《漏洞利用及渗透测试基础》实验报告

姓名：张洋　　学号：2111460　　班级：信安二班

实验名称：

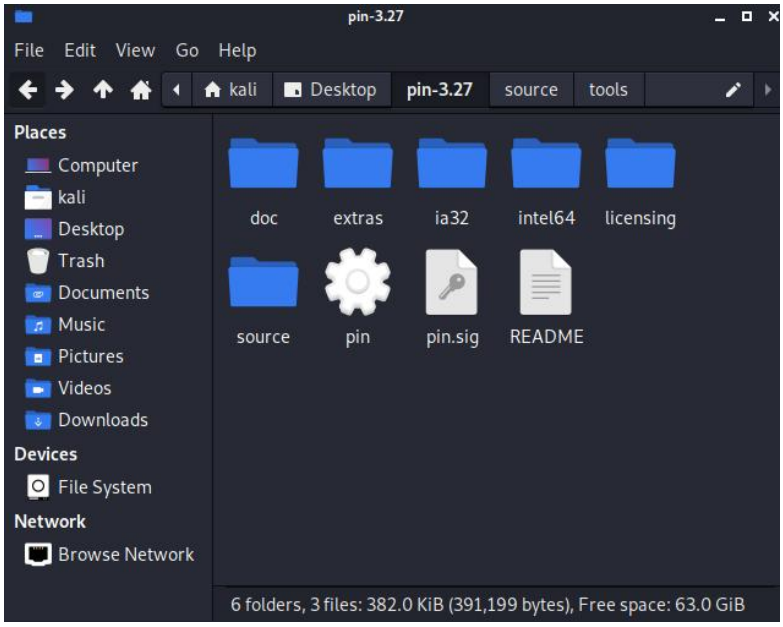
程序插桩及 Hook 实验

实验要求：

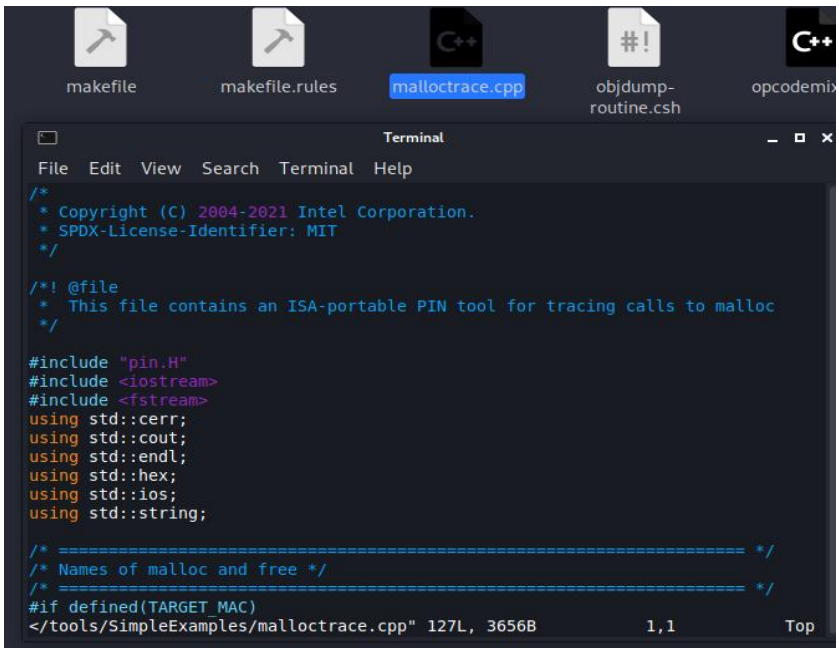
复现实验一，基于 Windows MyPinTool 或在 Kali 中复现 malloctrace 这个 PinTool，理解 Pin 插桩工具的核心步骤和相关 API，关注 malloc 和 free 函数的输入输出信息。

实验过程：

1. 安装 pin:下载 pin 后解压缩移入 kali



2. 打开 source-tools-manualExamples-malloctrace.cpp，查看 Pintool



3. Malloctrace.cpp 中的代码分析：

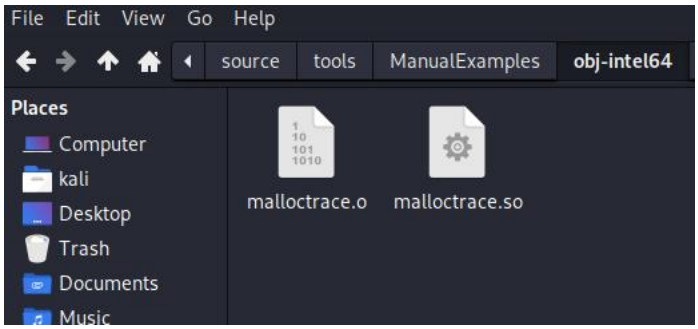
```
#include "pin.H"
#include <iostream>
#include <fstream>
using std::cerr;
using std::endl;
using std::hex;
using std::ios;
using std::string;
#if defined(TARGET_MAC)
#define MALLOC "_malloc"
#define FREE "_free"
#else
#define MALLOC "malloc"
#define FREE "free"
#endif
std::ofstream TraceFile;
KNOB< 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;
    }
```

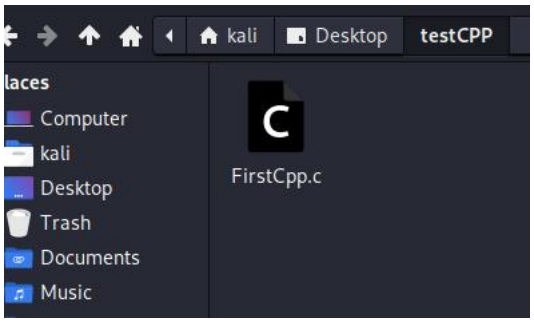
4. 编译运行，产生动态链接库

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

```
┌──(kali㉿kali)-[~/…/pin-3.27/source/tools/ManualExamples]
└─$ make malloctrace.test TARGET intel-64
mkdir -p obj-intel64/
g++ -Wall -Werror -Wno-unknown-pragmas -DPIN_CRT=1 -fno-stack-protector -fno-exceptions
./../../source/include/pin -I../../../source/include/pin/gen -isystem /home/kali/Desktop
ude/arch-x86_64 -isystem /home/kali/Desktop/pin-3.27/extras/crt/include/kernel/uapi -isy
4/include/xed -I../../../source/tools/Utils -I../../../source/tools/InstLib -O3 -fomit-f
g++ -shared -Wl,--hash-style=sysv  ../../../intel64/runtime/pincrt/crtbeginS.o -Wl,-Bsymb
.o -L../../../intel64/runtime/pincrt -L../../../intel64/lib -L../../../intel64/lib-ext -
lm-dynamic -lc-dynamic -lunwind-dynamic
/usr/bin/ld: warning: util_host_ia32e.spp.o: missing .note.GNU-stack section implies exe
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version
make -C  ../../../source/tools/Utils dir obj-intel64/cp-pin.exe
make[1]: Entering directory '/home/kali/Desktop/pin-3.27/source/tools/Utils'
mkdir -p obj-intel64/
g++ -DTARGET_IA32E -DHOST_IA32E -DFUND_TC_TARGETCPU=FUND_CPU_INTEL64 -DFUND_TC_HOSTCPU=F
intel64/cp-pin.exe cp-pin.cpp -no-pie
make[1]: Leaving directory '/home/kali/Desktop/pin-3.27/source/tools/Utils'
../../../pin   -t obj-intel64/malloctrace.so --  ../../../source/tools/Utils/obj-intel64/
  > obj-intel64/malloctrace.out 2>&1
cmp makefile obj-intel64/malloctrace.makefile.copy
rm obj-intel64/malloctrace.makefile.copy
rm obj-intel64/malloctrace.out
make: *** No rule to make target 'TARGET'.  Stop.
```



5. 生成 testCPP 文件，在 FirstCpp.c 中写入 malloc 和 free 函数

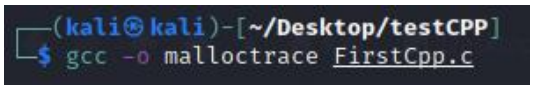


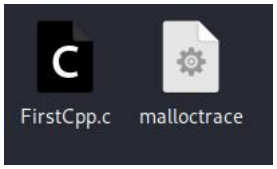
FirstCpp 中的代码：

```
#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;
}
```

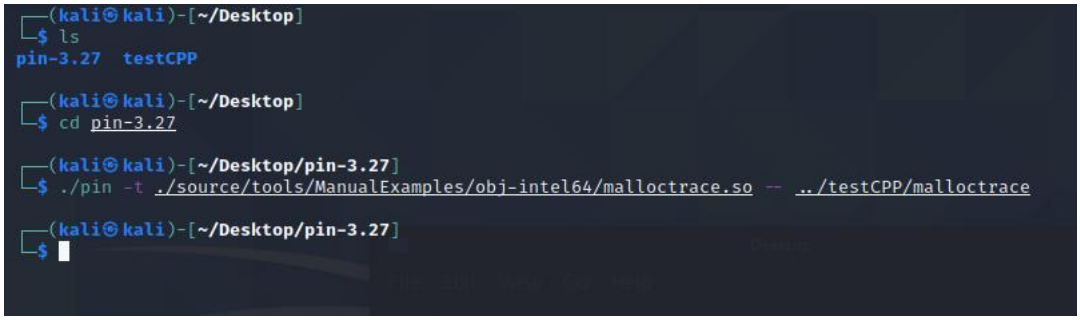
6. 使用语句 gcc -o malloctrace FirstCpp.c 进行编译



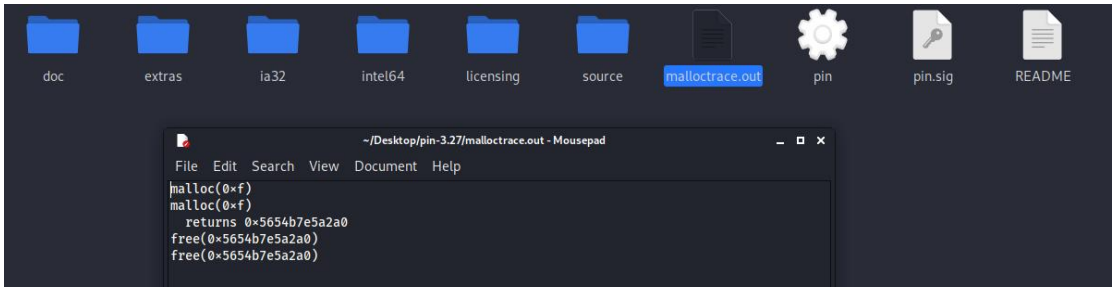


7. 用 pin 进行插桩

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：



8. Pin 文件夹下 malloctrace.out 中可以看到输出结果



9. malloctrace.out 中的内容

```
malloc(0xf)
malloc(0xf)
  returns 0x5654b7e5a2a0
free(0x5654b7e5a2a0)
free(0x5654b7e5a2a0)
```

0xf 即所预留空间大小(最初的内存分配为 15)，return 的值就是设定空间的首地址。

心得体会：

首先对这个实验进行总结，在 Kali 中复现 malloctrace 这个 PinTool 的步骤如下：

1. 在 Kali 中下载并安装 PinTool，可以从官方网站下载得到。
2. 下载 malloctrace 源代码，并编译生成 PinTool。使用 gcc 编译源代码，生成.so 文件。
3. 将生成的.so 文件和要跟踪的二进制文件放在同一个目录下。
4. 使用 PinTool 运行目标二进制文件，并指定要加载的 PinTool。

5. PinTool 开始工作后，会对目标程序进行插桩，当目标程序执行 malloc 和 free 函数时，PinTool 会记录下函数的输入输出信息，如分配内存的大小、返回的内存地址等。
6. 执行完成后，PinTool 会将记录的信息输出到指定文件中，可以通过解析输出文件，分析程序的内存分配情况。

在完成这个实验后，我学到了很多知识。我感觉 PinTool 是一种非常强大的插桩工具，可以帮助我们快速了解程序的行为。在使用 PinTool 进行插桩时，需要了解目标程序的运行环境和代码结构，以便更好地编写插桩代码。同时，通过本次实验，我更深入地理解了 malloc 和 free 函数的内部实现，以及它们对程序的内存使用情况的影响。

这个实验让我更深入地了解了 PinTool 插桩工具的使用方法和原理，以及程序内存分配的相关知识。这对我的编程和调试能力有很大的帮助，也让我更加熟悉了 Linux 操作系统的相关工具和 API。