《漏洞利用及渗透测试基础》实验报告

姓名：申宗尚 学号：2213924 班级：信息安全

**实验名称：**

程序插桩及Hook实验

**实验要求：**

复现实验一，基于Windows MyPinTool或在Kali中复现malloctrace这个PinTool，理解Pin插桩工具的核心步骤和相关API，关注malloc和free函数的输入输出信息。

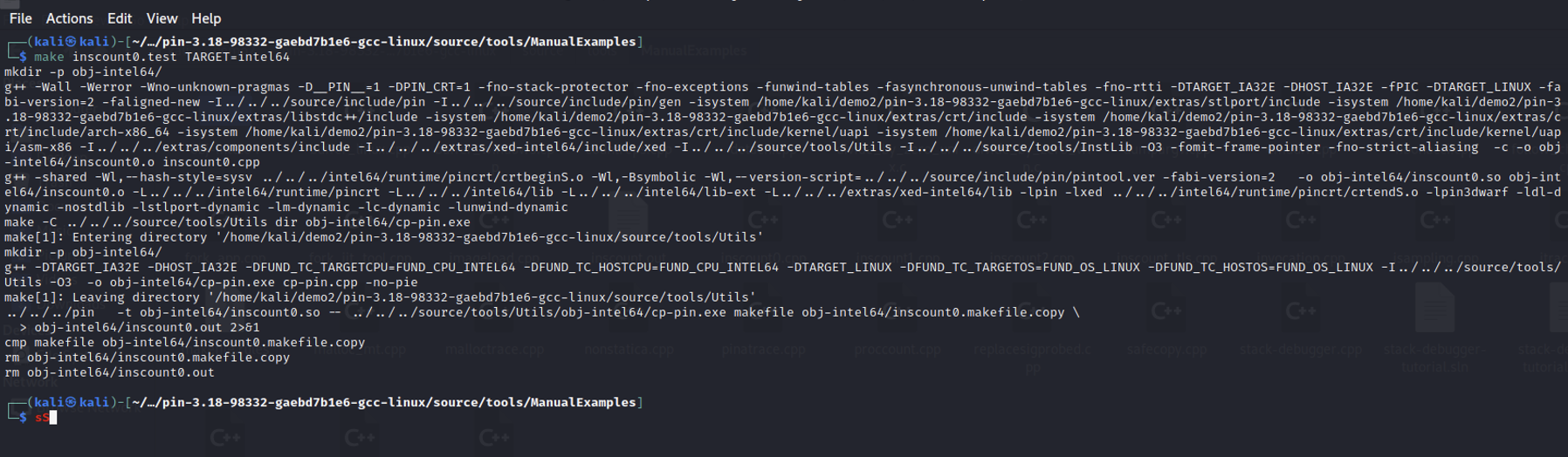
**实验过程：**

1. 首先，从intel官网下载pin-3.18 for linux，将下载好的pin-3.18压缩包拖动并移动到kali系统中的新建本实验文件夹kali/demo1文件夹中，解压。

**图形用户界面, 应用程序, Teams

描述已自动生成**

2. 进入该文件夹下的目录source/tools/ManualExamples，打开终端命令，输入命令：make inscount0.test TARGET=intel64来对inscount0.cpp进行编译。



编译成功，现在进入obj-intel64目录下查看动态链接库文件 inscount0.so ，并直接进入到其终端目录下（方便后续实验输入命令）。

文本

描述已自动生成

3.然后，尝试编写一个简单的控制台输出程序，并使用此动态链接库文件进行插桩实验。

代码如下：

#include <stdio.h>

void main(){Printf(“hello world!”);}

电脑萤幕的截图

描述已自动生成

编译指令：gcc -o testc test.c

pin工具插桩执行命令：/home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t inscount0.so -- /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples/obj-intel64/testc 图形用户界面, 文本

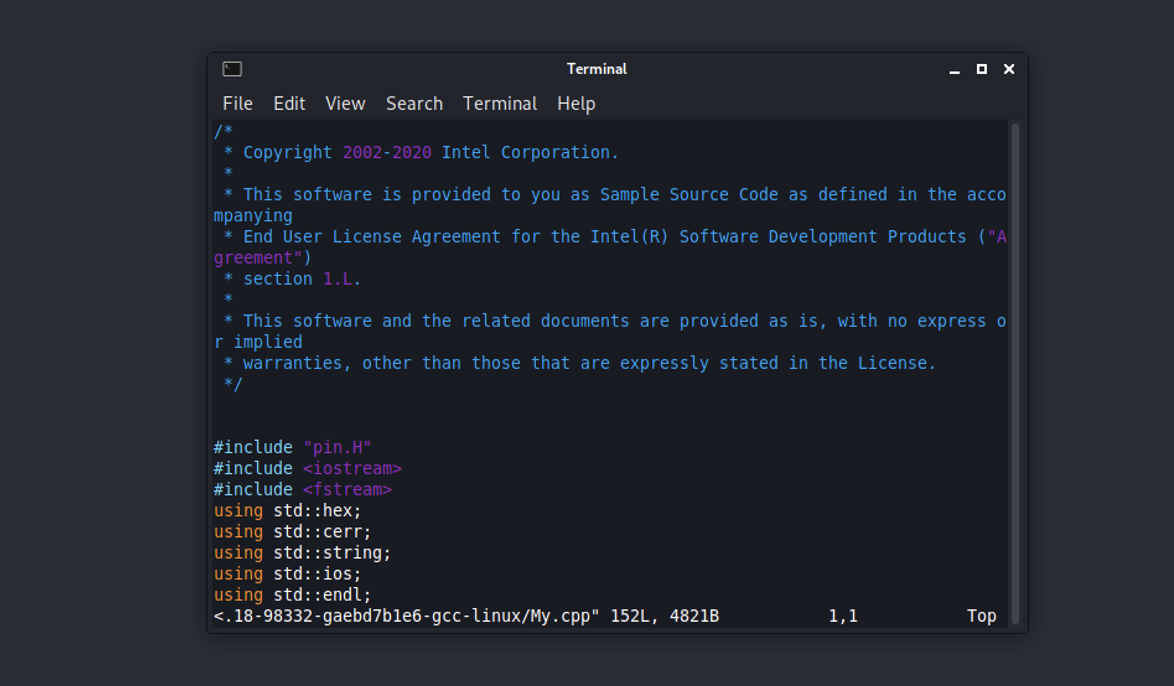
描述已自动生成

4.进行程序插桩完成后便可查看输出文件得知所用到的指令数，插桩实验完成。图形用户界面, 文本

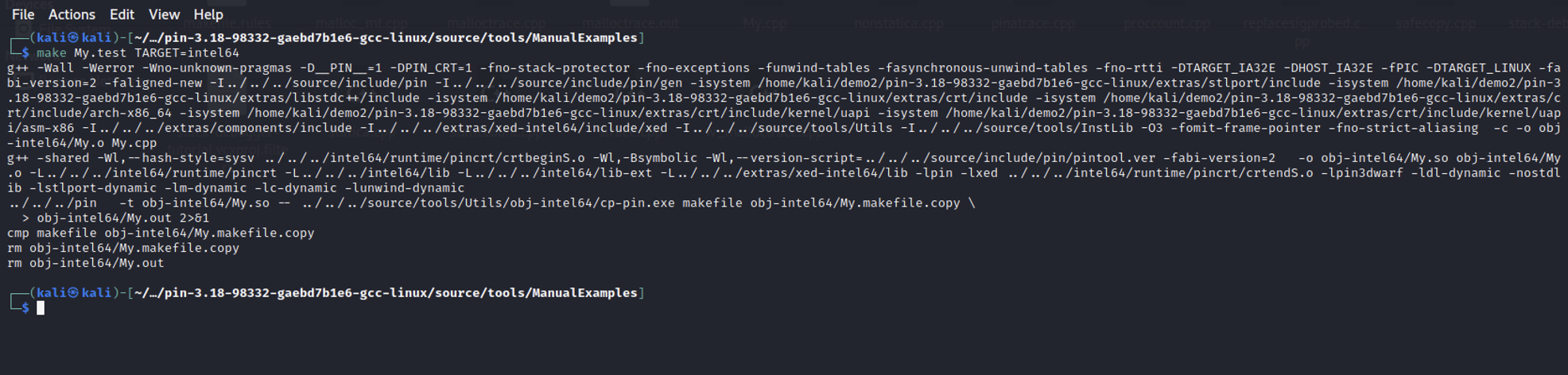
描述已自动生成

5.然后，我们尝试在Kali中复现malloctrace这个PinTool。

首先，新建cpp文件，并将malloctrace.cpp中的内容复制进去



6. 进行编译，得到其对应的动态链接库文件



7.新建测试cpp文件Test.cpp，代码如下：

#include <studio.h>

#include <stdlib.h>

void main(){int \*p = (int\*)malloc(sizeof(int));printf(“Hello,world!”);free(p);}

并且编译为可执行文件：gcc -o TEST Test.cpp

文本

描述已自动生成

8. 使用pin工具进行插桩操作：

命令行代码：/home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t inscount0.so -- TEST4.

图片包含 文本

描述已自动生成

输出文件中显示堆中空间地址的申请与释放情况，比如观察最后几行，可以看到，每次使用malloc申请堆内存空间的时候，都会输出申请的空间的大小，并且会有相应的地址记录。当free内存空间时，可以看到地址与申请时候的地址一一对应。

**核心实验内容：**

1.初始化：调用 PIN\_Init 函数进行初始化操作，解析命令行参数，并为 Pin 工具做好必要的准备工作。

2.注册插桩函数：根据需要的插桩粒度，使用合适的 API 注册插桩函数。例如使用 INS\_AddInstrumentFunction 注册指令级插桩函数:在程序执行每条指令前，会调用注册的插桩函数,使用 IMG\_AddInstrumentFunction 注册镜像级插桩函数，在加载每个可执行文件和共享库时调用插桩函数。

3.注册退出回调函数：使用 PIN\_AddFiniFunction 注册程序退出时的回调函数。当应用程序退出时，该回调函数会被调用，用于执行清理操作或输出统计信息。

4.启动程序：使用 PIN\_StartProgram 启动被插桩的目标程序。这个函数控制目标程序的执行，并开始插桩操作。

<相关 API 及其介绍>

1.PIN\_InitSymbols:初始化符号处理。使用该函数可以处理符号信息，如函数名和变量名，在插桩过程中提供更丰富的上下文信息。

2.PIN\_Init：初始化 Pin 工具，包括解析命令行参数和初始化内部数据结构，确保 Pin 环境准备就绪。

3.IMG\_AddInstrumentFunction：

注册镜像级插桩函数。该函数在每次加载新的可执行文件或共享库时被调用，可以在加载时进行相应的插桩操作。

4.PIN\_AddFiniFunction:注册程序退出时的回调函数。该回调函数在应用程序退出时被调用，用于进行资源释放或输出插桩结果。

5.PIN\_StartProgram：启动被插桩的目标程序，并开始插桩操作。调用此函数后，Pin 工具将控制目标程序的执行流程。

6.RTN\_FindByName：查找特定函数的运行时标识符（RTN）。该函数可以根据函数名查找函数，用于在特定函数执行前后插入自定义操作。

7.RTN\_InsertCall：在找到的函数执行前后插入回调函数。该函数可以在函数入口或出口处插入自定义操作，例如统计函数调用次数或记录参数值。

**心得体会：**

通过本次实验，我对Pin插桩工具的使用有了深刻的理解和实践体验。在实验中，我使用了Intel的Pin工具来进行程序插桩。Pin工具的强大之处在于它能够对程序进行动态二进制插桩，允许我们在不改变源代码的情况下插入自定义代码，从而实现对程序运行时行为的监控和分析。这对于漏洞利用和渗透测试等领域具有重要意义。

同时，我熟悉了Pin工具的核心API，包括PIN\_Init、IMG\_AddInstrumentFunction、PIN\_AddFiniFunction和PIN\_StartProgram等。这些API的使用使我能够在程序的不同阶段插入自定义代码，进行各种操作如统计指令数、记录函数调用等。

而且，我成功复现了malloctrace这个PinTool，跟踪了malloc和free函数的调用。通过插桩，我们可以实时监控内存的分配和释放情况，捕获到每次malloc申请的空间大小及相应的地址，并在free操作时进行对应的地址释放。这让我深刻体会到了插桩工具在内存管理监控中的重要作用。