

软件安全实验报告

姓名：辛浩然

学号：2112514

班级：信息安全、法学

1 实验名称

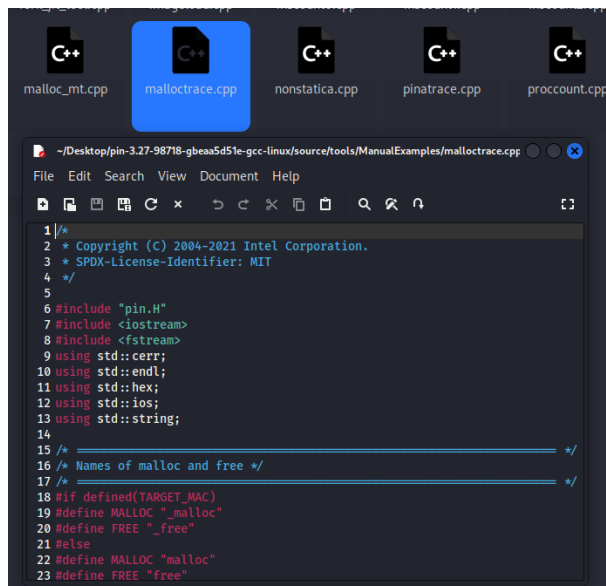
程序插桩及 hook 实验

2 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。

3 实验过程

1. 下载并解压 Pin。在文件夹 source/tools 里包含了大量的 PinTool；
2. 查看 Pintool 工具 malloctrace.cpp，其功能为记录 malloc 和 free 的调用情况；



```
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"
```

代码如下：

```
1 #include "pin.H"  
2 #include <iostream>  
3 #include <fstream>  
4 using std::cerr;  
5 using std::endl;  
6 using std::hex;
```

```

7      using std::ios;
8      using std::string;
9
10     // 如果是 Mac 平台, 使用下划线开头的函数名
11     #if defined(TARGET_MAC)
12     #define MALLOC "_malloc"
13     #define FREE "_free"
14     #else
15     #define MALLOC "malloc"
16     #define FREE "free"
17     #endif
18
19     std::ofstream TraceFile;
20
21     // 定义一个命令行参数 "-o", 用于指定输出文件名
22     KNOB< string > KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool", "o",
23     "malloctrace.out", "specify trace file name");
24
25     // 记录 malloc 函数调用前的参数, 即内存大小
26     VOID Arg1Before(CHAR* name, ADDRINT size) { TraceFile << name << "("
27     " << size << ")" << endl; }
28
29     // 记录 malloc 函数调用返回值, 即分配到的内存地址
30     VOID MallocAfter(ADDRINT ret) { TraceFile << " returns " << ret <<
31     endl; }
32
33     // 为每个被加载的二进制文件进行操作, 查找并 hook 目标函数
34     VOID Image(IMG img, VOID* v)
35     {
36
37         RTN mallocRtn = RTN_FindByName(img, MALLOC); // 查找 MALLOC 函
38         数
39         if (RTN_Valid(mallocRtn)) // 找到了函数
40         {
41             RTN_Open(mallocRtn);
42             // 在函数调用前插入一个回调函数 Arg1Before, 记录 malloc 函
43             数调用前的参数
44             RTN_InsertCall(mallocRtn, IPOINT_BEFORE, (AFUNPTR)
45             Arg1Before, IARG_ADDRINT, MALLOC, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
46             IARG_END);
47             // 在函数调用后插入一个回调函数 MallocAfter, 记录 malloc 函
48             数的返回值

```

```

42         RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR)
MallocAfter, IARG_FUNCRET_EXITPOINT_VALUE, IARG_END);
43
44         RTN_Close(mallocRtn);
45     }
46
47     RTN freeRtn = RTN_FindByName(img, FREE); // 查找 FREE 函数
48     if (RTN_Valid(freeRtn)) // 找到了函数
49     {
50         RTN_Open(freeRtn);
51         // 在函数调用前插入一个回调函数 Arg1Before, 记录 free 函数
调用前的参数
52         RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before,
IARG_ADDRINT, FREE, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
53             IARG_END);
54         RTN_Close(freeRtn);
55     }
56 }
57
58 // 当程序结束时关闭输出文件
59 VOID Fini(INT32 code, VOID* v) { TraceFile.close(); }
60
61 // 输出命令行参数用法
62 INT32 Usage()
63 {
64     cerr << "This tool produces a trace of calls to malloc." <<
endl;
65     cerr << endl << KNOB_BASE::StringKnobSummary() << endl;
66     return -1;
67 }
68
69 // 主函数
70 int main(int argc, char* argv[])
71 {
72     // 初始化Pin符号表
73     PIN_InitSymbols();
74     // 如果初始化Pin工具失败, 则返回使用信息
75     if (PIN_Init(argc, argv))
76     {
77         return Usage();
78     }
79     // 打开Trace文件

```

```

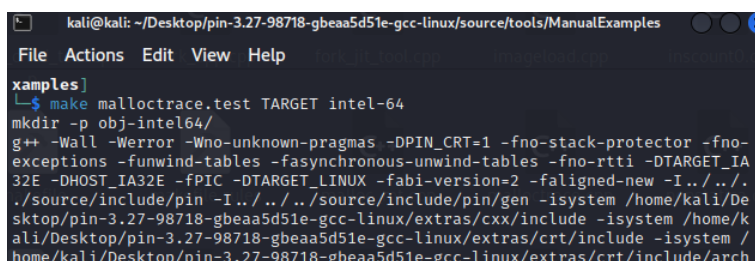
80     TraceFile.open(KnobOutputFile.Value().c_str());
81     // 将输出设置为16进制格式
82     TraceFile << hex;
83     // 显示地址前缀
84     TraceFile.setf(ios::showbase);
85     // 注册Image函数作为处理器函数，对加载的每个镜像进行处理
86     IMG_AddInstrumentFunction(Image, 0);
87     // 注册Fini函数作为结束函数，当程序结束时执行
88     PIN_AddFiniFunction(Fini, 0);
89     // 启动Pin工具并进入Pin指令执行循环，该函数不会返回
90     PIN_StartProgram();
91     return 0;
92 }
93

```

代码使用 Pin 工具插桩 malloc 和 free 函数，并输出它们的调用轨迹。具体来说，它使用 IMG_AddInstrumentFunction 函数来注册一个函数 Image，该函数会在加载新镜像时被调用，然后通过 RTN_FindByName 函数寻找 malloc 和 free 函数，并使用 RTN_InsertCall 在这些函数的入口处添加回调函数 Arg1Before，以输出函数的参数。对于 malloc 函数，还在它的出口处添加回调函数 MallocAfter，以输出函数的返回值。代码中还使用了一些 Pin 提供的工具函数，如 KNOB 和 ofstream 等。最后，它使用了 PIN_StartProgram 函数来启动 Pin 工具，并在程序结束时关闭输出文件。

代码调用了 Pin Tool 的 API 函数，包括：PIN_InitSymbols()、PIN_Init(argc,argv)、IMG_AddInstrumentFunction()、RTN_FindByName()、RTN_Valid()、RTN_Open()、RTN_InsertCall()、RTN_Close()、PIN_AddFiniFunction()、PIN_StartProgram() 等。

3. 进入 source/tools/ManualExamples，对 malloctrace.cpp 进行编译来产生其对应的动态链接库，所使用的命令为：make malloctrace.test TARGET=intel64

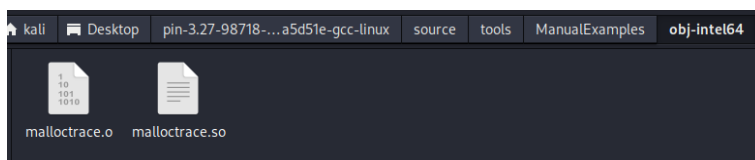


```

kali@kali: ~/Desktop/pin-3.27-98718-gbeaa5d51e-gcc-linux/source/tools/ManualExamples
File Actions Edit View Help
examples]
$ make malloctrace.test TARGET intel-64
mkdir -p obj-intel64/
g++ -Wall -Werror -Wno-unknown-pragmas -DPIN_CRT=1 -fno-stack-protector -fno-exceptions -funwind-tables -fasynchronous-unwind-tables -fno-rtti -DTARGET_IA32E -DHOST_IA32E -fPIC -DTARGET_LINUX -fabi-version=2 -faligned-new -I../.. ./source/include/pin -I../.. /source/include/pin/gen -isystem /home/kali/Desktop/pin-3.27-98718-gbeaa5d51e-gcc-linux/extras/cxx/include -isystem /home/kali/Desktop/pin-3.27-98718-gbeaa5d51e-gcc-linux/extras/crt/include -isystem /home/kali/Desktop/pin-3.27-98718-gbeaa5d51e-gcc-linux/extras/crt/include/arch

```

进入 obj-intel64 文件夹确认一下，已经生成了 malloctrace0.so 这个动态链接库文件。



4. 编写一个控制台命令程序 test.cpp，并进行测试，代码如下：

```

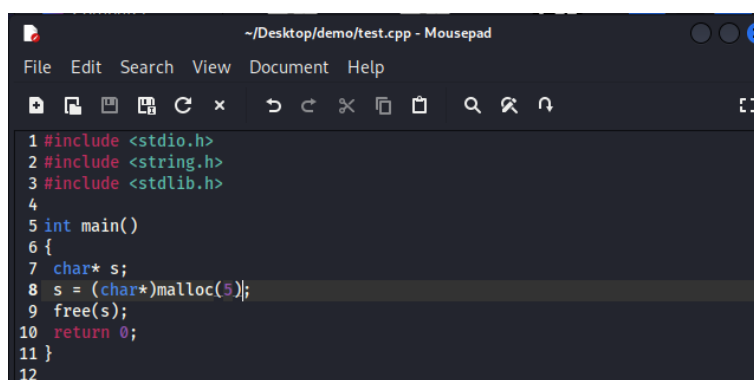
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>

```

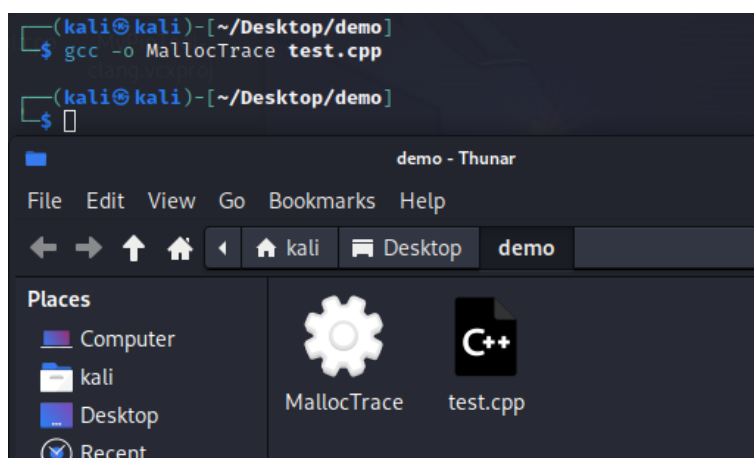
```

4  int main()
5  {
6      // 声明一个指向字符的指针变量s
7      char* s;
8      // 使用malloc分配5个字节的内存空间，并将其地址赋给指针变量s
9      s = (char*)malloc(5);
10     // 释放之前分配的内存空间
11     free(s);
12     return 0;
13 }

```

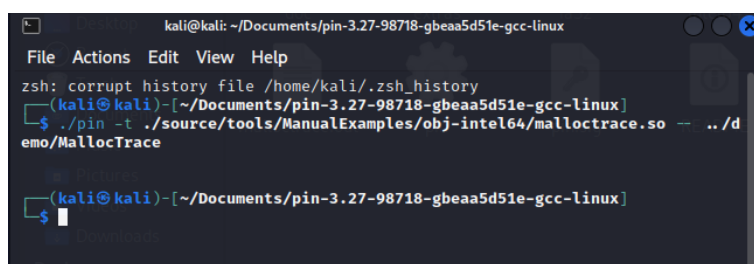


在 Linux 下编译该文件的命令为: `gcc -o MallocTrace test.cpp`, 生成 MallocTrace 可执行程序。

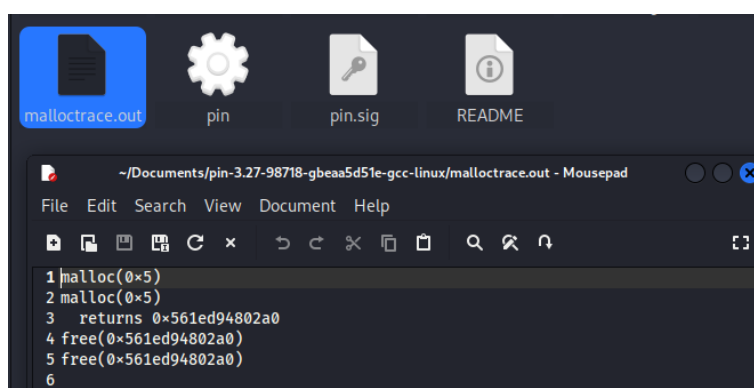


5. 对 MallocTrace 可执行程序进行程序插桩的 Pin 命令为: `./pin -t ./source/tools/ManualExamples/obj-intel64/malloctrace.so - ./demo/MallocTrace`

该命令将使用 Pin 工具 malloctrace0.so 来跟踪 MallocTrace 中的内存分配和释放操作。执行完命令后, 将输出应用程序执行期间的内存分配和释放的详细信息。



在路径下增加了一个输出文件 `malloctrace.out`，内容为：



`malloc(0x5)` 是分配大小为 5 个字节的内存块；`free(0x561ed94802a0)` 是释放 `0x561ed94802a0` 为起始地址的分配的内存空间。

6. 总结

Pin 通过已经定义的 `tools` 或者自己开发的 `tool` 来完成对目标程序的插桩。

在 Pin 的安装文件里，在 `source/tools` 里已经定义了大量 `PinTool`，可以编译后直接使用，具体步骤如下：

- (1) Linux 下编译现有 `Pintool`，产生动态链接库。
- (2) 编写简单程序，并进行编译，生成可执行程序。
- (3) 对可执行程序进行程序插桩。Pin 会启动目标程序，并在程序执行期间对指令进行插桩，记录下相关信息。
- (4) 插桩操作完成后，Pin 会在指定的输出文件（默认为 `inscount.out`）中输出插桩结果。

4 心得体会

通过本次实验，复现了 `malloctrace` 这个 `PinTool`，理解了 Pin 插桩工具的核心步骤和相关 API。