

# 《软件安全》实验报告

姓名：禹相祐

学号：2312900

班级：计算机科学与技术

## 实验名称：

程序插桩及 Hook 实验

## 实验要求：

复现实验一，基于 WindowsMyPinTool 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. 在 kali 内安装 WindowsMyPinTool 工具：

直接在官网下载最新的 linux 版本的压缩包并拖进 linux 虚拟机内解压即可，如图：



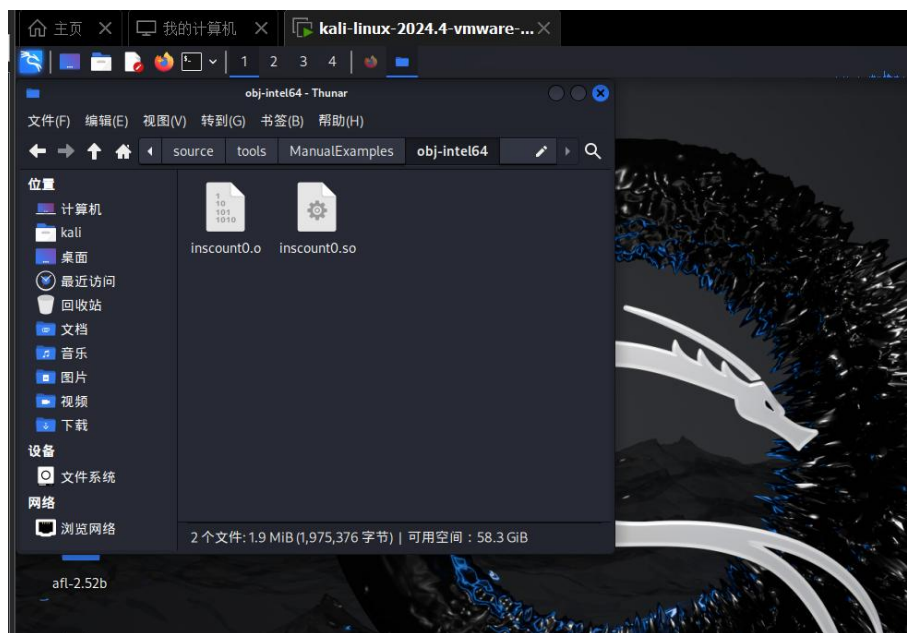
### 2. 编译 inscount0.cpp:

首先我们打开 pin 文件夹内的 source 文件夹，再打开 tools 文件夹中的 ManualExamples, 选择其中的 inscount0.cpp, 然后终端打开该文件并输入指令：

`make inscount0.test TARGET=intel64` 进行编译，如图：

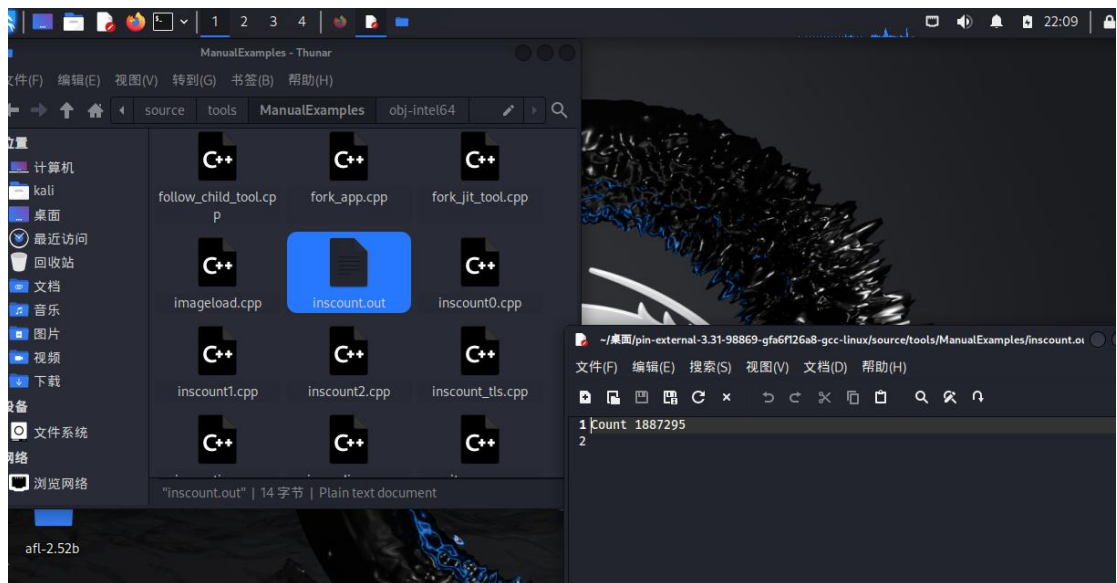
```
(kali@kali)~/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/ManualExamples]
$ make inscount0.test TARGET=intel64
mkdir -p obj-intel64/
g++ -Wall -Werror -Wno-unknown-pragmas -DPIN_CRT=1 -fno-stack-protector -fno-exceptions -funwind-tables -fasynchronous-unwind-tables -fno-rtti -DTARGET_IA32E -DHOST_IA32E -fPIC -DTARGET_LINUX -fabi-version=2 -faligned-new -I../..../source/include/pin -I../..../source/include/pin/gen -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/cxx/include -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/crt/include -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/crt/include/arch-x86_64 -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/crt/include/kernel/uapi -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/crt/include/kernel/uapi/asm-x86 -I../..../extras/components/include -I../..../extras/xed-intel64/include/xed -I../..../source/tools/Utils -I../..../source/tools/InstLib -O3 -fomit-frame-pointer -fno-strict-aliasing -Wno-dangling-pointer -c -o obj-intel64/inscount0.o inscount0.cpp
g++ -shared -Wl,--hash-style=sysv ../..../intel64/runtime/pincrt/crtbegin5.o -Wl,-Bsymbolic -Wl,--version-script=../..../source/include/pin/pintool.ver -fabi-version=2 -o obj-intel64/inscount0.so obj-intel64/inscount0.o -L../..../intel64/runtime/pincrt -L../..../intel64/lib -L../..../intel64/lib-ext -L../..../extras/xed-intel64/lib -lpin -lxd ../..../intel64/runtime/pincrt/crtend5.o -lpdwaf -ldwarf -ldl -dynamic -nostdlib -lc++ -lc++abi -lm -dynamic -lc -dynamic -lunwind-dynamic
/usr/bin/ld: warning: util_host_ia32e.spp.o: missing .note.GNU-stack section implies executable stack
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
make -C ../..../source/tools/Utils dir obj-intel64/cp-pin.exe
make[1]: 进入目录"/home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/Utils"
mkdir -p obj-intel64/
g++ -DTARGET_IA32E -DHOST_IA32E -DFUND_TC_TARGETCPU_FUND_CPU_INTEL64 -DFUND_TC_HOSTCPU_FUND_CPU_INTEL64 -DTARGET_LINUX -DFUND_TC_TARGETOS_FUND_OS_LINUX -DFUND_TC_HOSTOS_FUND_OS_LINUX -I../..../source/tools/Utils -O3 -std=c++11 -o obj-intel64/cp-pin.exe cp-pin.cpp -no-pie
make[1]: 离开目录"/home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/Utils"
../..../pin -t obj-intel64/inscount0.so -- ../..../source/tools/Utils/obj-intel64/cp-pin.exe makefile obj-intel64/inscount0.makefile.copy \
> obj-intel64/inscount0.out 2>01
cp makefile obj-intel64/inscount0.makefile.copy
rm obj-intel64/inscount0.makefile.copy
rm obj-intel64/inscount0.out
```

编译完后我们可以看到：



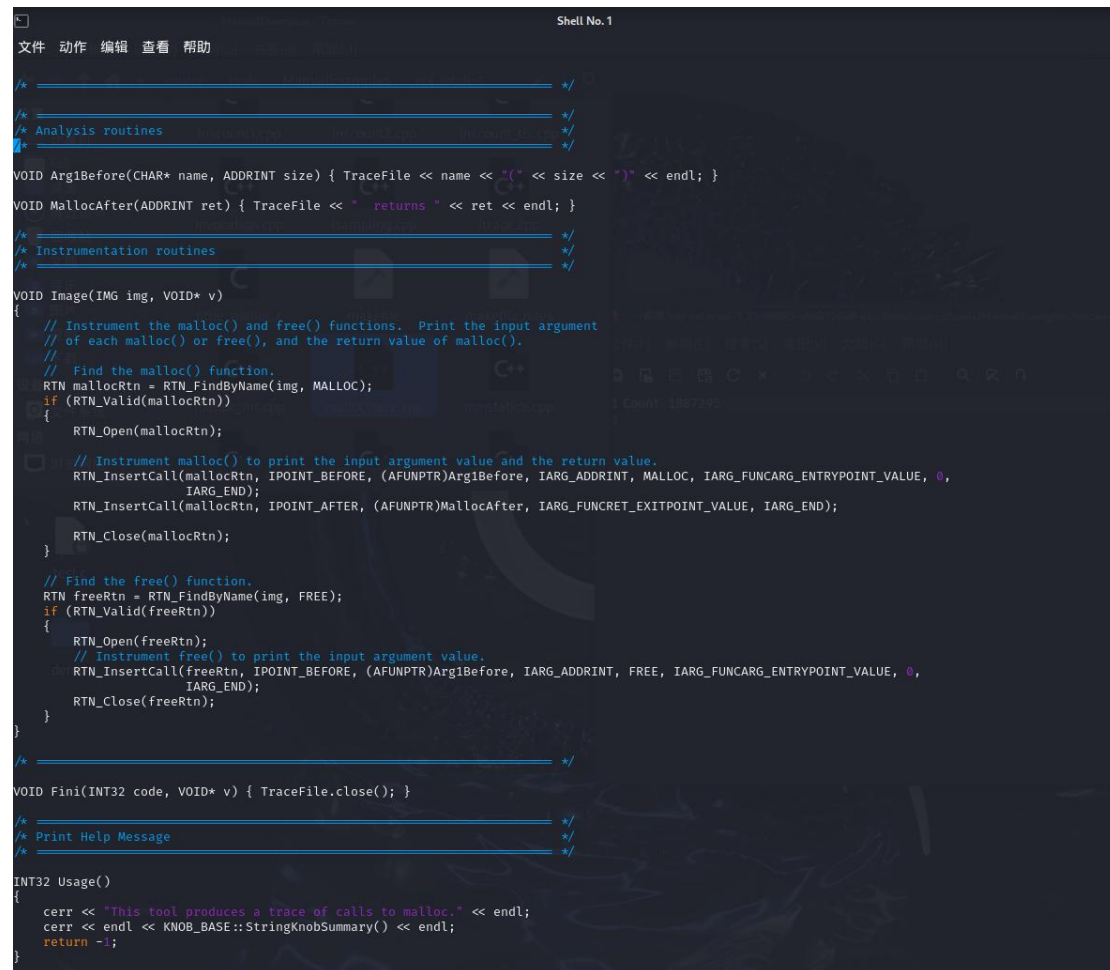
我们还可以打开文件 inscount.out 查看输出的 count 数量，此处为 1887295，

如图：



### 3. 编译 malloctrace.cpp:

依然打开 source/tools/ManualExamples, 打开 malloctrace.cpp 查看源码:



```
Shell No. 1
文件 动作 编辑 查看 帮助

/* ===== */
/* ===== */
/* Analysis routines */
/* ===== */
/* ===== */

VOID Arg1Before(CHAR* name, ADDRINT size) { TraceFile << name << "(" << size << ")" << endl; }

VOID MallocAfter(ADDRINT ret) { TraceFile << " returns " << ret << endl; }

/* ===== */
/* ===== */
/* Instrumentation routines */
/* ===== */
/* ===== */

VOID Image(IMG img, VOID* v)
{
    // Instrument the malloc() and free() functions. Print the input argument
    // of each malloc() or free(), and the return value of malloc().
    // Find the malloc() function.
    RTN mallocRtn = RTN_FindByName(img, MALLOC);
    if (RTN_Valid(mallocRtn))
    {
        RTN_Open(mallocRtn);
        // Instrument malloc() to print the input argument value and the return value.
        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_FUNCARG_EXITPOINT_VALUE, IARG_END);
        RTN_Close(mallocRtn);
    }
    // Find the free() function.
    RTN freeRtn = RTN_FindByName(img, FREE);
    if (RTN_Valid(freeRtn))
    {
        RTN_Open(freeRtn);
        // Instrument free() to print the input argument value.
        RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before, IARG_ADDRINT, FREE, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
            IARG_END);
        RTN_Close(freeRtn);
    }
}

/* ===== */
/* ===== */
VOID Fini(INT32 code, VOID* v) { TraceFile.close(); }

/* ===== */
/* ===== */
/* Print Help Message */
/* ===== */
/* ===== */

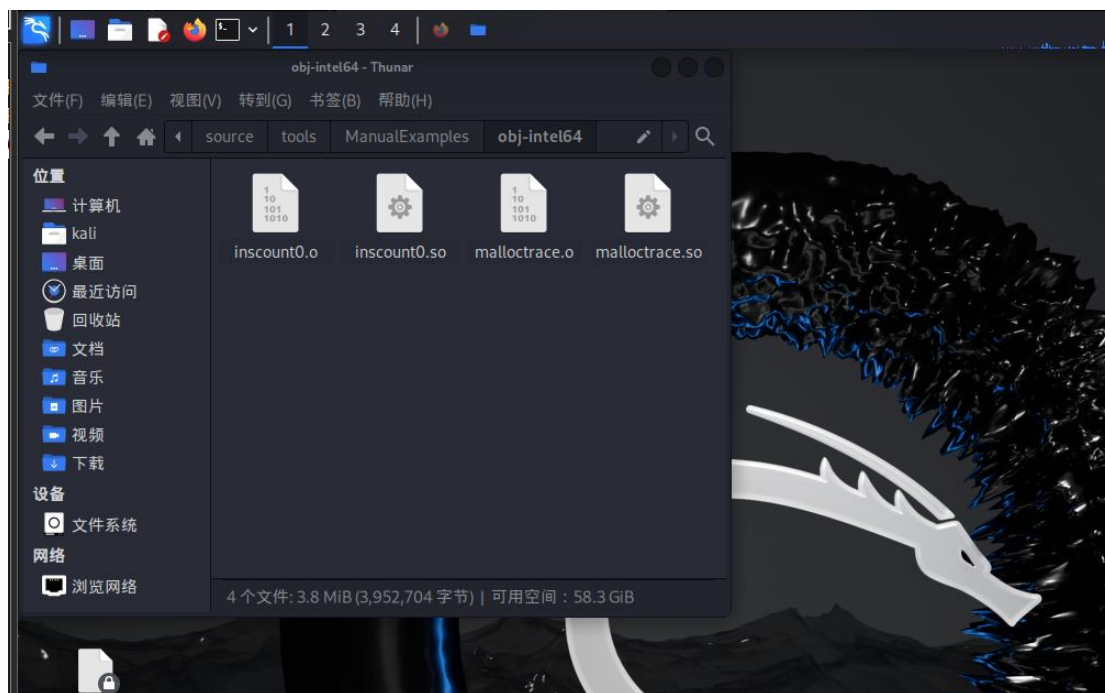
INT32 Usage()
{
    cerr << "This tool produces a trace of calls to malloc." << endl;
    cerr << endl << KNOB_BASE::StringKnobSummary() << endl;
    return -1;
}
```

依然新建一个终端输入命令: **make malloctrace.test TARGET=intel64** 进行编译, 如图:



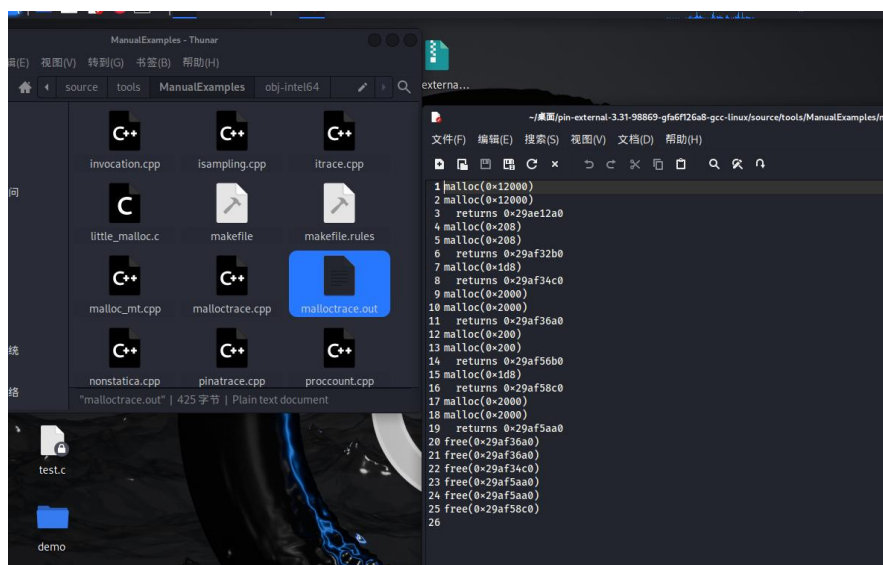
```
kali@kali: ~/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/ManualExamples
文件 动作 编辑 查看 帮助
sh: corrupt history file /home/kali/.zsh_history
--(kali@kali)-[~/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/ManualExamples]
$ make malloctrace.test TARGET=intel64
++ -Wall -Werror -Wno-unknown-pragmas -DPIN_CRT=1 -fno-stack-protector -fno-exceptions -funwind-tables -fasynchronous-unwind-tables -fno-rtti -DTARGET_IA32
E -DHOST_IA32E -fPIC -DTARGET_LINUX -fabi-version=2 -faligned-new -I../..../source/include/pin -I../..../source/include/pin/gen -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/cxx/include -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/crt/include -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/kernel/uapi/include -isystem /home/kali/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/extras/components/include -I../..../extras/xed-intel64/include/xed -I../..../source/tools/Utils -I../..../source/tools/InstLib -O3 -fomit-frame-pointer -fno-strict-aliasing -Wno-dangling-pointer -c -o obj-intel64/malloctrace.o malloctrace.cpp
++ -shared -Wl,--hash-style=sysv ../..../intel64/runtime/pincrt/crtbegin.S.o -Wl,-Bsymbolic -Wl,--version-script=../..../source/include/pin/pintool.ver -abi-version=2 -o obj-intel64/malloctrace.so obj-intel64/malloctrace.o -L../..../intel64/runtime/pincrt -L../..../intel64/lib -L../..../intel64/lib-ex -L../..../extras/xed-intel64/lib -lpin -lxd ../..../intel64/runtime/pincrt/crtend.S.o -lpwindwarf -ldwarf -ldl -dynamic -nostdlib -lc++ -lc++abi -lm -dynamic -lc -dynamic -lunwind-dynamic
usr/bin/ld: warning: util_host_ia32e.spp.o: missing .note.GNU-stack section implies executable stack
usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
../..../pin -t obj-intel64/malloctrace.so - ../..../source/tools/Utils/obj-intel64/cp-pin.exe makefile obj-intel64/malloctrace.makefile.copy \
> obj-intel64/malloctrace.out 2361
mp makefile obj-intel64/malloctrace.makefile.copy
m obj-intel64/malloctrace.makefile.copy
m obj-intel64/malloctrace.out
--(kali@kali)-[~/桌面/pin-external-3.31-98869-gfa6f126a8-gcc-linux/source/tools/ManualExamples]
$
```

看见下图证明已经完成:



#### 4. 进行插桩实验:

我们直接打开生成的 out 文件进行查看, 发现其输出了 malloc 和 free 函数的具体数值, 如图:



#### 5. 对输出结果的分析:

从上述的输出结果可知: 输出由 malloc 函数、free 函数以及 returns 构成, 每次使用 malloc 函数申请空间时, 都会输出申请空间的大小以及起始的地址; 每次调用 free 函数的时候, 都会输出释放空间的初始地址。

#### 6. Pintool 的基本分析:

Malloc 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;
std::ofstream TraceFile;
Knob< string > KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool", "o",
"malloctrace.out", "specify trace file name");
VOID Arg1Before(CHAR* name, ADDRINT size) { TraceFile << name << "(" <<
size <<
")" << endl; }
VOID MallocAfter(ADDRINT ret) { TraceFile << " returns " << ret << endl; }
VOID Image(IMG img, VOID* v)
{
    RTN mallocRtn = RTN_FindByName(img, MALLOC);
    if (RTN_Valid(mallocRtn))
    {
        RTN_Open(mallocRtn);
        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_FUNCARG_EXITPOINT_VALUE, IARG_END);
        RTN_Close(mallocRtn);
    }
    RTN freeRtn = RTN_FindByName(img, FREE);
    if (RTN_Valid(freeRtn))
    {
        RTN_Open(freeRtn);
        RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before,
IARG_ADDRINT,
FREE, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
IARG_END);
        RTN_Close(freeRtn);
    }
}
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[])
{
    // Initialize pin & symbol manager
    PIN_InitSymbols();
    if (PIN_Init(argc, argv))
    {
        return Usage();
    }
    // Write to a file since cout and cerr maybe closed by the application
    TraceFile.open(KnobOutputFile.Value().c_str());
    TraceFile << hex;
    TraceFile.setf(ios::showbase);
    // Register Image to be called to instrument functions.
    IMG_AddInstrumentFunction(Image, 0);
    PIN_AddFiniFunction(Fini, 0);
    // Never returns
    PIN_StartProgram();
    return 0;
}

```

由代码可知：其中包含两个分析函数:Arg1Before 和 MallocAfter。Arg1Before 在调用函数 malloc 和 free 前执行，拿来记录所调用的函数名称以及其对应参数；MallocAfter 则是在之后执行，拿来记录返回值。而 Image 函数则是拿来找到并插桩 malloc 和 free 函数对应的代码。而 RTN\_InsertCall 函数则会在找到 malloc 和 free 函数后前后插入分析函数 Arg1Before 和 MallocAfter。Fini 函数则负责关闭输出文件，Usage 函数负责在程序报错的时候输出有误的命令行参数。

## 心得体会：

通过此次试验，我对 Pin 插桩工具有了更深的理解，简单总结如下：

1. 首先是编写需要进行插桩的程序，然后通过编译生成可执行文件；
2. 选择需要使用的插桩工具，并同样对其进行编译以生成动态链接库；
3. 将 pintool 插入要执行的程序，就成功实现了插桩，打开 out 文件也能查看插桩后的对应输出。