

《程序插桩及 Hook 实验》实验报告

姓名：汤清云 学号：2013536 班级： 1075

实验名称：

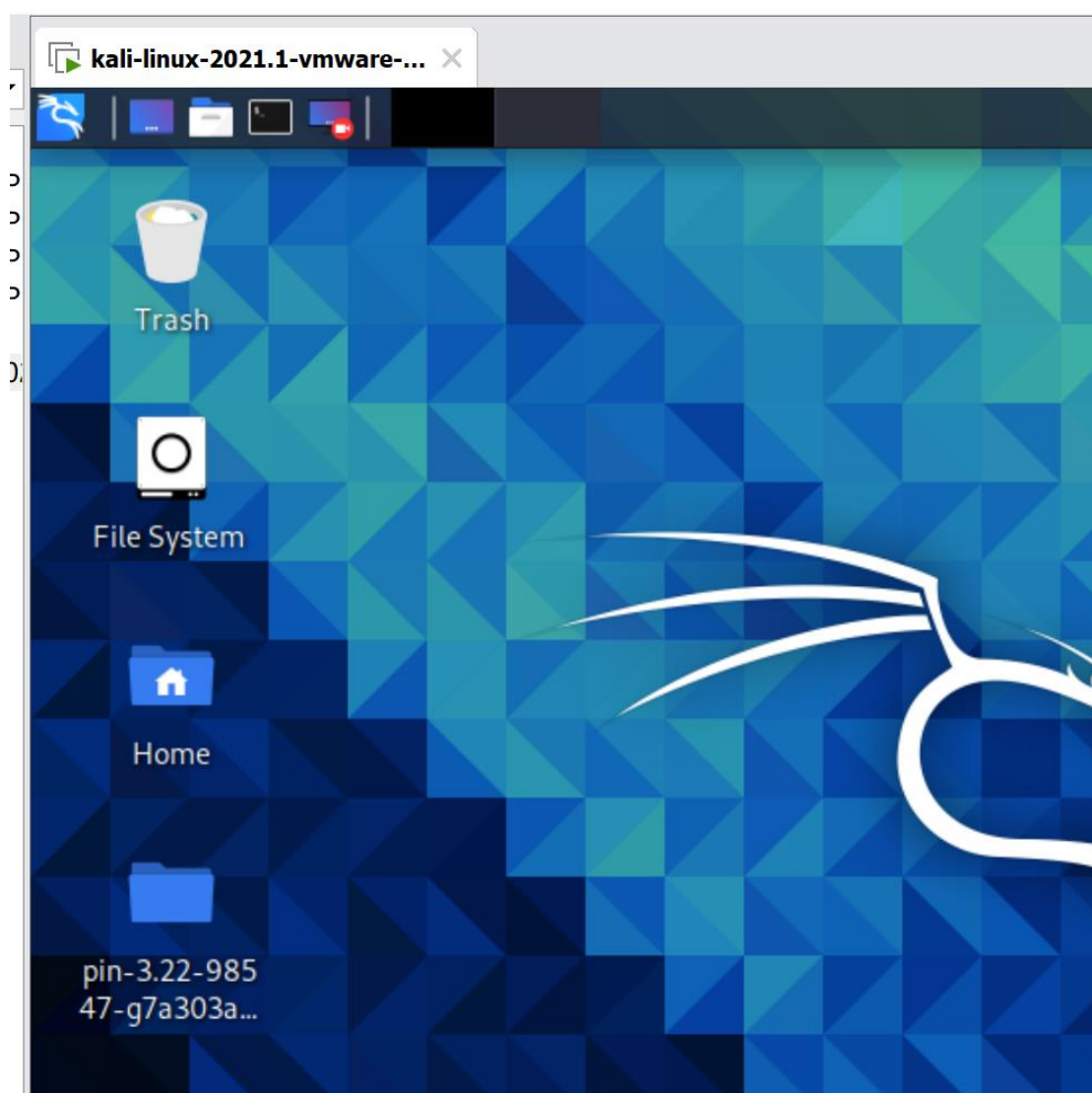
程序插桩以及 hook 实验

实验要求：

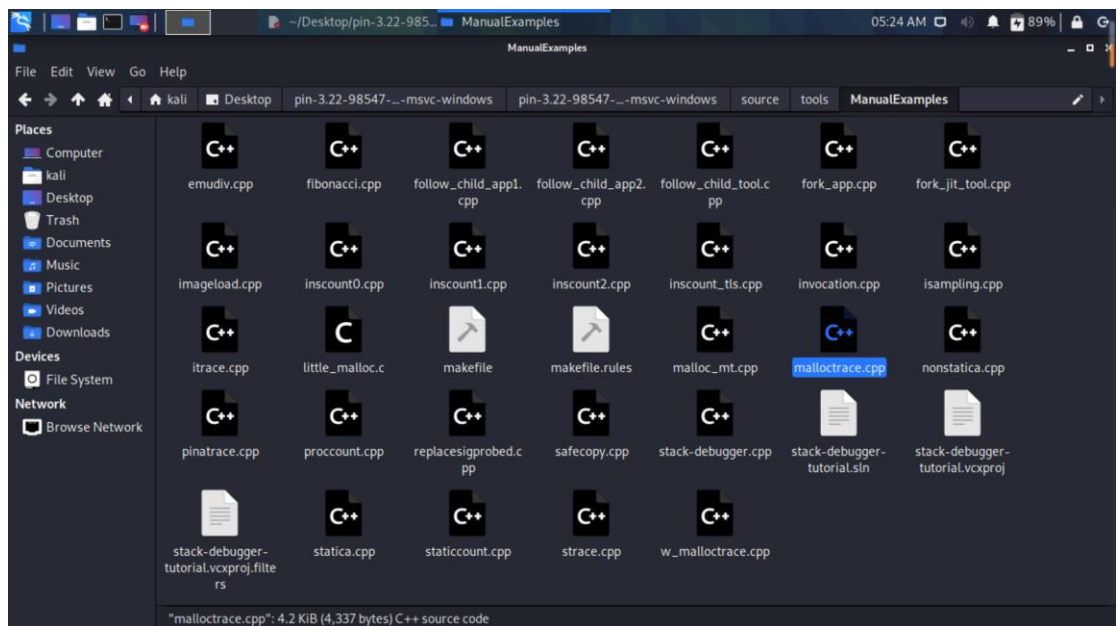
实验过程：

1. 安装 pin:

在官网下载 pin 压缩包，解压缩后移入 kali 虚拟机：



2. 打开 source-tools-manualExamples-malloctrace.cpp, 查看该 pintool



3. Malloctrace 中代码如下：

```
#include "pin.H"
#include <iostream>
#include <fstream>
using std::cerr;
using std::endl;
using std::hex;
using std::ios;
using std::string;
#if defined(TARGET_MAC)
#define MALLOC "_malloc"
#define FREE "_free"
#else
#define MALLOC "malloc"
#define FREE "free"
#endif
std::ofstream TraceFile;
KNOB< string > KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool", "o",
"malloctrace.out", "specify trace file name");
//指定输出文件为 malloctrace.out
VOID Arg1Before(CHAR* name, ADDRINT size) { TraceFile << name <<
 "(" << size << ")" << endl; }
VOID MallocAfter(ADDRINT ret) { TraceFile << " returns " << ret
<< endl; }
VOID Image(IMG img, VOID* v)
{
    //测试 malloc() 和 free() 函数。打印每个 malloc() 或 free() 的输入参
    数，以及 malloc() 的返回值。
    // 找到 malloc 函数
```

```

RTN mallocRtn = RTN_FindByName(img, MALLOC);
if (RTN_Valid(mallocRtn))
{
    RTN_Open(mallocRtn);
    //使用 malloc() 来打印输入参数值和返回值。
    RTN_InsertCall(mallocRtn, IPOINT_BEFORE,
(AFUNPTR)Arg1Before, IARG_ADDRINT, MALLOC,
IARG_FUNCARG_ENTRYPOINT_VALUE, 0, IARG_END);
    RTN_InsertCall(mallocRtn, IPOINT_AFTER,
(AFUNPTR)MallocAfter, IARG_FUNCRET_EXITPOINT_VALUE, IARG_END);
    RTN_Close(mallocRtn);
}
//找到 free 函数。
RTN freeRtn = RTN_FindByName(img, FREE);
if (RTN_Valid(freeRtn))
{
    RTN_Open(freeRtn);
    //使用 free() 输出输入参数值。
    RTN_InsertCall(freeRtn, IPOINT_BEFORE,
(AFUNPTR)Arg1Before, IARG_ADDRINT, FREE,
IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
IARG_END);
    RTN_Close(freeRtn);
}
}
//当应用退出的时候调用本函数
VOID Fini(INT32 code, VOID* v) { TraceFile.close(); }
INT32 Usage()
{
    cerr << "This tool produces a trace of calls to malloc." <<
endl;
    cerr << endl << KNOB_BASE::StringKnobSummary() << endl;
    return -1;
}

int main(int argc, char* argv[])
{
    //初始化引脚和符号管理器
    PIN_InitSymbols();
    if (PIN_Init(argc, argv))//调用函数 pin_init 完成初始化
    {
        return Usage();
    }
}

```

```

//写入文件，因为 cout 和 cerr 可能被应用程序关闭
TraceFile.open(KnobOutputFile.Value().c_str());
TraceFile << hex;
TraceFile.setf(ios::showbase);

//将要调用的图像配准到仪器函数中。
IMG_AddInstrumentFunction(Image, 0); //注册一个插桩函数，在原始
程序的每条指令执行前都会进入 image 函数中。
PIN_AddFiniFunction(Fini, 0); //注册退出回调函数，在退出时调用
该函数

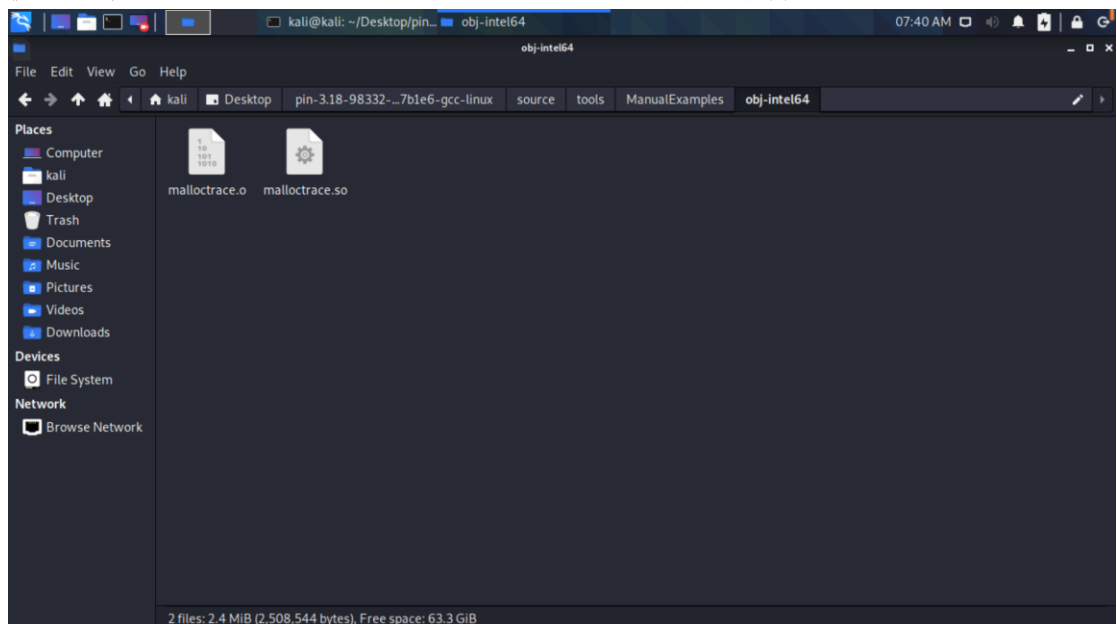
// Never returns
PIN_StartProgram(); //使用该函数启动程序

return 0;
}

```

4. 编译运行，产生动态链接库：

使用语句 `make malloctrace.test TARGET intel-64` 生成链接库



5. 在别处生成 test 文件，包含 malloc 和 free 函数。

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    char* str;
    /* 最初的内存分配 */
    str = (char*)malloc(15);
}

```

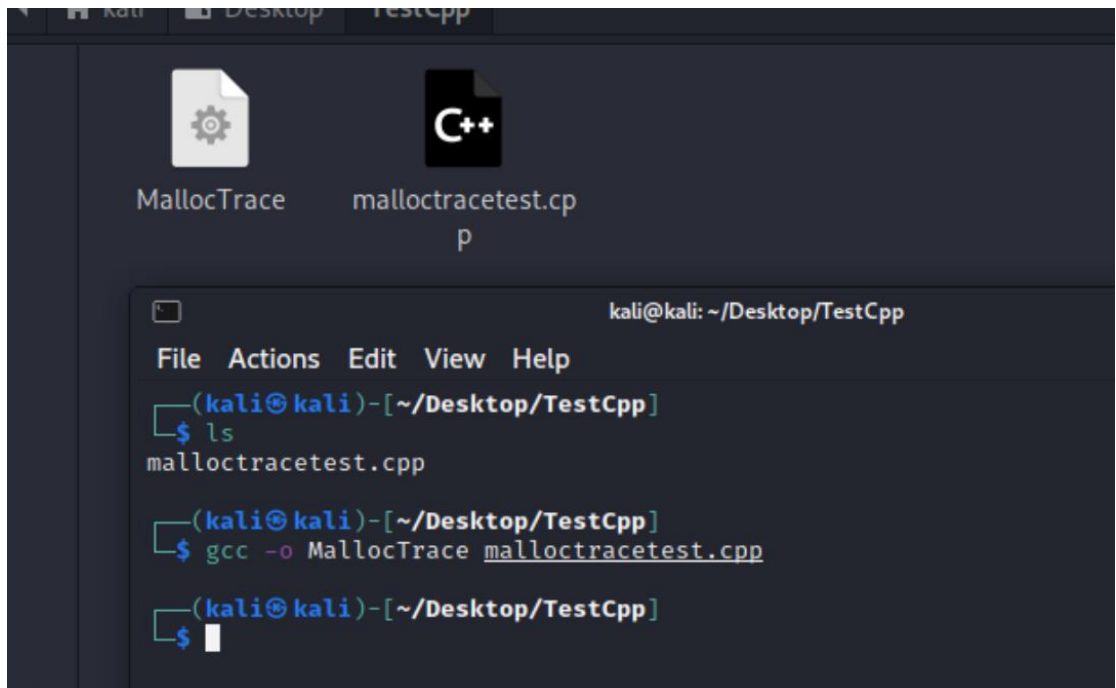
```

    free(str);

    return 0;
}

```

6. 使用语句 `gcc -o MallocTrace malloctracetest.cpp` 进行编译



The screenshot shows a terminal window with the following commands and output:

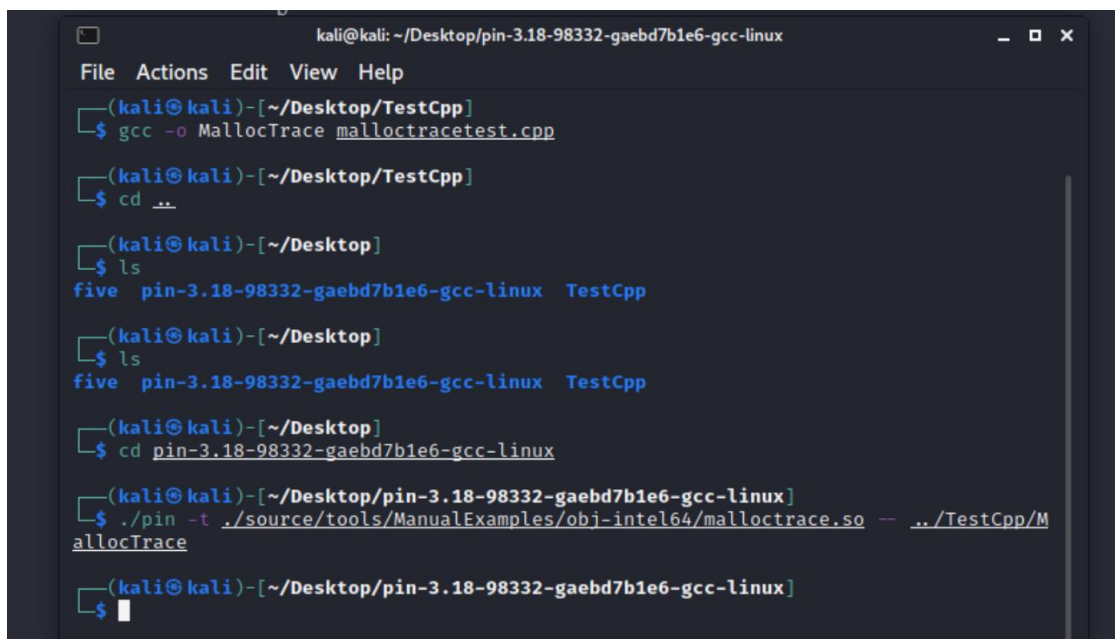
```

kali@kali: ~/Desktop/TestCpp
$ ls
malloctracetest.cpp

kali@kali: ~/Desktop/TestCpp
$ gcc -o MallocTrace malloctracetest.cpp

```

7. 使用语句 `./pin -t ./source/tools/ManualExamples/obj-intel64/malloctrace.so -- ../TestCpp/MallocTrace` 链接动态链接库并且运行:



The screenshot shows a terminal window with the following commands and output:

```

kali@kali: ~/Desktop/pin-3.18-98332-gaebd7b1e6-gcc-linux
$ gcc -o MallocTrace malloctracetest.cpp

kali@kali: ~/Desktop/TestCpp
$ cd ..

kali@kali: ~/Desktop
$ ls
five pin-3.18-98332-gaebd7b1e6-gcc-linux TestCpp

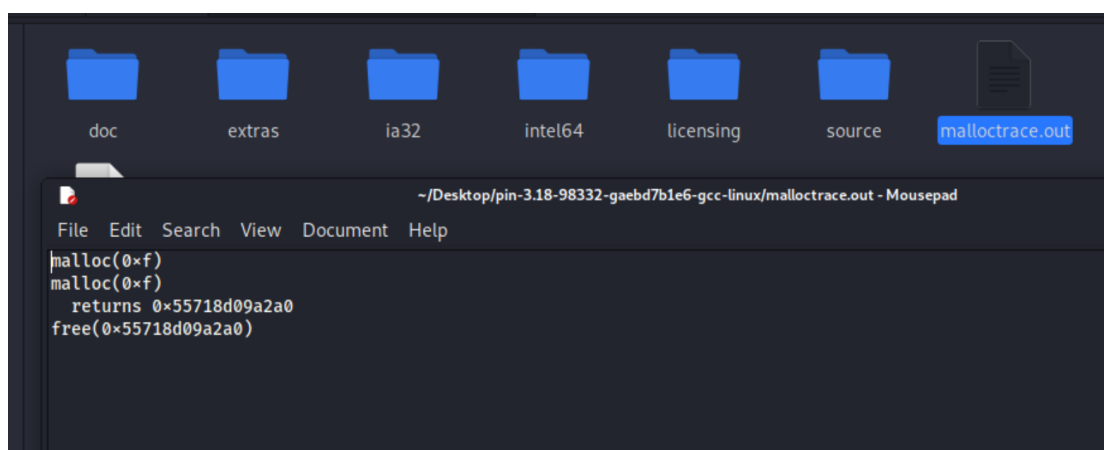
kali@kali: ~/Desktop
$ ls
five pin-3.18-98332-gaebd7b1e6-gcc-linux TestCpp

kali@kali: ~/Desktop
$ cd pin-3.18-98332-gaebd7b1e6-gcc-linux

kali@kali: ~/Desktop/pin-3.18-98332-gaebd7b1e6-gcc-linux
$ ./pin -t ./source/tools/ManualExamples/obj-intel64/malloctrace.so -- ../TestCpp/MallocTrace

```

8. 在 pin 文件夹下可看见输出结果:



9. 0xf 即所预留空间大小，return 的值就是设定空间的首地址。

心得体会：

学习了如何使用 pin 实现函数插桩。