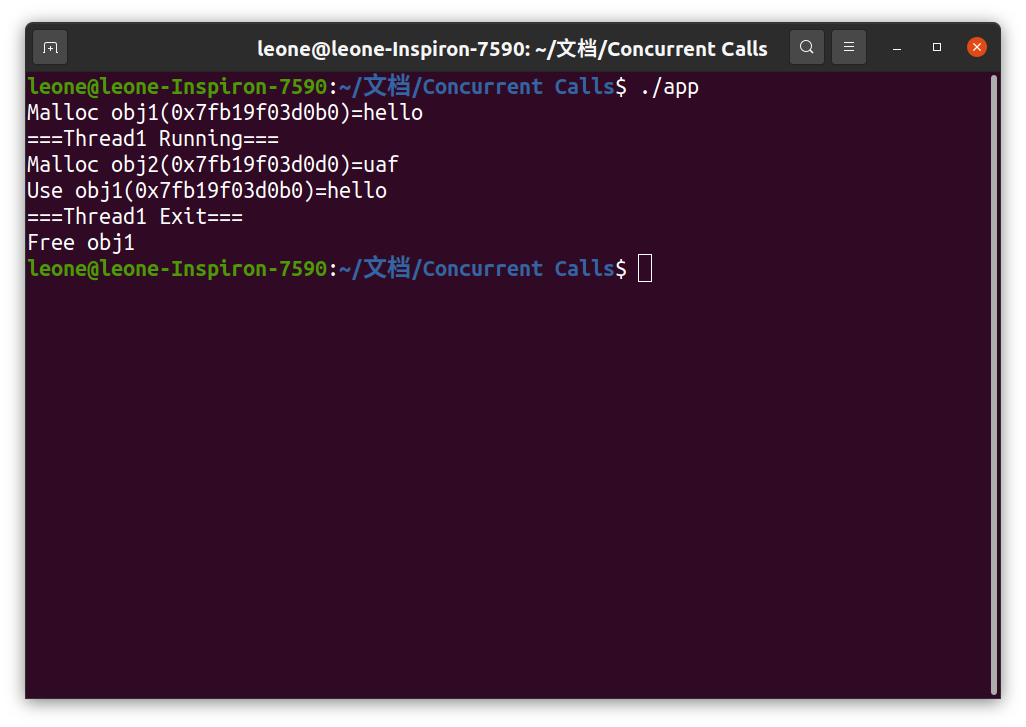
**图5.7**　SGX\_SQLite中调用排序攻击检测结果

**Figure5.7**　Detection results of Call Permutation Attack in SGX\_SQLite

除Glamdring外尚无防御方案针对SGX调用排序攻击向量进行有效防御。Glamdring减少ECALL数量以避免调用排序攻击，该方法效果有限，因为有些ECALL函数无法被裁剪。

### 5.3.2　针对“并发调用”攻击向量的防御效果验证

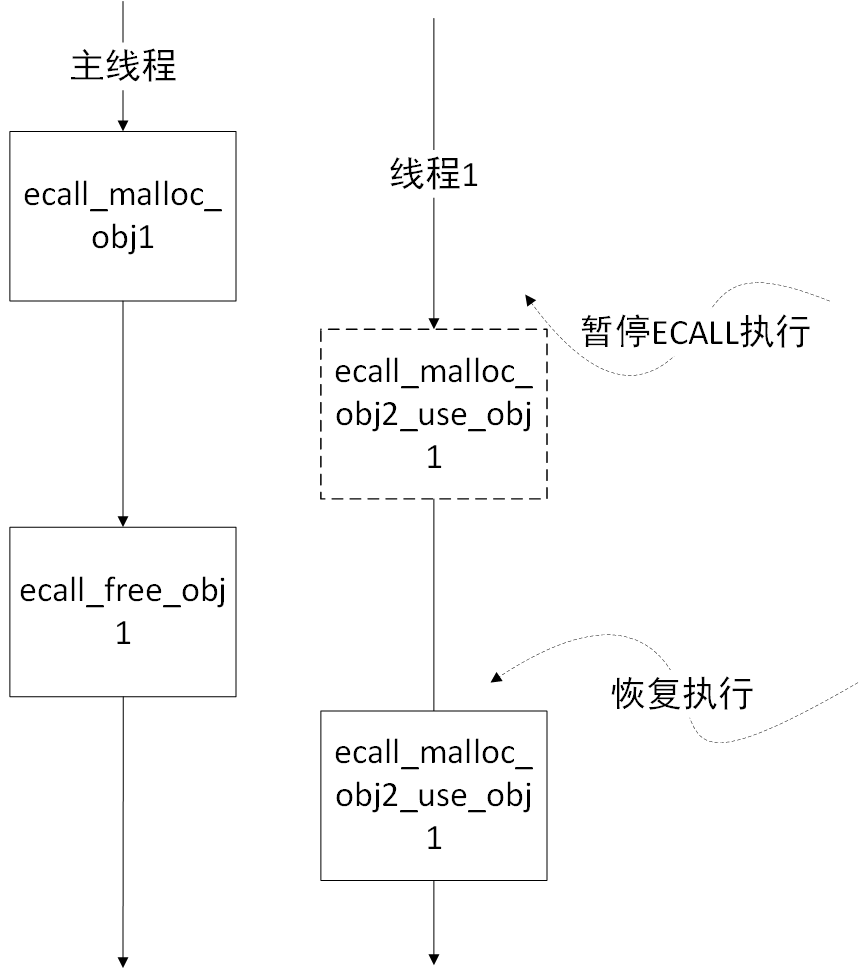
实验构造实现了“并发调用”攻击向量POC。并发调用攻击之前，ECALL调用顺序先后分别为ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数。程序执行结果如图5.8所示，主线程调用ecall\_malloc\_obj1函数在Enclave堆上创建对象1并等待线程1使用对象1，线程1在Enclave堆上创建对象2并使用对象1的值“hello”，线程1结束后主线程释放对象1但未将对象1置NULL（对象1指针成为悬空指针）。



**图5.8**　并发调用攻击之前程序执行结果

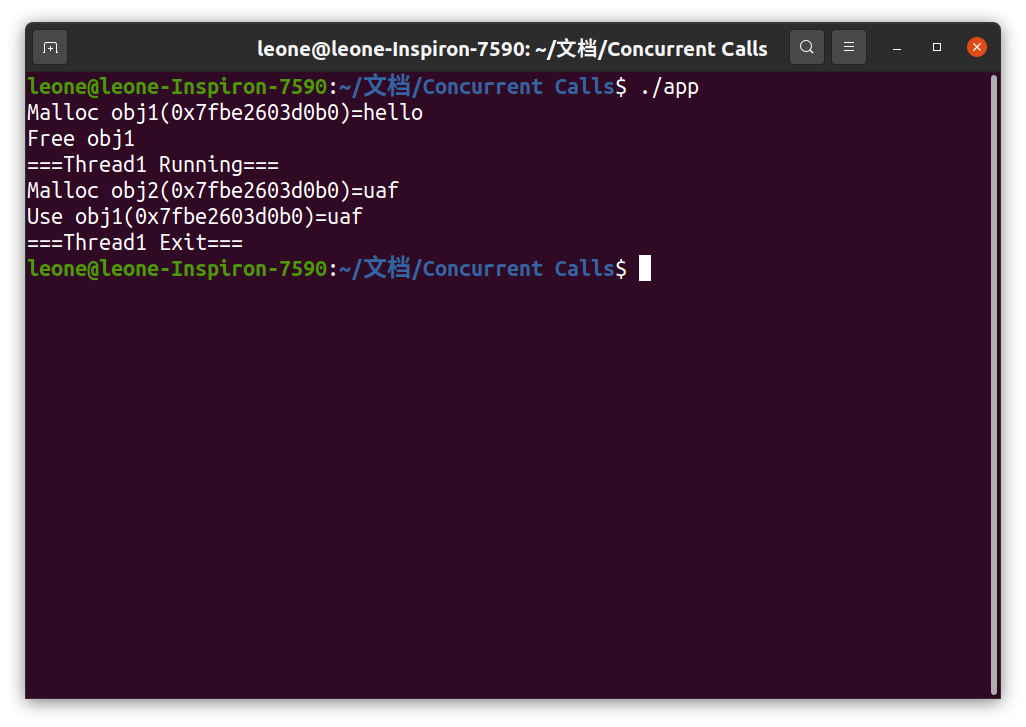
**Figure5.8**　Program execution result before Concurrent Calls Attack

恶意暂停线程1执行ecall\_malloc\_obj2\_use\_obj1函数，将ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数的ECALL调用顺序篡改为ecall\_malloc\_obj1函数、ecall\_free\_obj1函数和ecall\_malloc\_obj2\_use\_obj1函数的调用顺序（如图5.9所示），使得ecall\_free\_obj1中的悬空指针（对象1指针）被ecall\_malloc\_obj2\_use\_obj1使用，触发UAF。程序执行结果如图5.10所示，ecall\_malloc\_obj2\_use\_obj1函数中，对象2被分配在与悬空指针（对象1指针）相同的地址，使用悬空指针（对象1指针）所指内容等同于使用对象2的值“uaf”。



**图5.9**　恶意篡改并发调用

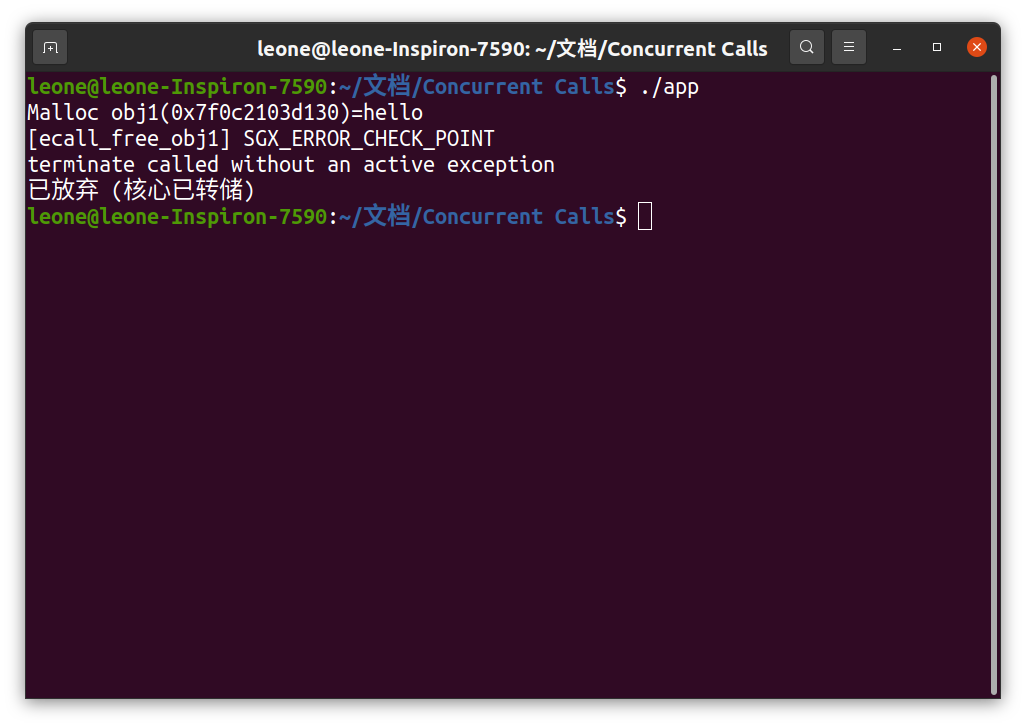
**Figure5.9**　Maliciously tampers with the concurrent calls



**图5.10**　并发调用攻击之后程序执行结果

**Figure5.10**　Program execution result after Concurrent Calls Attack

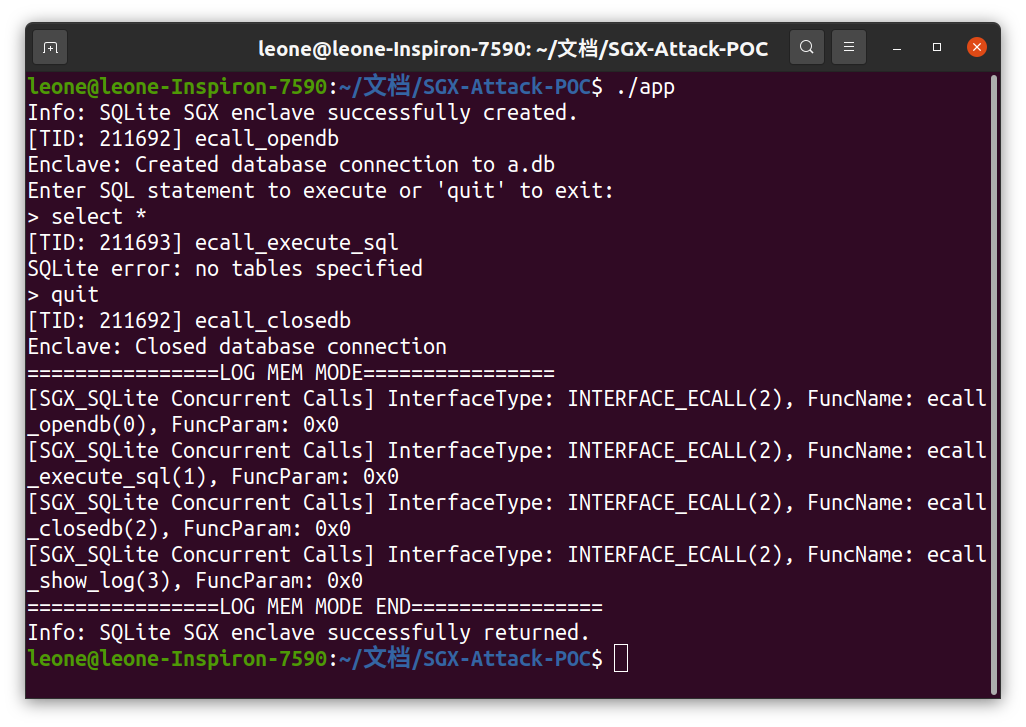
对此，Enclave开发者依据先验的预期执行情况编写策略文件，基于SGX-SEF设置“ECALL调用顺序白名单”安全策略，要求检测到的ECALL调用顺序满足ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数的先后顺序，否则SGX-SEF返回错误码SGX\_ERROR\_CHECK\_POINT并终止代码运行。如图5.11所示，在SGX-SEF中设置“ECALL调用顺序白名单”策略能成功检测非预期的调用顺序，核心已转储的原因是主线程调用ecall\_free\_obj1不符合安全策略被终止，线程1随主线程的终止而终止并发出核心已转储等信息。



**图5.11**并发调用攻击检测结果

**Figure5.11**　Detection result of Concurrent Calls Attack

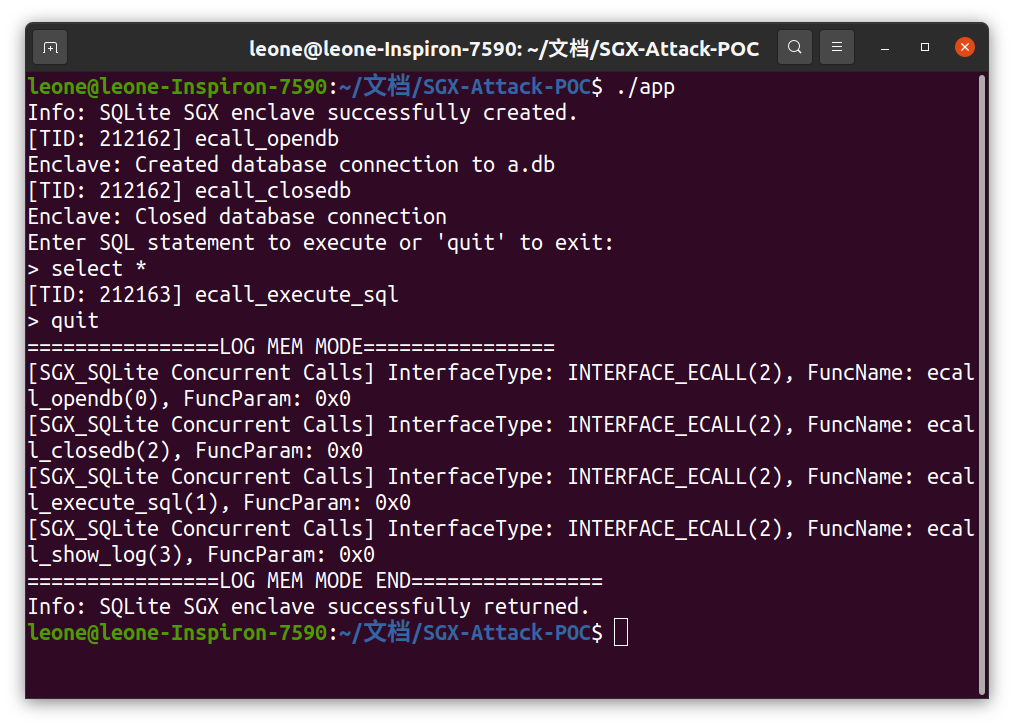
SGX\_SQLite中的悬空指针的情况，也能被“并发调用”攻击。攻击前SGX\_SQLite程序执行情况如图5.12所示，即主线程打开数据库句柄（ecall\_opendb），线程1使用数据库句柄执行SQL语句（ecall\_execute\_sql），主线程关闭数据库句柄（ecall\_closedb）。



**图5.12**　并发调用攻击前SGX\_SQLite执行情况图

**Figure5.12**　Diagram of SGX\_SQLite execution before Concurrent Calls attack

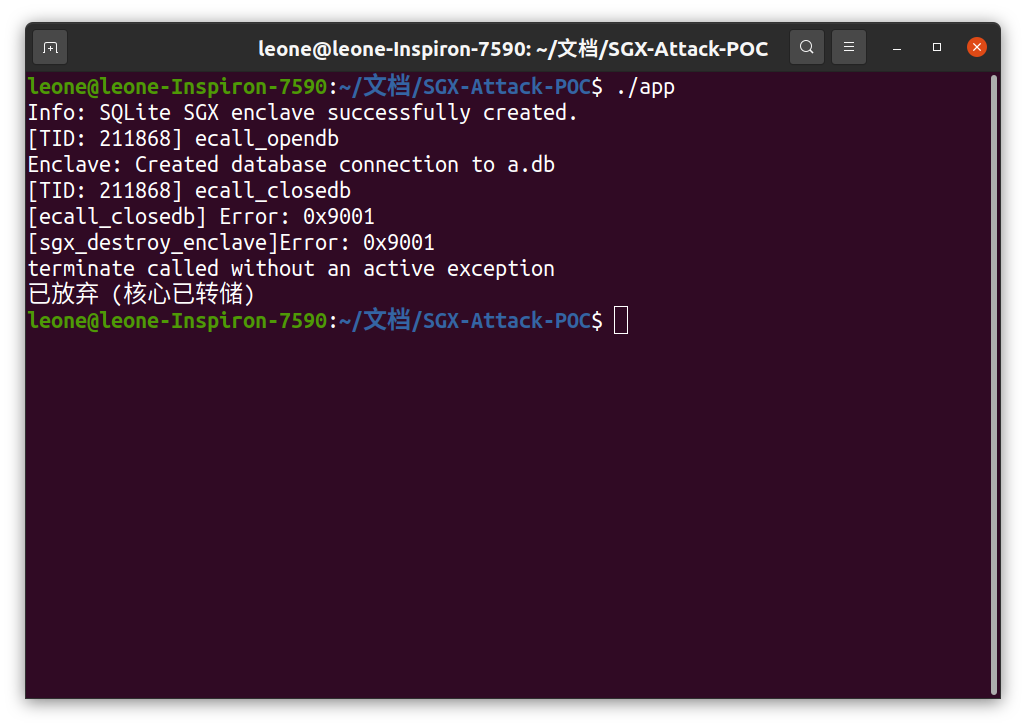
恶意拖延线程1执行SQL语句的时机，使得主线程中数据库句柄关闭（ecall\_closedb）后未置NULL形成的悬空指针被继续使用（ecall\_execute\_sql），形成安全隐患。程序执行情况如图5.13所示。



**图5.13**　并发调用攻击后SGX\_SQLite执行情况图

**Figure5.13**　Diagram of SGX\_SQLite execution after Concurrent Calls attack

为了缓解“并发调用”攻击，Enclave开发者依据先验的预期执行情况编写策略文件，在SGX-SEF上设置“ECALL调用顺序白名单”策略。如果执行情况违反策略，SGX-SEF将返回错误码SGX\_ERROR\_CHECK\_POINT（0x9001）并终止程序运行。如图5.14所示，在SGX-SEF中设置“ECALL调用顺序白名单”策略能成功检测非预期的调用顺序，核心已转储的原因是主线程调用ecall\_close\_db时不符合安全策略被终止，线程1随主线程的终止而终止并发出核心已转储等信息。



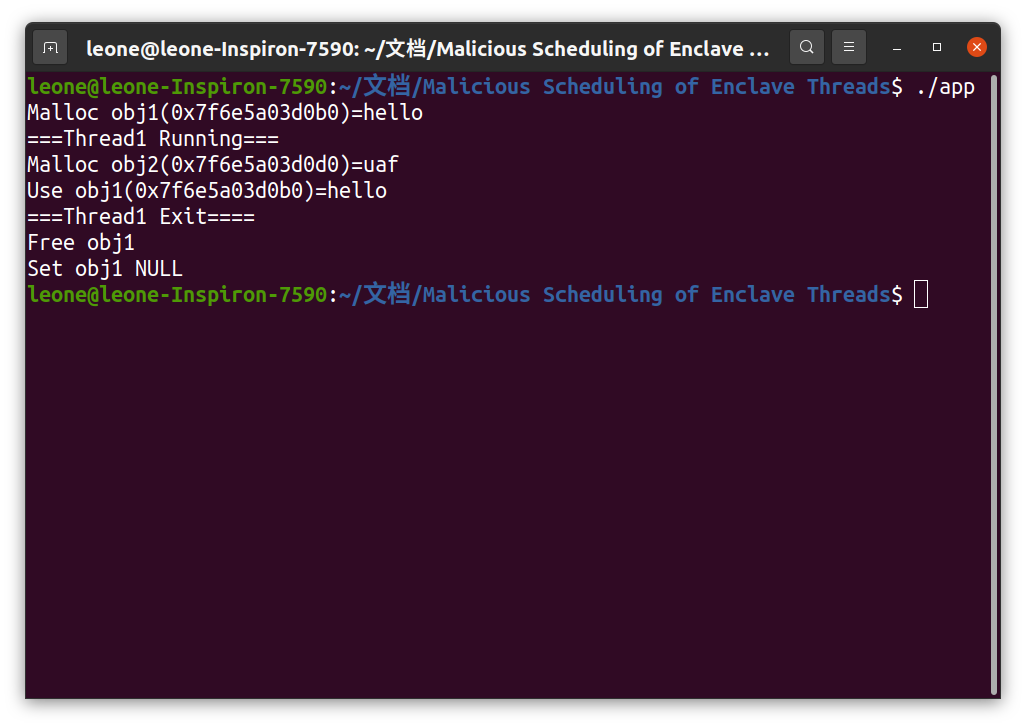
**图5.14**　SGX\_SQLite中并发调用攻击检测结果

**Figure5.14**　Detection results of Concurrent Calls Attack in SGX\_SQLite

目前尚未发现其它针对SGX并发调用攻击向量的防御方案。

### 5.3.3　针对“恶意Enclave线程调度”攻击向量的防御效果验证

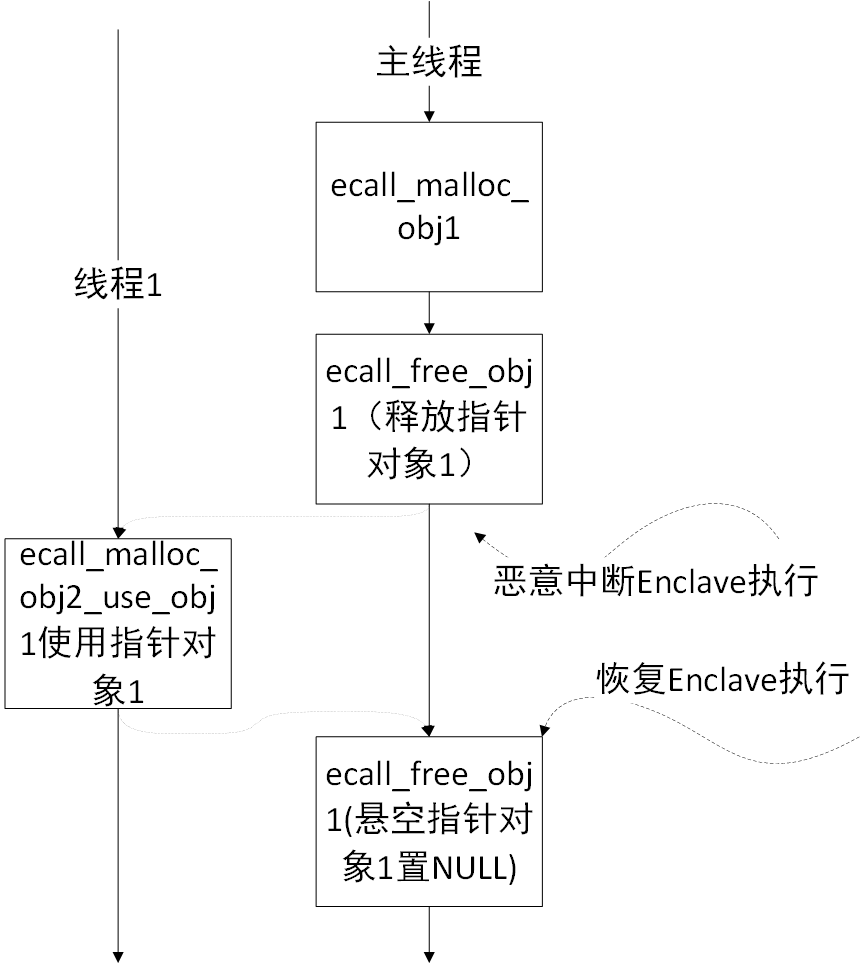
利用“线程暂停原语”恶意调度Enclave线程，构造实现“恶意Enclave线程调度”攻击向量POC。攻击之前，ECALL调用顺序为ecall\_malloc\_obj1函数、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数(悬空指针已通过置NULL修复)。程序执行结果如图5.15所示，主线程调用ecall\_malloc\_obj1函数在Enclave堆上创建值为“hello”的对象1并等待线程1执行，线程1调用ecall\_malloc\_obj2\_use\_obj1函数在Enclave堆上创建值为“uaf”的对象2并使用值为“hello”的对象1，线程1结束后主线程将对象1释放并置NULL。



**图5.15**　恶意线程调度攻击前程序执行结果

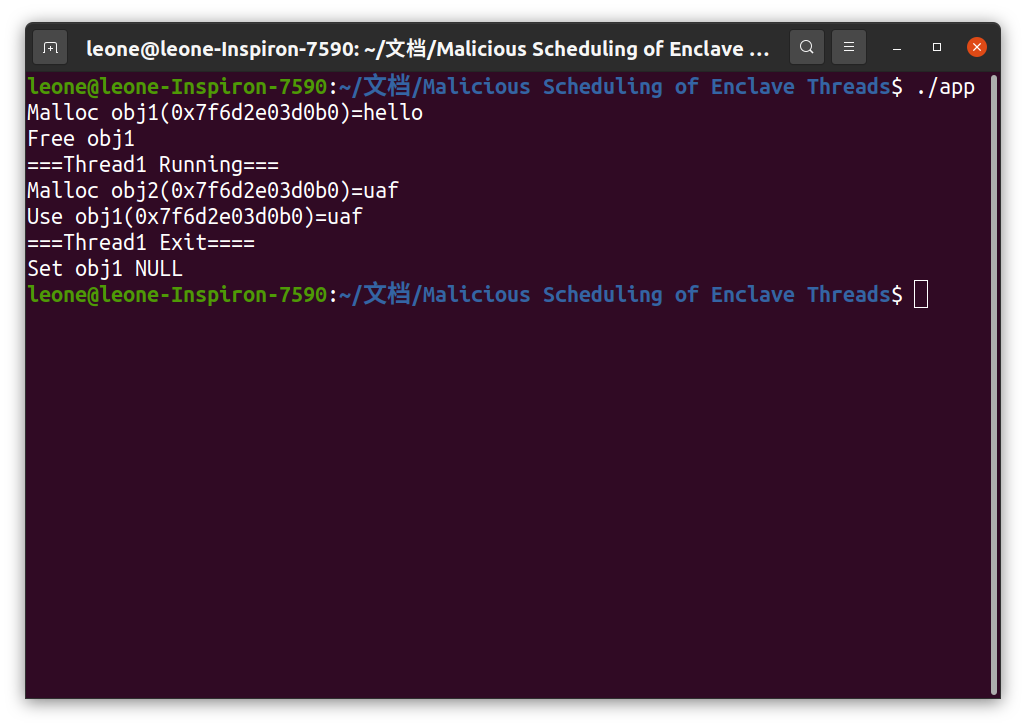
**Figure5.15**　Program execution result before Malicious Scheduling of Threads Attack

通过“线程暂停原语”将线程1的执行流停在ecall\_free\_obj1中释放指针后以及设置指针NULL前，形成ecall\_malloc\_obj1函数、ecall\_free\_obj1函数（释放对象1指针，指针成为悬空指针）、ecall\_malloc\_obj2\_use\_obj1函数和ecall\_free\_obj1函数(对象1指针置NULL)的调用顺序（如图5.16所示），使得ecall\_free\_obj1的悬空指针（对象1指针）置NULL之前被ecall\_malloc\_obj2\_use\_obj1使用，触发UAF。程序执行结果如图5.17所示，ecall\_free\_obj1函数中的释放过程和置NULL过程被中断，ecall\_malloc\_obj2\_use\_obj1函数中，对象2被分配了在与悬空指针（对象1指针）相同的地址，使用悬空指针（对象1指针）所指内容等同于使用对象2的值“uaf”。



**图5.16**　恶意中断ecall\_free\_obj1函数

**Figure5.16**　Malicious interruption of ecall\_free\_obj1 function

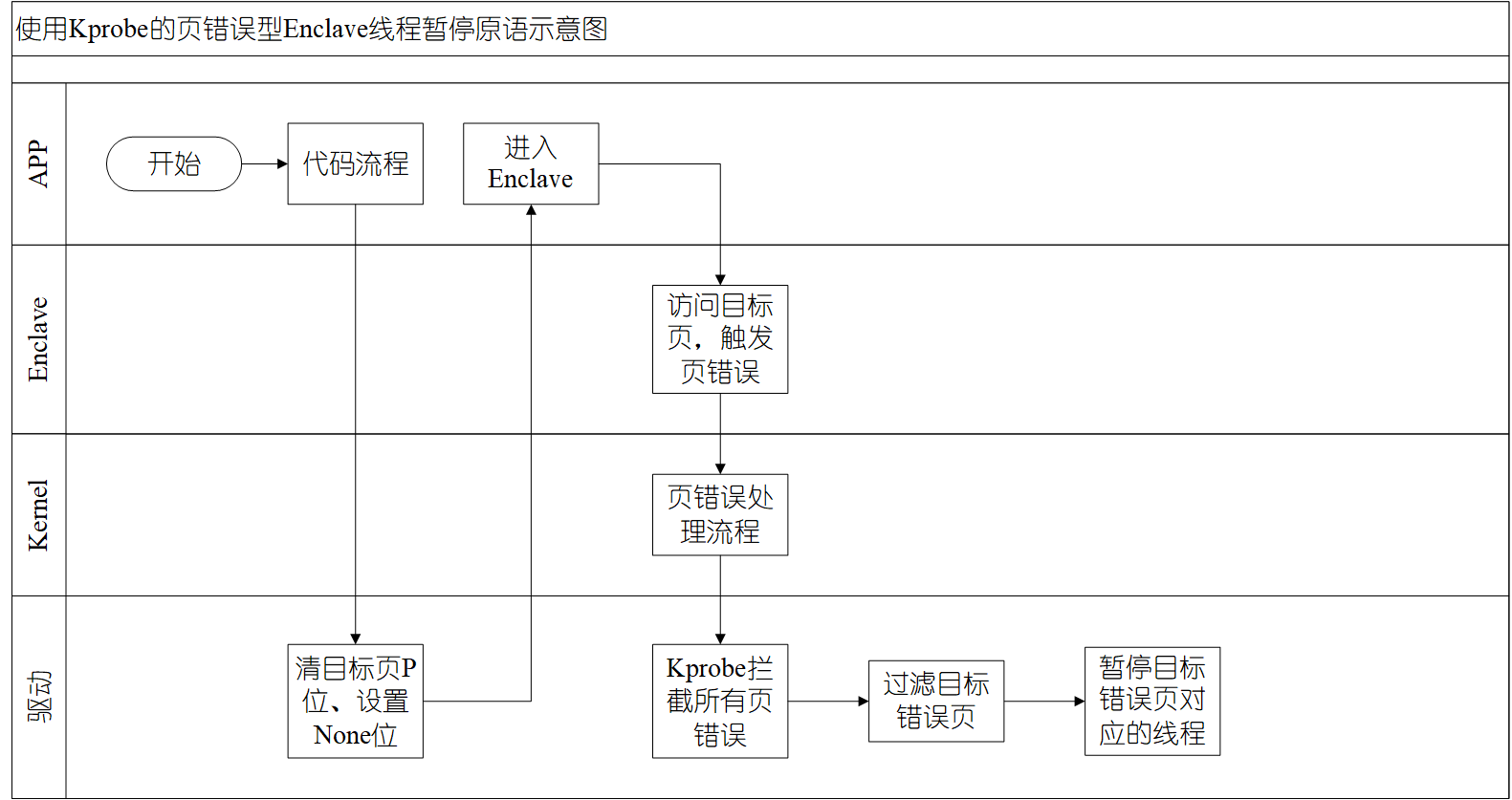


**图5.17**　恶意线程调度攻击后程序执行结果

**Figure5.17**　Program execution results after Malicious Scheduling of Threads Attack

实验中使用的“Enclave线程暂停原语”包括：

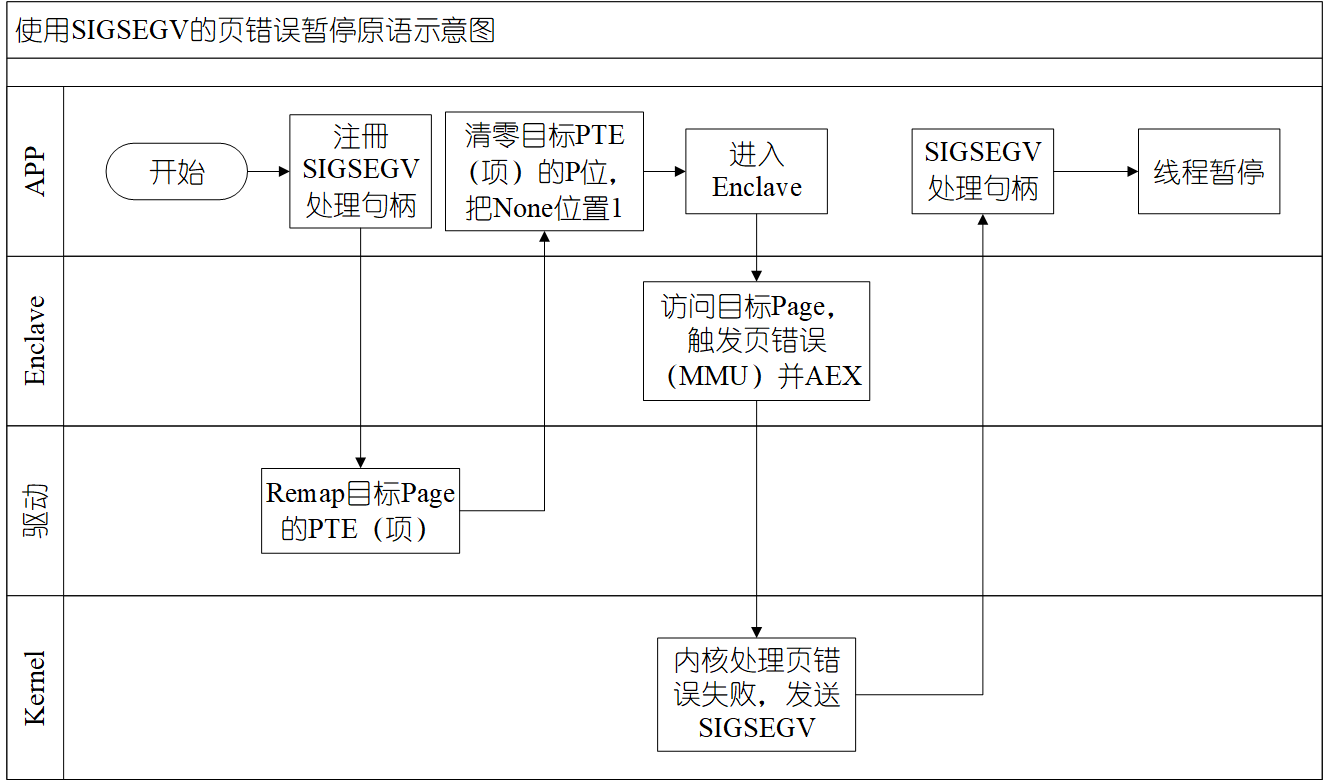
1. 使用Kprobe的页错误型Enclave线程暂停原语。如图5.18所示，攻击者清理目标页存在（Present）位，利用Kprobe拦截所有页错误并从中过滤出目标页错误，最后暂停目标错误页所对应的线程。



**图5.18**　使用Kprobe的页错误型Enclave线程暂停原语示意图

**Figure5.18**　Diagram of Enclave thread suspension primitive using page fault and Kprobe

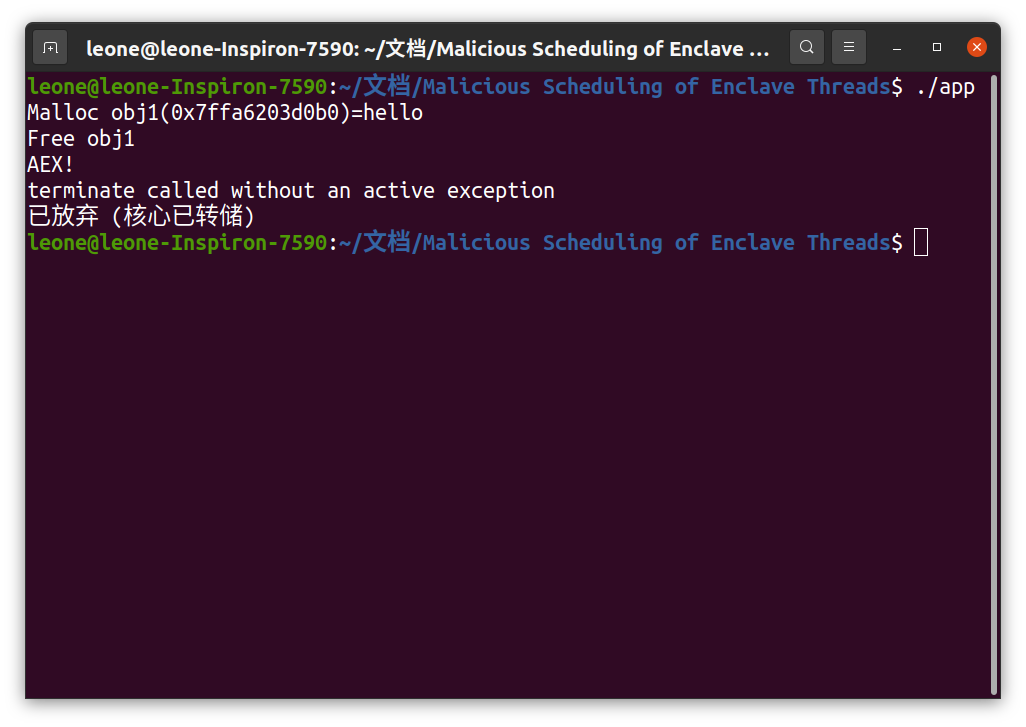
1. 使用SIGSEGV的页错误型Enclave线程暂停原语。如图5.19所示，攻击者预先清理目标页Present位，受害者Enclave线程访问此页时会触发页错误并进入攻击者事先注册的SIGSEGV处理句柄，受害者Enclave在SIGSEGV处理句柄中被暂停执行。由于SGX软件栈中的uRTS会直接报告Enclave处于崩溃状态，因此实现该线程暂停原语时需要修改uRTS端代码。



**图5.19**　使用SIGSEGV的页错误型Enclave线程暂停原语示意图

**Figure5.19**　Diagram of Enclave thread suspend primitive using page fault and SIGSEGV

对此，Enclave开发者编写策略文件，基于SGX-SEF设置“检测到AEX就报警”安全策略，要求程序执行过程中不能出现“Enclave线程暂停原语”会用到的AEX，否则SGX-SEF返回错误码SGX\_ERROR\_CHECK\_POINT并终止代码运行。检测结果如图5.20所示。



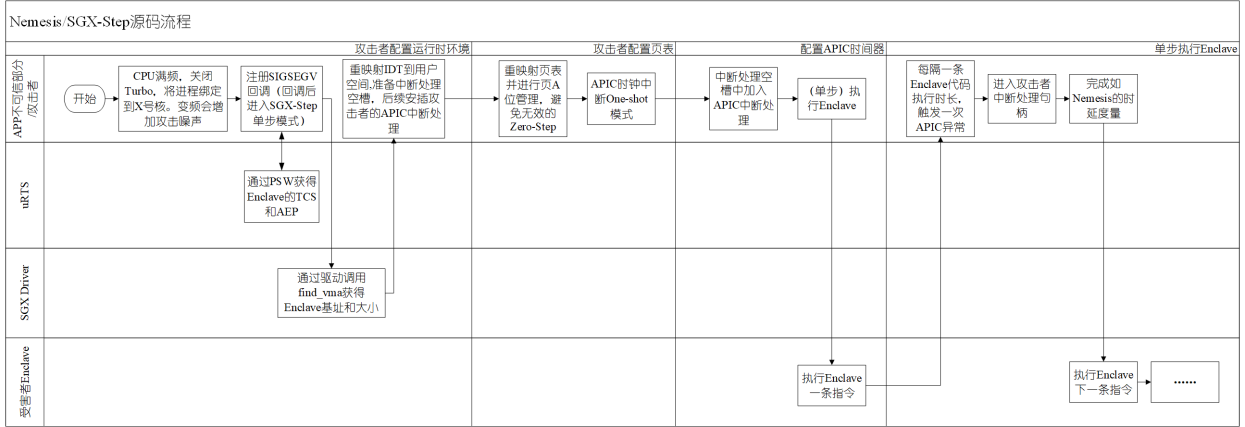
**图5.20**　恶意线程调度攻击检测结果

**Figure5.20**　Detection results of Malicious Scheduling of Threads Attack

但“检测到AEX就报警”策略过于简单，容易导致误报，需要进一步研究更合理的安全策略及策略匹配算法。目前尚未发现其它针对SGX恶意线程调度攻击向量的防御方案。

### 5.3.4　针对“时间侧信道”的防御效果验证

本文分析并复现了SGX-Step[43]/Nemesis攻击（使用时间侧信道攻击向量），其源码流程如图5.21所示。攻击者会频繁使用APIC时钟中断（时钟中断间隔为五十多个CPU周期，接近一条二进制代码执行时长），以中断Enclave代码执行，在AEX处理过程中计算一条二进制代码执行时延。Nemesis进一步利用该时延构建时延侧信道。



**图5.21**　SGX-Step/Nemesis源码分析流程图

**Figure5.****21**　Flowchart of SGX-Step/Nemesis

基于SGX-SEF设置“AEX超过10次即为攻击”安全策略。实验中，SGX-SEF能够成功检测该攻击，返回错误码并终止程序运行。由于所使用的攻击POC代码量较少，因此使用较小的阈值，实际应用中可以参考Varys[44]设置约5000次/秒的阈值来检测攻击。

## 5.4　本章小结

本章描述了实验环境。展示了审计功能的效果，评估了审计功能性能开销增幅在最优情况下为13%~18%，分析了引起性能开销的原因。针对调用排序、并发调用、恶意线程调度和时间侧信道攻击向量制定了安全策略验证SGX-SEF防御（检查功能）效果，指出SGX-SEF具有良好的防御效果，能灵活地兼容多种安全准则。

SGX关键执行路径及安全关键点分析启发“调用顺序白名单”安全策略以抵御调用排序和并发调用攻击向量，此前尚无防御方案能够有效防御这两类攻击向量。

# 第六章　总结与展望

## 6.1　总结

现有SGX攻击针对Enclave开发中的安全问题和SGX设计存在安全缺陷，透过Enclave接口及共享资源等攻击面，构建各类攻击向量，威胁SGX安全性。现有防御方案安全准则不统一，尚无防御方案分析SGX安全关键点并从SGX架构自身进行安全增强。本文提出了针对Intel SGX SDK的安全增强框架（简称SGX-SEF），旨在从SGX架构自身出发，全面加固SGX架构中的安全关键点，主要成果包括：

1. 产出SGX关键执行路径及安全关键点分析结果。本文以SGX软件栈为主刻画了SGX架构及关键执行路径，总结了其中的安全关键点，指导了对已有防御方案安全准则的兼容，启发了新的安全准则。
2. 设计实现了针对Intel SGX SDK的安全增强框架（SGX-SEF）。SGX-SEF审计了SGX关键执行路径，记录了上下文切换等安全事件，提供了函数名及参数等信息用于检测恶意行为，加固了安全关键点，将安全框架与安全策略分离，依据多种安全策略防御多种攻击向量，解决现有SGX架构难以有效兼容安全准则不统一的防御方案的问题。
3. 提出了“调用顺序白名单”安全策略。针对调用排序和并发调用攻击向量会恶意篡改调用逻辑违背开发者预期的情况，Enclave开发者能够基于SGX-SEF在SGX关键执行路径的安全关键点中插桩审计检查代码并部署“调用顺序白名单”安全策略以要求调用情况符合预期，从而抵御调用排序和并发调用攻击向量。针对“调用排序”和“并发调用”攻击向量的SGX-SEF防御效果验证中，依据预期执行情况设计的ECALL调用顺序白名单策略能很好地检测非预期的ECALL调用。调用顺序白名单策略还能进一步包含对OCALL调用的限定，使程序执行情况充分符合预期。
4. 针对Intel SGX SDK的安全增强框架（SGX-SEF）效果验证及性能评估。通过实验验证了SGX-SEF能够有效审计SGX架构的关键执行路径。评估了审计功能性能开销，分析了引起性能开销的原因。审计功能开销随日志容量增大而增大，开销包括日志使用维护开销及缓存未命中开销。当日志大小100项时，审计功能对各种调用方式引起的开销增幅不明显，100项大小的日志足够记录最近的安全事件用于审计检查功能。日志容量为100项时，开销增幅在最优情况（Ordinary调用方式）下为13%~18%。通过调用排序、并发调用、恶意线程调度和时间侧信道攻击向量这四个例子实验验证了防御者能在SGX-SEF中灵活部署多种安全策略以抵御多类攻击向量。

## 6.2　展望

结合当前可信执行环境的发展趋势，在本文工作基础上，还有以下几个研究点值得进一步探索。

1. 分析内核可信部分并联合SGX-SEF形成综合性安全增强框架。SGX设计中仅保护用户态程序敏感部分，将内核等模块排除在TEE之外并认为其不可信。SGX Enclave只具备用户态能力，为保证其能正常运行，Enclave严重依赖于内核提供的进程机制和段页表机制等。许多SGX攻击正是利用Enclave对内核的依赖发起攻击。在实际应用中完全假设内核不可信可能过于严苛，后续研究可深入分析内核可信部分和不可信部分，通过可信度量及挑战应答等手段从Enclave出发扩张可信域，更好地权衡可信域的TCB尺寸和安全性。当假设内核的部分模块可信时，可进一步将LSM为例的内核安全增强模块和SGX-SEF结合，实现综合性安全增强框架，保护Enclave安全。
2. 针对其他TEE技术设计安全增强框架。除Intel SGX外的其他TEE技术如ARM TrustZone也存在着可信环境功能依赖不可信环境及可信环境接口消毒不充分等问题。对此，后续研究可以借鉴本文的方案，深入刻画其它TEE架构并分析关键执行路径和安全关键点，设计实现对应的安全增强框架，以审计关键执行路径，加固安全关键点，全面提升安全性。此外，也非常需要进一步研究如何针对具体应用场景更好地权衡TEE技术的TCB尺寸、功能性及安全性等。

# 参考文献

1. Costan V, Devadas S. Intel SGX Explained[J]. IACR Cryptol. ePrint Arch., 2016, 2016(86): 1-118.
2. 董春涛,沈晴霓,罗武,吴鹏飞,吴中海.SGX应用支持技术研究进展[J].软件学报,2021,32(01):137-166.
3. Intel Corporation. Intel® Software Guard Extensions (Intel® SGX) Developer Guide, 2020. Revision Number 2.10.
4. Intel Corporation. Intel® Software Guard Extensions (Intel® SGX) SDK for Linux\* OS Developer Reference, 2020. Revision Number 2.10.
5. Intel Corporation. Intel64 and IA32 Architectures Software Developers Manual, 2020. Reference no. 325384-072US.
6. Jang I, Tang A, Kim T, et al. Heterogeneous isolated execution for commodity gpus[C]//Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 455-468.
7. Volos S, Vaswani K, Bruno R. Graviton: Trusted execution environments on gpus[C]//13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). 2018: 681-696.
8. Criswell J, Dautenhahn N, Adve V. Virtual ghost: Protecting applications from hostile operating systems[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 81-96.
9. Gueron S. A memory encryption engine suitable for general purpose processors[J]. IACR Cryptol. ePrint Arch., 2016, 2016: 204.
10. 王鹃, 樊成阳, 程越强, 等. SGX 技术的分析和研究[J]. Journal of Software, 2018, 9: 2778-2798.
11. Kim T. Security Issues on Intel SGX. ACM CCS 2017. https://www.youtube.com/watch?v=0TXR2JNsFBk&t=4s
12. Checkoway S, Shacham H. Iago attacks: why the system call API is a bad untrusted RPC interface[C]//Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems. 2013: 253-264.
13. Lee S, Kim T. Leaking uninitialized secure enclave memory via structure padding[J]. arXiv preprint arXiv:1710.09061, 2017.
14. Van Bulck J, Oswald D, Marin E, et al. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 1741-1758.
15. Khandaker M R, Cheng Y, Wang Z, et al. COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 971-985.
16. Wang J, Cheng Y, Li Q, et al. Interface-based side channel attack against intel SGX[J]. arXiv preprint arXiv:1811.05378, 2018.
17. Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 991–1008.
18. Chen G, Chen S, Xiao Y, et al. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution[C]//2019 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2019: 142-157.
19. Lee S, Shih M W, Gera P, et al. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing[C]//26th {USENIX} Security Symposium ({USENIX} Security 17). 2017: 557-574.
20. Xu Y, Cui W, Peinado M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 640-656.
21. Van Bulck J, Weichbrodt N, Kapitza R, et al. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution[C]//26th {USENIX} Security Symposium ({USENIX} Security 17). 2017: 1041-1056.
22. Weichbrodt N, Kurmus A, Pietzuch P, et al. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves[C]//European Symposium on Research in Computer Security. Springer, Cham, 2016: 440-457.
23. Sanchez Vicarte J R, Schreiber B, Paccagnella R, et al. Game of threads: Enabling asynchronous poisoning attacks[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 35-52.
24. Van Bulck J, Piessens F, Strackx R. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 178-195.
25. Van Schaik S, Milburn A, Österlund S, et al. RIDL: Rogue in-flight data load[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 88-105.
26. Baumann A, Peinado M, Hunt G. Shielding applications from an untrusted cloud with Haven[C]//Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation. 2014: 267-283.
27. 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). 2017: 645-658.
28. Arnautov S, Trach B, Gregor F, et al. {SCONE}: Secure linux containers with intel {SGX}[C]//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016: 689-703.
29. Hunt T, Zhu Z, Xu Y, et al. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data[C]//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016: 533-549
30. Lind J, Priebe C, Muthukumaran D, et al. Glamdring: Automatic application partitioning for intel {SGX}[C]//2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17). 2017: 285-298.
31. Shih M W, Lee S, Kim T, et al. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs[C]//NDSS. 2017.
32. Chen G, Wang W, Chen T, et al. Racing in hyperspace: Closing hyper-threading side channels on sgx with contrived data races[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 178-194.
33. Ahmad A, Joe B, Xiao Y, et al. OBFUSCURO: A Commodity Obfuscation Engine on Intel SGX[C]//NDSS. 2019.
34. 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. 2017: 238-253.
35. Orenbach M, Michalevsky Y, Fetzer C, et al. CoSMIX: a compiler-based system for secure memory instrumentation and execution in enclaves[C]//2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19). 2019: 555-570.
36. Orenbach M, Baumann A, Silberstein M. Autarky: closing controlled channels with self-paging enclaves[C]//Proceedings of the Fifteenth European Conference on Computer Systems. 2020: 1-16.
37. Gruss D, Lettner J, Schuster F, et al. Strong and efficient cache side-channel protection using hardware transactional memory[C]//26th {USENIX} Security Symposium ({USENIX} Security 17). 2017: 217-233.
38. Matetic S, Ahmed M, Kostiainen K, et al. {ROTE}: Rollback protection for trusted execution[C]//26th {USENIX} Security Symposium ({USENIX} Security 17). 2017: 1289-1306.
39. Morris J, Smalley S, Kroah-Hartman G. Linux security modules: General security support for the linux kernel[C]//USENIX Security Symposium. ACM Berkeley, CA, 2002: 17-31.
40. Smalley S, Vance C, Salamon W. Implementing SELinux as a Linux security module[J]. NAI Labs Report, 2001, 1(43): 139.
41. Tian H, Zhang Y, Xing C, et al. Sgxkernel: A library operating system optimized for intel SGX[C]//Proceedings of the Computing Frontiers Conference. 2017: 35-44.
42. Tian H, Zhang Q, Yan S, et al. Switchless calls made practical in intel SGX[C]//Proceedings of the 3rd Workshop on System Software for Trusted Execution. 2018: 22-27.
43. Van Bulck J, Piessens F, Strackx R. SGX-Step: A practical attack framework for precise enclave execution control[C]//Proceedings of the 2nd Workshop on System Software for Trusted Execution. 2017: 1-6.
44. Oleksenko O, Trach B, Krahn R, et al. Varys: Protecting {SGX} enclaves from practical side-channel attacks[C]//2018 {Usenix} Annual Technical Conference ({USENIX}{ATC} 18). 2018: 227-240.

# 致　谢

在中科院软件所基础软件实验室的三年硕士研究生生涯即将结束。大量调研经历使我对研究工作有了更深入的理解，研究过程中与他人的沟通合作锻炼了我的表达能力和合作能力，端正了我的科研态度。在此由衷感谢各位老师和同学的帮助。

首先非常感谢我的导师马恒太副研究员。马老师在我的科研和生活中提供了许多帮助。他丰富的阅历及严谨的治学风格对我具有很大的启发作用，指引我在科研路上脚踏实地地不断探索前行。对我的毕业设计提出了许多宝贵意见，帮助我改进毕业设计。

非常感谢周启明老师的悉心指导。周老师也极大地助力我的科研和生活。在我的科研方向和科研历程上提供了全面帮助。在毕业设计期间帮助我研究分析课题的框架和细节，在工程实践上提供积极的指导，不断帮助我完善论文。

感谢研究员贺也平老师。他教会我如何研究一门学问，如何在科研道路上发现问题并解决问题。他还经常指出我所陷入的思维困境，帮助我脱离困境继续研究。

感谢实验室的其他老师。他们对我的毕业设计提出了许多建设性的意见，帮助我更好地完成毕业设计。

感谢吴晓慧学姐、林少峰学长和霍天霖学长。他们帮助我不断推敲研究课题，让研究课题更具意义。与我不断讨论实现细节，确定下一步实施计划。

感谢丁羽。他对Intel SGX及其它安全相关领域都有长足的研究，并且不厌其烦地教会我许多关于SGX的细节知识，帮助我确认工程实践中的许多实现细节。

感谢家人和朋友的支持。他们在我硕士期间遇到各类学术及生活问题时悉心地照顾我，帮助我缓解心理压力并摆脱困境，替我解决了许多烦恼。

最后感谢参与论文评审答辩工作的各位老师，您辛苦了！

# 作者简历及攻读学位期间发表的学术论文与研究成果

**作者简历**

2014年9月——2018年7月，在北京交通大学计算机与信息技术学院获得学士学位。

2018年9月——2021年7月，在中国科学院大学软件研究所攻读硕士学位。

**已发表的学术论文**

1. Xiaohui W, Yeping H, Qiming Z, Hengtai M, Liang H, Wenhao W, Liheng C. Partial-SMT: Core-scheduling Protection Against SMT Contention-based Attacks[C]//2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 2020: 378-385.

**参加的研究项目**

1. 核高基课题：“开源操作系统内核分析和安全评估”（项目编号：2012ZX01039-004）
2. 中科院先导A课题：“关键软硬件系统安全性分析评估核心技术研发及验证”（项目编号：XDA-Y01-01）