《漏洞利用及渗透测试基础》实验报告

姓名：蒋薇　　学号：2110957　　班级：张健老师班

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，关于 malloc 和 free 函数的输入输出信息。

实验过程：

1. 消息 Hook 注入代码实现类似插桩作用

```
//Hook函数（键盘消息处理函数）
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)
{
    char szPath[MAX_PATH] = {0,};
    char *p = NULL;

    if( nCode >= 0 )
    {
        // lParam 第31位bit: 0 -> key press, 1 -> key release
        if( !(lParam & 0x80000000) )        //当按键被释放时
        {
            GetModuleFileNameA(NULL, szPath, MAX_PATH);
            p = strrchr(szPath, '\\');

          //比较当前进程名称，若为notepad.exe，则消息不会传递给应用程序及下一个“钩子”
            if( !_stricmp(p + 1, DEF_PROCESS_NAME) )
                return 1;       //丢弃该Keyboard消息
        }
    }

//若不为notepad.exe，调用CallNextHookEx()函数将消息传递给下一个“钩子”或应用程序
    return CallNextHookEx(g_hHook, nCode, wParam, lParam);
}
```

```
{
    HANDLE hProcess, hThread;
    LPVOID pRemoteBuf;
 DWORD dwBufSize = (DWORD)(_tcslen(szDllName) + 1) * sizeof(TCHAR);
    LPTHREAD_START_ROUTINE pThreadProc;

    // #1.使用dwPID获取目标进程(notepad.exe)句柄
    if ( !(hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPID)) )
 {
     DWORD dwErr = GetLastError();
        return FALSE;
 }
    if(hProcess!=NULL)
    // #2.在目标进程(notepad.exe)中分配szDllName大小的内存
    pRemoteBuf = VirtualAllocEx(hProcess, NULL, dwBufSize, MEM_COMMIT, PAGE_READWRITE);
    if(pRemoteBuf!=NULL)
    // #3.将szDll路径写入分配的内存
    WriteProcessMemory(hProcess, pRemoteBuf, (LPVOID)szDllName, dwBufSize, NULL);
    // #4.获取LoadLibraryA() API的地址
    pThreadProc = (LPTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(LPCTSTR("kerne13
    if(pThreadProc!=NULL)
    // #5.在exe进程中运行线程
    hThread = CreateRemoteThread(hProcess,              //hProcess
                                 NULL,          //lpThreadAttributes
                                 0,             //dwStackSize
                                 pThreadProc,       //lpStartAddress
                                 pRemoteBuf,        //lpParameter
```

KeyHook.cpp

KeyBoardProc()

return 1;//实现拦截记事本

Hook 函数

//导出函数

HookStart()

SetWindowsHookEx(WH_KEYBOARD,KeyboardProc,g_hInstance,0);

HookStop()

UnhookWindowsHookEx(g_hHook);

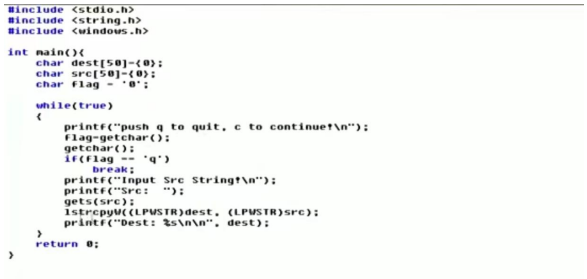
```
//自定义得到动态链接库特定函数的地址

HookStart= （PFN_HOOKSTART)GetProcAddress(hDll,DEF_HOOKSTART);
HookStop = (PFN_HOOKSTOP)GetProcAddress(hDll,DEF_HOOKSTOP);

HookStart();
...
FreeLibrary(hDll);
```

//打开记事本输入被拦截，输入 q 推出后，可正常输入

2.APIHook



```
#include <stdio.h>
#include <string.h>
#include <windows.h>

int main(){
    char dest[50]={0};
    char src[50]={0};
    char flag = '0';

    while(true)
    {
        printf("push q to quit, c to continuet\n");
        flag=getchar();
        getchar();
        if(flag == 'q')
            break;
        printf("Input Src String!\n");
        printf("Src:  ");
        gets(src);
        lstrcpyW((LPWSTR)dest, (LPWSTR)src);
        printf("Dest: %s\n\n", dest);
    }
    return 0;
}
```

strcpyW((LPWSTR)dest,(LPWSTR)src);//敏感函数，捕获输入输出
hook_iat();
//PE 文件 IAT table,更改导入地址表，
//遍历寻找 iat 表地址，更改内存属性 E/R/W,修改 IAT 值，恢复内存属性

//动态链接库，文件，注入 injection

3.Pin 利用 PinTool 完成插桩功能的包装和实现

```
    iscount0.cpp
    static Uinit64 icount = 0;
    void docount(){}
    voi instruction(){}
    int main(){
    //初始化 Pin
    //打开输出文件
    //调用指令级插桩，注册回调函数，回调函数调用自定义都 count()
    }
    //编译运行，动态链接库
    make inscount0.test target -intel64

    FirstCpp.c
    #include<stdio.h>
    void main(){
    printf(“hello world!”);
    }
    //编译
    gcc -o First FirstCPP.c
    //插桩(Linux)
./pin-t./source/tools/ManualExamples/obj-intel64/inscount0.so- ../testCPP/First
    //输出
    hello world!

    //inscount.out(统计指令数量)
```

Count 192994

//自定义
void Instruction(INS ins,VOID * v){
if(INS_OPcode(ins)==XED_ICLASS_MOV&&INS_ISMemoryRead(ins)&&INS_OperandIsReg(ins,0)&&INS_OperandIsMemory(ins,1))
{
icount++;
}
}
//重新编译再插桩
//打开 inscount.out
Count 1806

关于 malloc 和 free 函数的输入输出信息

（此处根据实际操作过程，留下具体操作步骤、附加一些自己的理解，即可）

心得体会：

消息 Hook 的是 Windows 的消息，APIhook 为一些函数