華中科技大學

课 程 设 计 报 告

基于 API HOOK 的软件行为分析系统

课　　程　软件安全课程设计

院　　系　网络空间安全学院

专业班级_____

学　　号_____

姓　　名　黄浩岩

指导教师　付才

2021 年　3　月　27　日

课程设计评分标准		
1	平时成绩（20 分）	1 考勤 10 分，按到课比例给分，全勤 10 分，无故缺勤每次扣 2 分； 2 小组讨论、作业 10 分，按作业组内互评参考，由指导教师评价后给分，优秀 9-10 分；中等 7-8 分；及格 6 分；
2	功能完成情况（40 分）	按任务的 1、框架设计 2、截获 3、分析 4、图形界面 5、样例 6、选做 7、创新 8、时间控制。完成度非常好的，按该项满分的 90%-100%；有小问题或缺陷的，按满分的 70-85%；有明显问题的，按满分的 50-65%；未完成的项目，该项不得分。此项由小组内进行，分数作为教师参考，可按得分折算作为完成情况。
3	报告格式（10 分）	按华中科技大学本科毕业论文规范，格式非常规范，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（9-10 分）；少量不规范（7-8 分）；格式不太符合规范（5-6 分）；未按格式书写（2-4 分）。
4	报告内容（30 分）	内容上，包括课设的任务分析、设计实现的过程、遇到的问题及分析解决的过程、对课设中的优点、不足，进行反思和总结；报告结构合理，语言通顺、流畅；优秀（27-30 分）；良好（24-26）；一般（21-23 分）；及格（18-20 分）。**报告需本人独立完成，其中的图片、文字都必须是作者本人课程的成果。对抄袭他人报告、代码的行为，按学校要求严肃处理。**

课程设计成绩

平时成绩	功能完成情况	课程报告		总评成绩
（20%）	（40%）	格式（10%）	内容（30%）	100%

目 录

1　课程设计任务书

1.1　课程设计目的

1. 熟悉 Windows 操作系统常用 API 函数
2. 理解 Detours 库的技术原理
3. 掌握无源码情况下分析样本程序行为的方法
4. 掌握使用 Detours 库提供的接口对简单程序进行行为分析的方法

1.2　课程设计要求

1. 实现基本的第三方进程 WindowsAPI 截获框架
   (1) 编译生成 Detours 库
   (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
   (3) 自编或者利用已有恶意代码样例（包含弹出对话框动作）
   (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
   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
   (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
   (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

(1) 判断是否 10 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，覆盖被检测的行为

10. 网络通信操作异常行为分析（选做）

(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

11. 内存拷贝监测与关联分析（选做）

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

操作系统：Windows 10 家庭中文版

开发语言：C/C++

集成开发环境：Visual Studio 2019 16.9.0，Qt Creator 4.14.1

1.4 实验过程记录

1. 使用 VS2019 编译 Detours 库

在 Detours 的 GitHub 库上下载 Detours 的源码，解压得到文件夹 Detours-master，然后，在开始菜单中找到 x86 Native Tools Command Prompt for VS 2019，定位路径到解压的文件夹的 src 目录下，使用如下命令编译:

```
cd Detours-master/src
nmake /f Makefile
```

编译后获得 bin.X86、lib.X86、include 这三个文件夹

2. 在 VS 上配置 Detours 库

在 VS 创建 DLL 项目，并在项目属性，VC++目录中将包含目录中加上刚才编译出的

include 文件夹路径，在库目录上加上 lib.X86 文件路径。

3. 编写 DLL 程序

（1）定义需要截获（Hook）的函数和引入替换后的新函数，该部分需要按照按照课程设计要求，在后续阶段不断完善截获的函数，以增加对 Windows API 的行为分析。

（2）在 DLL 入口 DllMain 函数中设置截获函数的绑定情况：在 DLL 加载到程序时使用“DetourAttach”挂载截获函数，在 DLL 卸载时使用“DetourDetach”卸载截获函数。

4. 完成开启和关闭注入的工具（注射器程序）

新建一个控制台程序，使用 DetourCreateProcessWithDllEx 函数将 DLL 注入目标程序。

5. 编写带注入的程序

根据课程设计要求，设计与 MessageBox、文件、注册表、堆等有关的操作程序，作为行为检测样本库，用于检测截获函数的执行情况，已经异常行为分析。该部分与 DLL 程序相同，在实验过程中需要不断更新完善。

6. 设计完成系统界面

使用 Qt 设计并编写图形界面，用于便于用户操作和执行结果的查看。

2　绪言

2.1　课程设计背景

Windows 是当下最为常用的操作系统之一，其安全问题一直受人们的关注。如今已经有了很多较为成熟的工具、开发库等用于 Windows 操作系统下的程序分析和函数截获。

而无源码情况下分析程序的行为也是如今软件安全领域的一个重要的研究方向，相比于在在有程序源码的情况下的程序行为分析，无源码程序分析适用范围更加广泛，但同时也需要更加有效底层的方法来对程序执行的函数进行截获分析。

本课程设计利用的是 Windows 平台下的微软的 Detours 开源库，该代码库提供了一系列针对 Windows 平台下的程序函数截获的函数，而通过结合 Windows 的 API 接口函数，可以达到在无源码的情况下对程序进行行为分析已经异常检测。

2.2　Detours 库原理

Detours 库时通过使用一个无条件转移指令来替换目标函数（即 Target 函数，一般为 Windows API 接口函数）的首部几条指令，具体而言是目标函数汇编代码的前 5 个字节。从而将控制流直接转移到一个用户自己实现的截获函数（即 Detour 函数）中，而原目标函数中被替换的指令被保存在一个被称为蹦床函数（即 Trampoline 函数）中。这些指令包括目标函数中被替换的代码，以及一个重新跳转到目标函数正确位置的无条件分支。截获函数可以替换目标函数，或者通过执行蹦床函数时，将目标函数作为子程序来调用的方法，在保留原来目标函数功能基础上，来完成扩展功能。

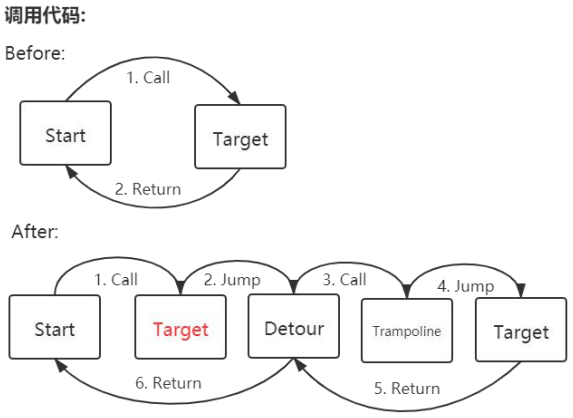

图 2-1 Detour 函数的调用过程

3　系统方案设计

3.1　系统模块

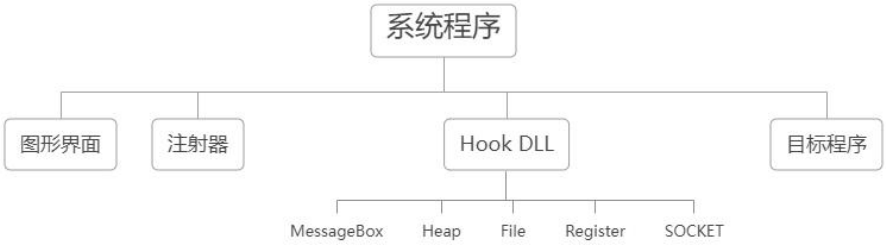

图 3-1 系统模块

本系统主要有四大模块组成，分别是图形界面、注射器程序、钩子程序 Hook DLL 以及目标的测试程序。

钩子程序 Hook DLL 是实验的关键部分，其下有分为对 MessageBox、堆、文件、注册表、套接字的相关操作的截获函数以及行为分析，借助 Detours 库实现自定义函数与目标 Windows API 函数的绑定。本实验主要围绕着该部分进行程序设计。

注射器程序主要是利用 Detour 提供的 DetourCreateProcessWithDllEx 函数将 DLL 注入目标程序，使得目标程序执行时使用 Hook DLL 模块中定义的自定义函数以便于程序行为分析。

目标程序即用于检测的样本库。

图形界面主要用来方便用户操作，以及用于更清晰简洁的展现截获程序的输出。

3.2　系统执行流程

系统整体展现在用户面前的是图形界面，用户通过图形界面选择要进行行为分析的目标程序。选择之后，系统会使用注射器模块对目标程序注入，将 Hook DLL 中定义的各类 Windows API 接口函数进行替换。注入之后，目标程序会开始运行，运行过程中一旦执行了在 Hook DLL 中定义截获的 Windows API，便会将执行相应的替换函数，记录下函数的输入参数，以及通过这些参数进行行为分析的相关信息。最后，所有目标程序记录的相关信息都会汇总到图形界面，有图形界面的相关程序进行处理，然后以较为简洁友好的形式向用户进行展示。

4 系统实现

4.1 API 截获的实现

截获 API 函数是 Hook DLL 模块的一大功能，其主要实现的是对常用的 Windows API 接口函数的截获、替换、拓展，从而在目标程序运行时将其 API 函数的调用信息予以截获记录，方便后续的行为异常分析。

具体而言，截获的 API 函数包括和弹窗相关的函数，如 MessageBoxA、MessageBoxW；和文件操作相关的函数，如 CreateFile、ReadFile、WriteFile、与堆操作相关的函数，如 HeapCreate、HeapAlloc；与注册表操作相关的函数，如 RegCreateKey、RegSetKeyValue 等，已经与套接字相关的函数，如 socket、send 等。由于对于这些 API 所设计的截获函数整体实现方法基本一致，此处统一进行阐述。

对于要截获的目标 API 函数，在此称之为 APIFunc，首先要定义一个函数指针 OldAPIFunc，来指向该 API 函数，相当于记录该目标 API 函数的原函数入口地址。接下来定义截获后目标函数的替换函数，称之为 NewAPIFunc，该函数与目标 API 函数的函数声明是完全相同的，以保证能够正确替换。在 DLL 入口函数 DllMain 中，通过 DetourAttach 函数，将 OldAPIFunc 和 NewAPIFunc 绑定在一起，并在执行时进行替换。

在 NewAPIFunc 中，主要将函数的参数信息进行输出记录，由于替换后该函数代替了原来的 APIFunc，因此输出该函数的参数信息实际上就输出了要截获的目标 API 函数 APIFunc 的参数信息。此外，通常仍需要目标程序能够继续运行，因此需要在 NewAPIFunc 函数中调用原来的目标 API 函数，由 Detours 库的技术原理可知，在 Detours 进行函数替换时，会将原函数开始部分代码进行修改，因此在 NewAPIFunc 调用原 API 函数不能使用 APIFunc，原因是会产生递归的截获，而是使用上述的函数指针，用于执行原来 API 函数。

需要额外说明一下 Windows 函数参数的_In_和_Out_修饰符，前者表示参数是输入，后者表示参数是用于存放输出的，比如 ReadFile 中表示实际读取个数的 lpNumberOfBytesRead 参数，该参数只有当函数调用之后才能得到其正确的值，因此对于带_Out_修饰符的函数，需要在替换函数中执行了原函数之后，才能得到相应的参数值。

4.2 WriteFile 和 HeapAlloc 的截获

此处需要额外说明一下 WriteFile 和 HeapAlloc 两个 API 函数的截获。由于这两个函数需要在替换后的函数 NewWriteFile 和 NewHeapAlloc 函数中使用 printf 等函数来输出函数信息，

而这些函数又会调用 WriteFile 和 HeapAlloc 函数，会产生递归现象，致使程序崩溃，因此，需要在上述的截获函数上进行调整，以避免递归现象的产生。

对于 NewWriteFile 函数，在输出参数信息前，通过使用代码 GetFileType(hFile) != FILE_TYPE_DISK 来对文件类型进行判断再选择输出，当文件类型不是磁盘文件时则选择不输出其信息。对于 printf 产生的 WriteFile 操作，其写入的文件类型不为磁盘文件，因此可以屏蔽掉 printf 产生的 WriteFile 截获操作，从而避免了递归。

对于 HeapAlloc 操作，则是采用了全局变量标记的方法防止递归。此处设置一个布尔型全局变量 CantPrintAlloc，初始值为假，当其值为假时则进行 printf 信息输出，同时在输出前将其值值为真输出后重置为假；当其值为真是表明不能进行信息输出，函数则直接返回。当该函数执行时，由于 CantPrintAlloc 初始值为假因此可以可以进行信息输出，此时输出的就是真正要截获的目标 HeapAlloc 函数，在输出之前由于将 CantPrintAlloc 值为真，因此在输出过程中 printf 产生的 HeapAlloc 不会触发信息输出而是正常执行因此不会产生 HeapAlloc 的递归。当信息输出完毕后，再将 CantPrintAlloc 值为假，确保后续的 HeapAlloc 能够正常被截获。由此，解决了 HeapAlloc 的递归问题。

4.3 堆操作异常行为分析

堆操作异常行为分析主要是检测堆申请与释放是否一致，以及是否发生重复的多次释放。为了确定堆的释放情况，此处使用一个哈希集合 unordered_set 来记录堆的句柄。

当使用 HeapAlloc 时，在替换的函数 NewHeapAlloc 中将调用 OldHeapAlloc 分配的堆的句柄添加到哈希集合中。

当使用 HeapFree 进行堆释放时，首先检查当前待释放的堆句柄是否在哈希集合中，如果在哈希集合中，证明当前堆被分配还未释放，此时将该堆进行释放，输出相应的参数信息，并将该堆句柄从哈希集合中移除。若当前待释放的堆句柄不在哈希集合中，则表明要么当前句柄不为堆句柄，即释放不一致；要么当前堆句柄对应的堆已经释放，已经从哈希集合中移除，为重复释放。

通过上述方法，即可对堆操作异常进行分析。

4.4 文件操作异常行为分析

文件操作异常行为分析，在本次课设中个人主要实现了判断当前文件是否存在自我复制的情况，以及是否修改了可执行代码。

对于文件是否存在自我复制情况，主要是通过截获 CopyFile 函数实现的。一旦目标程序

调用了该函数，则证明该程序存在复制文件的情况，此时进一步判断是否复制的文件是自身。使用 GetModuleFileName 函数获取当前运行的程序的文件，即运行的目标程序的绝对路径。此时在判断 CopyFile 参数中的复制的源文件 lpExistingFileName，使用 GetFullPathName 函数将路径转换为绝对路径后与上述当前程序的路径进行比对，若比对一致，则证明发生了文件的自我复制情况。

对于是否修改了可执行代码，则是通过截获 WriteFile 和 DeleteFile 进行分析，判断当前操作的文件是否为以 exe，dll，ocx 为拓展名的可执行文件。对于 DeleteFile 的截获分析比较简单，因为该函数的参数即为待删除的文件名，可以直接对该文件名提取拓展名并进行比对，从而判断当前删除的文件是否为可执行文件。而对于 WriteFile 情况则不同，该函数传入的参数不为文件名，而是文件对应的句柄，因此不能直接进行拓展名的比较。此处采用了类似上述堆异常分析的方法。当试图写文件时，一定在这之前使用 CreateFile 打开了文件，因此此处定义了一个哈希表，键名为文件的句柄键值为文件的路径，这样在 WriteFile 修改文件时，可以通过访问哈希表来获取文件句柄对应的文件完整路径，从而再进行文件拓展名的比对，来判断当前是否修改了可执行文件。

4.5 注册表操作异常行为分析

对于注册表的修改，可以直接通过截获可注册表操作有关的 API 实现，如 RegSetKeyValue、RegGetValue 等。此处对注册表操作异常行为分析主要是判断新增注册表项是否为自启动执行文件项。

对于 Windows，在特定的注册表位置添加注册表项，即可达到自启动执行文件的效果。在本课设中，主要考虑的是在注册表路径“HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run”下，如果添加了注册表项，则该项就会作为开机自启动项。因此，在 RegSetKeyValue 截获函数 NewRegSetKeyValue 中，对当前的注册表的根键和子项进行判断，若根键为 HKEY_CURRENT_USER 同时子项为 Software\Microsoft\Windows\CurrentVersion\Run，则证明当前设置的注册表项为自启动项，进行异常行为信息的记录。

4.6 系统界面和数据传输的实现

对于系统界面，此处使用了 Qt 桌面应用程序开发框架进行系统界面的开发。Qt 作为当下 C++图形界面程序开发的主流框架，其具有良好跨平台属性，且易于上手，界面设计简单而多样，因此用于本次课设的系统界面的开发十分合适。

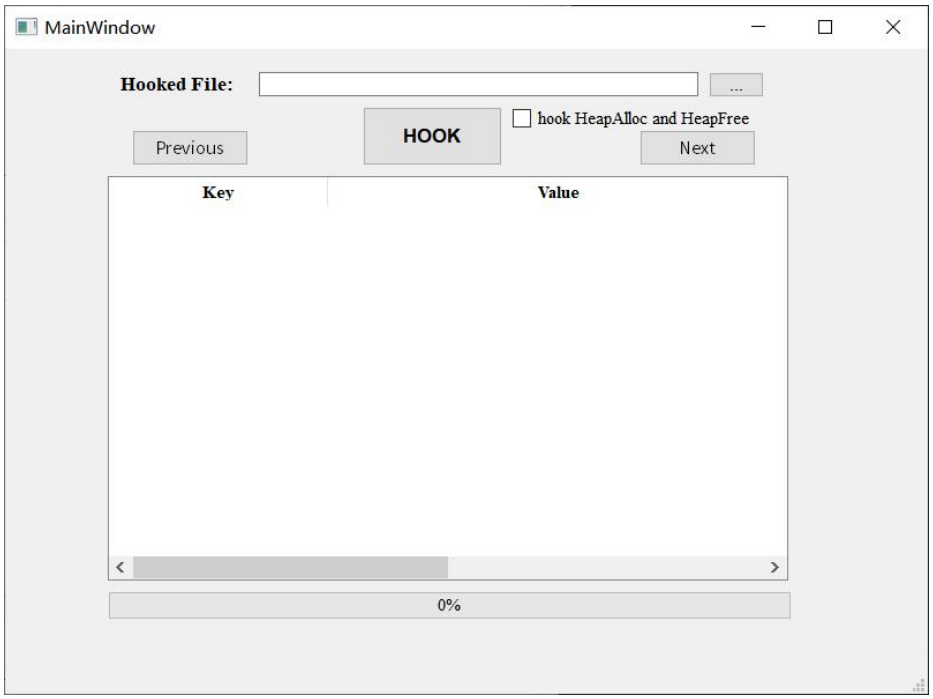

图 4-1 系统界面

系统界面如图 4-1 所示，在界面右上角，用户可以通过“...”选择需要注入的目标程序，点击 HOOK 即可运行目标程序，同时使用 Detours 库进行 API 的截获分析。对于截获的信息，则以每个函数为一页在下面的列表中进行显示，用户可以通过“Previous”和“Next”两个按钮切换查看截获的信息，下方的进度条则反应了当前截获信息的总进度。

在 HOOK 的右上角有一个复选框，勾选上则可以对 HeapAlloc 和 HeapFree 两个函数进行截获，由于 HeapAlloc 函数在操作系统中调用十分频繁，因此若不是为了专门截获这里个函数，不建议勾选，否则大量的 HeapAlloc 截获信息会淹没其他函数的截获信息。对于该复选框的底层实现，在此处实际上是通过编译生成两个 DLL 实现的，一个 DLL 中实现对 HeapAlloc 两个函数的截获，另一个不实现；用户通过复选框来实际上就是在两个 DLL 中进行选择，用于注射器程序来注入目标程序。

对于注射器程序和系统界面的数据传输，此处使用 Qt 自带的 QProcess::readAllStandardOutput 方法，来读取注射器程序的输出，其本质就是使用的进程间通信的管道通信方式，将注射器进程的输出重定向到系统界面。对于截获的 API 信息，专门设置了不同的分隔符，用于对每个函数的截获信息、每一项的信息分别进行标记，用于系统界面程序对其进行划分处理转换成界面上列表可以输出的样式。

4.7 行为检测样本库

本课设中，根据检测的对象，编写了弹窗样本、文件操作样本、堆操作样本、注册表操作样本、修改 exe 文件操作样本以及套接字操作样本，共 6 个简单的样本程序。每个样本中涉及的便是其相关的 Windows API 以及相应的异常操作，作为样本库来进行分析。

以文件操作样本为例，在该样本中，首先调用了 CreateFile 打开一个文件，然后是用 WriteFile 进行写操作，接着使用 ReadFile 进行读操作，接下来尝试使用 CopyFile 进行自我复制，最后使用 DeleteFile 尝试删掉一个文件。整个样本中就涉及了系统中文件操作相关的 API，以及自我复制的异常行为操作。其他样本也类似，在此不赘述。

5　系统测试

5.1　弹窗 API 截获

如图 5-1 所示，截获了目标程序的弹窗 API 函数 MessageBoxW，并获取了其函数的参数信息。

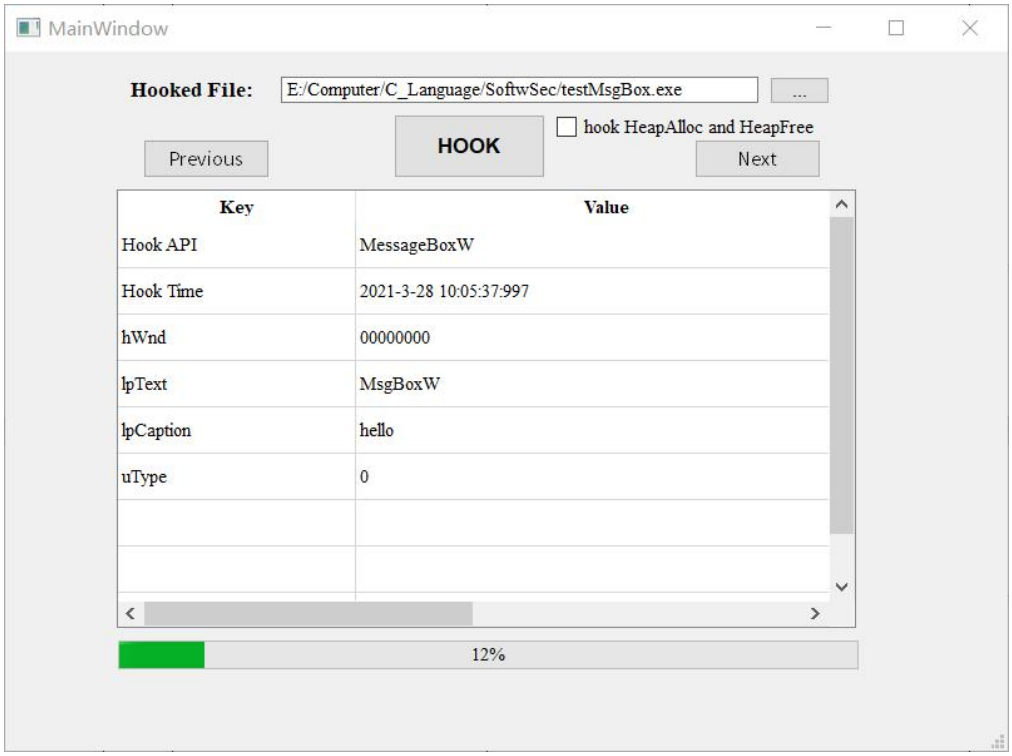

图 5-1 弹窗 API 截获

5.2　堆操作 API 截获

如图 5-2 和 5-3 所示，截获了目标程序的堆操作 API 函数，包括 HeapCreate、HeapDestroy，并输出了截获的函数参数信息。此处未勾选 HeapAlloc 和 HeapFree 的复选框，因此并不会截获这两个函数。

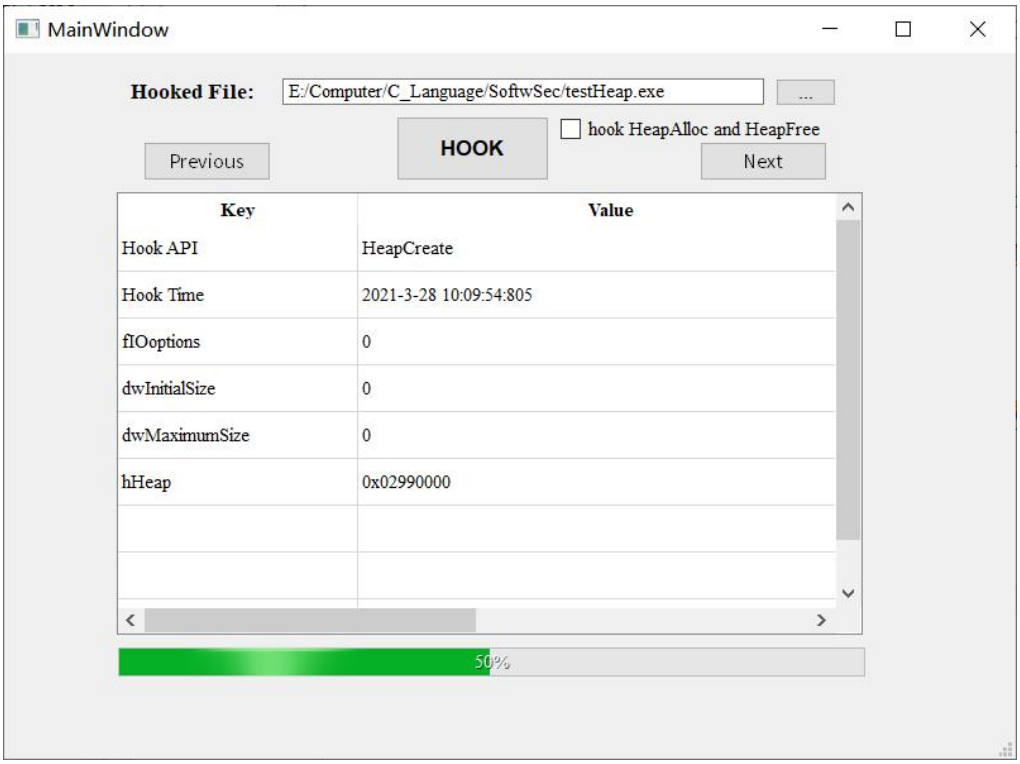

图 5-2 HeapCreate 的截获

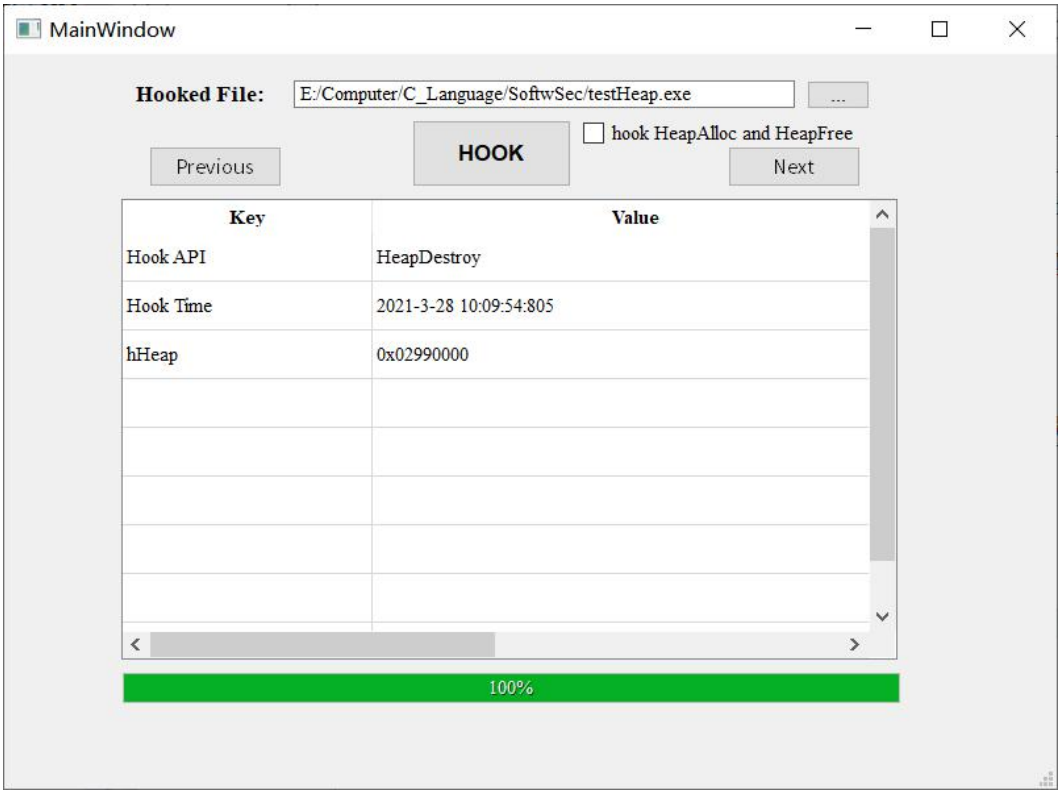

图 5-3 HeapFree 的截获

5.3 文件操作 API 截获

如图 5-4 和图 5-5，实现了文件操作的 API 截获，不限于图中的 CreateFile 和 ReadFile，还包括 WriteFile、DeleteFile 等，并输出了截获的相应信息。

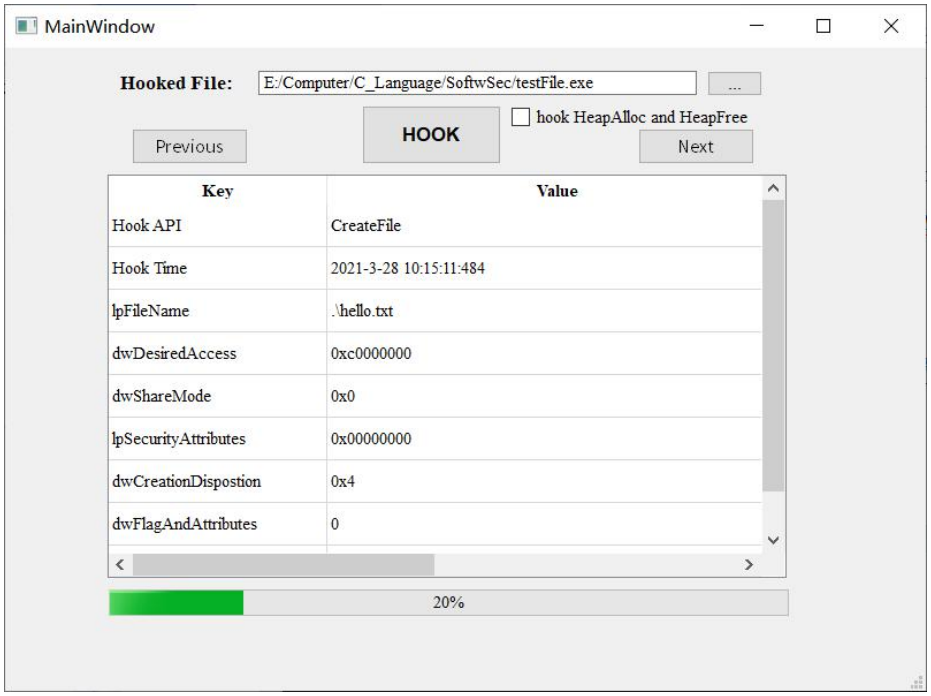

图 5-4 CreateFile 的截获

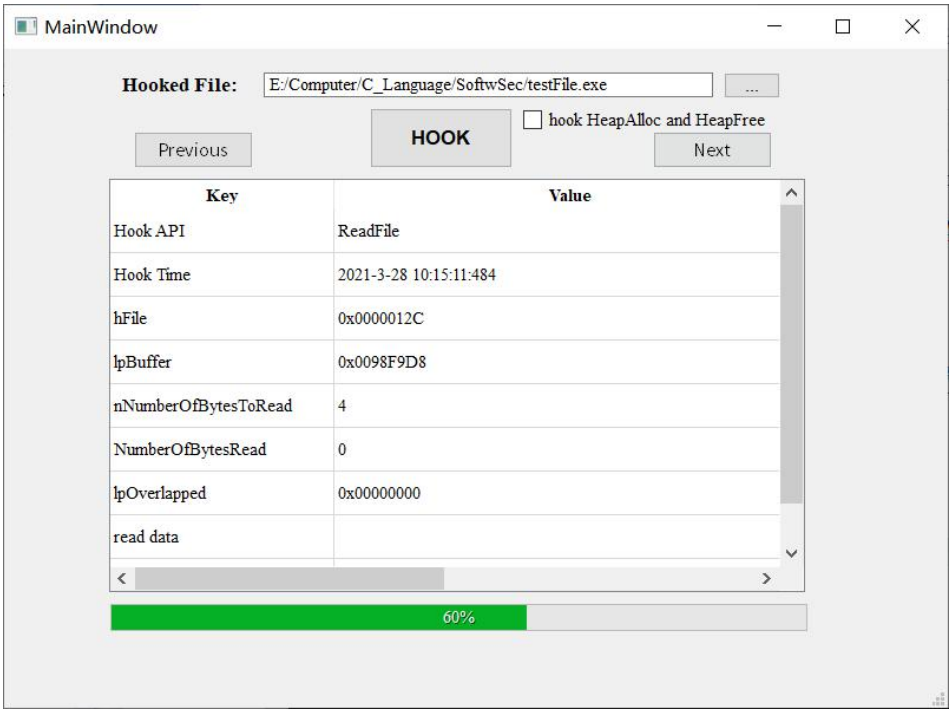

图 5-5 ReadFile 的截获

5.4 注册表操作 API 截获

实现了对注册表 API 的截获，包括 RegCreateKey、RegSetKeyValue、RegGetKeyValue、RegCloseKey 等和注册表相关的函数，并输出了相关信息。此处仅列出了 RegCreateKey 和 RegGetKeyValue 的截获信息，如图 5-6 和 5-7 所示。

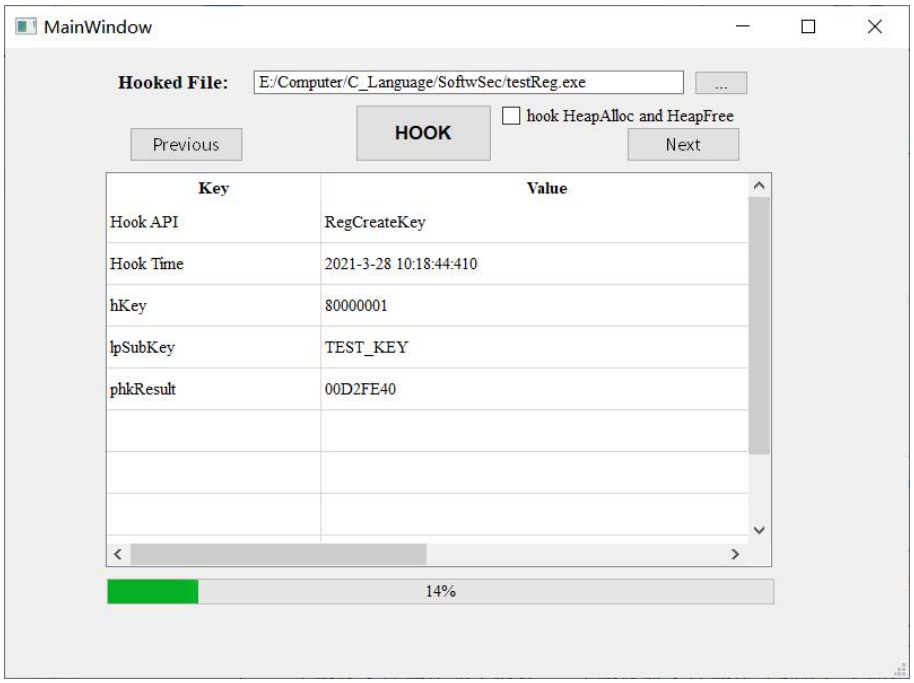

图 5-6 RegCreateKey 的截获

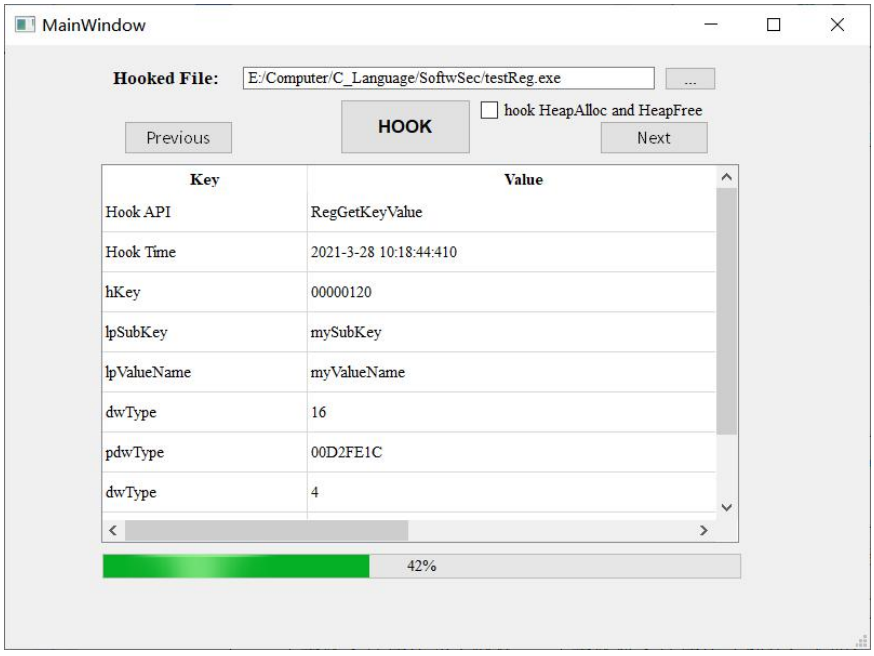

图 5-7 RegGetKeyValue 的截获

5.5 堆操作异常行为分析

在勾选对 HeapAlloc 和 HeapFree 的截获复选框后，运行堆的目标程序。当目标程序尝试对一个堆进行重复释放时，后续的释放会被截获到，并在列表中显示堆释放错误“Heap Free ERROR”，同时提示可能存在堆重复释放或者释放的句柄不为堆句柄的情况，如图 5-8 所示。

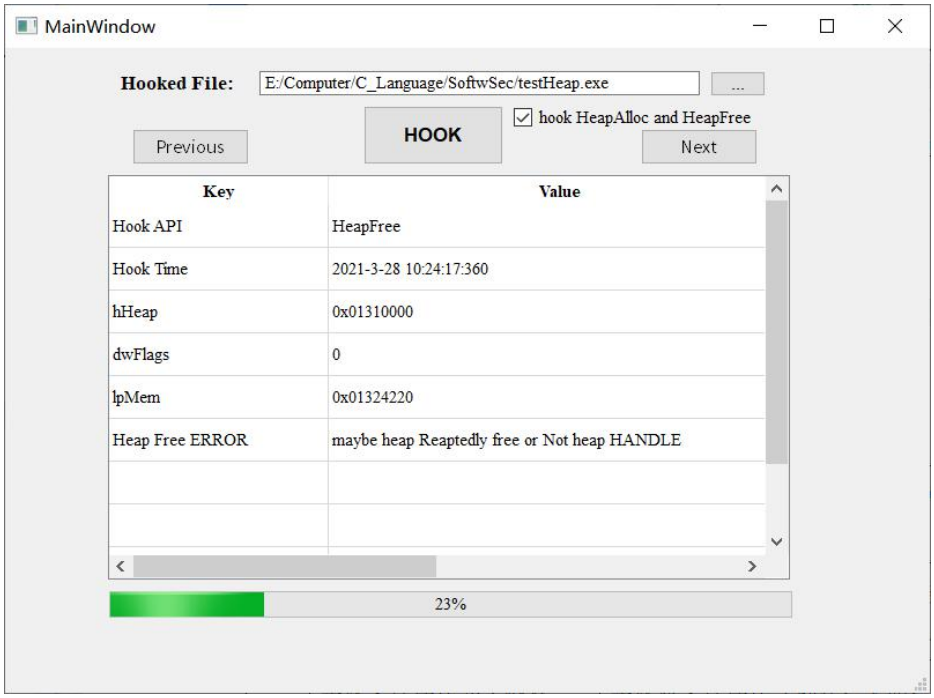

图 5-8 堆重复释放的截获

5.6 文件操作异常行为分析

当目标程序尝试进行自我复制时，系统会对其信息进行截获并输出，输出的信息包括当前目标程序的文件路径以及被复制的文件路径，当二者一致时则说明有自我复制现象，会提示“File Copies Itself！”如图 5-9 所示。

当目标程序试图使用 WriteFile 修改一个可执行文件时，也会被截获到，并提示该程序修改了可执行文件“Modified executable file”，同时输出了修改的可执行文件路径，如图 5-10 所示。

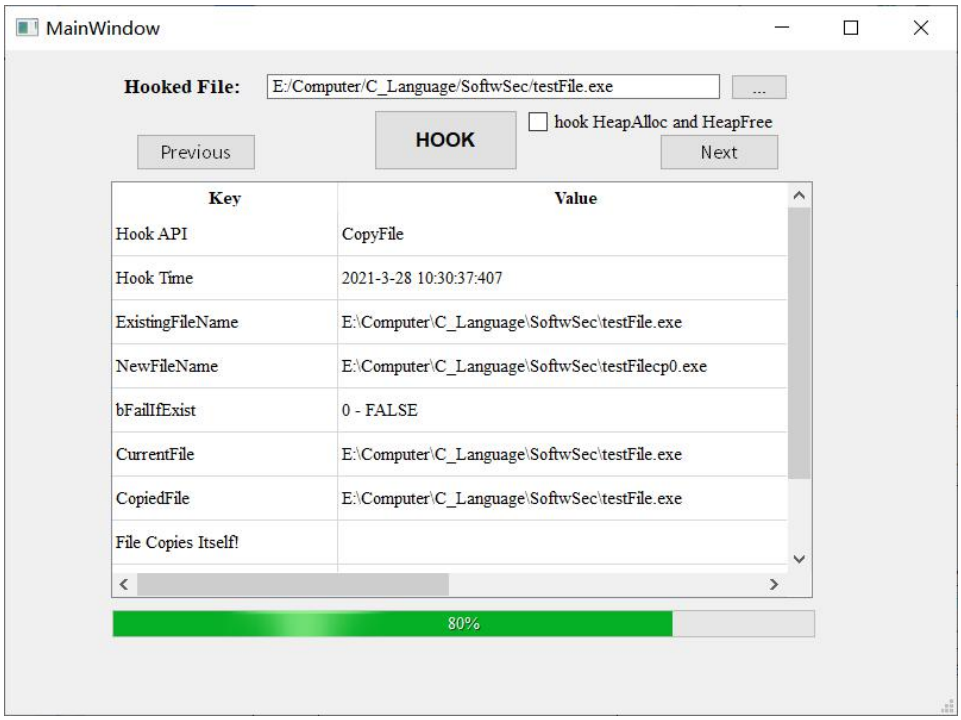

图 5-9 文件自我复制的截获

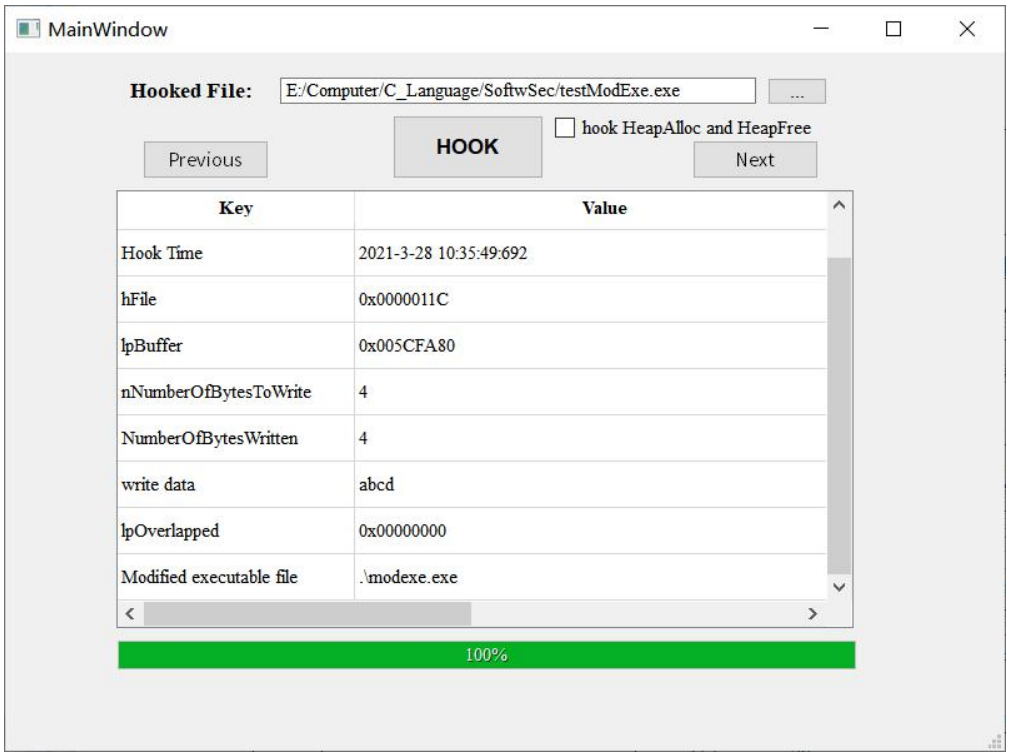

图 5-10 修改可执行文件的截获

5.7 注册表操作异常行为分析

注册表操作的异常行为主要是针对是否修改了系统注册表的自启动项，会进行截获并输出相应的提示信息。如图 5-11 所示，目标程序尝试使用 RegSetKeyValue 添加系统注册表项，此处被截获，并输出了提示信息。

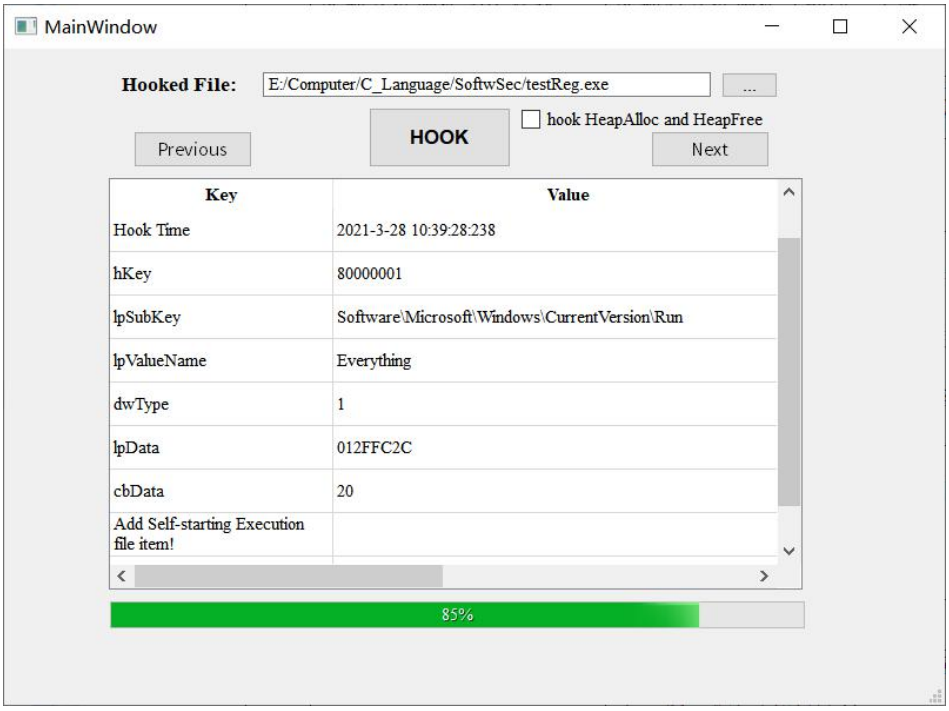

图 5-11 添加注册表自启动项的截获

5.8 套接字 API 截获

对于套接字 API 的截获，此处需要开启一个用于测试的服务端进程，然后使用系统程序注入待测试的客户端程序，即可对套接字相关的 API 进行截获。

如图 5-12 所示，截获了 connect 函数的参数信息，包括其套接字描述符信息、进程的地址类型、目标 IP 地址和端口号。

如图 5-13 所示，截获了客户端进程发给服务端进程的 send 函数，其中就记录的发送的数据大小，以及发送的数据“74126985”。

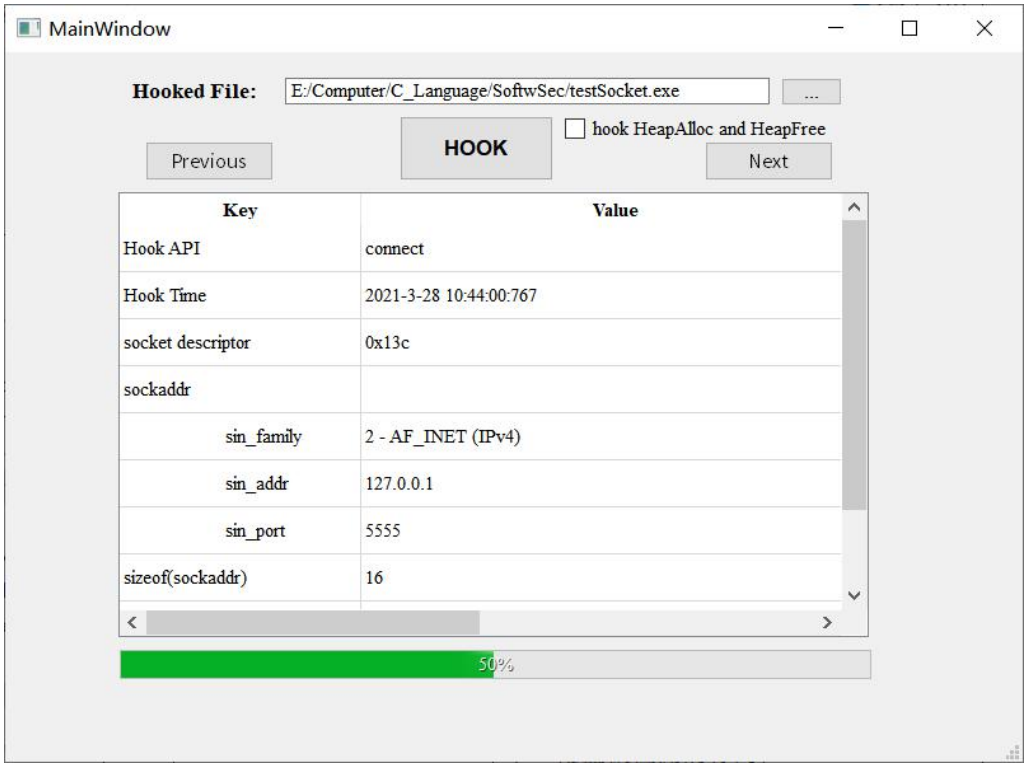

图 5-12 connect 函数的截获

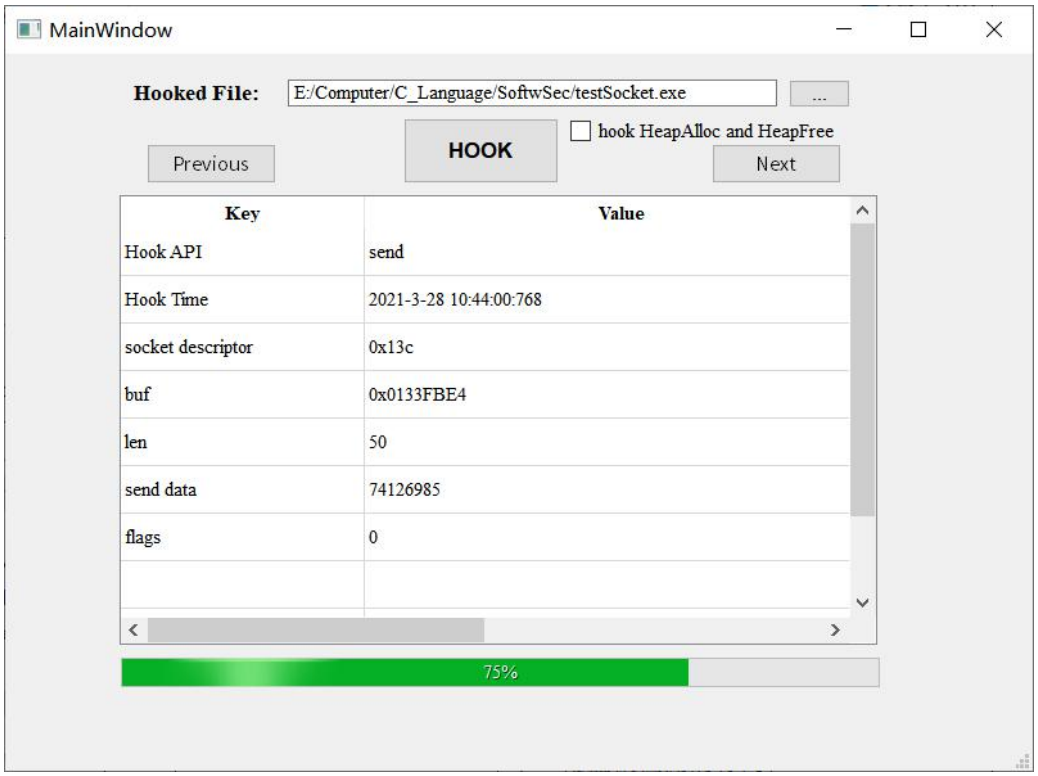

图 5-13 send 函数的截获

6　总结与展望

6.1　总结

在本次课程设计中，我了解了 Windows 操作系统常用的 API 函数，涉及弹窗操作、堆操作、文件操作、注册表操作等，学习使用了微软 Detours 开源库，实现了对上述 API 函数的截获。在这个过程中，同时学习到了这些函数操作背后隐藏的一些异常行为，如何在无源码情况下对样本程序进行行为分析、异常分析，以及如何通过截获到的函数信息来推断出可能存在的异常行为。

同时，通过对程序的异常行为分析，也反过来让我注意到在平常编程的过程中需要注意哪些方面，对如何编写出对系统更加安全、无异常行为的缜密的程序有很多启发和帮助。

此外，在本次课设中，也提高了我的编程能力，了解了动态链接库的函数的编写方式、DLL 源程序结构等。同时在系统界面设计方面，本次摒弃了传统而繁琐的 MFC 框架，选择了当下较为主流的 C++图形界面框架 Qt，学习了 Qt 的图形编程，及其其相应的信号与槽等触发机制等。

6.2　展望

本次课程设计同样还有许多值的改进和提高的地方，比如由于使用了 Qt 提供的进程间的管道通信，虽然数据读取方面较为便捷，但缺少了一定的实时性，所有的数据结果是在程序执行完毕后统一发送到系统前端界面的，可以考虑使用共享内存等方法进行完善，使得截获的信息具有实时性。

此外，对于任务中的选做内容，比如判断是否将文件内容读取后发送的网络，套接字传输的信息是否为 HTTP 报文、是否为明文，以及内存拷贝检测等都没有进行实现，这些方面还可以进行设计实现，以进一步提高程序的异常行为分析能力。

最后，在系统界面上也有完善的空间，当前提供了用户选择目标程序的选项，只提供了是否截获 HeapAlloc 和 HeapFree 函数的选项，应该更进一步，能让用户所有可截获的 API 函数都能选择，这样就更进一步提供了系统的交互性，也对目标程序的 API 截获更有针对性。

7　课程建议与意见

本次课程设计主要是借助 Detours 库进行 API 截获和异常行为分析，多组函数的截获方式实际上是一致的，感觉有些冗余，可以选择一两个进行深入挖掘。

程序的异常分析没有涉及过于多的算法知识，可以简单使用一些标准库函数或者找到对应的 Windows API 函数即可完成，希望能够更具挑战性一些。

最后，希望课程设计之后能够设计 Linux 操作系统以及网络的部分多一些。

8 参考文献

[1] GitHub-Microsoft/Detours https://github.com/microsoft/Detours

[2] Windows 修改注册表实现开机自启动 - CSDN https://blog.csdn.net/hxxjxw/article/details/89785345

[3] Programming reference for the Win32 API - Microsoft https://docs.microsoft.com/en-us/windows/win32/api/

[4] Qt Documentation - https://doc.qt.io/qt-5/index.html