西安電子科技大學
XIDIAN UNIVERSITY

Web 安全
课 程 作 业

作业内容：	小组作业 3
小组成员：	姜欣悦
	李颖
	盖乐
	职泉
	陈静怡
论文名称：	SymGX: Detecting Cross-boundary Pointer Vulnerabilities of SGX Applications via Static Symbolic Execution
作者：	Yuanpeng Wang Ziqi Zhang Ningyu He Zhineng Zhong Shengjian Guo Qinkun Bao Ding Li Yao Guo Xiangqun Chen
其他信息：	2023 CCS

一、论文阅读

1.1 摘要

英特尔安全防护扩展（Security Guard Extensions，简称 SGX）在关键数据保护方面显示出了有效性。最近基于符号执行的技术揭示了 SGX 应用程序容易受到内存损坏漏洞的影响。虽然现有方法专注于 SGX 应用程序中的常规内存损坏，但它们忽视了一种重要的 SGX 专用漏洞：跨边界指针漏洞。对于 SGX 应用程序来说，这种漏洞非常关键，因为它们在安全飞地和不受信任的环境之间广泛使用指针来交换数据。不幸的是，现有的符号执行方法无法有效地检测到跨边界指针漏洞，因为缺乏一个适当处理 SGX 应用程序三个独特特征的 SGX 特定分析模型：多入口任意顺序执行、有状态执行和上下文感知指针。为了解决这些问题，我们提出了一个名为“具有上下文感知指针的全局状态转换图（GSTG-CAP）”的新分析模型，该模型模拟了 SGX 应用程序的属性保持执行行为，并驱动符号执行进行漏洞检测。基于 GSTG-CAP，我们构建了一个名为 SymGX 的新颖符号执行漏洞检测器，用于检测跨边界指针漏洞。根据我们的评估，SymGX 在 14 个开源项目中发现了 30 个 0-DAY 漏洞，其中三个已被开发人员确认。在效果、效率和准确性方面，SymGX 也优于两个现有的最先进工具 COIN 和 TeeRex。

关键词：

Intel SGX；符号执行；漏洞检测

1.2 简介

现有的最先进的解决方案都不能有效地检测跨边界指针漏洞。主要原因是现有的符号执行技术试图将常规程序的执行和分析模型应用于 SGX 应用程序，而没有遵循 SGX 应用程序的执行模型。与常规程序相比，SGX 应用程序具有一种新的执行模型，具有三个独特的属性：

- 多入口任意顺序执行(P1)
- 有状态执行(P2)
- 上下文感知的指针(P3)

不幸的是，现有的方法不能正确处理这三个独特的属性，导致跨边界指针漏洞的检测无效和低效。

为了解决这些问题，提出了一种新的工具 SymGX 检测 SGX 应用程序中的跨境指针漏洞。

总之，我们在本文中做出以下贡献：

•我们开发了一种基于符号执行的新检测器 SymGX，用于解决跨境指针漏洞，这是 SGX 应用程序数据交换边界上出现的一个关键安全问题。

•我们为 SGX 应用程序构建了一个新的分析模型 GSTG-CAP，该模型可准确处理其三个独特特性：

多条目任意顺序执行、状态执行和上下文感知指针。

•我们对 SymGX 的开源项目进行了评估行业级项目中的已知 CVE。发现 SymGX30 个新的 0 天漏洞和 3 个已被开发人员确认。

1.3 背景和动机

1.3.1 软件保护扩展（SGX）

图 1 展示了编程模型 SGX 软件应用程序的一部分，它由两个世界组成：受保护内存区域内的受信任飞地和不受信任的主机上的环境。不受信任的环境无法直接访问包围区中的内存区域，从而确保将这块飞地与不受信任的世界隔离开来。

Intel SGX SDK[78]为飞地和不受信任的世界之间的数据交换提供了软件接口：enclave 呼叫（ECall）和外部呼叫（OCall）。这些接口允许不可信世界调用 ECalls 来与飞地通信，以及向不可信世界发布 OCalls 的飞地。

```
enclave {
    ...
    // define ECALLs
    trusted {
        public int ecall_create_wallet(
            [in, string]const char* master_password
        );
        public int ecall_show_wallet(
            [in, string]const char* master_password,
            [out, size=wallet_size] wallet_t* wallet,
            size_t wallet_size
        );
        ...
    };
};
```

Listing 1: A sample EDL file from sgx_wallet.

1.3.2 跨境指针漏洞

我们将跨界指针漏洞分为两类：直接跨界指针(1)和间接的交叉边界指针(2).前者(1）当 at tacker 从不受信任的世界传递指向飞地内的地址。后者(2）当攻击者间接操纵包围区内的指针以访问 SGX 保护的内存。

清单 2 展示了一个示例，这使对手能够写入飞地中的任何地址。

```
//Enclave.edl
// msg is a safe pointer, SGX SDK will automatically
// check msg and move it to enclaves. encrypted is an
// unsafe pointer (decorated by user_check]) that the
// adversary controls
void simple_encrypt([in, size=len] char *msg,
   [user_check] char* encrypted, unsigned len);

//Enclave.cpp
char book[10] = {52, 48, 55, 51, 56, 54, 50, 49, 57, 53};
//Attackers set encrypted to an enclave memory address.
void simple_encrypt(char *msg, char* encrypted,
        unsigned len) {
  if (!msg || !encrypted || len<=0 || len>10)
    return;

  for (unsigned i = 0; i < len; ++i) {
    if (msg[i]<48 || msg[i]>57)
        return;
    // if (!sgx_is_outside_enclave(encrypted + i,
    //  sizeof(char)))
    //       exit();
    // a possible fix with intrinsic functions
    encrypted[i] = book[msg[i]-48];
    // arbitrary write to addresses in enclaves
  }
}
```

Listing 2: The sample code of unprotected pointers (V_1).

清单 3 显示了的一个示例，使对手能够访问飞地内的机密数据。

```
//Enclave.edl
// cnt is an offset controlled by the attacker
void ecall_copy_information([out, count=cnt] int* ptr,
         unsigned cnt);

//Enclave.cpp
void ecall_copy_information(int *ptr, unsigned cnt) {
    int meta[8] = {0, 1, 2, 3, 4, 5, 6, 7};
    int secret[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    //Attacker set cnt > meta size to steal secret
    memcpy(ptr, meta, sizeof(int) * cnt);
}
```

Listing 3: The sample code of unprotected array indices (V_2).

1.4 本论文方法

本节介绍 SymGX 的核心组件。我们首先介绍威胁模型和 SymGX 试图解决的挑战。然后，我们解释了处理 SGX 应用程序执行属性的底层 GSTG-CAP 模型。接下来，我们将讨论 SymGX 如何遍历图并对数据流执行细粒度污染分析。最后，我们介绍 SymGX 如何识别 SGX 应用程序中的漏洞。

1.4.1 威胁模型和研究挑战

我们采用了与 TeeRex[18]相同的威胁模型，该模型假设飞地保护数据不受对手的攻击。我们还假设对手可以访问任意内存地址，并在不受信任的世界中执行任何 ECalls 组合。我们的方法侧重于 SGX 应用程序特有的跨境指针漏洞。我们不解决可能影响 SGX 应用程序的其他内存安全相关漏洞，因为它们与本工作的兴趣正交。基于威胁模型，我们旨在解决 SymGX 中的以下研究挑战：

- 挑战 1（C1）：如何对 SGX 应用程序的多条目任意顺序执行行为进行建模？
- 挑战 2（C2）：如何跟踪通过全球状态的欧洲法院间信息流？
- 挑战 3（C3）：如何在符号执行中处理上下文感知指针？

1.4.2 GSTG-CAP 模型

总的来说，我们通过构建一个名为“具有上下文感知指针的全局状态转换图”（GSTG-CAP）的分析模型来应对这些挑战。对于 C1，我们将 ECalls 表示为全局状态转换图（GSTG）中的边。这种设计使 SymGX 能够通过在图上重新启动的情况下执行随机遍历来生成不同的 ECall 序列，模拟多条目和任意顺序的 ECall 执行。对于 C2，我们让 GSTG 中的每个节点充当唯一的 SGX 程序全局状态。因此，当在 GSTG 上随机行走时，对相关节点的更新自然地跟踪通过全局状态的 ECall 间信息流。对于 C3，我们在符号执行器内部开发了一个边界感知内存模型（BAMM），以在 ECall 序列执行期间处理上下文感知指针并执行漏洞检测。

1.4.3 SymGX 的体系结构

我们基于 GSTG-CAP 模型开发了一个新的符号执行器来检测跨界指针漏洞，如图 3 所示。SymGX 接收 SGX 应用程序的源代码，并使用 GSTG-CAP 分析模型输出漏洞报告。它构建了一个 GSTG 来对 ECall 交互进行建模，并将执行模拟为在 GSTG 上重新启动的随机行走。我们的 SymGX 框架使用 SECAP 来分析 GSTG 中节点之间的转换。

1.4.4 代码覆盖引导的图随机漫步

使用 Restart，SymGX 通过从 GSTG-CAP 中的初始状态进行随机漫步来开始其分析。纯随机方法效率低下的原因有两个：

（1）它可能探索 GSTG 中的有限路径；

（2）GSTG 可能具有无限多的节点。

为了解决第一个问题，我们分配了一个重新启动的概率随机行走。因此，在每一个步骤中，SymGX 都有可能返回到初始状态。

对于第二个问题，我们设计了一个代码覆盖引导算法来提高 SymGX 的效率。算法 1 提出了代码覆盖引导的随机游动算法。

Algorithm 1: Code coverage Guided Random Walk with Restart

Input : $\mathbb{G}_{state}$
Output: ECall Sequences

```
1  δ0.path = {δ0}
2  prio_queue = init(A_r, δ0)
3  visited = ∅
4  while prio_queue ≠ ∅ do
5      current=prio_queue.pop()
6      visited = visited ∪ {current}
7      yield current.path
8      for each ECall e ∈ E do
9          next=e(current)
10         next.path=current.path ∪ {next}
11         if not A_p(next) then
12             prio_queue.push(next)
13         end
14     end
15     Restart with probability p^r
16 end
17 return G_state
```

1.4.5 细粒度污染数据流分析器

此组件区分 V1 和 V2 漏洞。我们做出这一区分的依据是 V1 带来的风险比 V2 更多。V1 使攻击者能够操纵基指针，从而导致飞地中的任意读/写。相反 V2 只允许攻击者控制的索引。只要没有缓冲区溢出，V2 的损坏是有限的。通过区分 V1 和 V2，我们可以通过推理数组索引是否会导致溢出来减少误报 V2。

为了进行细粒度分析，我们为每个变量分配一组属性，以捕获不同的属性。特别地，我们设计了三个属性：污点、指针和 base_taint，如表 1 所示。

1.4.6 漏洞检测

对于 SGX 程序中的每个内存操作,我们定义了两个检查规则来检测 V1 和 V2。算法 3 给出了我们用于识别漏洞的详细算法。

SymGX 识别 V1，通过在算法 3 中应用两个条件：（1）指针源自不可信世界（第 1-2 行）和（2）指针指向包围区内的地址（第 3 行）。

为了检测漏洞 V2，SymGX 应用两个条件：（1）偏移量来自不可信的世界（第 1 行）和（2）地址超出缓冲区大小（第 6 行）。

1.5 评估

问题 1：SymGX 的漏洞检测能力有多强?——4.3
问题 2：SymGX 的漏洞报告有多精确?
问题 3：SymGX 能否实现比基线更高的代码覆盖率?
问题 4：GSTG-CAP 在没有代码覆盖率引导的随机行走和重启的情况下表现如何?
问题 5：SymGX 的运行内存消耗量是多少?

1.5.1 基准

构建一个包含 27 个表现出跨境指针漏洞的应用程序的基准集，包括开源 GitHub 应用程序和现实世界的 CVE。

1.5.2 实施与实验设置

基于 SGX 函数的软件模拟创建 SymGX，与 TeeRex 和 COIN 这两个最先进的工具进行了比较。

1.5.3 有效性

比较了 SymGX 与基准工具 TeeRex 和 COIN 的漏洞检测性能：

(1)0-DAY 漏洞搜索。将 SymGX 应用于 14 个开源 GitHub 项目并搜索 0-DAY 漏洞

(2) 已知漏洞验证。测量 SymGX 检测 13 个真实 CVE 所需的时间

SymGX 结合 Ar 和 Ap 发现了 30(13+17=30)个漏洞，比基线的综合结果(TeeRex 发现 10 个，COIN 发现 6 个)要多，这证明了它的有效性。

问题 1 回答：在有限的时间内，SymGX 比最先进的基线能检测到更多的漏洞。

1.5.4 精确度

SymGX 比现有工具更好地平衡了精确度和召回率。

误报的三个来源：

(1)SMT 解算器的结果不准确，SymGX 探索无法到达的分支；

(2)SymGX 在编码未仿真的函数时，返回一个无约束符号；

(3)污点和指针分析错误

可以通过模拟更多的函数和提高静态分析的准确性来减少误报。

V1 的两个条件的有效性：

(1)检测对手是否提供了地址基指针；

(2)检测指针是否指向飞地内部的地址（必要的）

定量研究 V1 的第二个条件的有效性发现,V1 的第二个条件有效地减少了假阳性。

问题 2 回答：SymGX 的精确度达到 11.8%,比最先进的 TeeRex 高出近 59 倍。此外是，SymGX 发现的漏洞比 COIN 多 5 倍，尽管它的精度较低。

1.5.5 覆盖范围

ECall 序列生成算法和优化（修剪和排序方法）使 SymGX 的平均覆盖率在所有应用程序中始终优于其他两个基线

问题 3 回答：SymGX 更有效的因素之一是其更高的代码覆盖率。与 TeeRex 和 COIN 相比，SymGX 的代码覆盖率平均分别高出 12%和 20%。

1.5.6 案例研究

SymGX 如何检测现实中的跨界指针漏洞

代码中有两个 ECall: add_to_store 和 get_from_store，在 get_from_store 未验证 out_len 的代码中存在 V_2 漏洞。

SymGX 检测漏洞的过程

1.5.7 消融研究

评估 GSTG-CAP 以及 Ar 和 Ap 这两个优化函数的有效性，比较在使用和不使用这些优化功能时的性能。结果证明了内存模型的有效性；Ar 和 Ap 可将覆盖率提高 8.60%。

Ap 可以跳过重访相同代码块的 ECall 序列集，从而节省资源，而 Ar 可以专注于探索更多新代码块的更重要 ECall 序列。因此推断优化函数 Ar 和 Ap 可以提高 SymGX 的效率。

问题 4 回答：只有在使用 GSTG-CAP 时，SymGX 才能实现优于基线的性能。应用 Ar 和 Ap 可以进一步提高 SymGX 的有效性。

1.5.8 内存消耗

测量了工具及其基线的内存使用情况，COIN 的内存使用量是 SymGX 的 5.67 倍，而 TeeRex 的内存使用量与 SymGX 相当。

问题 5 回答：SymGX 的内存消耗比 COIN 低得多。

1.6 相关工作

SGX 已广泛应用于许多应用程序。漏洞检测器如 COIN、TeeRex 和其他类似方法都无法处理跨边界指针漏洞或捕捉 SGX 应用程序的三个独特属性。

1.7 结论

新型符号执行技术 SymGX,建立在一种名为 GSTG-CAP 的新型分析模型上，该模型可处理多入口任意顺序执行、有状态执行和上下文感知指针三种特性。总体而言，SymGX 在检测跨境指针漏洞方面优于现有解决方案。

二、重要参考文献的学习

2.1 参考文献

Victor Costan, Ilia Lebedev, Srinivas Devadas, et al. 2017. Secure processors

part I: background, taxonomy for secure enclaves and Intel SGX architecture.

Foundations and Trends® in Electronic Design Automation 11, 1-2 (2017), 1 – 248.

论文是有关安全处理器系统的最新技术调研和分析的两篇论文之一，重点关注远程软件认证和软件隔离。首先探讨了计算机体系结构和密码学相关概念，然后调查攻击向量和现有处理器系统声称具有远程计算和/或软件隔离功能。详细研究了现代隔离容器（enclave）原语，作为在实际可信硬件和合理性能开销下最小化可信软件的手段。具体而言，详细研究了英特尔软件保护扩展（SGX）的编程模型和软件设计考虑，因为它是一种可用和有文档支持的 enclave-capable 系统。第二部分则深入研究了两个现代的 enclave-capable 安全处理器系统的实现和安全评估：SGX 和 MIT 的 Sanctum。SGX 采用复杂但不足够的威胁模型，推动了 Sanctum 的发展，后者在软件攻击下实现了更强的安全保证，并具有相同的编程模型。

这篇文章探讨了远程计算的复杂权衡问题，用户需要在远程系统中放置多少信任？在给定安全性属性的情况下，接受多少性能开销？远程系统可以抵御多强大的对手？理想的系统将提供无额外开销的可信私密远程计算，而不需要任何信任假设，然而目前并不存在这样的系统。

在一个极端，昂贵的加密技术，包括混淆电路[Yao, 1986]和全同态加密[Gentry, 2009]，提供了高昂代价的无信任计算。而典型的云计算场景更接近相反的极端：在几乎无法检查的对远程系统的信任下，实现的安全性保证较弱，且开销最小。这项工作旨在说明，在对远程系统的信任非常有限的情况下，仍然可以实现重要的安全性质。一系列安全处理器的发展探索了可信硬件领域，实现了针对多种威胁模型的廉价远程计算。

对于安全性的深入讨论需要一个明确定义的威胁模型：可信硬件必须是安全的，即必须对一个明确定义的威胁模型表现出强大的抵御能力。例如，很少有系统能够对能够物理篡改系统硬件的对手提供有意义的保证。尽管“安全处理器”

描述的项目空间相当广泛，但这项工作专注于实现安全远程计算的系统，其定义见于§ 1.1。具体而言，该工作旨在阐明与安全软件隔离领域相关的编程模型、历史背景、设计决策和威胁模型，这是目前最先进且最具能力的安全远程计算范式。通过对 Intel 的软件保护扩展（SGX）和 MIT 的圣所系统的调查，该工作对隔离技术进行了范例说明。

2.2 参考文献 2

Tobias Cloosters, Michael Rodler, and Lucas Davi. 2020. TeeRex: discovery and

exploitation of memory corruption vulnerabilities in SGX enclaves. In Proceedings

of the 29th USENIX Conference on Security Symposium. 841 – 858.

本文讲述了 Intel 的软件保护扩展（SGX）引入了新的指令，将处理器切换到隔离模式，以保护其免受内省的影响。虽然隔离模式强大地保护了内存和处理器状态，但无法抵御在隔离代码内部的内存损坏错误。论文指出，SGX 隔离环境的攻击面为隔离开发者提供了新的挑战，易受内存损坏漏洞的影响。作者开发了名为 TeeRex 的工具，通过符号执行自动分析隔离二进制代码，以检测主机到隔离边界引入的漏洞。实证评估揭示了许多隔离二进制文件存在内存损坏错误，允许攻击者破坏函数指针或执行任意内存写入。TeeRex 提供了专门为 SGX 隔离环境定制的框架，简化了构建利用的过程，以评估发现的漏洞。研究发现了多个隔离，包括 Intel、Baidu 和 WolfSSL 开发的隔离以及部署在流行笔记本品牌上的生物特征指纹软件，存在漏洞。

这篇论文总结了 Intel 引入的 SGX 技术，该技术允许创建隔离环境以保护敏感应用代码和数据，但难以抵御内存损坏攻击。作者提出了 TeeRex 工具，旨在自动分析 SGX 隔离二进制代码，发现漏洞并构建利用。研究强调了主机到隔离边界的漏洞，特别关注了指针验证的困难。TeeRex 的评估揭示了许多已发布的隔离文件存在漏洞，作者强调了 SGX 隔离环境面临的挑战，包括来自 Intel、Baidu 和其他公司的隔离。研究的发现指出了隔离技术的潜在漏洞，展示了 TeeRex 的有效性，为隔离安全性研究领域做出了贡献。

第二三部分内容主要介绍了 Intel 的 Software Guard Extensions（SGX）技术以及在 SGX 中存在的内存腐败漏洞

Intel 的 Software Guard Extensions（SGX）引入了新的指令，将处理器切换到“enclave”模式，以保护其免受内省。虽然“enclave”模式可以强力保护处理器的内存和状态，但无法抵御“enclave”代码内的内存腐败错误。研究者发现 SGX enclave 存在攻击面，易受内存腐败漏洞的影响。为此，他们开发了 TEEREX，通过符号执行自动分析 enclave 二进制代码中在主机到 enclave 边界引入的漏洞。他们发现许多公共的 enclave 二进制文件都存在内存腐败错误，允许攻击者破坏函数指针或执行任意内存写入。

研究重点关注 SGX enclaves 边界处的验证，发现 Intel、Baidu 和 WolfSSL 等多个 enclaves 存在漏洞。TEEREX 提供了专门定制的框架，使简单的 POC（proof-of-concept）攻击构建成为可能，以评估发现的漏洞。结果显示，甚至包括 Intel、Baidu 等公司开发的 enclaves 也存在漏洞。

在第二部分，作者讨论了 SGX 中的内存腐败问题。由于常见的系统级编程语言 C/C++缺乏内建的内存安全性，过去三十年中出现了大量内存腐败漏洞。这些

漏洞允许攻击者对内存执行有限或(通常是)任意写入。作者认为，SGX enclaves与其他系统软件一样容易受到内存腐败攻击，大多数 enclaves 主要使用 C/C++进行开发。

研究者指出在针对 SGX enclaves 的内存腐败攻击中存在一个挑战：由于 SGX enclaves 在内存中是加密的并且可以作为加密的二进制文件传输，攻击者无法对 enclave 的二进制文件执行静态分析以搜索有趣的 ROP（return-oriented programming）小工具。为了解决这个问题，有学者通过反复执行 enclave 并在不同入口点触发执行，动态识别 ROP 小工具。然而，这种攻击不适用于代码地址在每次实例化 enclave 时都被随机化的 enclaves。作者还讨论了 SGX 的初步背景，解释了 SGX 技术和 Intel SGX SDK 的相关概念，以及 SGX 中的主机到 enclave 接口和 EDL 接口规范。

第二三部分展示了 SGX enclaves 中存在的内存腐败漏洞，并介绍了一个用于分析这些漏洞的框架 TEEREX。作者的发现强调了在开发和部署 SGX enclaves 时需要更加谨慎，并建议开发者关注主机到 enclave 接口的验证，以防范内存腐败漏洞。

第四部分向我们阐述 TEEREX 是一种新颖的符号执行框架，专为自动识别 SGX（Software Guard Extensions） enclave 的漏洞而设计。该框架不仅能够识别漏洞，还生成详细的漏洞报告，极大简化了构建对受影响 SGX enclave 的概念验证攻击的过程。它支持 Intel SGX SDK 支持的所有平台，包括 Windows（PE）和 Linux（ELF）二进制文件以及 32 位和 64 位 enclave。TEEREX 通过在二进制级别上应用符号执行，能够分析封闭源、专有 enclave。

TEEREX 架构主要包括以下几个组件：

1. Preprocessor： 用于预处理 enclave 二进制，以提高分析性能和覆盖范围。
2. Enclave Loader： 设置初始 enclave 执行环境，将标识的通用函数和特殊指令替换为模拟的 Python 代码。
3. Symbolic Explorer： 利用 ANGR 框架进行符号执行，从 enclave 代码中提取内存约束，用于漏洞分析。
4. Vulnerability Detection： 分析符号状态以检测 enclave 中的漏洞，特别关注访问内存和跳转的指令。
5. Pointer Tracking：通过分析所有指针解引用，跟踪标签以区分来自 enclave 和主机内存的数据，以识别可能的漏洞。
6. Vulnerability Report： 生成漏洞报告，包含漏洞类型、位置、受控指针和其在攻击者控制的输入中的位置，以及执行跟踪。

TEEREX 克服了多个挑战，包括符号执行的准确性和可扩展性，SGX-specific 指令的处理，对标准内存函数的支持，以及 enclave 的全局状态和 ECALL 链的分析。它能够检测控制流劫持、受控写入和 NULL 指针解引用等漏洞类型。

TEEREX 为 SGX enclave 提供了强大的漏洞分析工具，可帮助识别并报告潜在的安全问题。

TEEREX 分析结果总结：

TEEREX 通过对开源和专有公共 enclaves 的数据集进行分析，检测到了各种漏洞。分析中包括了 Intel 和 Baidu 等知名公司开发的 enclaves，以及 Dell 和 Lenovo 笔记本中使用的 SGX 保护的指纹软件。通过构建 PoC 攻击，TEEREX 证实了它发现的漏洞是严重的。

使用标准 SGX 对手模型进行分析，TEEREX 发现所有分析的 SGX enclaves 中

存在漏洞，除了 SignalApp 的 contact discovery service。对于每个有漏洞的 enclave，都成功构建了 PoC 攻击，并进行了负责任的披露。所有供应商都承认了发现的问题，除一家供应商外，其他供应商都开发了修复漏洞的解决方案。

在根本原因分析中，TEEREX 识别了多种导致漏洞的问题代码模式。表 2 展示了分析结果的概览，发现并成功利用了 TEEREX 检测到的所有不同类型的攻击原语。

具体来说，作者在 Intel GMP 示例、WolfSSL 示例和 TaLoS 等 enclaves 中发现了不同类型的漏洞。例如，在 Intel GMP 示例中，发现了一个关于 GMP 大整数对象内部指针未经过验证的漏洞。在 WolfSSL 示例中，通过传递带有恶意控制流的数据结构来利用 enclave，而 Rust SGX SDK 的 tlsclient 中的漏洞则显示了使用内存安全语言不会自动确保 enclaves 的内存安全。

对于 TaLoS，发现了使用用户检查标记的指针的漏洞，以及一个与 NULL 指针解引用相关的漏洞。在 Synaptics SynaTEE 驱动中，发现了一个由于 NULL 指针解引用而引起的控制流劫持漏洞。这些结果突显了 enclave 开发者需要更加谨慎，并指出了自动化分析工具如 TEEREX 的重要性。

三、结尾

3.1 深入理解 Intel Security Guard Extension 以保护关键数据

Intel Security Guard Extensions（SGX）是一项关键技术，旨在应对当今计算环境中对敏感和机密数据保护的紧迫需求。本文将全面探讨 SGX 的目的和功能，阐明其在加固计算系统安全性方面的作用。

3.1.1 SGX 的目的

SGX 的核心目标是在处理器内部建立一个安全区域，通常称为“enclave”（飞地），在该区域内进行的敏感计算和数据操作可以与系统的其余部分隔离开来。其主要目的是保护关键数据免受潜在对手的威胁，确保即使整个系统受到威胁，飞地仍然是保护机密操作的安全避风港。

3.1.2 保密性保证

SGX 通过硬件和软件相结合的机制，确保数据的机密性。它支持创建安全的飞地，其中存储和处理加密密钥、敏感算法和其他关键信息，远离未经授权的实体的窥探目光。

3.1.3 防范内部威胁

SGX 的显著特点之一是其能够防范内部威胁。即使恶意行为者获得对主机系统的访问权限，飞地中的数据仍然是加密的，并且没有适当授权，无法访问。这极大地减轻了源于受信任环境内部的数据泄露风险。

3.1.4 动态代码加载

SGX 的另一个显著功能是支持飞地内的动态代码加载。这使开发人员能够更新和修补飞地应用程序，而不会 compromise 飞地的安全性。动态加载代码的能力增强了 SGX 受保护应用程序的灵活性和适应性。

3.1.5 飞地的创建和管理

SGX 允许开发人员将应用程序的特定部分定义为飞地。这些飞地充当隔离的安全容器，其中进行敏感计算。开发人员对这些飞地中存在的数据和代码具有细粒度的控制权。

3.1.6 内存隔离

SGX 采用先进的内存隔离技术，防止未经授权的访问飞地数据。飞地的内存是加密的，只有在飞地内运行的经过授权的代码才能解密和操作数据。这确保即使攻击者获得对主机内存的访问权限，飞地的内容也保持不可理解。

3.1.7 远程证明

为了在不同系统组件之间建立信任，SGX 引入了一种称为远程证明的功能。该过程允许一方验证另一方飞地的真实性。远程证明在分布式系统中发挥着关键作用，确保飞地能够在不损害整个系统完整性的情况下进行安全通信。

3.1.8 安全密钥管理

SGX 为密钥管理提供了安全的环境，这是加密操作的关键方面。存储在 SGX 飞地中的密钥受到未经授权访问的保护，增强了加密过程的整体安全性，如安全通信和数据完整性验证。

3.2 符号执行：在 SGX 应用中利用静态符号执行进行漏洞检测

符号执行是一种先进的静态分析技术，被广泛应用于软件安全领域，特别是在 Intel Security Guard Extensions（SGX）应用程序的漏洞检测中发挥了关键作用。本文将深入研究符号执行的概念，着重探讨在 SGX 应用中使用静态符号执行进行漏洞检测的方法和意义。

3.2.1 符号执行的基本概念

符号执行是一种程序分析技术，旨在推导程序在各种输入条件下的行为。与传统的具体执行不同，符号执行使用符号变量来代表程序中的未知值，从而产生关于程序路径和状态的符号约束。这使得符号执行能够探索程序的多个执行路径，发现潜在的漏洞和安全隐患。

3.2.2 静态符号执行在 SGX 应用中的应用

在 SGX 应用中，静态符号执行被广泛应用于漏洞检测。SGX 的安全性建立在其能力以及应用程序在安全的飞地中执行的事实上。通过利用静态符号执行，我们可以在不运行实际代码的情况下分析程序的执行路径，揭示潜在的漏洞，从而提高 SGX 应用的安全性。

3.2.3 漏洞检测的挑战

SGX 应用的特殊性质使得传统的漏洞检测方法面临一些挑战。由于 SGX 飞地中的代码是加密的，传统的动态分析方法受到限制，因为无法直接观察和分析飞地内的执行。这时，静态分析方法变得尤为重要，而静态符号执行作为其中的一种方法，为我们提供了深入挖掘程序潜在漏洞的途径。

3.2.4 静态符号执行的优势

1. 路径探索： 静态符号执行能够深入分析程序的不同路径，发现潜在的错误和漏洞。在 SGX 应用中，这尤为重要，因为飞地的特性要求我们在设计阶段就考虑到各种可能的执行情景。
2. 约束求解： 符号执行生成的符号约束可以通过约束求解器进行求解，帮助我们理解在不同输入条件下程序的行为。这有助于发现潜在的漏洞，并提供改进代码的线索。
3. 安全规约验证： 静态符号执行可以帮助验证 SGX 应用中的安全规约是否得到了正确的实现。通过分析程序的符号执行路径，我们可以检查是否存在违反

规约的情况。

3.2.5 SGX 应用中的符号执行流程

在 SGX 应用中，利用静态符号执行进行漏洞检测通常包括以下关键步骤：

1.飞地分析：首先，对 SGX 应用的飞地进行分析。这包括识别飞地中的关键代码块、函数和数据结构。由于飞地的特殊性质，需要特别考虑加密和内存隔离。

2.符号变量引入：为了进行符号执行，引入符号变量以代表程序中的未知值。这些符号变量在路径探索中将被用于推导路径上的约束条件。

3.路径探索：通过静态符号执行，探索程序的不同执行路径。在每个路径上，收集符号约束，以描述在特定输入条件下程序的状态。

4.符号约束求解：利用约束求解器对收集的符号约束进行求解。这可以帮助理解程序在不同输入条件下的行为，并揭示潜在的漏洞。

5.安全规约验证：检查程序的符号执行路径是否符合预定义的安全规约。这有助于确保 SGX 应用在实际执行中能够遵循预期的安全行为。

3.2.6 应用案例

为了更好地理解静态符号执行在 SGX 应用中的应用，我们可以考虑一个具体的应用案例。假设我们有一个处理敏感数据的 SGX 应用，使用了自定义加密算法。通过静态符号执行，我们可以发现在某些输入条件下，加密算法可能存在缺陷，导致数据泄漏的风险。通过早期的漏洞检测，我们能够及时修复这些问题，提高 SGX 应用的安全性。

3.3 跨边界指针漏洞：识别和解决 SGX 应用程序数据交换边界的特定漏洞类型

随着信息技术的不断发展，数据安全性成为计算领域中的首要关注点。Intel Security Guard Extensions（SGX）技术的引入为数据的安全存储和处理提供了一种强大的机制。然而，即便在这样的安全环境中，我们依然面临着一些特定类型的漏洞，其中之一就是跨边界指针漏洞。本文将深入探讨跨边界指针漏洞的特性、识别方法和解决策略。

3.3.1 跨边界指针漏洞的基本概念

跨边界指针漏洞是一种特定类型的漏洞，它发生在数据交换边界，涉及指针的使用和管理。在 SGX 应用程序中，由于飞地（enclave）的特殊性质，这类漏洞具有一些独特的特征。指针漏洞通常涉及到对内存的非法访问，可能导致敏感数据泄露、程序崩溃或者远程执行代码攻击。

3.3.2 SGX 应用的数据交换边界

在 SGX 应用中，数据交换通常涉及到飞地与非飞地之间的边界。飞地是一种受保护的执行环境，而非飞地则是普通的执行环境。数据在这两个环境之间进行交互，指针被用于在不同环境中传递引用。跨边界指针漏洞即发生在这种边界处。

3.3.3 特定漏洞类型的影响

跨边界指针漏洞可能导致的问题包括：

1.数据泄露： 恶意攻击者可以通过利用跨边界指针漏洞，尝试读取飞地中的敏感数据，危及数据的保密性。
2.程序崩溃： 非法指针操作可能导致程序的不稳定性，甚至引发严重错误，从而影响应用程序的可靠性。
3.远程执行代码攻击： 攻击者通过精心构造的指针操作，可能实现在飞地中执行恶意代码，从而远程控制应用程序的执行。

3.3.4 识别跨边界指针漏洞的方法

为了提高 SGX 应用程序的安全性，我们需要有效地识别并解决跨边界指针漏洞。以下是一些常用的识别方法：

1.静态代码分析：通过对 SGX 应用程序进行静态代码分析，特别关注指针操作的位置和边界条件。静态分析工具可以帮助发现潜在的跨边界指针漏洞，提前防范可能的安全威胁。
2.动态代码分析：通过在实际运行时监视指针操作，动态代码分析可以帮助捕获实际执行中可能出现的漏洞。这种方法更接近应用程序实际的运行情况，有助于发现动态生成的漏洞。
3.符号执行：符号执行技术可以用于路径探索，识别在不同输入条件下可能导致跨边界指针漏洞的执行路径。这种方法对于深入理解程序的潜在问题和漏洞非常有帮助。

3.3.5 解决跨边界指针漏洞的策略

一旦识别到跨边界指针漏洞，采取适当的解决策略至关重要。以下是一些常见的解决方法：

1.严格的输入验证：确保对输入数据进行严格的验证，防止恶意输入触发跨边界指针漏洞。这包括验证指针是否指向有效的内存区域以及在交互边界处进行必要的边界检查。
2.内存保护机制：使用内存保护机制，如地址空间布局随机化（ASLR）和数据执行保护（DEP），以减轻跨边界指针漏洞可能导致的潜在危害。
3.静态和动态检测工具：定期使用静态和动态检测工具对 SGX 应用程序进行审查。这些工具可以帮助发现潜在的跨边界指针漏洞，并及时修复它们。
4.安全编程实践：采用安全编程实践，例如避免使用裸指针、使用安全的指针操作函数等，以减少指针漏洞的发生概率。

3.3.6 应用案例

为了更具体地理解跨边界指针漏洞的影响和解决方法，我们可以考虑一个实际的 SGX 应用案例。假设一个 SGX 应用程序涉及到与外部服务的通信，而在通信时使用了不经过严格验证的指针。通过静态和动态分析，我们可能发现在特定条件下，这些指针可能导致对飞地中敏感数据的非法访问。通过采取适当的解决策略，如增强输入验证和使用安全的通信协议，我们可以提高应用程序的安全性。

3.4 全局状态转换图与上下文感知指针模型（GSTG-CAP）：处理 SGX 应用程序独特属性的新颖分析模型

在当今数字化时代，数据安全性成为信息技术领域的核心挑战之一。Intel Security Guard Extensions（SGX）技术的出现为解决敏感数据的安全存储和处理提供了有力的解决方案。然而，由于 SGX 应用程序具有多入口、任意顺序执行、有状态执行和上下文感知指针等独特属性，传统的分析模型往往难以满足其特殊需求。本文将深入探讨全局状态转换图与上下文感知指针模型（GSTG-CAP）的特性、设计原理以及如何应对 SGX 应用程序的独特挑战。

3.4.1 GSTG-CAP 模型的基本概念

全局状态转换图与上下文感知指针模型（GSTG-CAP）是一种新颖的分析模型，专门设计用于处理 SGX 应用程序的独特属性。以下是 GSTG-CAP 模型的基本概念：

1. 全局状态转换图（GSTG）：全局状态转换图是一种用于表示系统行为的图形模型。在 SGX 应用程序中，由于存在多入口和任意顺序执行的特性，传统的有限状态机等模型往往无法充分表达其复杂的状态转换。GSTG 通过更全面的状态表示，能够更好地捕捉 SGX 应用程序中的多样性执行路径。
2. 上下文感知指针：上下文感知指针是指能够理解和适应当前执行上下文的指针。在 SGX 应用程序中，由于存在飞地和非飞地之间的数据交换，传统指针模型可能难以准确跟踪和管理指针的状态。上下文感知指针的引入使得模型能够更好地处理不同上下文中的指针行为，提高了分析的精度。
3. 模型设计原理：GSTG-CAP 模型的设计原理基于对 SGX 应用程序独特属性的深入理解。模型通过整合全局状态转换图和上下文感知指针，实现对 SGX 应用程序多入口、任意顺序执行、有状态执行等特性的综合考虑。这种综合性的设计使得 GSTG-CAP 模型能够更好地适应 SGX 应用程序的复杂性。

3.4.2 GSTG-CAP 模型的优势

GSTG-CAP 模型相较于传统的分析模型，具有明显的优势，尤其是在处理 SGX 应用程序的独特属性时：

1. 多入口任意顺序执行：由于 SGX 应用程序允许多入口和任意顺序执行，传统的有限状态机等模型可能无法捕捉所有可能的执行路径。GSTG-CAP 模型通过全局状态转换图的设计，能够更全面地覆盖多入口任意顺序执行的情况，提高了对系统行为的全面把握。
2. 有状态执行：SGX 应用程序常常涉及到有状态的执行，而这在传统的状态机模型中可能难以刻画。GSTG-CAP 模型通过对状态的更灵活表示，能够更好地适应 SGX 应用程序中的有状态执行，提供更精准的分析结果。
3. 上下文感知指针：引入上下文感知指针使得 GSTG-CAP 模型能够更好地理解和适应于不同上下文中的指针行为。在处理飞地和非飞地之间的数据交换时，上下文感知指针的使用提高了对指针状态的准确追踪，增强了分析

的精度。

3.4.3 GSTG-CAP 模型的应用

GSTG-CAP 模型在实际应用中具有广泛的潜力，特别适用于对 SGX 应用程序进行深入分析和漏洞检测。以下是 GSTG-CAP 模型的应用案例：

1. SGX 应用程序安全分析：通过构建 SGX 应用程序的全局状态转换图，GSTG-CAP 模型可以用于进行安全性分析，识别潜在的安全风险。模型的全面性和灵活性使得其能够更全面地评估 SGX 应用程序的安全性。
2. 漏洞检测：GSTG-CAP 模型可以用于检测 SGX 应用程序中的跨边界指针漏洞等问题。通过对指针行为的细致分析，模型能够发现潜在的漏洞，并提供有效的解决方案。
3. 性能优化：通过对 SGX 应用程序的执行路径进行全局状态转换图分析，可以帮助发现性能瓶颈和优化机会。GSTG-CAP 模型的设计使得其不仅能够关注安全性问题，还能够关注系统性能的细节。

3.5 总结

在 SGX 应用程序的独特环境中，全局状态转换图与上下文感知指针模型（GSTG-CAP）的引入提供了有力的工具，深入理解系统行为、分析安全性和解决性能问题。通过考虑 SGX 应用程序特性，GSTG-CAP 模型通过全局状态转换图和上下文感知指针的整合，提供更全面、更准确的分析结果。在信息安全领域不断演进的背景下，全面分析和理解 SGX 应用程序变得愈发关键，而 GSTG-CAP 模型成为应对这一挑战的强大工具。结合模型的理论基础和实际应用，可以更好地应对 SGX 应用程序的安全性和性能优化需求，确保其在数字化时代发挥最大潜力。

跨边界指针漏洞是 SGX 应用中常见但严重的安全威胁，可能导致敏感数据泄露和系统不稳定。通过深入理解漏洞特性、有效的识别方法和解决策略，可以更好地保护 SGX 应用程序免受这类漏洞的威胁。结合静态和动态分析、符号执行等技术，并采取安全的编程实践，能够建立更加坚固的 SGX 应用程序，确保其在数据交换边界处安全可靠地运行。对这类漏洞的认识和防范在信息安全领域中变得愈发重要，成为确保 SGX 技术发挥最大优势的重要一环。Intel Security Guard Extensions（SGX）技术是一项创新性的改变数据安全性处理方式的技术。通过提供安全的飞地，SGX 有效地应对网络威胁，确保敏感计算和数据操作的安全性。其目的是保证保密性和防范内部威胁，并得到一系列丰富功能的支持，使得开发人员能够构建具有弹性和安全性的应用程序。在不断演变的网络威胁面前，SGX 如一盏创新之灯，强化着我们数字世界的安全性。在复杂的网络安全领域中，SGX 作为一项创新技术，持续推动着我们对安全性的不断追求。

作业 3 小组分工

姜欣悦 20%	信息收集与查找 作业版式制作
盖乐 20%	论文总结分析
李颖 20%	论文学习整理
陈静怡 20%	相关文献学习
职泉 20%	论文学习整理