

# 上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

## 课程论文

COURSE PAPER



论文题目： 基于系统调用重载的系统级资源访问审计

学生姓名： 麻家乐、王梓睿、张天铄、汪何希、孙靖轩

学生学号： 521021910559、521021911110、521021910631、521021910523、521021910040

课程名称： 信息安全科技创新

指导教师： 陈秀真、姚立红

学院(系)： 电子信息与电气工程学院

## 基于系统调用重载的系统级资源访问审计

### 摘 要

本项目旨在开发一种基于系统调用重载的系统级资源访问审计解决方案。随着技术的不断发展，系统资源的访问控制和监控变得越来越重要，需要新的方法来实现对系统级资源的审计，以便及时发现和应对潜在的安全漏洞和威胁。重载系统调用接口，可以帮助识别异常行为、检测潜在的威胁以及追溯安全事件的源头，从而加强系统的安全性和完整性。将相关信息保存为日志，可以更好地分析性能和排查问题。

#### 本项目的重点工作包括：

1. 获取系统调用表 `sys_call_table`。
2. 实现系统函数重载和地址写入。
3. 实现内核和用户层间的信息交互。
4. 将信息保存为日志及尝试相关扩展处理。
5. 实现了图形化界面，提供可视化操作方便用户使用。

最终，本项目在 Ubuntu22.04 操作系统上完成，内核版本号为 5.19.0-xx-generic。本项目实现了上述功能，通过了项目测试，实现了基于系统调用重载的系统级资源访问审计的一般过程。

测试和试用结果表明，该解决方案具有较高的准确性和可靠性，并取得了令人满意的效果。我们的解决方案提供了全面的系统级资源访问审计，为保护系统免受潜在的内部和外部威胁提供了有力支持。

通过本项目，我们深入研究了系统资源访问审计的相关问题，并提供了创新的解决方案。我们的努力为进一步加强系统的安全性和完整性提供了有力的基础。未来的工作可以进一步改进和优化我们的解决方案，并在实际生产环境中实施和推广。

**关键词：**系统调用、内核通信、资源审计

## 目 录

第一章 需求分析 .....	1
1.1 项目需求分析 .....	1
1.2 项目功能 .....	1
第二章 总体设计 .....	2
2.1 总体结构 .....	2
2.2 内核态模块 .....	2
2.3 用户态模块 .....	2
第三章 详细设计 .....	3
3.1 内核态模块详细设计 .....	3
3.2 用户态模块详细设计 .....	4
第四章 系统实现 .....	6
4.1 实现工具与开发环境 .....	6
4.2 源文件及其功能介绍 .....	6
第五章 系统测试 .....	8
5.1 测试流程 .....	8
5.2 通信测试 .....	8
5.3 重载函数实现 .....	9
5.3.1 openat 与 unlink 函数重载 .....	9
5.3.2 execve 函数重载 .....	9
5.3.3 finit_module 函数重载 .....	10
5.3.4 reboot 与 shutdown 函数重载 .....	10
5.3.5 mount 与 umount 函数重载 .....	11
5.3.6 Mknodat 函数重载 .....	12
5.4 日志管理 .....	12
5.5 图形界面 .....	13
5.6 结论 .....	15
第六章 项目小结 .....	16
6.1 项目实施情况 .....	16
6.2 代码管理 .....	16
6.3 不足和展望 .....	17
6.4 体会、感受与建议 .....	17
6.5 附：项目组成员贡献表 .....	18

## 第一章 需求分析

### 1.1 项目需求分析

安全性是系统必不可少的性质。基于系统调用重载的文件访问日志可以用于监控和记录系统中发生的文件访问事件，从而有助于提高系统的安全性。为了提高系统的可扩展性和灵活性，获取系统调用表可以让我们掌握实现其他函数重载与扩展的方法与信息。而实现资源访问审计的功能，可以在其基础上实现一些定制化的功能和安全控制，例如针对某些特定的系统调用进行拦截或限制等，同时记录日志也能帮助管理员和团队监控系统的运行情况，如有异常行为即可及时作出调整，从而加强系统的安全性。

### 1.2 项目功能

1. 通过注册内核探针的方式获取了系统调用表，同时实现了函数重载与地址写入，便于项目的进行和未来的优化。
2. 完成内核态与用户态程序的数据传递，用户态程序接受到内核态程序信息后将其储存在日志文件。
3. 实现日志的管理，如按关键字排序、搜索关键字、合并同内容日志等功能。
4. 实现了图形化界面，提供可视化操作方便用户使用。

## 第二章 总体设计

### 2.1 总体结构

项目的总体结构如图2-1所示：

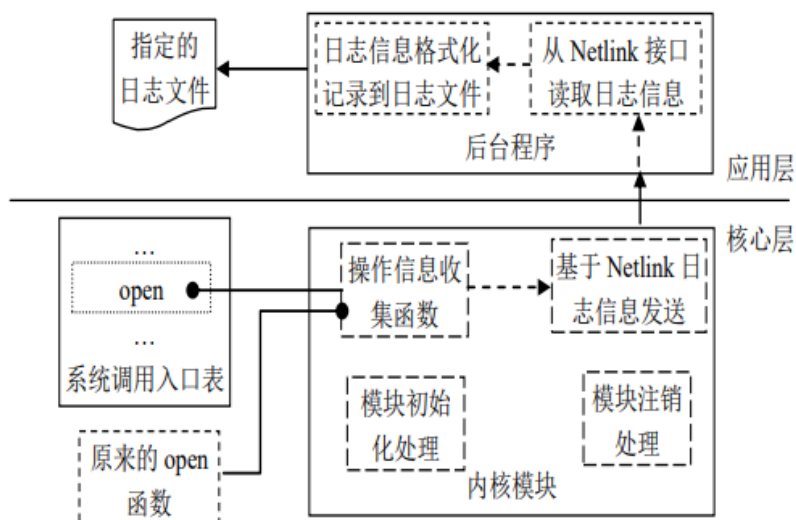


图 2-1 项目总体架构图

本项目可以分为两个模块：内核态模块和用户态模块。

总体的流程是先进行模块的初始化处理，在内核模块获取系统调用表后，实现重载的函数并将其地址写入接口，通过 netlink 方式将获得的日志数据发往应用层，实现日志信息收集和通信；用户态模块从 netlink 接口读取日志信息后，将其格式化记录到日志文件，在此基础上实现图形界面和日志的审计功能，使其功能更完善，便于交互。

### 2.2 内核态模块

借助系统调用重载技术，从 open 系统调用中获得文件读写操作的日志信息，并通过 netlink 方式将获得的日志数据发往应用层，在应用层创建一个 netlink socket 来接收内核模块发送的日志信息，实现日志信息收集和通信。

### 2.3 用户态模块

在应用层接受从核心层以 netlink 机制传来的日志信息，然后以格式化的形式存储在指定的日志文件中，并提供了图形化界面方便用户的使用与交互。其中日志模块实现了显示、交换、排序、删除、搜索等功能方便用户的管理，且具有可扩展性，方便附加其他功能。

## 第三章 详细设计

### 3.1 内核态模块详细设计

本模块的功能与目的是实现系统调用重载，基本过程如下：先实现各个重载函数，之后获取系统调用表 `sys_call_tabel`，写入重载的系统调用功能函数入口地址，根据需求编写新函数并将其接入系统调用表中对应的项，并在各个重载函数中将对应的日志信息发送给用户态的程序。其处理流程图如图3-1所示：

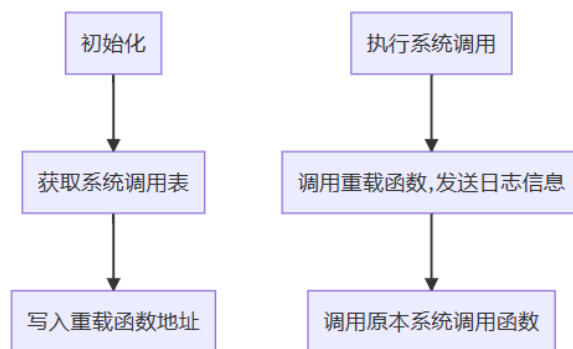


图 3-1 内核模块详细设计流程图

文件 `netlink_kernel.c` 的相关模块功能解释如下：

- **DEFINITION**: 定义了一些全局变量、结构体、工具函数等。包括获取和存储系统调用表，获取文件名和时间的函数等。对 `openat` 函数进行了钩子操作，其中包括处理函数 `handle_openat`，安装钩子的函数 `hook_openat`，恢复函数 `restore_openat`，以及假函数 `fake_openat`。另外还定义了一个指向真实 `openat` 函数和假的 `openat` 函数的函数指针。对 `unlink`、`execve`、`shutdown`、`reboot`、`finit_module`、`mount`、`umount2`、`mknodat` 函数也进行了同样的钩子操作。还定义了接受和发送消息的函数用于进行 `netlink` 操作。

- **KERNEL**: 定义了模块的初始化和退出函数，加载模块时自动执行。

- **GET\_SYS\_TABLE**: 获取系统调用表。

- **UTILS**: 定义了三个工具函数。`get_fullname` 函数用于获取文件的路径名，`get_cur_name` 函数用于获取当前时间并用字符串表示，`path_cmp` 函数用于比较两个路径名字符串是否相同。

- **OPENAT**: `handle_openat()` 实现对系统调用的控制；`fake_openat()` 用于替代真实的系统调用；`hook_openat()` 用于实现 `fake_unlink` 函数的替换并打印成功的信息；`restore_openat()` 用于恢复原始的系统调用并打印成功信息。

- **UNLINK、EXECVE、SHUTDOWN、REBOOT、FINIT\_MODULE**: 实现的功能与上一个模块相似，实现了对系统调用的控制、替换、恢复和打印相关信息的功能。

- **NETLINK**: 接收 `netlink` 消息，根据 `flag` 进行相应处理并给用户反馈。

## 3.2 用户态模块详细设计

本模块的功能与目的是信息接收与日志记录，过程分为两个模块：信息接收与日志管理。在程序运行的时候，需要不断接受来自内核态的信息，并将其保存在文件中。此外，还需要接收来自用户的指令，对日志信息进行修改、筛选与展示。

文件 `clean_log.c` 实现了打开文件并用写入模式关闭的功能，如果文件不存在，会创建一个新的空文件。

有关文件控制的功能实现在文件 `log_control.c` 中，其中包括的主要函数有：

- **help()**: 显示帮助信息.
- **show()**: 显示日志信息.
- **log\_val\_init()**: 初始化日志变量，分配内存空间。
- **release()**: 释放内存空间。
- **open\_log()**: 返回日志条目数量
- **swap()**: 交换日志内容。
- **cmp\_xxxx()**: 根据日志的指定字段进行排序，如 `cmp_uid` 是根据用户 `id` 字段排序，还有相关的倒序排序函数。
- **\*search()**: 搜索指定的日志条目并返回。
- **merge()**: 返回相同的日志条目并返回数量。
- **run()**: 运行函数，其中包括了上述功能的调用。

`run` 函数的执行流程如图3-2所示：

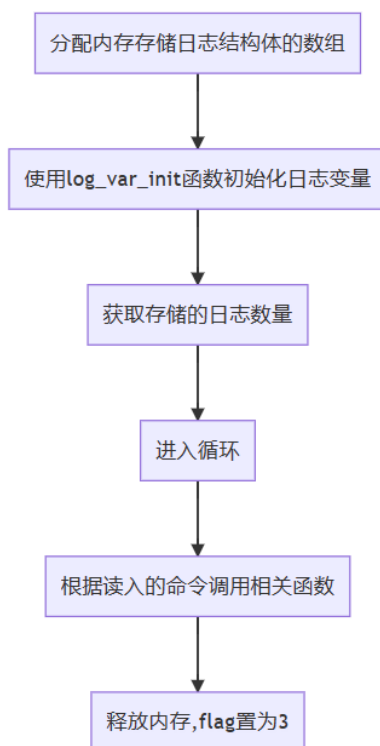


图 3-2 `run` 函数的执行流程图

对内核态与用户态之间的通信功能的实现在文件 `recv_msg_from_kernel.c` 和

send\_msg\_to\_kernel.c 中，下面介绍这两段代码的实现逻辑：

recv\_msg\_from\_kernel.c 中的主要函数及功能包括：

- **clear\_output:** 用于清空文件内容。打开文件并检查是否成功打开。若失败会输出错误信息并返回。成功打开文件后关闭文件。
- **save\_buffer:** 用于保存接收到的缓冲区数据到日志文件。首先获取格式化的命令名称和文件路径。然后检查命令格式是否有效，如果无效则返回。接着使用 Log 函数记录日志信息，其中包括命令名称、接收到的用户 ID、接收到的进程 ID 和文件路径。
- **main:** 首先获取当前进程的 ID，并使用 signal 函数设置信号处理函数。然后创建一个 Netlink 套接字，并设置源地址和目标地址，分配并初始化 Netlink 消息头，并向内核发送消息。之后进入主循环，接收消息并保存到日志文件中。

send\_msg\_to\_kernel.c 中的主要函数及功能包括：

- **send\_message\_to\_kernel:** 函数用于向内核发送消息。它首先创建一个 Netlink 套接字并设置源地址。然后设置目标地址为内核，并分配并初始化 Netlink 消息头。根据传入的标志和类型参数，设置消息头的标志和类型字段。然后使用 sendmsg 函数向内核发送消息。最后关闭套接字并释放内存。
- **main:** 根据命令行参数来确定需要发送的标志和类型。然后调用 send\_message\_to\_kernel 函数发送消息给内核。

send\_control\_path.c 实现了通过 Netlink 套接字将控制路径发送给内核的功能。程序创建 Netlink socket，发送控制路径消息给内核，实现特定功能或请求内核响应。

有关图形化界面的功能实现在文件 program.py 中，其中包括的主要类与功能有：

- **class LOG:** 有关页面展示的类，包括了一些基本的功能函数如页面初始化、更换、退出、输入密码、展示帮助菜单等。
- **class cmds:** 有关命令封装的类。
- **class Button:** 有关按钮对象的类，包括的功能函数有按钮的显示、状态检查等。
- **class textBox:** 有关文本框对象的类，实现的功能有接受、删除内容，颜色更换、显示等。
- **class ListBoxes:** 实现了列表框的初始化和在其中对文本框进行添加、删除、修改、处理事件等功能。
- **class LISTS:** 实现了一个多列的列表的初始化及相关功能。
- **class Dropdown:** 实现了一个下拉框的初始化、更改设置选项、绘制图案、处理事件等功能。
- **class TimeDropdown:** 用上面 textBox 和 Dropdown 类创建的一个带时间选择的下拉框的类，包括了绘制图案、设置时间、处理事件等功能。
- **class Timer:** 计时器类，包括获取时间、绘制计时器等功能。



## 第四章 系统实现

### 4.1 实现工具与开发环境

软件开发平台：

- 编程语言支持：C/C++
- 开发工具和集成环境 (IDE)：GNU 编译器集合 (GCC)、GNU 调试器 (GDB)、GNU 自动构建工具 (Make)、集成环境 Visual Studio Code
- 版本控制系统：Git
- 调试和测试工具：Visual Studio Code、Ubuntu 虚拟机

运行环境：

Ubuntu 22.04

内核 linux-5.19

### 4.2 源文件及其功能介绍

• **netlink\_kernel.mod.c**: 60 lines。定义和构建了一个基本的 Linux 内核模块的源代码框架，为进一步的扩展和实现提供了基础。它包含必要的模块信息和许可证声明等，用于正确加载和管理内核模块。

• **netlink\_kernel.c**: 821 lines。该文件实现了重载系统调用的功能函数入口地址，并将新函数接入系统调用表的相应项中，从而实现自定义的功能。它涉及到操作内核的底层机制，通过修改系统调用表来改变特定系统调用的行为。

• **log\_control.c**: 475 lines。该文件实现了日志管理的功能，包括按关键字排序、搜索关键字、合并相同内容的日志等功能。它提供了对日志数据的处理和操作方法，使用户能够有效地管理和分析日志信息。

• **recv\_msg\_from\_kernel.c**、**send\_msg\_to\_kernel.c**: 319 lines。这两个文件实现了内核态与用户态之间的通信功能。通过使用 Netlink 套接字，它们使用户空间的应用程序能够与内核进行双向的数据传输，以实现内核模块与用户程序之间的交互和信息传递。。

• **send\_control\_path.c**: 68 lines。该文件实现了通过 Netlink 套接字将控制路径发送给内核的功能。它负责将用户提供的控制路径信息发送到内核空间，以便内核模块能够读取并作出相应的响应和处理。

• **program.py**: 446 lines。这个文件实现了一个图形化界面，包括页面的显示与更换、按钮的显示与状态识别、文本框、列表、下拉框的显示与输入等功能。它作为用户与系统交互的界面，提供了友好的操作界面，使用户能够方便地使用和控制系统的各个功能。

文件的调用关系如图4-1所示：

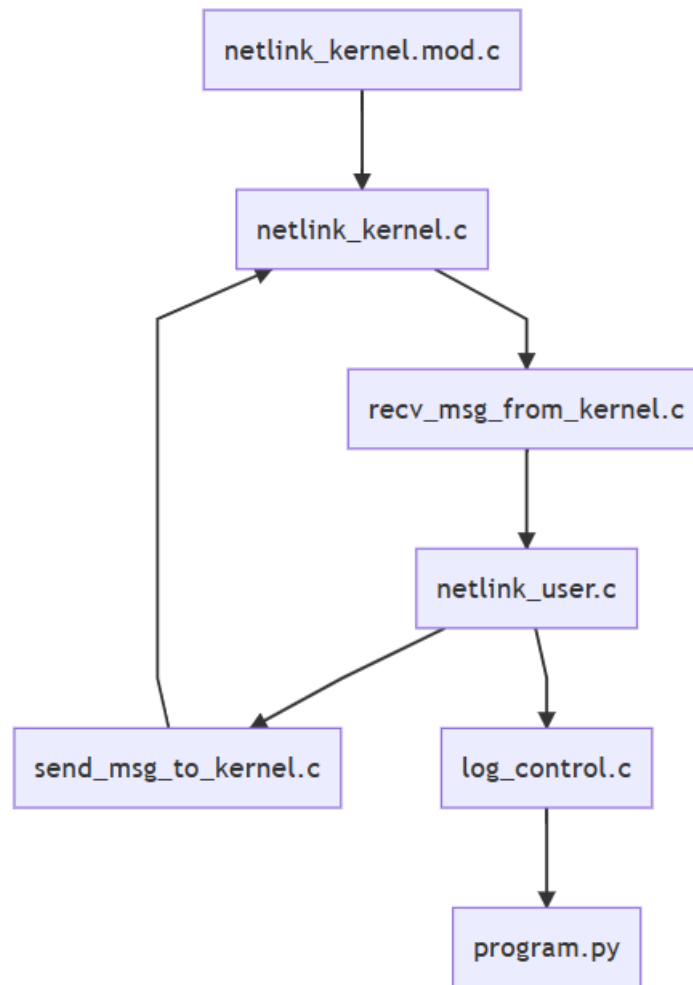


图 4-1 源文件调用关系

文件的主要函数与数据结构在“详细设计”一节已说明。

## 第五章 系统测试

### 5.1 测试流程

结合项目功能的实现顺序，本项目的一般测试流程如图5-1所示：

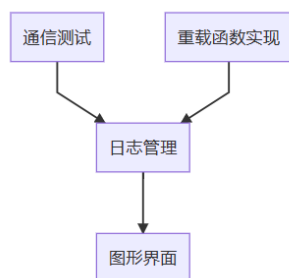


图 5-1 项目测试流程图

### 5.2 通信测试

实验中，需要将内核态的程序信息传递到用户态，再保存为日志文件以执行审计与日志管理操作。程序编译并运行，测试后的信息保存在文件 log 中。执行以下命令以实现清空日志、记录日志：

```
./clean_log  
./recv_msg_from_kernel
```

观察到的信息如图5-2所示（部分）：

```
runner 1001 EXECVE 3597 2023-07-19 14:33:27 /home/runner/work/NIS3302-okftools/NIS3302-okftools/okftools/bin/sh  
runner 1001 EXECVE 35981 2023-07-19 14:33:27 /home/runner/work/NIS3302-okftools/NIS3302-okftools/okftools/usr/bin/clear  
runner 1001 EXECVE 3600 2023-07-19 14:33:33 /home/runner/work/NIS3302-okftools/NIS3302-okftools/okftools/./send_msg_to_kernel  
runner 1001 RESTORE 3600 2023-07-19 14:33:33 OPENAT  
runner 1001 NETLINK 3600 2023-07-19 14:33:33 Hello from kernel  
runner 1001 EXECVE 3601 2023-07-19 14:33:34 /home/runner/work/NIS3302-okftools/NIS3302-okftools/okftools/./send_msg_to_kernel  
runner 1001 RESTORE 3601 2023-07-19 14:33:34 UNLINK  
runner 1001 NETLINK 3601 2023-07-19 14:33:34 Hello from kernel  
runner 1001 EXECVE 3602 2023-07-19 14:33:34 /home/runner/work/NIS3302-okftools/NIS3302-okftools/okftools/./send_msg_to_kernel  
runner 1001 RESTORE 3602 2023-07-19 14:33:34 EXECVE  
runner 1001 NETLINK 3602 2023-07-19 14:33:34 Hello from kernel  
runner 1001 RESTORE 3603 2023-07-19 14:33:34 SHUTDOWN  
runner 1001 NETLINK 3603 2023-07-19 14:33:34 Hello from kernel  
runner 1001 RESTORE 3604 2023-07-19 14:33:34 REBOOT  
runner 1001 NETLINK 3604 2023-07-19 14:33:34 Hello from kernel  
runner 1001 RESTORE 3605 2023-07-19 14:33:34 FINIT_MODULE  
runner 1001 NETLINK 3605 2023-07-19 14:33:34 Hello from kernel  
runner 1001 RESTORE 3606 2023-07-19 14:33:35 MOUNT  
runner 1001 NETLINK 3606 2023-07-19 14:33:35 Hello from kernel  
runner 1001 RESTORE 3607 2023-07-19 14:33:35 UMOUNT2  
runner 1001 NETLINK 3607 2023-07-19 14:33:35 Hello from kernel  
runner 1001 RESTORE 3608 2023-07-19 14:33:35 MKNODAT  
runner 1001 NETLINK 3608 2023-07-19 14:33:35 Hello from kernel
```

图 5-2 通信测试结果图

可以看到程序信息的相关字段如时间、路径等均已成功打印，说明这一部分的测试成功。

## 5.3 重载函数实现

检测相关函数重载功能，过程如下：

### 5.3.1 openat 与 unlink 函数重载

安装、卸载模块：

```
sudo ./send_msg_to_kernel hook openat
sudo ./send_msg_to_kernel restore openat
sudo ./send_msg_to_kernel hook unlink
sudo ./send_msg_to_kernel restore unlink
```

安装模块之后，日志信息被打印在 log 中：

```
40 heatingma 1000 EXECVE 9774 2023-07-17 14:35:34 ./send_msg_to_kernel
41 heatingma 1000 RESTORE 9774 2023-07-17 14:35:34 EXECVE
42 heatingma 1000 NETLINK 97744 2023-07-17 14:35:34 Hello from kernel
43 heatingma 1000 RESTORE 9776 2023-07-17 14:35:34 UNLINK
44 heatingma 1000 NETLINK 97764 2023-07-17 14:35:34 Hello from kernel
45 heatingma 1000 RESTORE 9777 2023-07-17 14:35:34 OPENAT
46 heatingma 1000 NETLINK 97774 2023-07-17 14:35:34 Hello from kernel
47 heatingma 1000 NETLINK 5443 2023-07-18 22:42:34 Hello from kernel
48 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.CVH971
49 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.SCV971
50 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.XTZ971
51 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.USQ971
52 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.VVU971
53 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.ZFMB81
54 heatingma 1000 OPENAT 2073 2023-07-18 22:42:34 /home/heatingma/.cache/tracker3/files/last-crawl.txt.98RB81
```

图 5-3 openat、unlink 函数重载测试结果图

### 5.3.2 execve 函数重载

安装、卸载模块：

```
sudo ./send_msg_to_kernel hook execve
sudo ./send_msg_to_kernel restore execve
```

安装 execve 模块之后，可以发现此时执行程序的信息被打印在内核日志和 log 中：

```
[ 163.490267] Receive the message from user: flag = 3, type = 0
[ 163.490273] real_execve:ffffffffffa3bfe300
[ 163.490275] Disable write-protection of page with sys_call_table
[ 163.490277] Restart write-protection successfully
[ 163.490278] the execve function has been successfully hooked
[ 163.507507] Program /usr/libexec/tracker-extract-3 is executed by user 1000
[ 178.175735] Program /usr/bin/ls is executed by user 1000
[ 178.190032] Program /usr/libexec/tracker-extract-3 is executed by user 1000
[ 193.485838] Program /usr/bin/mkdir is executed by user 1000
[ 193.526849] Program /usr/libexec/tracker-extract-3 is executed by user 1000
[ 202.615598] Program /usr/bin/touch is executed by user 1000
[ 205.008161] Program /usr/bin/rm is executed by user 1000
```

图 5-4 execve 函数重载测试结果图-内核日志

```
1 halsayxi 1000 NETLINK 2700 2023-07-19 21:26:28 Hello from kernel
2 halsayxi 1000 HOOK 2752 2023-07-19 21:27:07 EXECVE
3 halsayxi 1000 NETLINK 2752 2023-07-19 21:27:07 Hello from kernel
4 halsayxi 1000 EXECVE 2756 2023-07-19 21:27:07 /usr/libexec/tracker-extract-3
5 halsayxi 1000 EXECVE 2766 2023-07-19 21:27:22 /usr/bin/ls
6 halsayxi 1000 EXECVE 2768 2023-07-19 21:27:22 /usr/libexec/tracker-extract-3
7 halsayxi 1000 EXECVE 2776 2023-07-19 21:27:37 /usr/bin/mkdir
8 halsayxi 1000 EXECVE 2779 2023-07-19 21:27:37 /usr/libexec/tracker-extract-3
9 halsayxi 1000 EXECVE 2786 2023-07-19 21:27:46 /usr/bin/touch
10 halsayxi 1000 EXECVE 2789786 2023-07-19 21:27:48 /usr/bin/rm
11 halsayxi 1000 EXECVE 2796 2023-07-19 21:28:04 /usr/bin/gedit
12 halsayxi 1000 EXECVE 2798 2023-07-19 21:28:04 /usr/libexec/tracker-extract-3
```

图 5-5 execve 函数重载测试结果图-log

## 5.3.3 finit\_module 函数重载

安装、卸载模块：

```
sudo ./send_msg_to_kernel hook finit_module
sudo ./send_msg_to_kernel restore finit_module
```

安装 finit\_module 模块之后，在命令行执行 insmod 命令，将在内核日志和 log 中打印信息：

```
719.073327] Receive the message from user: flag = 6, type = 0
719.073332] real_finit_module:ffffffff398fef0
719.073334] Disable write-protection of page with sys_call_table
719.073336] Restart write-protection successfully
719.073337] the finit_module function has been successfully hooked
719.086616] Program /usr/libexec/tracker-extract-3 is executed by user 1000
729.258247] Program /usr/bin/sudo is executed by user 1000
729.275044] Program /usr/libexec/tracker-extract-3 is executed by user 1000
731.629706] Program /usr/sbin/insmod is executed by user 0
731.632182] Module initialized by user: 0, fd: 3
731.632186] 555
731.632205] 666
731.645783] hello
```

图 5-6 finit\_module 函数重载测试结果图-内核日志

```
8 halsayxi 1000 HOOK 3197 2023-07-19 21:36:15 EXECVE
9 halsayxi 1000 NETLINK 3197 2023-07-19 21:36:15 Hello from kernel
10 halsayxi 1000 EXECVE 3203 2023-07-19 21:36:26 ./send_msg_to_kernel
11 halsayxi 1000 HOOK 32031000 2023-07-19 21:36:26 FINIT_MODULE
12 halsayxi 1000 NETLINK 32033 2023-07-19 21:36:26 Hello from kernel
13 halsayxi 1000 EXECVE 3205 2023-07-19 21:36:26 /usr/libexec/tracker-extract-3
14 halsayxi 1000 EXECVE 3213 2023-07-19 21:36:36 /usr/bin/sudo
15 halsayxi 1000 EXECVE 3215 2023-07-19 21:36:36 /usr/libexec/tracker-extract-3
16 root 0 EXECVE 3223 2023-07-19 21:36:38 /usr/sbin/insmod
17 root 0 FINIT_MODULE 3223 2023-07-19 21:36:38 A new module is initialized, fd 03
```

图 5-7 finit\_module 函数重载测试结果图-log

## 5.3.4 reboot 与 shutdown 函数重载

安装卸载模块同理。

但是在安装模块之后执行 reboot、shutdown 命令，log 中并没有出现相应的日志，同时重启之后原本在缓存区中的内核日志也被清空，故都没有输出。

我们认为执行 reboot 之后会调用一个可执行程序，这里面涉及包括停止所有进程、保存信息等等的内容，故怀疑这个时候我们的监听程序被停止了。

我们目前的解决办法是：在 fake\_shutdown 和 fake\_reboot 函数中使系统睡眠 10s，再执行关机或重启，并在内核中打印 “After 10 seconds, the computer will be shutdown/reboot”，并监测关机重启过程中的系统日志。

测试发现，关机重启时间将延长 10s，并有如下输出：

```
[ 549.412870] After 10 seconds, the computer will be shutdown
```

图 5-8 shutdown 函数重载测试结果图

### 5.3.5 mount 与 umount 函数重载

执行命令

```
sudo ./send_msg_to_kernel hook mount
```

mount 函数模块被成功安装，同理，执行命令

```
sudo ./send_msg_to_kernel hook umount2
```

安装了 umount 函数。

测试两个模块能否正常运行的测试脚本 test 详见附录 A：

执行如下命令以运行脚本：（在运行脚本前，先输入配置环境的指令）

```
dd if=/dev/zero of=file.img bs=1M count=1000
sudo touch /mnt/myfile
./test_mount
./test_mount
sudo dmesg | grep mount
```

结果如图5-9，说明信息成功打印到日志中。

```
[18611.525885] real_mount:ffffffffff89026320
[18611.525888] the mount function has been successfully hooked
[18637.280856] real_umount2 : ffffffff89023720
[18637.280860] the umount2 function has been successfully hooked
[18737.519451] Z\xbc\xff\xff\xp\x4\xd3\x1b\x9b\xff\xffH\r\xc6Z\xbc\xff\xff\r\xc6Z\xbc\xff\xff\x80d\x4\xc1
\x1b\x9b\xff\xff0\r\xc6Z\xbc\xff\xffD\r\xc6Z\xbc\xff\xff(\r\xc6Z\xbc\xff\xff E\xbe\xd1\x1b\x9b\xff\xff is m
ounted at \xff\xff\xff\xff0\r\xc6Z\xbc\xff\xff@N\x80\x1c\x1c\x9b\xff\xff
[18737.570017] EXT4-fs (loop19): mounted filesystem with ordered data mode. Quota mode: none.
[18742.280119] \xedG-\x98)V is unmounted
[18768.830410] \xed\x87{\xcdxU is unmounted
[18768.831866] \xedG-\x98)V is unmounted
[18768.834726] EXT4-fs (loop19): unmounting filesystem.
[18768.912851] Z\xbc\xff\xff\x10\xbc8\xd1Z\xbc\xff\xff E\xbe\xd1\x1b\x9b\xff\xff@N\x80\x1c\x1c\x9b\xff\xff
is mounted at
[18768.919455] EXT4-fs (loop19): mounted filesystem with ordered data mode. Quota mode: none.
```

图 5-9 mount 与 umount 函数重载测试结果图

执行命令

```
sudo ./send_msg_to_kernel restore mount
sudo ./send_msg_to_kernel restore umount2
```

卸载函数模块。

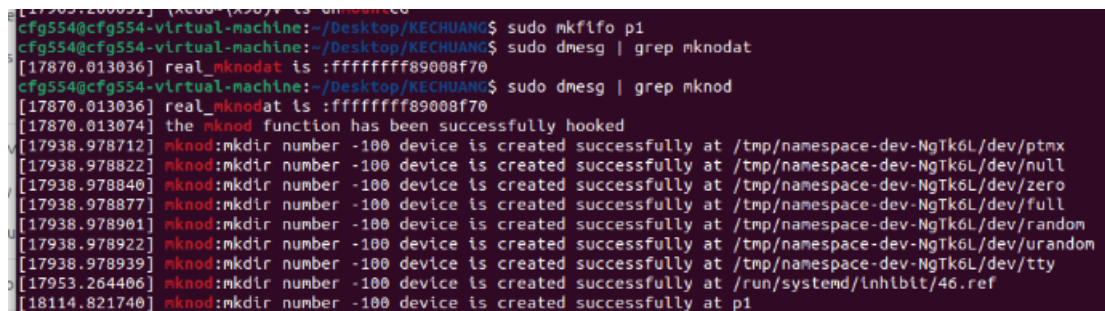


### 5.3.6 Mknodat 函数重载

执行命令

```
sudo ./send_msg_to_kernel hook mknodat  
sudo mkfifo p1  
sudo dmesg | grep mknod
```

表示安装模块后在当前目录下创建了一个 p1 管道文件，结果如图5-10：



```
[17938.978931] (kernel) CS Unhooked  
cfg554@cfg554-virtual-machine:~/Desktop/KECHUANG$ sudo mkfifo p1  
cfg554@cfg554-virtual-machine:~/Desktop/KECHUANG$ sudo dmesg | grep mknodat  
[17870.013036] real_mknodat is :ffffffff89008f70  
cfg554@cfg554-virtual-machine:~/Desktop/KECHUANG$ sudo dmesg | grep mknod  
[17870.013036] real_mknodat is :ffffffff89008f70  
[17870.013074] the mknodat function has been successfully hooked  
[17938.978712] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/ptmx  
[17938.978822] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/null  
[17938.978840] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/zero  
[17938.978877] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/full  
[17938.978901] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/random  
[17938.978922] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/urandom  
[17938.978939] mknod:mkdir number -100 device is created successfully at /tmp/namespace-dev-NgTk6L/dev/tty  
[17953.264406] mknod:mkdir number -100 device is created successfully at /run/systemd/inhibit/46.ref  
[18114.821740] mknod:mkdir number -100 device is created successfully at p1
```

图 5-10 Mknodat 函数重载测试结果图

可以看到相关信息以打印到日志中。

最后同理，输入

```
sudo ./send_msg_to_kernel restore mknodat
```

以卸载函数模块。

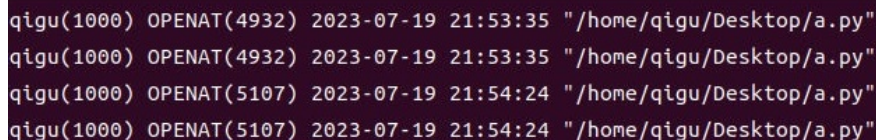
## 5.4 日志管理

程序信息保存为日志文件后，需要接受用户的指令以实现一般的审计功能。程序运行后输入对应指令，日志信息的控制如下所示：

执行 log\_control 之后，输入

```
search 文件名
```

进行搜索，搜索 a.py 的测试结果如图5-11



```
qigu(1000) OPENAT(4932) 2023-07-19 21:53:35 "/home/qigu/Desktop/a.py"  
qigu(1000) OPENAT(4932) 2023-07-19 21:53:35 "/home/qigu/Desktop/a.py"  
qigu(1000) OPENAT(5107) 2023-07-19 21:54:24 "/home/qigu/Desktop/a.py"  
qigu(1000) OPENAT(5107) 2023-07-19 21:54:24 "/home/qigu/Desktop/a.py"
```

图 5-11 日志管理搜索功能测试结果图

输入

sort 6

对第六项日志内容进行排序，结果如图5-12。

```
qigu(1000) OPENAT(1640) 2023-07-19 21:55:00 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZA4871"
qigu(1000) OPENAT(1640) 2023-07-19 21:54:58 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZEY671"
qigu(1000) OPENAT(1640) 2023-07-19 21:55:00 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZFV571"
qigu(1000) OPENAT(1640) 2023-07-19 21:54:57 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZK9E81"
qigu(1000) OPENAT(1640) 2023-07-19 21:54:58 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZZ1E81"
qigu(1000) OPENAT(5143) 2023-07-19 21:54:57 "/home/qigu/.cache/tracker3/files/meta.db-wal"
qigu(1000) OPENAT(5144) 2023-07-19 21:54:57 "/home/qigu/.cache/tracker3/files/meta.db-wal"
qigu(1000) OPENAT(1640) 2023-07-19 21:54:58 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZIBM81"
qigu(1000) OPENAT(1640) 2023-07-19 21:54:58 "/home/qigu/.cache/tracker3/files/last-crawl.txt.ZRMM81"
qigu(1000) OPENAT(5144) 2023-07-19 21:54:57 "/home/qigu/.cache/tracker3/files/meta.db-wal"
qigu(1000) OPENAT(4932) 2023-07-19 21:53:35 "/home/qigu/Desktop/a.py"
qigu(1000) OPENAT(4932) 2023-07-19 21:53:35 "/home/qigu/Desktop/a.py"
qigu(1000) NETLINK(5141) 2023-07-19 21:54:57 "Hello from kernel"
```

图 5-12 日志管理排序索引功能测试结果图

综上所述，日志信息进行搜索、排序与展示功能均成功进行。

## 5.5 图形界面

本项目还实现了图形界面，方便用户的使用与交互，其使用流程如下：  
输入 root 密码进行登录，如图5-13所示：

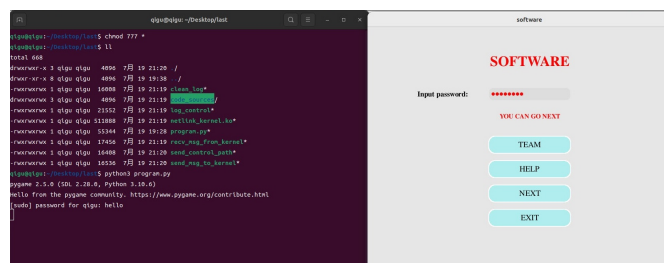


图 5-13 图形化登录界面展示图

输入路径，使得只有在该路径下的文件打开时才会进行记录，发送成功结果如图5-14：



图 5-14 图形化路径控制展示图



进行路径控制之后，结果如图5-15所示：

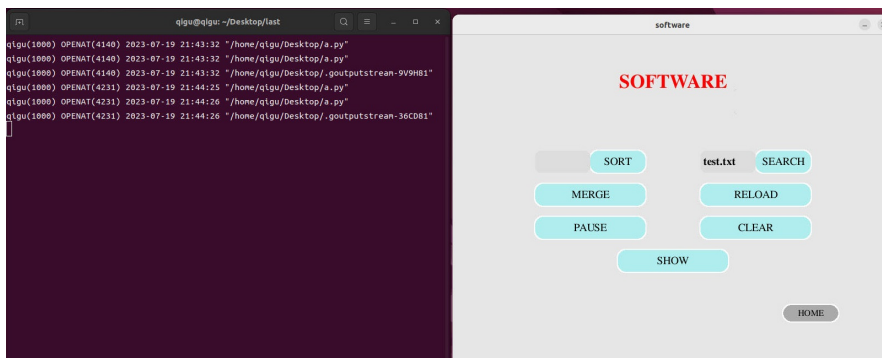


图 5-15 图形化控制界面展示图

在 log 模块下，可以进行排序、搜索等操作，结果如图5-16、5-17：

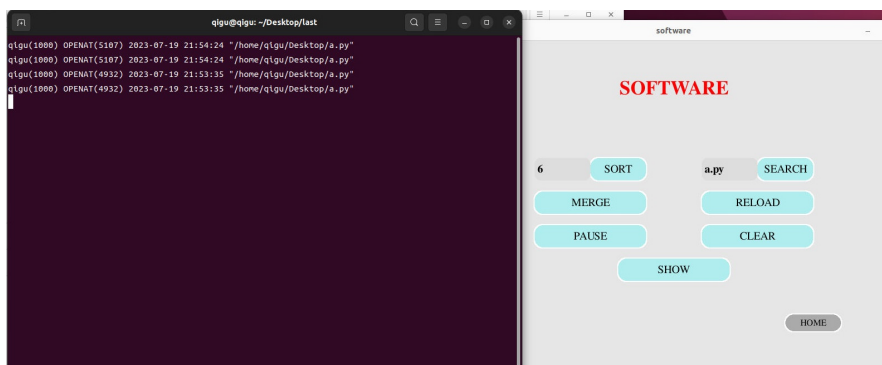


图 5-16 图形化日志模块展示图-搜索

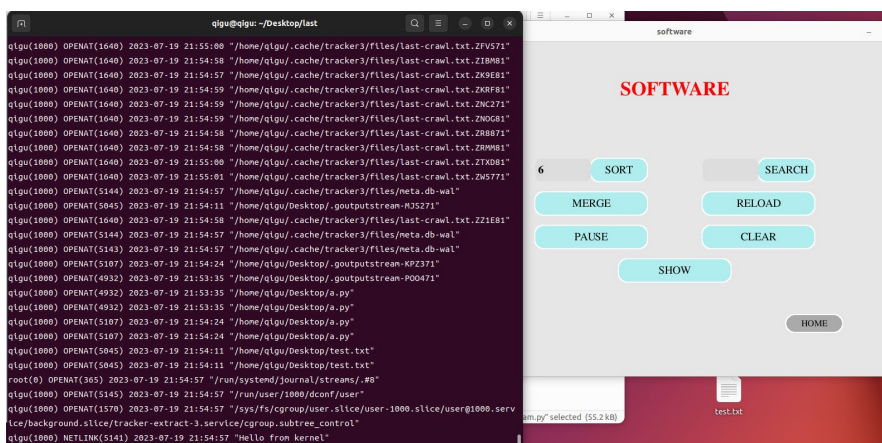


图 5-17 图形化日志模块展示图-排序

基本功能均能正常进行。

### 5.6 结论

经过测试和试用，我们项目表现出了令人满意的效果。通过提供全面的系统级资源访问审计，我们能够对系统中的各项操作进行准确的记录和审计。这使得我们能够及时识别和应对潜在的安全威胁，保护系统和数据的完整性和机密性。

## 第六章 项目小结

### 6.1 项目实施情况

整个项目的进行过程较为顺利，上述相关工作均在总体计划的时间内完成，达到了预期的目标和质量标准。小组召开会议，及时沟通项目进展和相关问题，保证了项目的顺利进行和成功完成。

### 6.2 代码管理

整个项目采用 github 进行代码管理，利用 github 的 Pull Request 和 workflow 进行任务的分工与代码的检测。在整个团队的共同努力下，整个项目经过了 56 次 commit 和 4 次 PR，成功的开发了 okftools 包，实现了 linux 内核函数的重载功能。

当前 okftools 最新版本为 v0.0.6

Github 源代码链接: <https://github.com/heatingma/NIS3302-okftools>

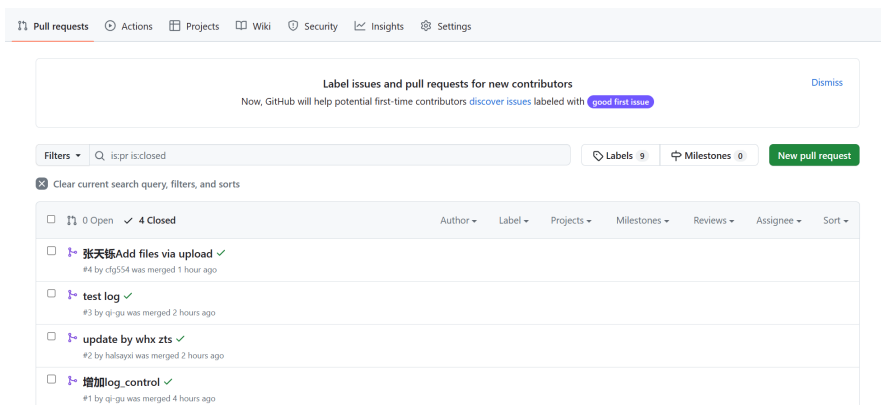


图 6-1 Pull Request

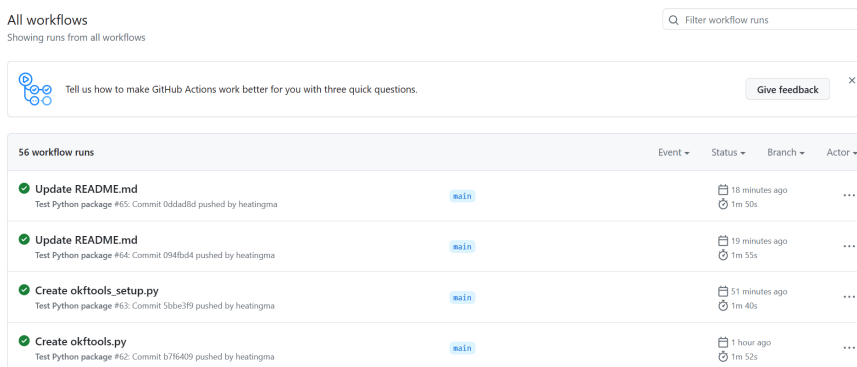


图 6-2 workflow

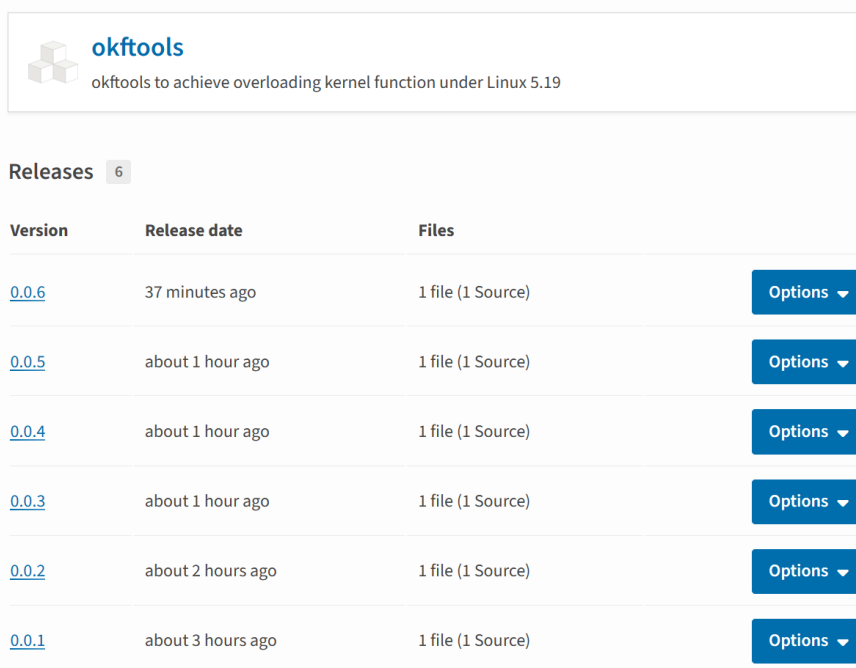


图 6-3 okftools

## 6.3 不足和展望

在本项目的基础上，按照资源访问审计的一般功能，本项目还有如下这些未来可以开发的功能：

1. 安全机制：可以加入权限控制，或恶意代码注入的防止等功能，避免数据丢失。
2. 异常处理：增加对异常情况的处理及交互，如内存泄漏，存储空间不足等情况的处理。
3. 日志分析：添加日志分析工具，如用户行为分析工具等，进一步提升用户使用体验。
4. reboot 和 shutdown 的重载实现地不理想。

## 6.4 体会、感受与建议

结束了本次信息安全科技创新的课程，我们感到受益良多。本次的项目引发了我们对系统安全和资源保护的深入思考：在现代计算机系统中，任何资源的访问都必须受到记录和控制，以确保系统的安全性和可靠性。而通过系统调用重载，我们可以在系统内核层面对资源进行审计和控制，有助于发现未经授权的行为，提高系统的安全性和可信度。在未来的学习里，我们可以结合本项目掌握相关知识，进一步理解资源保护的原理和实现。

在今后的学习中，希望增加关于课程项目具体实现方面的内容的讲解，加强理论与实际操作的结合。深入了解和掌握信息安全技术的实际应用，培养解决实际问题的能力，并将理论知识转化为实际技能。

## 6.5 附：项目组成员贡献表

成员姓名	是否组长	具体承担任务	组长评分
张天铄	是	内核函数设计与实现、脚本编写、代码调试	20
麻家乐		整体框架设计、内核函数与 netlink 机制的设计实现、GitHub 代码管理	20
王梓睿		日志控制设计与实现、脚本编写、代码测试	20
汪何希		内核函数设计与实现、脚本编写、代码调试	20
孙靖轩		内容整理与结题报告撰写	20

补充：本项目实现了图形化界面以及 okftools 包的开发与维护，完成者为麻家乐。

## 附录 A mount、umount 函数测试脚本

```
1  int main() {
2      char *file = "file.img";
3      char *dir = "/mnt/myfile";
4      char cmd[256];
5      bool mounted = false;
6
7      mounted = is_mounted(file, dir);
8      if (mounted) {
9          printf("File system is already mounted, unmounting...\n");
10         sprintf(cmd, "sudo umount %s", dir);
11         system(cmd);
12     }
13
14     sprintf(cmd, "sudo mount -o loop %s int ret = system(cmd);
15     if (ret == 0) {
16         printf("Mount succeeded\n");
17     } else {
18         printf("Mount failed: %d", ret);
19     }
20
21     return 0;
22 }
```