《漏洞利用及渗透测试基础》实验报告

姓名：齐明杰　　学号：2113997　　班级： 信安 2 班

实验名称：

程序插桩及 Hook 实验

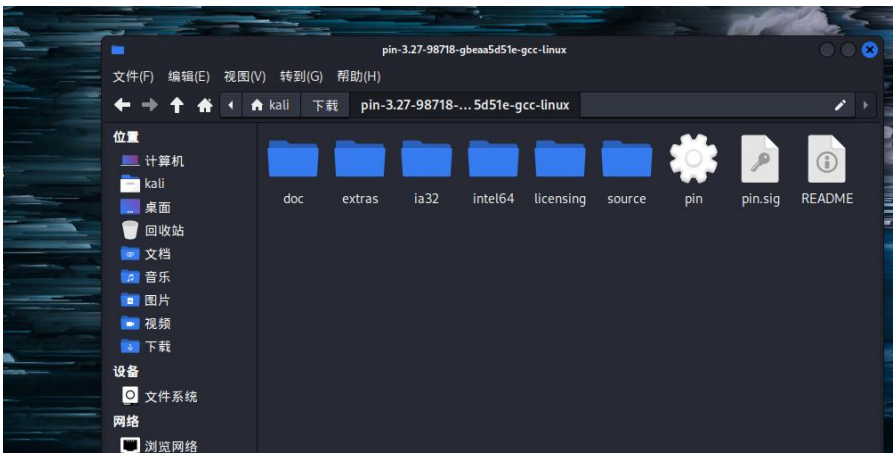
实验要求：

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。

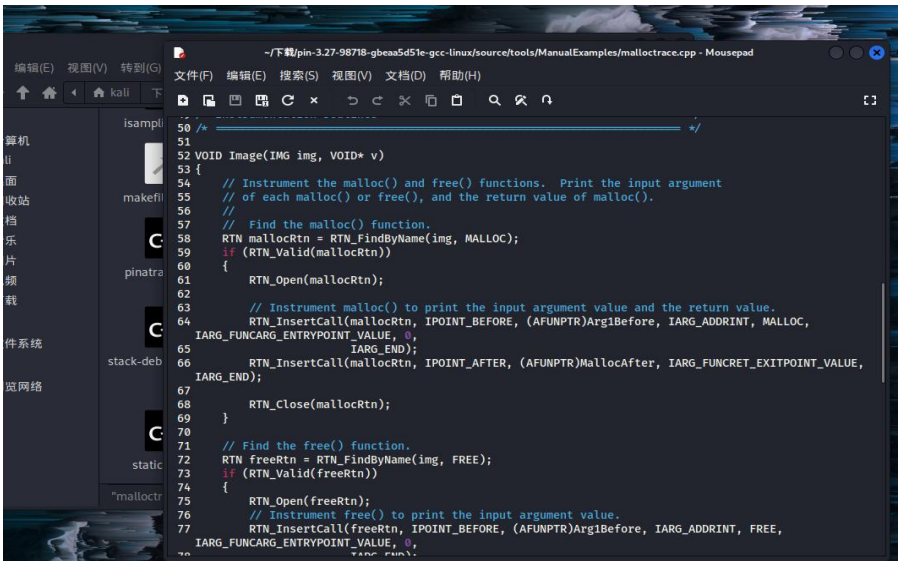
实验过程：

一、 安装 Pin

进入官网下载 linux 版本的 pin，放入 kali 虚拟机中，如下图所示：



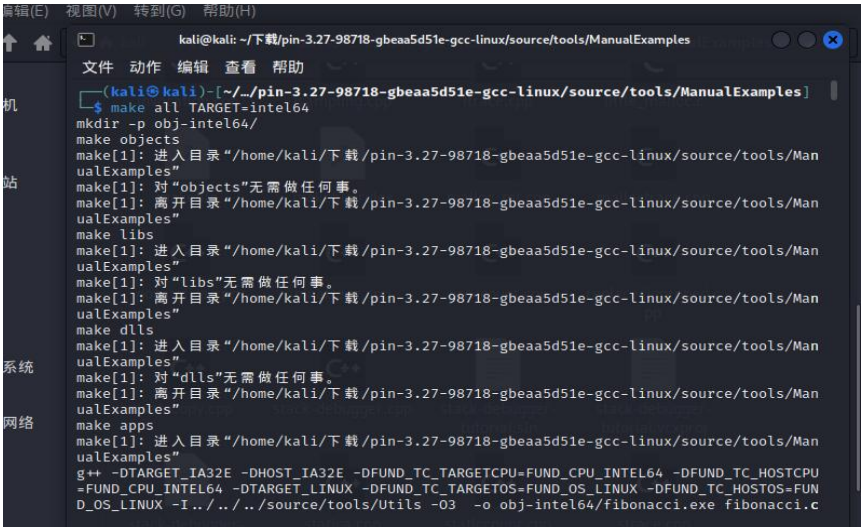
其中我们可以找到 malloctrace.cpp 的代码：



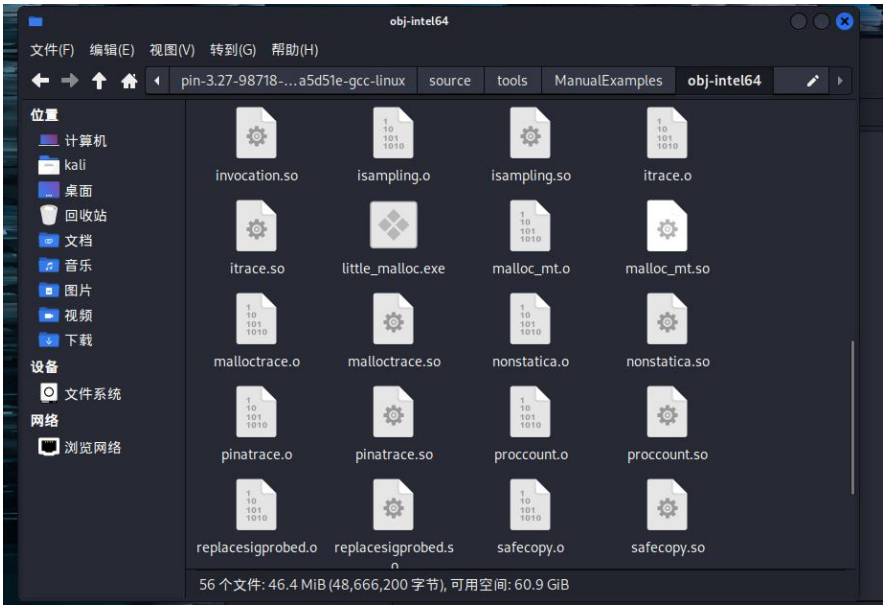
二、 使用 Pintool

首先编译现有 Pintool。在 ManualExamples 文件夹下打开终端，输入命令：

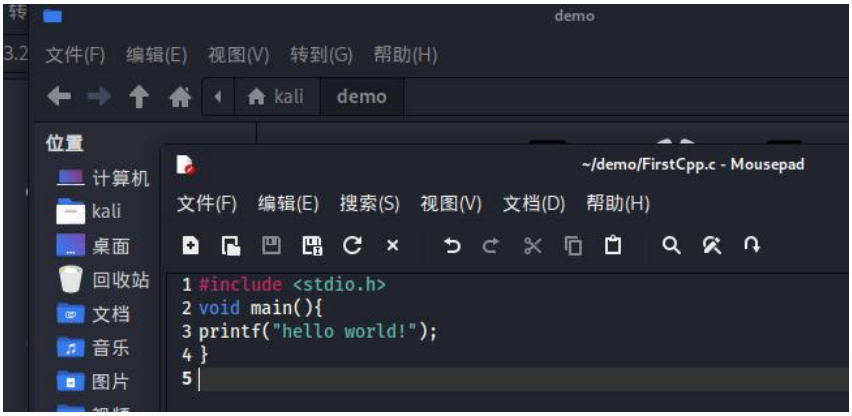
make all TARGET=intel64，即可编译所有 Pintool，如下图所示：



完成后，我们可以在同目录下找到编译出的动态链接库，其中包括本次实验使用的 malloctrace.so：



我们创建一个新文件 FirstCpp.c，作为测试文件，内容如下：

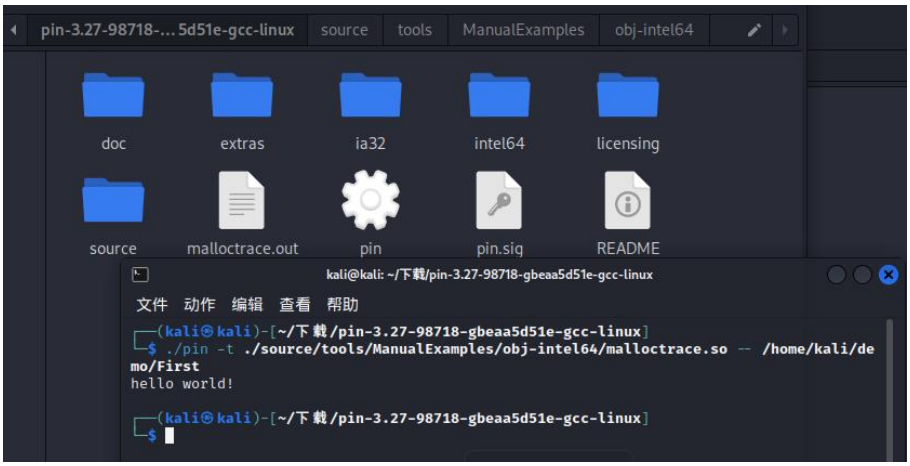


在 Linux 下编译 c 文件的命令为：gcc -o First FirstCpp.c，结果如下图：

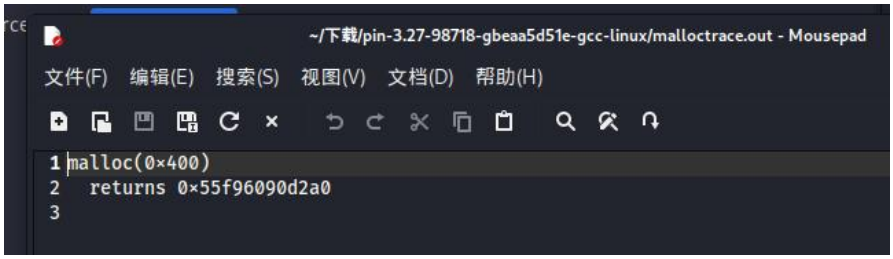
对 First 可执行程序进行程序插桩的 Pin 命令为：

./pin -t ./source/tools/ManualExamples/obj-intel64/malloctrace.so -- /home/kali/demo/First

结果如下图：



如上所示，成功执行。同时，在 pin-3.27 路径下增加了一个输出文件 malloctrace.out，文件内容如下图所示：



输出结果 malloc(0x400)表示调用了 malloc 函数，并请求分配 0x400（即 1024）字节的内存。returns 0x55f96090d2a0 表示 malloc 函数成功地分配了内存，并返回了指向该内存块的指针（0x55f96090d2a0）。

三、 Pintool 基本框架

malloctrace.cpp 代码如下：

```
1.  /*
2.   * Copyright (C) 2004-2021 Intel Corporation.
3.   * SPDX-License-Identifier: MIT
4.   */
5.
6.  #include "pin.H"
7.  #include <iostream>
```

```
8.  #include <fstream>
9.  using std::cerr;
10. using std::endl;
11. using std::hex;
12. using std::ios;
13. using std::string;
14.
15. /* ================================================================== */
16. /* Names of malloc and free */
17. /* ================================================================== */
18. #if defined(TARGET_MAC)
19. #define MALLOC "_malloc"
20. #define FREE "_free"
21. #else
22. #define MALLOC "malloc"
23. #define FREE "free"
24. #endif
25.
26. /* ================================================================== */
27. /* Global Variables */
28. /* ================================================================== */
29.
30. std::ofstream TraceFile;
31.
32. /* ================================================================== */
33. /* Commandline Switches */
34. /* ================================================================== */
35.
36. KNOB< string > KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool", "o", "malloctrace.out
    ", "specify trace file name");
37.
38. /* ================================================================== */
39.
40. /* ================================================================== */
41. /* Analysis routines                                                  */
42. /* ================================================================== */
43.
44. VOID Arg1Before(CHAR* name, ADDRINT size) { TraceFile << name << "(" << size << ")"
     << endl; }
45.
46. VOID MallocAfter(ADDRINT ret) { TraceFile << "  returns " << ret << endl; }
47.
48. /* ================================================================== */
49. /* Instrumentation routines                                           */
```

```
50. /* ===================================================================== */
51.
52. VOID Image(IMG img, VOID* v)
53. {
54.     // Instrument the malloc() and free() functions.  Print the input argument
55.     // of each malloc() or free(), and the return value of malloc().
56.     //
57.     //  Find the malloc() function.
58.     RTN mallocRtn = RTN_FindByName(img, MALLOC);
59.     if (RTN_Valid(mallocRtn))
60.     {
61.         RTN_Open(mallocRtn);
62.
63.         // Instrument malloc() to print the input argument value and the return val
    ue.
64.         RTN_InsertCall(mallocRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before, IARG_ADDRINT,
     MALLOC, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
65.                        IARG_END);
66.         RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR)MallocAfter, IARG_FUNCRET_
    EXITPOINT_VALUE, IARG_END);
67.
68.         RTN_Close(mallocRtn);
69.     }
70.
71.     // Find the free() function.
72.     RTN freeRtn = RTN_FindByName(img, FREE);
73.     if (RTN_Valid(freeRtn))
74.     {
75.         RTN_Open(freeRtn);
76.         // Instrument free() to print the input argument value.
77.         RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before, IARG_ADDRINT, F
    REE, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
78.                        IARG_END);
79.         RTN_Close(freeRtn);
80.     }
81. }
82.
83. /* ===================================================================== */
84.
85. VOID Fini(INT32 code, VOID* v) { TraceFile.close(); }
86.
87. /* ===================================================================== */
88. /* Print Help Message                                                    */
89. /* ===================================================================== */
```

```
90.
91. INT32 Usage()
92. {
93.     cerr << "This tool produces a trace of calls to malloc." << endl;
94.     cerr << endl << KNOB_BASE::StringKnobSummary() << endl;
95.     return -1;
96. }
97.
98. /* ===================================================================== */
99. /* Main                                                                  */
100. /* ===================================================================== */
101.
102. int main(int argc, char* argv[])
103. {
104.     // Initialize pin & symbol manager
105.     PIN_InitSymbols();
106.     if (PIN_Init(argc, argv))
107.     {
108.         return Usage();
109.     }
110.
111.     // Write to a file since cout and cerr maybe closed by the application
112.     TraceFile.open(KnobOutputFile.Value().c_str());
113.     TraceFile << hex;
114.     TraceFile.setf(ios::showbase);
115.
116.     // Register Image to be called to instrument functions.
117.     IMG_AddInstrumentFunction(Image, 0);
118.     PIN_AddFiniFunction(Fini, 0);
119.
120.     // Never returns
121.     PIN_StartProgram();
122.
123.     return 0;
124. }
125.
126. /* ===================================================================== */
127. /* eof */
128. /* ===================================================================== */
```

其中，两个分析函数是 Arg1Before 和 MallocAfter。Arg1Before 在调用 malloc 或 free 之前执行，用于记录函数名称和参数。MallocAfter 在调用 malloc 之后执行，记录返回值。

之后的 Image 函数是用于在程序映像中找到并插桩 malloc 和 free 函数的代码。它会在程序加载时被调用。对于找到的 malloc 和 free 函数，代码使用 RTN_InsertCall 在函数调用前后插入分析函数（如 Arg1Before 和 MallocAfter）。

Fini 函数在 Pin 工具完成后关闭输出文件。

Usage 函数提供了一个帮助信息，当命令行参数有误时显示。

一般 **Pintool 的基本框架**，在 main 函数中：

1、初始化。通过调用函数 PIN_Init 完成初始化。

2、注册插桩函数。通过使用 INS_AddInstrumentFunction 注册一个插桩函数，其第 2 个参数为一个额外传递给 Instruction 的参数，即对应 VOID *v 这个参数，这里没有使用。而 Instruction 接受的第一个参数为 Image 结构，用来表示 malloc 和 free 的调用。

3、注册退出回调函数。通过使用 PIN_AddFiniFunction 注册一个程序退出时的回调函数 Fini，当应用退出的时候会调用函数 Fini。

4、启动程序。使用函数 PIN_StartProgram 启动程序。

心得体会：

通过本次实验，我掌握了如何使用 Intel Pin 工具进行动态二进制插桩，以及如何利用插桩来跟踪和分析程序中的 malloc 和 free 函数调用。此外，我也了解到了程序底层实现中可能存在的间接内存分配行为。这些知识和技能为我在未来研究程序性能、优化内存管理和深入理解底层实现提供了宝贵的经验。