

《漏洞利用及渗透测试基础》实验报告

姓名：申宗尚 学号：2213924 班级：信息安全

实验名称：

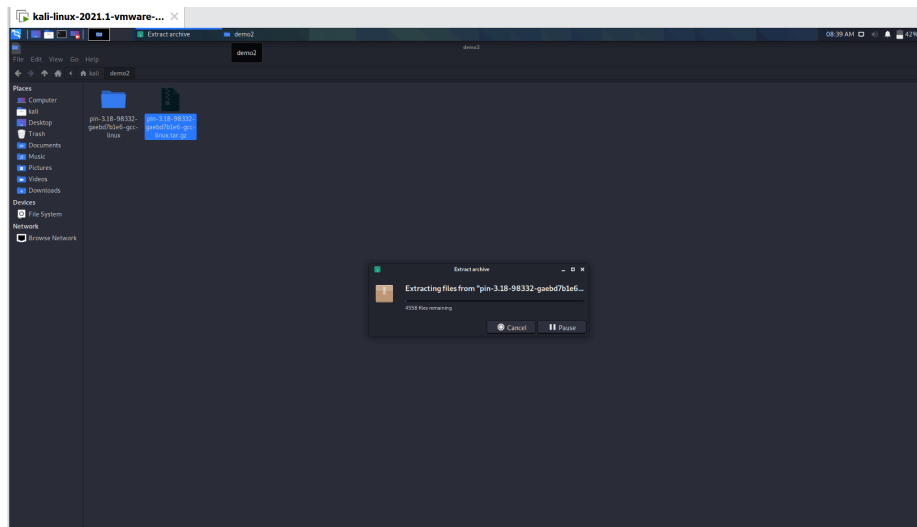
程序插桩及 Hook 实验

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：

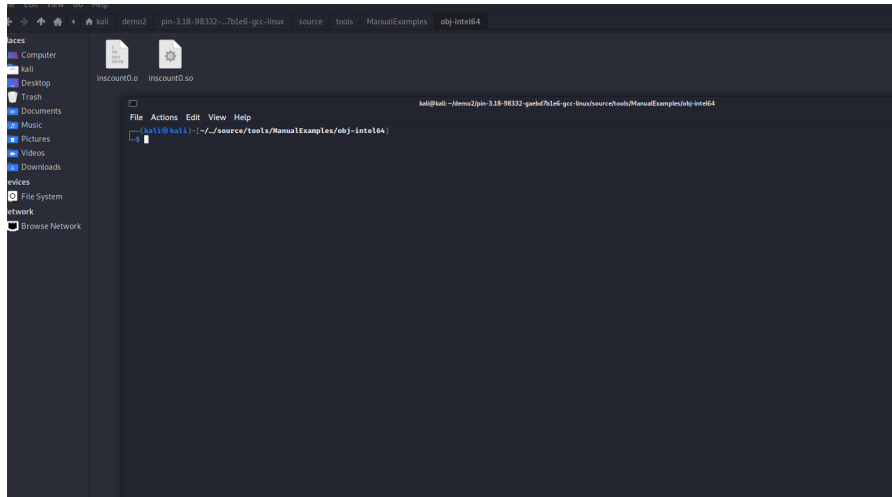
- 首先，从 intel 官网下载 pin-3.18 for linux，将下载好的 pin-3.18 压缩包拖动并移动到 kali 系统中的新建本实验文件夹 kali/demo1 文件夹中，解压。



- 进入该文件夹下的目录 source/tools/ManualExamples，打开终端命令，输入命令：
make inscount0.test TARGET=intel64 来对 inscount0.cpp 进行编译。

```
File Actions Edit View Help
kali@kali:~/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples$
$ make inscount0.test TARGET=intel64
mkdir -p obj-intel64/
g++ -Wall -Werror -Wno-unknown-pragmas -D_PIN_1 -DPIN_CRT-1 -fno-stack-protector -fno-exceptions -fno-unwind-tables -fno-asynchronous-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/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/stlport/include -isystem /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/libstdc++/include -isystem /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/crt/include -isystem /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/crt/include/arch-x86_64 -isystem /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/crt/include/kernel/uapi -isystem /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/extras/crt/include/kernel/uapi/arm-a64 -I../..//extras/components/include -I../..//extras/xed-intel64/include/xed -I../..//source/tools/Utils -I../..//source/tools/instlib -D3 -fomit-frame-pointer -fno-strict-aliasing -c -o obj-intel64/inscount0.o inscount0.cpp
g++ -shared -Wl,-hash-style=sysv ../..//intel64/runtime/pincrt/crtbegin.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/crtend.o -lpinldwarf -ldl -dynamic -nostdlib -lstlport-dynamic -lm-dynamic -lc-dynamic -lunwind-dynamic
make -C ../..//source/tools/Utils dir obj-intel64/cp-pin.exe
make[1]: Entering directory /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-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 -o obj-intel64/cp-pin.exe cp-pin.cpp -no-pie
make[1]: Leaving directory /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/Utils'
../..//pin -c obj-intel64/inscount0.so - ../..//source/tools/Utils/obj-intel64/cp-pin.exe makefile obj-intel64/inscount0.makefile.copy \
> obj-intel64/inscount0.out 2>1
cmp makefile obj-intel64/inscount0.makefile.copy
rm obj-intel64/inscount0.makefile.copy
rm obj-intel64/inscount0.out
kali@kali:~/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples$
```

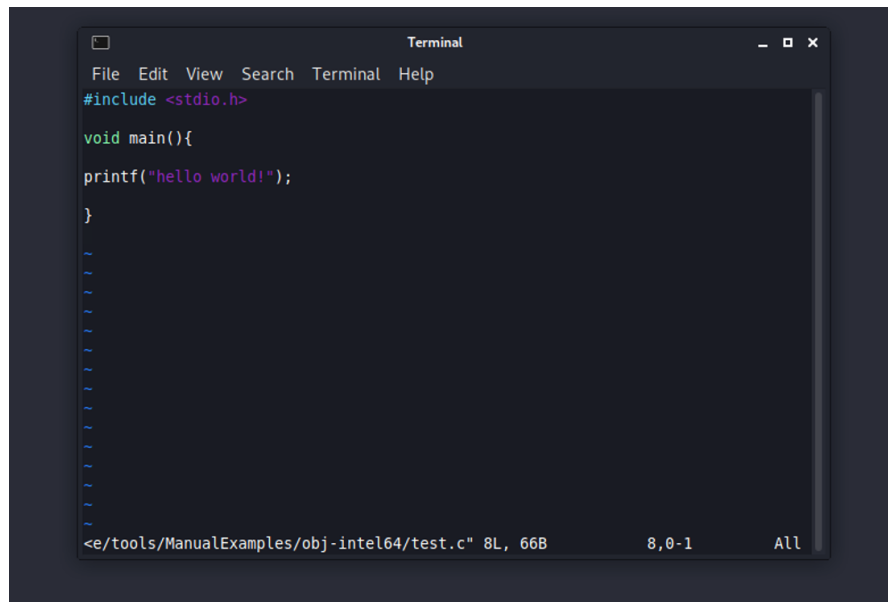
编译成功，现在进入 obj-intel64 目录下查看动态链接库文件 inscount0.so，并直接进入到其终端目录下（方便后续实验输入命令）。



3. 然后, 尝试编写一个简单的控制台输出程序, 并使用此动态链接库文件进行插桩实验。
代码如下:

```
#include <stdio.h>

void main() {Printf( "hello world!" );}
```



编译指令: `gcc -o testc test.c`

pin 工具插桩执行命令: `/home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t inscount0.so -- /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/ManualExamples/obj-intel64/testc`



4. 进行程序插桩完成后便可查看输出文件得知所用到的指令数, 插桩实验完成。


```
File Actions Edit View Help
(kali@kali)-[~/source/tools/ManualExamples/obj-intel64]
$ gcc TEST Test.cpp
(kali@kali)-[~/source/tools/ManualExamples/obj-intel64]
$
```

8. 使用 pin 工具进行插桩操作:

命令行代码: /home/kali/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t inscount0.so -- TEST4.

```
File Edit Search View Document H kali@kali: ~/demo2/pin-3.18-98332-gaebd7b1e6-gcc-linux/pin-intel64
returns 0x7f5da8ad4ec0
malloc(0x20)
returns 0x7f5da8ad4ee0
free(0)
malloc(0x4aa)
returns 0x7f5da8ad4f00
malloc(0x20)
returns 0x7f5da8ad53b0
malloc(0x58)
returns 0x7f5da8ad53d0
malloc(0x28)
returns 0x7f5da8ad5430
malloc(0x28)
returns 0x7f5da8ad5460
malloc(0x38)
returns 0x7f5da8ad5490
malloc(0x68)
returns 0x7f5da8ad54d0
malloc(0xd8)
returns 0x7f5da8ad5540
malloc(0x48)
returns 0x7f5da8ad5620
malloc(0x5b8)
returns 0x7f5da8ad5670
malloc(0x180)
returns 0x7f5da8ad5c30
malloc(0x348)
returns 0x7f5da8585000
malloc(0x1b0)
returns 0x7f5da8585350
malloc(0x90)
returns 0x7f5da8585500
malloc(0x420)
returns 0x7f5da8585590
malloc(0x1088)
returns 0x7f5da85859b0
malloc(0x120)
returns 0x7f5da8586a40
malloc(0x3e0)
returns 0x7f5da8586b60
malloc(0x7a0)
returns 0x7f5da85804000
free(0x7f5da8586b60)
malloc(0xfe0)
returns 0x7f5da85047a0
free(0x7f5da8504000)
malloc(0x11c00)
returns 0x204d2a0
malloc(0x208)
returns 0x205eeb0
malloc(0x1d8)
returns 0x205f0c0
malloc(0x2000)
returns 0x205f2a0
malloc(0x200)
returns 0x20612b0
malloc(0x1d8)
returns 0x20614c0
malloc(0x2000)
returns 0x20616a0
free(0x205f2a0)
free(0x205f0c0)
free(0x20616a0)
free(0x20614c0)
```

输出文件中显示堆中空间地址的申请与释放情况,比如观察最后几行,可以看到,每次使用 malloc 申请堆内存空间的时候,都会输出申请的空间的大小,并且会有相应的地址记录。当 free 内存空间时,可以看到地址与申请时候的地址一一对应。

核心实验内容:

1. 初始化:调用 PIN_Init 函数进行初始化操作,解析命令行参数,并为 Pin 工具做好必要的准备工作。

2. 注册插桩函数：根据需要的插桩粒度，使用合适的 API 注册插桩函数。例如使用 `INS_AddInstrumentFunction` 注册指令级插桩函数：在程序执行每条指令前，会调用注册的插桩函数，使用 `IMG_AddInstrumentFunction` 注册镜像级插桩函数，在加载每个可执行文件和共享库时调用插桩函数。
3. 注册退出回调函数：使用 `PIN_AddFiniFunction` 注册程序退出时的回调函数。当应用程序退出时，该回调函数会被调用，用于执行清理操作或输出统计信息。
4. 启动程序：使用 `PIN_StartProgram` 启动被插桩的目标程序。这个函数控制目标程序的执行，并开始插桩操作。

＜相关 API 及其介绍＞

1. `PIN_InitSymbols`：初始化符号处理。使用该函数可以处理符号信息，如函数名和变量名，在插桩过程中提供更丰富的上下文信息。
2. `PIN_Init`：初始化 Pin 工具，包括解析命令行参数和初始化内部数据结构，确保 Pin 环境准备就绪。
3. `IMG_AddInstrumentFunction`：
注册镜像级插桩函数。该函数在每次加载新的可执行文件或共享库时被调用，可以在加载时进行相应的插桩操作。
4. `PIN_AddFiniFunction`：注册程序退出时的回调函数。该回调函数在应用程序退出时被调用，用于进行资源释放或输出插桩结果。
5. `PIN_StartProgram`：启动被插桩的目标程序，并开始插桩操作。调用此函数后，Pin 工具将控制目标程序的执行流程。
6. `RTN_FindByName`：查找特定函数的运行时标识符（RTN）。该函数可以根据函数名查找函数，用于在特定函数执行前后插入自定义操作。
7. `RTN_InsertCall`：在找到的函数执行前后插入回调函数。该函数可以在函数入口或出口处插入自定义操作，例如统计函数调用次数或记录参数值。

心得体会：

通过本次实验，我对 Pin 插桩工具的使用有了深刻的理解和实践体验。在实验中，我使用了 Intel 的 Pin 工具来进行程序插桩。Pin 工具的强大之处在于它能够对程序进行动态二进制插桩，允许我们在不改变源代码的情况下插入自定义代码，从而实现对程序运行时行为的监控和分析。这对于漏洞利用和渗透测试等领域具有重要意义。

同时，我熟悉了 Pin 工具的核心 API，包括 `PIN_Init`、`IMG_AddInstrumentFunction`、`PIN_AddFiniFunction` 和 `PIN_StartProgram` 等。这些 API 的使用使我能够在程序的不同阶段插入自定义代码，进行各种操作如统计指令数、记录函数调用等。

而且，我成功复现了 `malloctrace` 这个 PinTool，跟踪了 `malloc` 和 `free` 函数的调用。通过插桩，我们可以实时监控内存的分配和释放情况，捕获到每次 `malloc` 申请的空间大小及相应的地址，并在 `free` 操作时进行对应的地址释放。这让我深刻体会到了插桩工具在内存管理监控中的重要作用。