

华中科技大学

《软件安全课程设计》

报告

基于 API HOOK 的软件行为
分析系统

姓 名：邬雪菲

学 号：U202112131

专业班级：网安 2104 班

指导教师：周威

日 期：2024 年 4 月 14 日

评分标准		
1	个人与团队 (10 分)	<p>1 具有团队意识和团队能力,能够在多学科背景下的团队中承担个体、团队成员以及负责人的角色。</p> <p>2 参与小组安排的交流、讨论;能够理解团队的意义,了解团队的角色,在团队中主动与其他成员沟通、合作、开展工作;或承担成员角色,完成个体工作;优 9-10 分;良 8 分;中 7 分;及格 6 分;不及格 0-5 分。教师根据答辩中分工、报告中团队相关介绍,给出该项成绩。</p>
2	设计解决方案 (40 分)	<p>能够针对信息安全领域特定需求,利用数学模型和工具,规划和设计完整的系统。包括:对行为分析需求任务的分析;初步总体设计、各模块的详细设计;能在设计环节体现创新意识,考虑社会、安全、法律、文化、环境等因素。优(34-40 分);良(30-33);中(26-29 分);及格(24-25 分);不及格(0-23 分)。教师根据学生答辩中思路讲解环节、提交的报告中设计内容,综合给出该项成绩。</p>
3	实现解决方案 (40 分)	<p>根据设计内容,能开发、实现系统的功能。按任务要求完成内容 1、框架设计、行为截获 2、行为分析 3、交互界面 4、测试样本 5、创新。</p> <p>优(34-40 分);良(30-33);中(26-29 分);及格(24-25 分);未完成的项目设计的,不及格(0-23 分)。教师根据小组答辩中演示环节、报告中成果展示部分、最终实现源代码等,综合给出该项成绩。</p>
4	报告规范 (10 分)	<p>具有良好的人文社会科学素养、社会责任感,能够在工程实践中理解并遵守工程职业道德和规范,履行责任。报告结构合理,语言通顺、流畅;按华中科技大学本科毕业论文格式规范,字体、段落、图片、表格的版面美观、对他人资料列入参考文献并标注引用。非常规范,优(9-10 分);少量不规范,(1-3 项)良(8 分);不规范项目(4-6 项),中(7 分);不规范项目(7-9 项),及格(6 分),不规范项目(10 项以上)(0-5 分)。教师根据报告,给出该项成绩。</p>

评分表

个人与团队 (10%)	设计解决方案 (40%)	实现解决方案 (40%)	报告规范(10%)	成绩	教师签名

摘 要

本次课程设计是基于 Detours 开源项目实现的 win32 API 截获与异常行为分析，通过 Gitee 平台小组协作完成了一个具有图形界面的 API HOOK 行为分析客户端。

课设目的是在理解微软 Detours 开源库的工作原理基础上，借助该库开发一个 API 截获和行为异常分析程序，以此锻炼学生的安全思维和自主开发能力。并通过小组协作形式，让学生切身体会项目构建和管理的实践过程，更好地培养对项目整体的把握和设计能力、团队协作能力。

本组通过 Gitee 合作实现了 API HOOK 行为检测分析项目。后端部分使用 c++ 语言开发，主要包括注射器、DLL 和服务端程序三部分，在完成整体框架的基础上添加了堆、文件、注册表、网络通信、内存拷贝等 API 截获与异常行为分析，并构建检测样本库完成测试。前端部分采用 QT 开发实现 GUI，最终完成了一个行为分析检测客户端。本人主要负责对文件操作、网络通信操作的 API 截获和异常行为分析，并编写对应样本完成测试，在实验过程中不仅了解了 Detours 开源项目的原理和使用方式，而且对安全分析加深了知识理解和实践性体会。

目 录

摘 要	III
目 录	IV
1 课程设计任务书	1
1.1 课程设计目的	1
1.2 课程设计的要求	1
1.3 系统环境	3
1.4 实验过程记录	3
1.5 团队分工与个人负责内容说明	4
2 绪言	5
2.1 API 截获与分析背景	5
2.2 Detours 开源库介绍	5
3 系统方案设计	6
3.1 系统需求分析	6
3.2 系统模块与结构设计	6
4 系统实现	10
4.1 钩子 DLL	10
4.2 测试程序	16
5 系统测试	18
5.1 客户端测试	18
5.2 文件操作测试	20
5.3 网络通信操作测试	22
6 总结与展望	25
6.1 总结	25
6.2 展望	26
自评及反馈	27
参考文献	29

1 课程设计任务书

1.1 课程设计目的

本次课程设计的目的是利用 Detours 开源项目包提供的接口，在无源码情况下对程序进行行为分析，通过小组协作完成基本的第三方进程 windowsAPI 截获框架，并添加对堆、文件、注册表、内存等方面的 API 截获与行为异常分析，最终实现一个完整的进程分析项目。

1.2 课程设计要求

对于无源码情况下分析样本程序的行为，有多种方法。本次课程设计是利用 Detours 开源项目包提供的接口，完成基本的程序行为分析。具体课程设计任务见表 1。任务主要分为 API 调用截获及分析两大部分。编程语言为 C 或者 C++。使用的 Detours 开源库可以在 VS2019 环境下编译后使用。

表 1 课程设计任务表

序号	任务	要求
1	实现基本的第三方进程 WindowsAPI 截获框架	框架包括 1.1 编译生成 Detours 库；1.2 完成挂钩框架 DLL，实现对 MessageBox 调用截获，能打印出调用的参数、进程名称以及进程 Exe 文件信息；1.3 自编或者利用已有恶意代码样例（包含弹出对话框动作）；1.4 完成注入动作开启和关闭的“注射器”控制程序
2	实现堆操作 API 截获	修改 1.2-1.4，实现堆操作（创建，释放）API 进行截获，打印出所有参数信息。
3	实现文件操作 API 截获	实现对文件操作（创建，关闭，读写）API 进行截获，打印出所有参数信息。
4	注册表操作 API 截获	实现对注册表操作（创建，关闭，读写）API 进行截获，打印出所有参数信息。

5	堆操作异常行为分析	设计并完成算法，记录并给出提示： 1. 检测堆申请与释放是否一致（正常）； 2. 是否发生重复的多次释放（异常）
6	文件操作异常行为分析	设计并完成算法，记录并给出提示： 1. 判断操作范围是否有多个文件夹； 2. 是否存在自我复制的情况； 3. 是否修改了其它可执行代码包括 exe, dll, ocx 等； 4. 是否将文件内容读取后发送到网络（选做）；
7	注册表操作异常行为分析	设计并完成算法，记录并给出提示： 1. 判断是否新增注册表项并判断是否为自启动执行文件项； 2. 是否修改了注册表； 3. 输出所有的注册表操作项；
8	提供系统界面	所设计实现的功能，有图形界面展示
9	行为检测样本库	提供 5 个待检测的可能存在恶意的 Exe 样本；
10	项目开源	在 Gitee 平台上开源各自完成的行为分析系统，小组成员通过 Gitee 协同开发
11	项目文档撰写	在开源项目中编写项目构建与使用文档
12	在指定的开源仓库完成开源贡献	在以下开源仓库中提交 PR： (1) Gitee 的 HustDetours（待公布具体网址） (2) 软件安全课程平台的基础架构仓库 (https://github.com/HUSTSecLab/dojo) (3) 理论课实验课实验仓库 (https://github.com/HUSTSecLab/software-security-dojo)
13	网络通信操作异常行为分析（选做）	设计并完成算法，记录并给出提示： 1. 实现对网络传输 SOCKET 操作（连接、发送与接收）API 的截获； 2. 打印进程连接端口、协议类型、IP 信息 3. HTTP 连接协议的解析，判断传输的内容是否为明文
14	内存拷贝监测与关联分析（选做）	设计并完成算法，记录并给出提示： 能够输出内存拷贝信息，并分析拷贝的内容流向。

1.3 系统环境

开发与测试环境：

Window11

Visual Studio2022

QT Creator

本次课设在安装配置微软 Detours 库基础上实现，后端使用 c++语言 win32 编程实现基本功能，前端基于 QT 封装图形化界面。项目程序和待测程序均为 32 位应用程序。

1.4 实验过程记录

课设任务下发后，小组首先是对项目进行整体理解和把握，在了解 Detours 库实现原理和使用方法后，我们将整个项目分为前端和后端两个板块，完成了后端的截获分析框架后，主要根据后端 DLL 程序中截获行为的不同分类进行分工。

具体来说，为了更好地实现协作，我们的分工相对独立，各自部分的相互约束和影响较小。对于前端和后端的连接，前端的客户端主体直接获取后端进程的输出生成统计、分析和回显，工作相对集中，由一个同学完成。而后端部分则根据截获行为的特征分为较为独立的几个部分，各自完成 DLL 项目中对应方向的头文件，

本人主要负责的是文件和网络通信操作的异常行为分析，并进行对应的功能测试。

在 API 截获方面遇到的问题不多，Detours 极大地简化了这个过程。查询 win32API 函数原型并按照 Detours 库原则在 DLL 中添加截获即可。

异常行为分析部分则不太顺利，首要的问题就是课设任务的理解，这极大地影响了最终实现，不同的理解会导向完全不同的解决方式。

其次是具体的算法设计难度大。由于异常分析是在 API 截获上进行的，但某些行为同时涉及多个 API 函数，要想完成分析只修改自己的头文件是不行的，必须要在基本框架和服务端进行功能添加和修改，这就打破了原本较为独立的分工模块，对模块间的协调同步提出了较大挑战。比如判断进程是否将文件读取后发送到网络，就是文件和网络通信在时间上的一个先后关联操作。我最初的想法是在服务器端增加一个统计模块，

统计时间维度上的 API 调用情况，并将该异常行为的分析交由服务器完成，即如果读取文件和发送消息这两个行为是先后进行的则认为是异常。这种方式显然过于简便，且误判率高。之后想到的一个改进方式是对读取的文件内容进行记录，每次截获到发送消息的时候判断是否为己读取内容。不过最后并未能完成这个改进方案。

此外还有一些实验测试过程的细节问题，将在之后的系统实现和系统测试模块中具体介绍。

1.5 团队分工与个人负责内容说明

1. 团队分工：

李婧瑶：实现基本第三方进程 WindowsAPI 截获框架，内存拷贝监测与关联分析

王婧薇：实现堆操作 API 截获与异常行为分析，提供行为检测样本库

邬雪菲：实现文件操作 API 截获与异常行为分析，网络通信操作异常行为分析

陈欣然：提供系统界面，开发客户端，服务端，注射器程序，并进行样本测试

胡展豪：实现注册表 API 截获与异常行为分析，整合分支并审计代码提交，分配任务，撰写项目文档

2. 个人负责内容说明：

本人负责文件和网络通信模块的 API 截获与异常分析，并完成对应的测试，主要添加修改了项目 DLL 中的 Base.h、File.h、Socket.h 和项目 test_app。

2 绪言

2.1 API 截获与分析背景

API 截获与分析是一种重要的技术手段，用于监控和干预应用程序的行为。这项技术的应用范围广泛，涵盖了软件逆向工程、安全审计、行为分析等领域。

在软件开发过程中，开发者常常需要了解应用程序与操作系统、硬件等底层资源之间的交互过程，以便进行性能优化、错误调试等工作。而 API 截获技术能够提供一个非常便捷的途径，帮助开发者捕获应用程序与系统之间的交互细节。另一方面，安全领域也对 API 截获与分析技术有着强烈的需求。恶意软件常常利用 API 来与系统进行交互，执行恶意操作，如窃取敏感信息、植入后门等。通过对 API 的截获与分析，安全研究人员可以及时发现恶意软件的行为，并采取相应的防御措施，保护系统的安全。

2.2 Detours 开源库介绍

微软 Detours 是一款强大的开源库，用于在 Windows 操作系统上进行 API 截获和重定向。其核心原理是通过修改目标函数的入口点，将其重定向到一个用户自定义的截获函数，从而实现对目标函数的拦截和重定向。Detours 的独特之处在于，它不直接修改目标函数的二进制代码，而是通过将截获函数插入到目标函数的执行流程中来实现截获功能。这种设计保证了在截获过程中不会影响到其他进程或系统对目标函数的调用，从而提高了系统的稳定性和可靠性。

Detours 还提供了丰富的功能和灵活的使用方式，可以方便地编辑 DLL 导入表，向二进制代码中注入任意数据节表，甚至向正在运行的进程中注入 DLL。这些功能使得 Detours 成为了一个强大而灵活的工具，可以应用于各种不同的场景，包括软件开发、安全研究、系统调试等。通过 Detours 库，研究人员可以更加方便地进行 API 截获与分析，并加强对系统安全的监控和防护。

3 系统方案设计

3.1 系统需求分析

根据课程设计任务表，可知最终实现的程序客户端应能够选择目标测试程序，完成对若干 API 的截获和异常行为分析，并回显截获信息与分析结果，进行更深的统计分析。在此基础上还可以添加一些扩展功能，如添加更多的行为异常分析、对不同的程序可选择不同的截获 API 配置等等。

3.2 系统模块与结构设计

基于需求分析与任务表，本组将系统模块分为客户端、注射器、DLL 中的钩子函数、统计分析服务器模块四个部分，系统架构如图 3.1 所示：

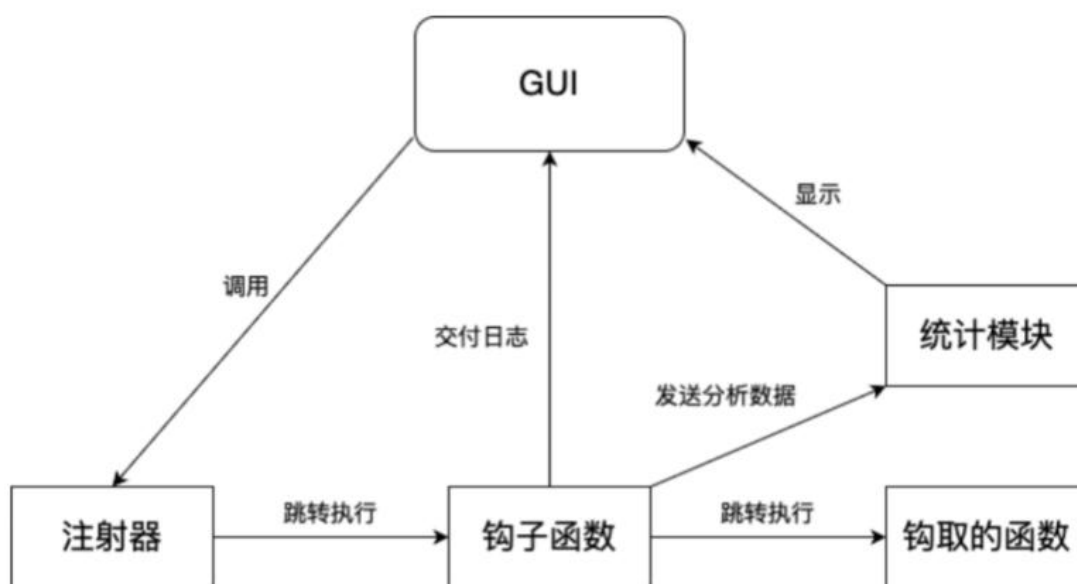


图 3.1 系统架构图

从前端后端的角度看，前端即客户端，负责实现人机交互的 GUI 图形化界面；后端即注射器、DLL 钩子函数和统计分析服务器，由注射器完成对目标程序的 DLL 注入，DLL 钩子中实现对基本 API 的截获与大部分异常行为分析，服务器负责进一步的统计和分析。具体的模块介绍将在子节中阐述。

3.2.1 注射器

注射器模块是一个控制台程序,利用 DetourCreateProcessWithDllEx 函数完成开启和关闭注入 DLL 的操作,是用户截获程序控制流的开始。目标测试文件路径由前端客户端根据用户选择或输入获取,并作为参数启动注射器。

3.2.2 DLL 钩子函数

Detours 的原理是向目标进程注入 DLL,而这个 DLL 动态库可以实现对相关 API 的截获与分析,即将控制流直接转移到用户自己实现的截获函数中,而原目标函数中被替换的指令被保存在 Trampoline 函数中。故 DLL 是本项目的功能核心,所有的 API 截获与大部分异常行为分析都是在 DLL 中实现的。由注射器程序将 DLL 注入目标进程中。

DLL 文件分为两大部分,分别是钩子主函数以及截获和异常分析函数群。

1) 钩子主函数

钩子主函数即 DetourAttach 和 DetourDettach 函数,它们分别完成挂钩和解除挂钩的操作,以将截获函数插入目标代码之中。这部分构建了基本的 API HOOK 截获框架。

2) 截获函数

截获函数是需要 HOOK 的函数的修改替代,Detours 通过将 API 函数替换为用户自定义的截获函数,获取 API 调用参数的同时进行行为分析。

本项目针对不同类别的待钩取函数编写了不同的头文件用以具体函数的编写。截获函数对目标 API 操作进行截获,并将相关参数打印到终端显示。同时,截获函数中包含了对程序行为的异常分析,通过分析 API 调用的参数可判断是否为异常行为,并将结果发送给用于统计分析的服务器程序。最终要实现了对堆、文件、注册表、网络通信操作的 API 截获与异常行为分析。

本人主要负责的是文件和网络通信部分。对于基本的行为截获,只需要添加 CreateFile, CloseHandle, ReadFile, WriteFile, CopyFile 以及 WSAConnet, WSA Send, WSARecv 这些 API 的截获函数。对于异常分析,设计思路具体如下:

a) 判断操作范围是否存在多个文件夹

文件相关操作总需要提供实际的文件路径,可在截获函数中添加获取当前操作路径

的操作，通过递归遍历当前路径，即可计算出路径下的文件夹数目，得出判断结果。另一种理解方式是对于单个进程，它完成的若干文件操作是否是在不同路径下进行的，这可以借助服务器统计模块实现，每次截获到文件操作就发送路径参数至服务器，当进程结束时，由服务器对路径进行统计分析，判断是否存在多个不同路径。

b) 是否存在自我复制

自我复制有多种情况。若待测目标有可能直接使用 CopyFile API 进行复制，也有可能不通过 CopyFile 实现复制，但程序总需要创建文件并对自身内容进行读取。故判断逻辑可按如下设计：

添加对 CopyFile 函数的截获，并在其中判断要复制的目标文件是否为进程本身，是则认为进行了自我复制；

CreateFile 中添加对新建文件名是否和进程自身文件名相同的判断，是则说明进程进行了自我复制且复制程序名仍和原名称一致；

ReadFile 中判断进程是否尝试读取自身内容，是则认为进行了自我复制，可以检测出不直接使用 API 复制、部分复制、复制文件名不同等情况。

c) 是否修改了可执行程序

修改操作涉及写权限，可执行程序可根据后缀名判断，故只需要在截获函数中增加相关判断代码即可。

d) 是否将文件内容读取后发送到网络（选做）

每截获到一次 API 便向服务器发送截获到的 API 信息，服务器通过分析前后两次 API 是否分别为读取文件和发送消息操作来判断异常。

e) 打印进程连接端口、协议类型、IP 信息（选做）

WSAConnet, WSASend, WSAREcv 等截获函数中对参数进行解析并打印即可。

f) HTTP 连接协议解析（选做）

HTTP 协议不提供加密保护，用户发送信息的隐私无法保障，而 HTTPS 协议中则提供了加密。通过解析数据包头中的连接协议版本即可判断是否为 HTTP 协议。

3.2.3 服务器程序

服务端程序主要用于接收 API 截获情况信息和异常分析信息进行进一步统计分析，并将结果送回客户端进行显示。

3.2.4 客户端

客户端主体是整个项目的控制核心，通过 QT 开发实现 GUI 界面以便与用户交互。用户在客户端上选择需要分析的程序，客户端将调用注射器模块将 HOOK DLL 注入目标程序并启动分析。DLL 中的截获和分析函数会将运行信息打印到终端，客户端可以直接获取进程的输出信息并显示在 GUI 的日志模块中，同时接收服务器的统计分析结果进行显示。

可以考虑将客户端作为父进程，注射器作为子进程进行，父进程可获取子进程的输出并显示。

3.2.5 检测样本库

对于每个类型的 API 截获与异常行为分析，在完成功能代码编写的过程中也会构建专门的测试程序进行测试，此外还专门构建和收集了一些测试样本，每个样本包含多种 API 调用与异常行为。

4 系统实现

由于本人负责的模块主要涉及 DLL 部分，注射器部分相对原仓库文件改动不大，主要是在客户端中完成对目标程序的选择和启动注射器子进程，而客户端和服务端并非本人负责的内容，其中的代码实现无法详细地全部介绍，故在此仅重点介绍 DLL 部分。

4.1 钩子 DLL

DLL 中通过调用 DetourAttach(&(PVOID&)OldFunction, Newfunction) 实现对目标进程函数的钩取和替换，DetourDetach(&(PVOID&)OldFunction, Newfunction) 解除钩子。

```
case DLL_PROCESS_ATTACH: {
    DisableThreadLibraryCalls(hModule);
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());
    //MessageBoxHookApiAttach
    DetourAttach(&(PVOID&)OldMessageBoxW, NewMessageBoxW);
    DetourAttach(&(PVOID&)OldMessageBoxA, NewMessageBoxA);
    DetourAttach(&(PVOID&)OldHeapCreate, NewHeapCreate);
    DetourAttach(&(PVOID&)OldHeapDestroy, NewHeapDestroy);
    //HeapOperationHookApiAttach
    DetourAttach(&(PVOID&)OldHeapFree, NewHeapFree);
    DetourAttach(&(PVOID&)OldHeapAlloc, NewHeapAlloc);
    //FileOperationHookApiAttach
    DetourAttach(&(PVOID&)OldCreateFile, NewCreateFile);
    DetourAttach(&(PVOID&)OldCloseHandle, NewCloseHandle);
    DetourAttach(&(PVOID&)OldReadFile, NewReadFile);
    DetourAttach(&(PVOID&)OldWriteFile, NewWriteFile);
    //RegOperationHookApiAttach
    DetourAttach(&(PVOID&)OldRegCreateKeyExW, NewRegCreateKeyExW);
    DetourAttach(&(PVOID&)OldRegSetValueExW, NewRegSetValueExW);
    DetourAttach(&(PVOID&)OldRegQueryValueExW, NewRegQueryValueExW);
    DetourAttach(&(PVOID&)OldRegCloseKey, NewRegCloseKey);
    DetourAttach(&(PVOID&)OldRegOpenKeyExW, NewRegOpenKeyExW);
    DetourAttach(&(PVOID&)OldRegDeleteValueW, NewRegDeleteValueW);
    //WinSocketHookApiAttach
    DetourAttach(&(PVOID&)OldWSAConnet, NewWSAConnet);
    DetourAttach(&(PVOID&)OldWSASend, NewWSASend);
    DetourAttach(&(PVOID&)OldWSARecv, NewWSARecv);
    //MemcpyHookApiAttach
    DetourAttach(&(PVOID&)OldMemcpy, NewMemcpy);
    DetourTransactionCommit();
    break;
}
```

最终实现的 API 截获和异常行为分析如下宏定义所示，由服务器端统计分析并回显到客户端。

<pre> 3 // API截获***** 4 // 弹窗操作----- 5 #define _MessageBox 1 6 #define _CreateKey 2 7 // 堆操作----- 8 #define _HeapCreate 3 9 #define _HeapDestroy 4 10 #define _HeapAlloc 5 11 #define _HeapFree 6 12 // 文件操作----- 13 #define _CreateFile 7 14 #define _CloseHandle 8 15 #define _WriteFile 9 16 #define _CopyFile 10 17 #define _ReadFile 11 18 // 注册表操作----- 19 #define _RegCreateKeyExW 12 20 #define _RegSetValueExW 13 21 #define _RegQueryValueExW 14 22 #define _RegCloseKey 15 23 #define _RegOpenKeyExW 16 24 #define _RegDeleteValueW 17 </pre>	<pre> 25 // 网络通信操作----- 26 #define _WSAConnet 18 27 #define _WSASend 19 28 #define _WSARecv 20 29 // 内存拷贝操作----- 30 #define _NewMemcpy 21 31 // 异常行为分析***** 32 #define _SetKeyValue 22 33 #define _ModifyReg 23 34 #define _CreateSelfBoot 24 35 #define _CreateKeyValue 25 36 #define _MultiFree 26 37 #define _MultiDir 27 38 #define _ChangeEXE 28 39 #define _CopySelf 29 40 #define _SendFileToNet 30 41 #define _HttpPlainText 31 42 // 进行分析后调用该函数，参数为异常种类对应的宏，如 43 bool CreatingPipeAndSendingMessage(int type); </pre>
---	--

4.1.1 API 截获函数框架

API 截获函数均采用同样的框架，先定义被替换的原函数和替换的新函数，注意原函数的定义应查询 Windows API 手册，且原函数和替换函数的参数需要一致。在替换函数中，首先需要调用原函数，然后可以添加参数获取和打印操作，以及异常分析操作。举例来说，WSAconnect 函数的截获部分如下：

```

12 // WSAconnect连接操作-----
13 static int(WINAPI *OldWSAConnet)(
14     SOCKET s, // 标识未连接套接字的描述符
15     const struct sockaddr FAR *name, // 指向指定要连接到的地址的 sockaddr 结构的指针
16     int namelen, // sockaddr 结构的长度（以字节为单位）
17     LPWSABUF lpCallerData, // 指向在连接建立期间要传输到另一个套接字的用户数据的指针
18     LPWSABUF lpCalleeData, // 指向在建立连接期间要从另一个套接字传回的用户数据的指针
19     LPQOS lpSQOS, // 指向套接字 s 的 FLOWSPEC 结构的指针
20     LPQOS lpGQOS) = WSAConnect; // 指向套接字组的 FLOWSPEC 结构的指针
13 Codiumate: Options | Test this function
21 extern "C" __declspec(dllexport) int WINAPI NewWSAConnet(
22     SOCKET s,
23     const struct sockaddr FAR *name,
24     int namelen,
25     LPWSABUF lpCallerData,
26     LPWSABUF lpCalleeData,
27     LPQOS lpSQOS,
28     LPQOS lpGQOS)
29 {
30     int ret = OldWSAConnet(s, name, namelen, lpCallerData, lpCalleeData, lpSQOS, lpGQOS);

```


4.4.2 文件操作截获与分析

对于文件操作，我实现了对 CreateFile, CloseHandle, ReadFile, WriteFile, CopyFile 函数的截获。

对于异常行为分析，我编写了两个函数来分别获取当前进程路径与路径中的文件名，便于分析时使用，具体如下：

```
// 获取当前进程路径
extern "C" __declspec(dllexport) BOOL GetNowProcessPath(WCHAR *lpModuleName)
{
    if (!GetModuleFileNameW(NULL, lpModuleName, MAX_PATH))
    {
        printf("Failed to get nowprocesspath!\n");
        return FALSE;
    }
    return TRUE;
}
```

```
// 获取文件名
// 把lpPathName的末尾（最后一个'\\'之后）放到lpFileName
extern "C" __declspec(dllexport) int GetFileName(const wchar_t *lpPathName, wchar_t *lpFileName)
{
    lpFileName[0] = 0;
    for (int i = wcslen(lpPathName) - 1; i >= 0; i--)
        if (lpPathName[i] == '\\')
        {
            wcscpy_s(lpFileName, MAX_PATH, &lpPathName[i + 1]);
            break;
        }
    return wcslen(lpFileName);
}
```

此外，为了防止注射器程序自身加载 DLL 导致自身 API 被截获，在截获函数中增加判断操作看目前程序是否为注射器程序，不是则进行截获。

```
// 获取当前进程的文件名
GetNowProcessPath(ProcessPath);
GetFileName(ProcessPath, FileName);
if (wcscmp(FileName, INJECTEXE)) // 过滤器设置
{
    if (GetFileType(hFile) == FILE_TYPE_DISK) // 过滤磁盘文件

```

1) 判断操作范围是否存在多个文件夹

通过递归遍历获取当前路径的子文件夹个数，如果大于 2 则认为操作范围内存在多个文件夹。具体来说，通过调用 NumberOfSubfolders 函数实现。函数将通过 for 循环

去掉绝对路径的文件名，即得到文件夹路径，然后补充“*.*”并循环调用 FindFirstFile 函数搜索该文件夹中的所有文件，如果搜索到的是文件夹则计数器增加。具体代码实现如下：

```
34 extern "C" __declspec(dllexport) int NumberOfSubfolders(LPCWSTR lpFileName)
35 {
36     WIN32_FIND_DATA fd;
37     HANDLE hFirst;
38     int count = 0;
39     WCHAR tempPath[MAX_PATH];
40     wcsncpy_s(tempPath, lpFileName);
41     for (int i = wcslen(tempPath); i; i--){
42         if (tempPath[i] == L'\\'){
43             tempPath[i] = 0;
44             break;
45         }
46     }
47     wscat_s(tempPath, L"\\*.*");
48     for (hFirst = FindFirstFile(tempPath, &fd); hFirst != INVALID_HANDLE_VALUE && FindNextFile(hFirst, &fd);){
49         if (fd.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY && fd.cFileName[0] != '.')
50             count++;
51     }
52     return count;
53 }
```

2) 是否修改了可执行程序

修改操作涉及写权限，可执行程序可根据后缀名判断，故只需要在截获函数中增加相关判断代码即可。即通过 wcsstr 函数判断后缀“.exe”、“.dll”、“.ocx”三种字符串是否在文件路径中出现，以及是否有文件写权限，可以判断是否在修改可执行文件。代码实现如下：

```
// 检查是否试图修改可执行程序—目标文件为可执行文件且文件访问权限拥有写权限
if ((wcsstr(lpFileName, L".exe") || wcsstr(lpFileName, L".dll") || wcsstr(lpFileName, L".ocx")) && (dwDesiredAccess & GENERIC_WRITE))
{
    cout << "*****" << endl;
    cout << "ERROR: 可能正在试图修改可执行程序" << endl;
    cout << "*****" << endl;
    CreatingPipeAndSendingMessage(_ChangeEXE);
    Sleep(1);
}
```

3) 是否存在自我复制

自我复制有多种情况。若待测目标有可能直接使用 CopyFile API 进行复制，也有可能不通过 CopyFile 实现复制，但程序总需要创建文件并对自身内容进行读取。故判断逻辑按如下设计：

a) 添加对 CopyFile 函数的截获，并在其中判断要复制的目标文件是否为进程本身，是则认为进行了自我复制。代码实现如下：

```
GetFileName(lpExistingFileName, CopiedFileName);
// 复制文件名当前进程文件名相同
if (FileName != NULL && !wcscmp(FileName, CopiedFileName)){
    cout << "*****" << endl;
    cout << "ERROR: 可能正在试图复制自身" << endl;
    cout << "*****" << endl;
    CreatingPipeAndSendingMessage(_CopySelf);
}
```

```
static BOOL(WINAPI *OldCopyFile)(
    LPCTSTR lpExistingFileName, // 源路径
    LPCTSTR lpNewFileName,      // 目标路径
    BOOL bFailIfExists          // 是否覆盖文件
) = CopyFile;
extern "C" __declspec(dllexport) BOOL WINAPI NewCopyFile(
    LPCTSTR lpExistingFileName,
    LPCTSTR lpNewFileName,
    BOOL bFailIfExists)
```

b) CreateFile 中添加对新建文件名是否和进程自身文件名相同的判断，是则说明进程进行了自我复制且复制程序名仍和原名称一致。代码实现如下：

```
// 有读权限且新建文件名与当前进程文件名相同
if (dwDesiredAccess & GENERIC_READ && FileNameAddress != NULL && (FileNameAddress == lpFileName || *(FileNameAddress - 1) == L'\\'))
{
    cout << "*****" << endl;
    cout << "ERROR: 可能正在试图复制自身" << endl;
    cout << "*****" << endl;
    CreatingPipeAndSendingMessage(_CopyShelf);
}
```

c) ReadFile 中判断进程是否尝试读取自身内容，是则认为进行了自我复制，可以检测出不直接使用 API 复制、部分复制、复制文件名不同等情况。

```
// 读取文件名与当前进程文件名相同
if (FileNameAddress != NULL && (FileNameAddress == lpFileName || *(FileNameAddress - 1) == L'\\'))
{
    cout << "*****" << endl;
    cout << "ERROR: 可能正在试图复制自身" << endl;
    cout << "*****" << endl;
    CreatingPipeAndSendingMessage(_CopyShelf);
}
```

4.4.3 网络通信操作截获与分析

对于网络通信操作，我添加了对 WSAConnet, WSASend, WSARecv 等函数的截获，并解析参数获取到进程的连接端口、协议类型、IP 等信息打印。部分代码实现如下：

```
12 // WSAconnect连接操作-----
13 static int(WINAPI *OldWSAConnet)(
14     SOCKET s, // 标识未连接套接字的描述符
15     const struct sockaddr FAR *name, // 指向指定要连接到的地址的 sockaddr 结构的指针
16     int namelen, // sockaddr 结构的长度（以字节为单位）
17     LPWSABUF lpCallerData, // 指向在连接建立期间要传输到另一个套接字的用户数据的指针
18     LPWSABUF lpCalleeData, // 指向在建立连接期间要从另一个套接字传回的用户数据的指针
19     LPQOS lpSQOS, // 指向套接字 s 的 FLOWSPEC 结构的指针
20     LPQOS lpGQOS) = WSAConnect; // 指向套接字组的 FLOWSPEC 结构的指针
21 extern "C" __declspec(dllexport) int WINAPI NewWSAConnet(
22     SOCKET s,
23     const struct sockaddr FAR *name,
24     int namelen,
25     LPWSABUF lpCallerData,
26     LPWSABUF lpCalleeData,
27     LPQOS lpSQOS,
28     LPQOS lpGQOS)
29 {
```

```

30     int ret = OldWSAConnect(s, name, namelen, lpCallerData, lpCalleeData, lpSQOS, lpGQOS);
31     char ip[MAX_NUM];
32     struct sockaddr_in *sockaddr_ipv4 = (struct sockaddr_in *)name;
33     // 获取IP信息并转化为可读格式
34     wchar_t ipWideString[MAX_NUM];
35     InetNtop(AF_INET, &sockaddr_ipv4->sin_addr, ipWideString, sizeof(ipWideString) / sizeof(ipWideString[0]));
36     size_t convertedChars;
37     wcstombs_s(&convertedChars, ip, ipWideString, sizeof(ip));
38     int port = ntohs(sockaddr_ipv4->sin_port);
39     string af = (sockaddr_ipv4->sin_family) == 2 ? "AF_INET" : "AF_INET6";
40     cout << "WSAconnect Hooked!" << endl;
41     cout << "*****" << endl;
42     cout << "Connect to IP: " << ip << endl;
43     cout << "Port: " << oct << port << endl;
44     cout << "Address family: " << af << endl;
45     DilLog();

```

对于 HTTP 连接协议的解析，则通过分析数据包头部的协议版本字段进行。因为 http 请求的头部字段跟数据字段是通过两组回车换行符来间隔的，故可以通过查找该间隔确定头部的结尾处，从而取出头部进行分析。代码实现在 WSARev 和 WSASend 截获函数中进行，具体如下所示：

```

159     // 解析HTTP请求的内容
160     char *dataBuffer = lpBuffers->buf;
161     int dataSize = *lpNumberOfBytesRecv;
162     // 查找HTTP头部的结束标识
163     char *headerEnd = strstr(dataBuffer, "\r\n\r\n");
164     if (headerEnd != NULL) // HTTP报文
165     {
166         // 获取数据包头部的内容
167         ptrdiff_t headerSize = headerEnd - dataBuffer;
168         char *header = new char[headerSize + 1];
169         strncpy(header, dataBuffer, headerSize);
170         header[headerSize] = '\0';
171         cout << "HTTP Header:\n"
172             << header << endl;
173         // 判断是否为HTTP传输
174         char *securityFlag = strstr(dataBuffer, "HTTP");
175         if (securityFlag != NULL){
176             cout << "*****" << endl;
177             cout << "The message is http data." << endl;
178             cout << "*****" << endl;
179         }
180         else{
181             cout << "*****" << endl;
182             cout << "The message is https data." << endl;
183             cout << "*****" << endl;
184         }
185         delete[] header;
186     }

```

4.2 测试程序

编写完成 HOOK 函数代码后，我自建样本程序进行测试。对于文件操作，参数的传入和函数调用较为简单，在此不做赘述。但是对于网络通信操作，需要熟悉网络连接和传输的 win32API 调用流程，才能构建出能正常运行的样本程序。且需要服务器和客户端两个程序进行联合测试。

1. 服务器通信流程：

- 1) 初始化 Winsock 库：使用 `WSAStartup` 函数初始化 Winsock 库，确保网络功能的正常运行。
- 2) 创建服务器监听套接字：通过 `socket` 函数创建一个用于监听客户端连接请求的套接字。
- 3) 绑定本地地址：将创建的套接字绑定到服务器的本地地址上。指定服务器的 IP 地址为本地任意地址（`INADDR_ANY`），并指定服务器监听的端口号。
- 4) 开始监听连接请求：使用 `listen` 函数开始监听客户端的连接请求，并指定同时允许连接的最大数量。
- 5) 接受客户端连接请求：使用 `accept` 函数接受客户端的连接请求，并获取客户端的地址信息。一旦有客户端连接成功，服务器将进入循环接收消息的阶段。
- 6) 接收消息：使用 `WSARecv` 函数循环接收客户端发送的消息，处理并响应客户端请求。
- 7) 关闭套接字：使用 `closesocket` 函数关闭服务器监听套接字和与客户端的通信套接字。
- 8) 清理 Winsock 库：使用 `WSACleanup` 函数清理 Winsock 库，释放相关资源。

2. 客户端通信流程：

- 1) 初始化 Winsock 库：使用 `WSAStartup` 函数初始化 Winsock 库，确保网络功能的正常运行。
- 2) 指定服务器地址：指定待连接服务器的地址信息，包括 IP 地址和端口号。
- 3) 创建客户端套接字：通过 `socket` 函数创建一个用于与服务器通信的套接字。
- 4) 发起连接请求：使用 `WSAConnect` 函数向服务器发起连接请求，建立与服务器

的连接。如果连接成功建立，则进入下一步操作。

5) 接收服务器消息：使用 `WSARecv` 函数接收服务器发送的消息，并显示在客户端控制台上。

6) 发送消息：根据客户端选择的操作，使用 `WSASend` 函数发送文本消息或调用自定义函数发送文件给服务器。

7) 关闭套接字：使用 `closesocket` 函数关闭客户端套接字。

8) 清理 Winsock 库：使用 `WSACleanup` 函数清理 Winsock 库，释放相关资源。

5 系统测试

系统实现的功能有很多，限于报告篇幅，在此仅展示最终实现客户端的部分功能展示，和我在编写负责的文件和网络通信操作时做的测试。

5.1 客户端测试

使用 QT 生成最后的项目文件夹 `release`，根目录下的 `client.exe` 文件即为最终的客户端可执行程序，可直接在 win11 环境下双击运行，无需额外的环境配置。进入客户端后的界面如下图 5.1 所示：

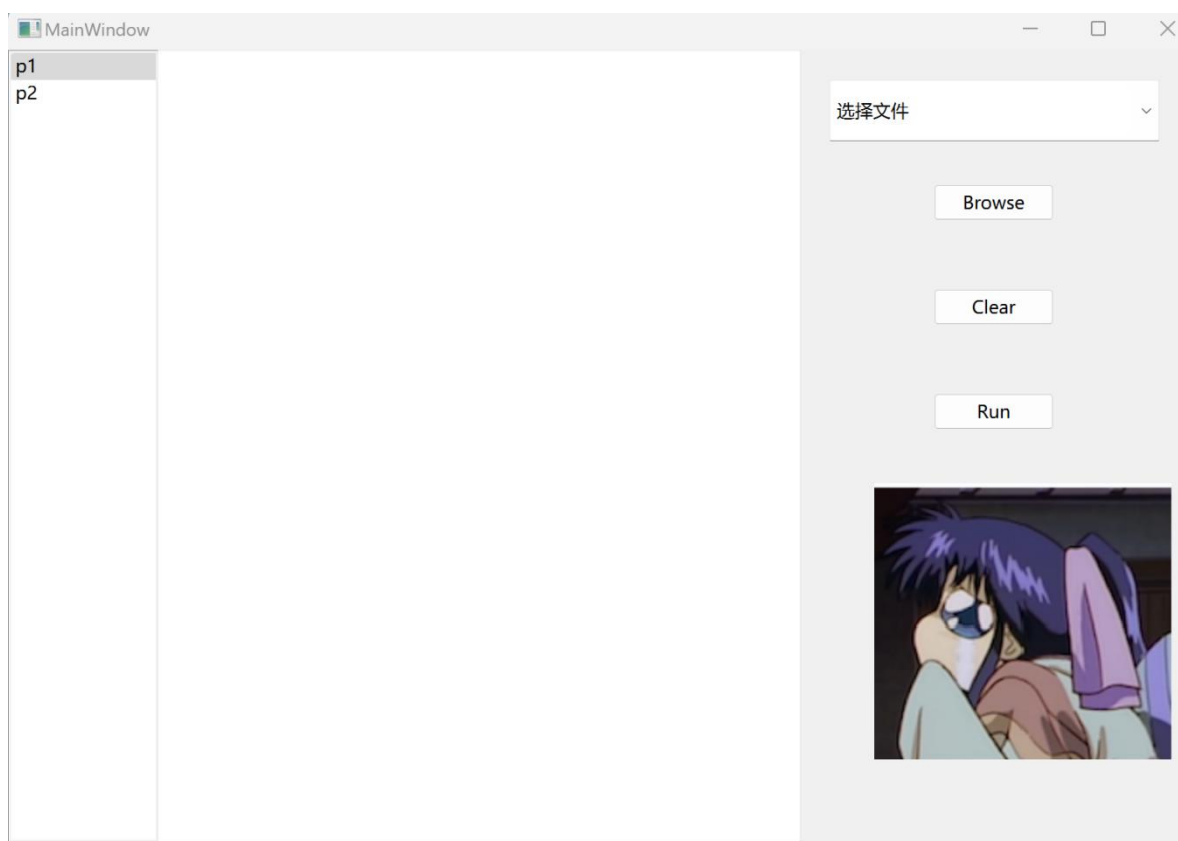


图 5.1 客户端界面

默认进入左侧菜单栏的 `p1` 界面，可在窗口中显示截获的日志信息和异常行为分析结果。右侧菜单栏 `browse` 按钮可以浏览本地文件并选择待目标测试程序，选择之后的文件路径回显在消息框中。选择过程如图 5.2 所示：

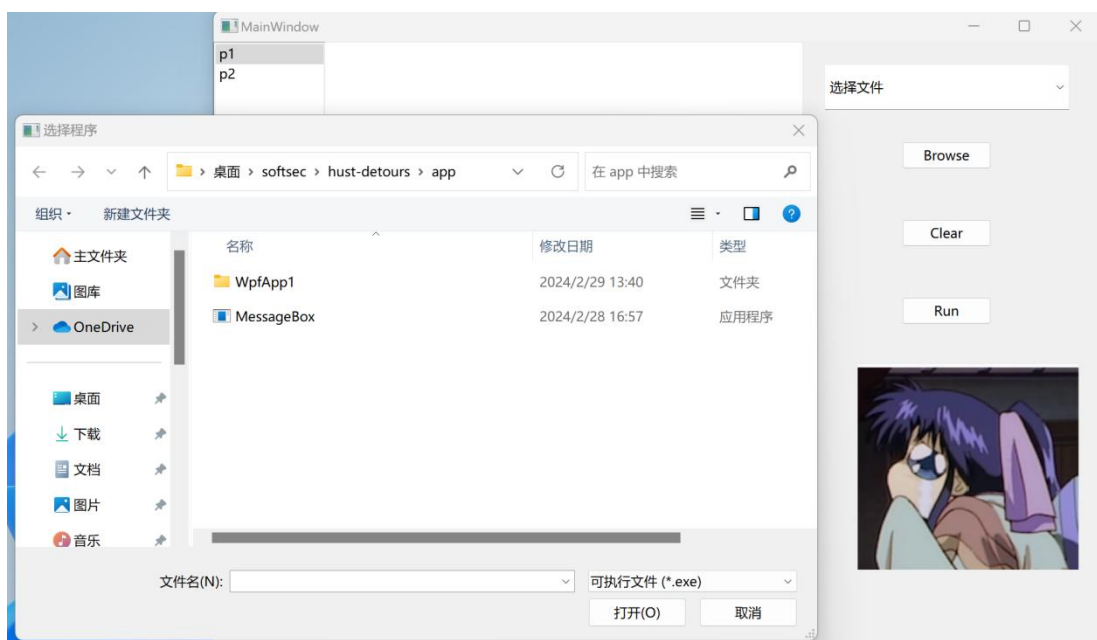


图 5.2 待测程序选择

选择程序后单击 run 按钮将启动测试，客户端会运行目标程序并调用注射器模块将 DLL 注入。本次测试弹窗程序，结果如图 5.3 所示。可以看到窗口中除了预期的 MessageBox 函数截获，还截获到了若干文件操作，这是因为弹窗函数存在重入问题，其内部调用了许多其它的 API 函数。

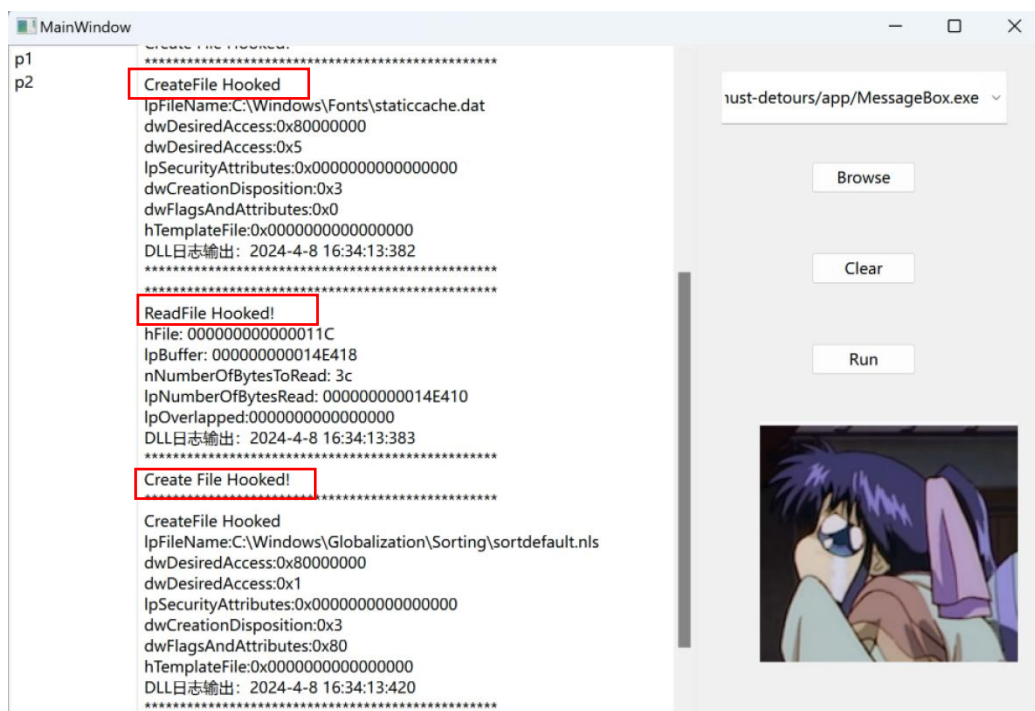
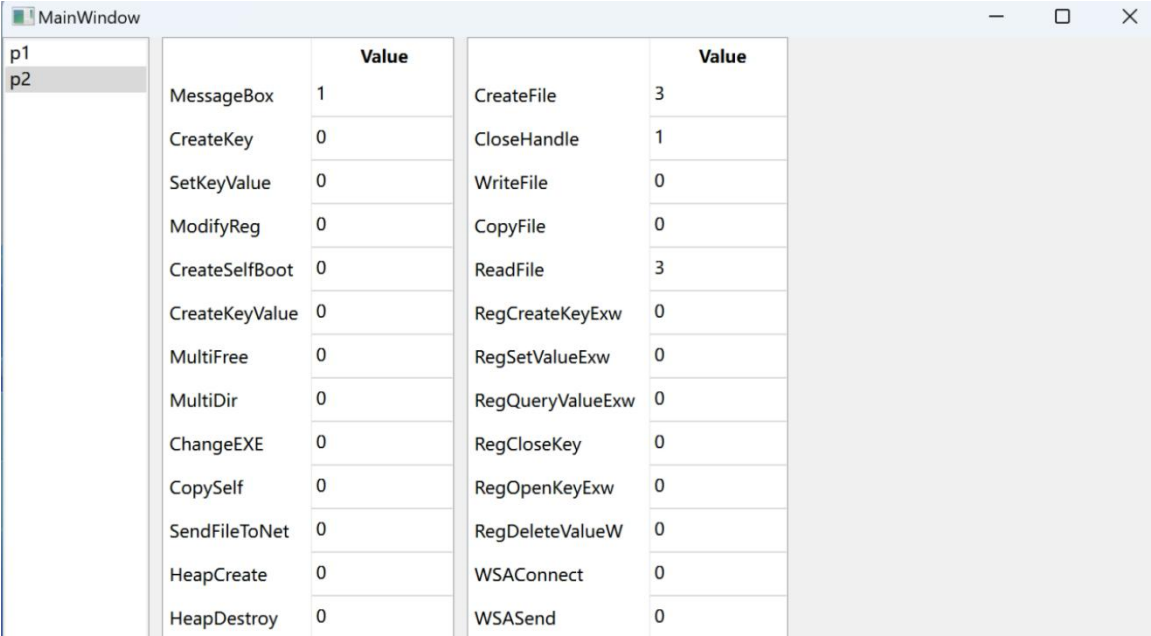


图 5.3 弹窗程序测试结果

单击 **clear** 按钮可以清屏，方便切换不同文件测试观察。

左侧菜单栏的 **p2** 界面用于显示 API 截获情况和异常行为触发次数。单击进入可观察到服务端回显的统计信息如下图 5.4 所示：



The screenshot shows a window titled 'MainWindow' with a sidebar on the left containing 'p1' and 'p2'. 'p2' is selected. The main area displays two tables of API statistics. The first table lists 15 APIs with their respective values, and the second table lists 10 more APIs with their values. The right side of the window is a large empty gray area.

	Value
MessageBox	1
CreateKey	0
SetKeyValue	0
ModifyReg	0
CreateSelfBoot	0
CreateKeyValue	0
MultiFree	0
MultiDir	0
ChangeEXE	0
CopySelf	0
SendFileToNet	0
HeapCreate	0
HeapDestroy	0

	Value
CreateFile	3
CloseHandle	1
WriteFile	0
CopyFile	0
ReadFile	3
RegCreateKeyExw	0
RegSetValueExw	0
RegQueryValueExw	0
RegCloseKey	0
RegOpenKeyExw	0
RegDeleteValueW	0
WSAConnect	0
WSASend	0

图 5.4 服务器分析统计结果

可见程序设计成功，客户端可通过 GUI 实现人机交互，且能正常运行测试样本并对 API 进行截获分析，以及分析结果的回显和日志记录。

5.2 文件操作测试

下面是文件截获函数的测试。运行 `inject.exe` 程序，可以看到程序成功执行待测目标 `filetest.exe`，该测试样本对文件操作进行了测试，提供操作菜单供用户选择不同的行为执行。测试结果说明基本的文件 API 截获和异常行为分析成功。

首先是创建和写文件操作测试，控制程序输入文件路径和写入的字符串，如图 5.5 所示，可观察到窗口中能返回截获信息和日志打印，`CreateFile`、`WriteFile` 和 `CloseHandle` 函数均成功被截获，检查本地路径，发现目标文件确实成功创建了，且写入正确的字符串。说明相关功能设计成功！

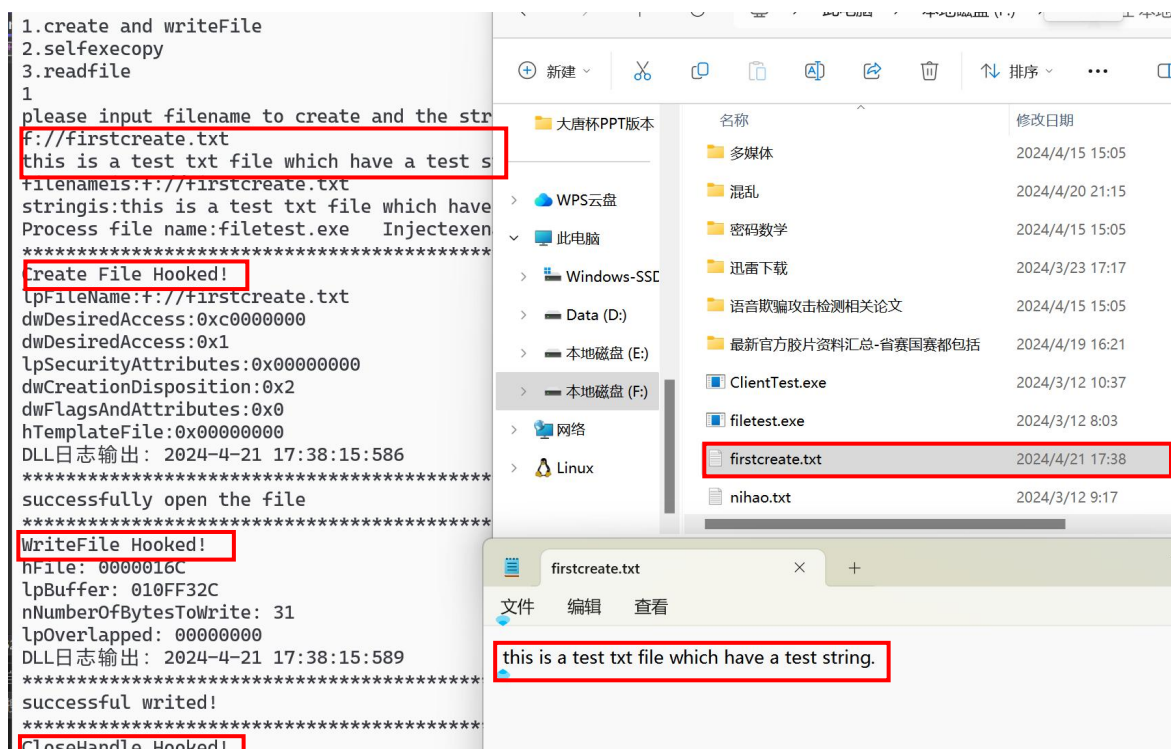


图 5.5 文件基本操作测试结果

接着样本程序试图将自身复制到 F 盘中并命名为 copiedfiletest.exe，观察图 5.6 可见客户端成功截获了 CopyFile 操作且本地路径下如期出现了复制成功的文件。

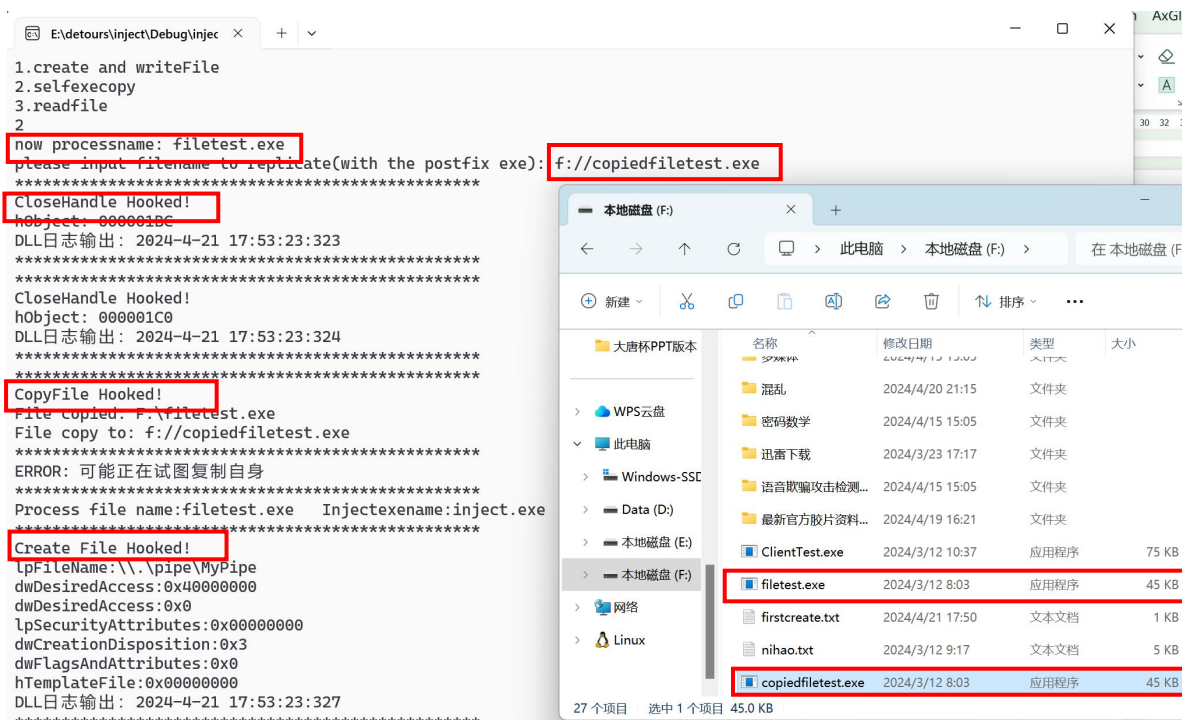


图 5.6 自我复制测试结果

接着尝试创建一个可执行程序，从结果图 5.7 中可观察到异常行为被截获！

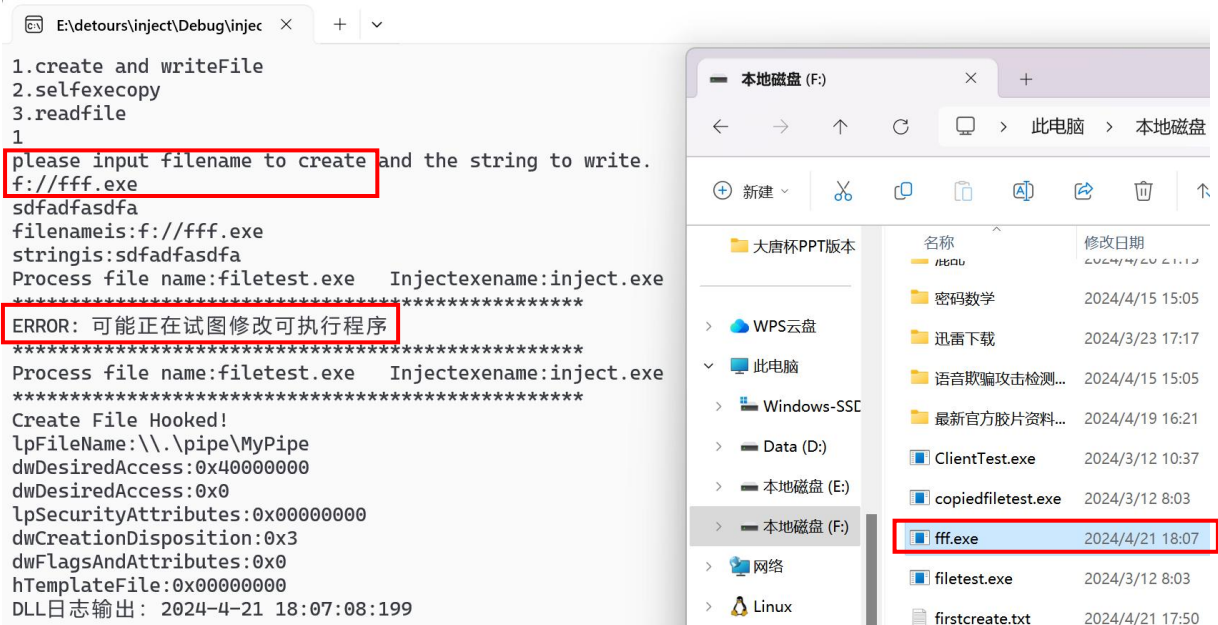


图 5.7 修改可执行文件测试结果

5.3 网络通信操作测试

我分别编写了服务器和客户端程序用于测试。

测试过程中首先需要双击启动服务器，服务器初始化后进入监听状态等待客户端连接，此时窗口无回显信息。

接着启动注射器对客户端测试程序进行分析，可观察到客户端成功启动，且启动后在初始化连接的过程中有很多文件操作被截获，因为网络通信 API 中内嵌了许多文件操作。

同时观察到服务器显示客户端连接成功，服务器连接成功后向客户端发送了一条打招呼信息，客户端成功收到，且 WSAConnect 和 WSAREv 操作被成功截获了并打印了通信 IP、端口和网络协议等连接信息。由于服务器和客户端在本地同 IP 地址下连接，因此观察到服务器端显示连接到的客户端 IP 和注射器中截获的 IP 是一致的，只是两者端口不同。

具体结果如图 5.8 所示：

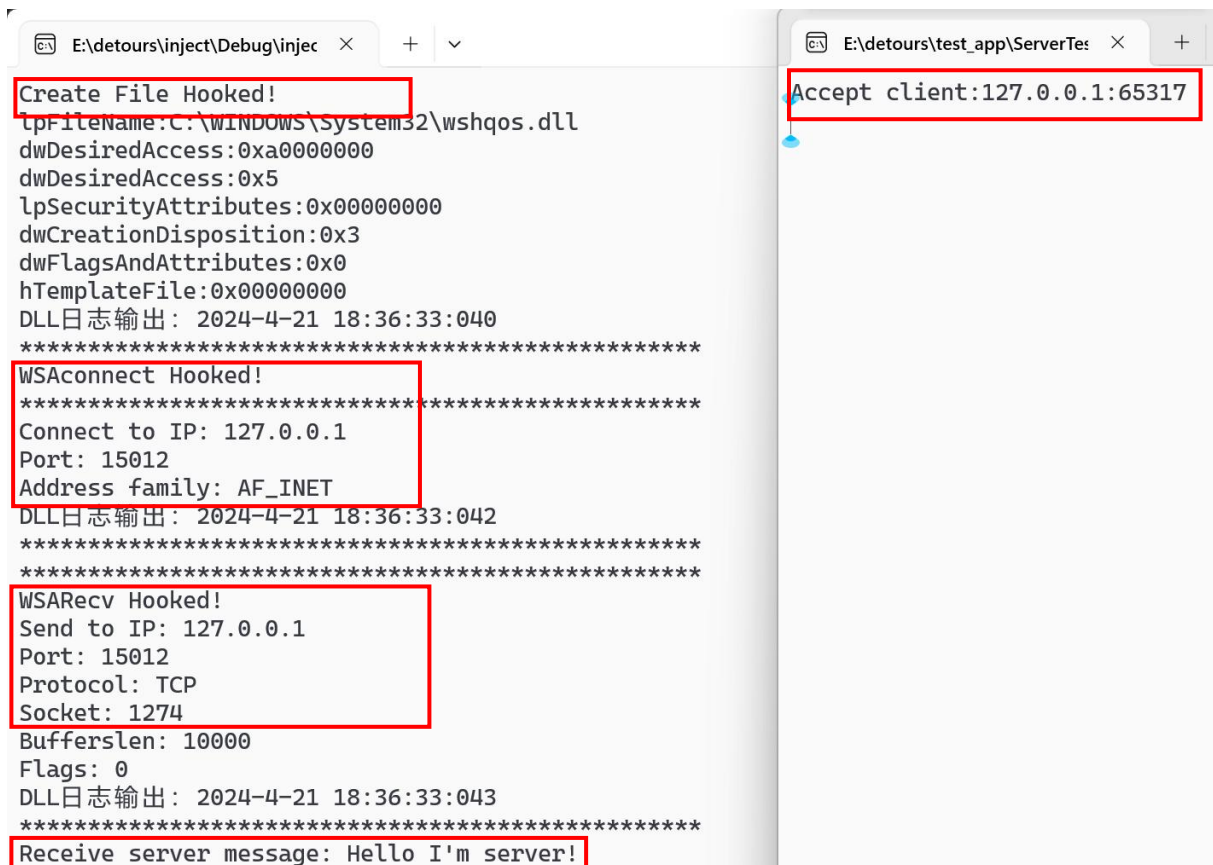


图 5.8 客户端连接服务器测试

接着控制客户端向服务器发送一条消息，可以观察到发送操作被成功截获，且服务器正常接收到了该消息，如图 5.9 所示：

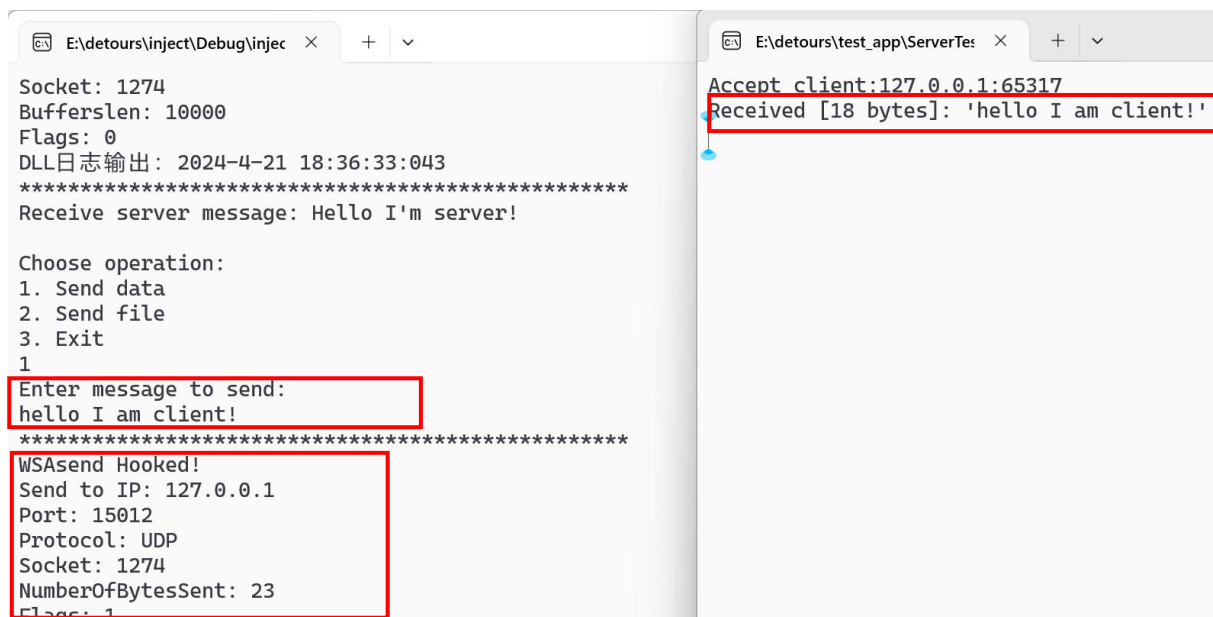


图 5.9 客户端发送消息测试

最后控制客户端读取本地文件并打包成数据包发送到服务器，观察到客户端 CreateFile 打开文件操作被截获，ReadFile 和 WSASend 操作被连续截获，说明进行了读取文件并发送到网络的操作，此时分析项目的统计服务器端分析出异常行为。测试的服务器端正常收到了该文件。如图 5.10 所示：

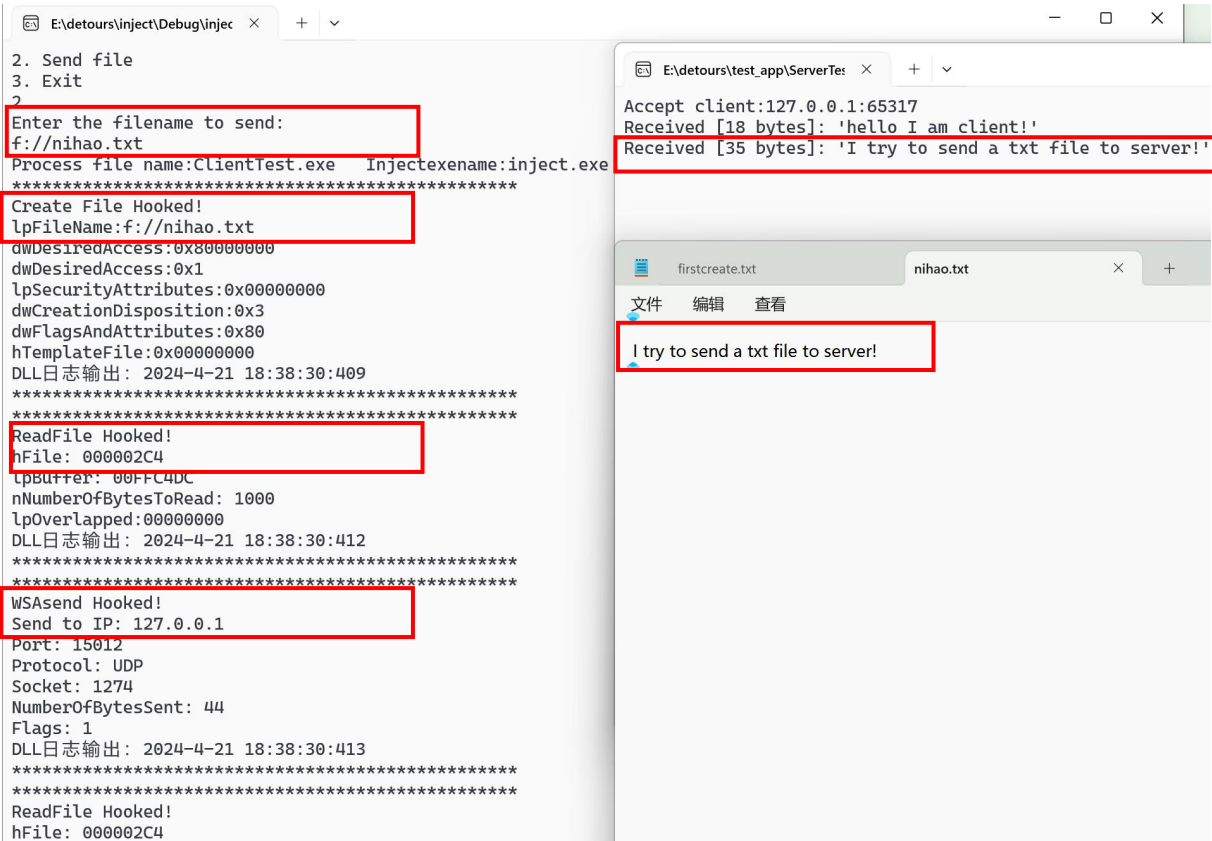


图 5.10 发送文件到网络测试

由以上测试可见网络通信和文件操作的截获分析程序设计成功！

6 总结与展望

6.1 总结

本次课设是一个比较大的综合任务，需要实现一个完整的安全分析工具，涉及多方面知识。不过相比于以往课设，本课设最大的特点是借助 Gitee 实现小组合作开发管理项目，这对团队协同和沟通提出了不小的挑战，也让同学们切身体会项目构建和管理的实践过程，更好地培养对项目整体的把握和设计能力。

刚开始以为课设的任务量巨大，所以需要多人合作完成。但在共同研读任务书、理解课设流程、掌握 Detours 原理和使用方法后我发现其实任务量并不大，拆分为前端和后端两部分即可。不过考虑到每个人工作量的问题，我们对后端的钩子函数按照类型分工，也算比较好地分摊了任务，不过弊端是由于分工相对独立，每个人各自完成自己的部分并进行测试，对于其他人的工作可能了解得不是很清楚，加上前后端的连接由负责前端的同学一人负责，当项目最终的测试出现问题时其他人难以对解决问题做出贡献。这也导致答辩前我们组为了添加功能反而把做好的程序搞崩了，版本回溯不及时导致展示的时候陷入窘迫。不过这个经验教训也让人受益匪浅，团队协作的项目不仅需要一强有力的领导核心来统筹分工与管理推进项目，还需要每个成员的互相协作，更需要有人对整体项目实现有着较为清晰的把握和理解并主导展示过程，这样才能更好地呈现出我们组努力的结果和项目特色。总之对于答辩时的失误我感到遗憾。

对于本人负责的文件和网络通信部分，看似工作量不大，但实际实现的过程中遇到了非常多的问题，下面我将主要的几个一一阐述。

首先是环境配置测试与 Gitee 协作的冲突，由于频繁更新项目文件，而每个人在本地都需要自己进行测试，导致测试更新的项目时总是需要重新本地配置 Detours 库，且操作频繁了还会发生一些未知问题导致项目运行失败。后来的解决方案是我们在本地另外设置一个备用仓库进行测试，由于分工较为独立，实际上只需要对原项目中各自的头文件进行修改即可，再由一个人负责最终客户端的测试，这样就不会造成因为本地测试生成方案导致的 pr 频繁冲突等问题。但是我认为这样的解决方案导致依托 Gitee 平台

进行的协作稍显累赘，后端部分基本上仅修改各自的头文件，前后端协调的工作均由一人完成，虽然有版本控制的好处，但实际上每个人本地自我管理已经足够，定位问题的时候也相对方便。

其次是异常分析算法的设计。在询问老师之后才对具体的异常行为分析有了较为清晰的认知，但实现过程还是经过和其他组负责相同模块的讨论后才能得出一个较为完善的解决思路。比如对文件发送到网络的异常行为分析还需要结合统计服务器端进行，也就是我要和负责前端的同学共同商量如何完成这个任务，打破了原本独立的分工，但好处是两个脑袋想问题能更好地解决。还有一些细节性的实现问题，如没有对文件操作对象进行过滤，http 头部数据无法解析等小的问题就不赘述了，最后大部分都解决了。

然后是测试样本的构建，测试过程中我发现很多其它 API 都会内嵌文件操作，比如弹窗函数、网络通信操作，这就导致了分析结果比预期的截获要多得多。以及在测试网络通信样本时也不太顺利，API 截获只需要改写函数，但是测试的时候却需要找到合适的样本。我在自己编写初期样本的时候，需要对网络通信服务器和客户端的通信有着较为清晰的理解，也算是对计算机网络知识的复习和实践了。

最后，虽然这个项目完成得不是很顺利，但过程中得到的收获还是很大的，非常感谢老师的耐心指导、同组甚至异组同学们的帮助，协作完成一个项目的成就感还是很大的。

6.2 展望

我们最后开发的程序或许还算不上一个安全工具，他的分析相当简陋而且冗长。界面和实现的功能也有所欠缺，离真正能应用的安全工具还有鸿沟般的差距。且由于实践关系，还有很多方面可以提升，比如对异常行为的分析可以采用更复杂但贴合实际的方式，以及更美观的图形界面和更多样化的功能，我们的目光不应该仅仅局限在完成一个课程任务上，而应该采用更实际的安全思维，向已落地的安全应用靠拢。

希望这次课设带来的收获和体会能帮助自己在未来更好地提升能力，在安全领域能有更深的研究和理解，最终能有机会参与实现一个可推广使用的安全工具。

自评及反馈

指标点	评价标准				自评
	优	良	中	差	
3. 设计 / 开发 解决方案	设计的方案能够完全覆盖所需要解决的问题, 使用掌握的技术, 并能够在实验报告中完整准确的说明设计思路。	设计的解决方案能大部分覆盖所需要解决的问题, 并能够在实验报告中较完整准确的说明设计思。	实现了任务书中提出的主要功能要求, 具备在调试和实现过程中分析问题和解决问题的能力, 熟悉开发过程中的细节, 但对关键部分解释含糊	设计的解决方案不能覆盖解决所提出的提问题, 使用当前掌握的技术不具备实现可行性, 在实验报告中未能清楚说明设计思路。	优
9. 个人和团队	在分小组进行课程设计的过程中, 能非常积极主动地承担具有难度的任务, 按时完成个人负责的模块, 并能主动协助组内成员; 善于跨组纵向学习和组内合作。	在分小组进行课程设计的过程中, 能保证质量的完成个人负责的模块, 能配合组内其他成员的工作。能较好完成跨组纵向学习和组内合作。	在分小组进行课程设计的过程中, 基本能完成个人负责的模块, 通过组内学习, 完成程序的装配。能基本完成跨组纵向学习和组内合作。	在分小组进行课程设计的过程中, 不能按时完成个人负责的模块, 不能与其他组员合作开展工作。	优
10. 沟通	能针对本课程设计的问题, 灵活运用专业知识, 通过撰写报告或者陈述发言, 清晰准确的说明系统的设计方案、实现方法和测试结果; 文字或语言表达通顺流畅。	能针对本课程设计的问题, 运用专业知识, 通过撰写报告或者陈述发言, 较为准确的说明系统的设计方案、实现方法和测试结果; 文字或语言表达较为通顺流畅。	能针对本课程设计的问题, 运用专业知识, 通过撰写报告或者陈述发言, 基本能说明系统的设计方案、实现方法和测试结果; 文字或语言表达比较通顺。	能针对本课程设计的问题, 运用专业知识, 通过撰写报告或者陈述发言, 系统的设计方案、实现方法和测试结果说明不充分; 文字或语言表达不通顺。	优

意见或建议反馈：

这次实验我们面向课设任务需求开发了一个安全工具，并且是以小组合作的形式，过程中收获很大。最后答辩的过程可以看到其他组的展示都非常优秀，优美简约的 GUI 和流畅的工具使用体验以及诸多扩展功能。但只要对整个项目有比较清晰的认知就会发现，后端的工作量在借助 Detours 库的情况下其实并不大，而且课设为了方便评分任务点给的也比较具体，大家基本都实现了任务，主要是在 GUI 和展示方面下功夫。虽然良好的 GUI 是任何软件都应当具备的，但作为一个小组协作项目，如果重点是展示而非核心安全功能的设计创新的话，感觉有点脱离原本的课程设计目的了。

故我在此提出一点小小的建议，课设可以不拘泥于使用某个具体的库，而是让同学们自行搜索学习和选择，提供更大的发挥空间和小组合作任务量。或者也可以保持当前的任务量，减少小组人数，个人认为分为前端和后端即可，甚至一人完成也是可以的。在和其他组的交流过程中了解到个别组的项目实际上是由部分人完成的，而且即使是个人项目，同学们间也不会缺少沟通交流，也可以自己进行版本管理。所以我觉得老师们可以重新考虑一下这个课设任务量的安排。

参考文献

- [1] 软件安全 课程设计 实验指导书 2024（整合）v2.0
- [2] <https://blog.csdn.net/tutucoo/article/details/85235478>
- [3] <https://blog.csdn.net/tcjiaan/article/details/8497535?spm=1001.2014.3001.5501>
- [4] <https://github.com/microsoft/Detours>
- [5] <https://learn.microsoft.com/zh-cn/windows/win32/apiindex/windows-api-list>
- [6] https://blog.csdn.net/weixin_38793855/article/details/104563652
- [7] https://blog.csdn.net/qq_39741222/article/details/128206096