

**恶意代码课程实验报告**

**实验五：IDA Python**

****

学 院 网络安全学院

专 业 信息安全

学 号 2110688

姓 名 史文天

班 级 1063

1. **实验目的**

**完成教材Lab5的实验内容，编写Lab5样本的Yara引擎规则，并测试规则的执行效率。尝试编写IDA Python脚本辅助分析。**

1. **实验原理**

**配置XP虚拟机环境，使用IDA完成对给定病毒样本的分析。**

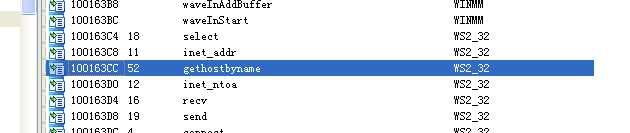
1. **实验过程**
2. **DllMain的地址是什么?**

**使用IDA打开文件，定位DLLMain，可以看到其地址为.text 0x1000D02E。**



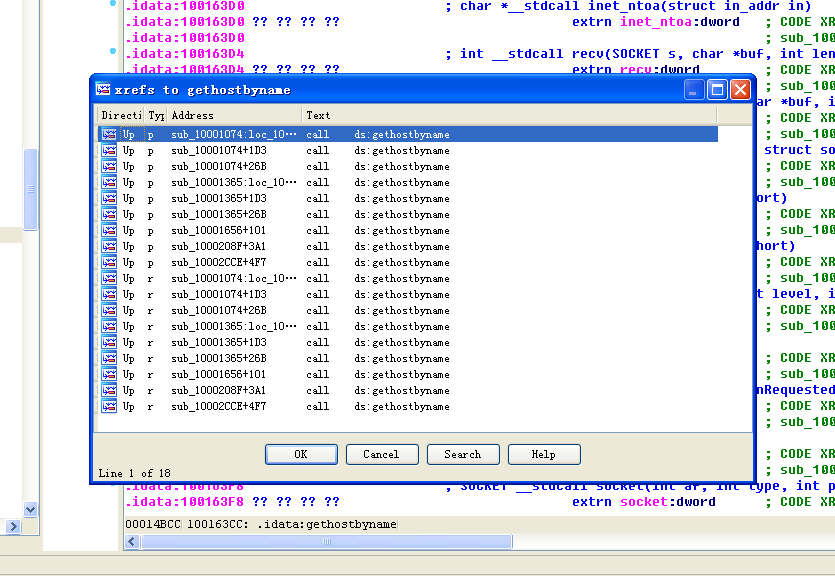
1. **使用Imports窗口并浏览到gethostbyname, 导入函数定位到什么地址？**

**打开Imports窗口，查找搜索定位该函数，可以看到其地址为.idata 0x100163CC。**



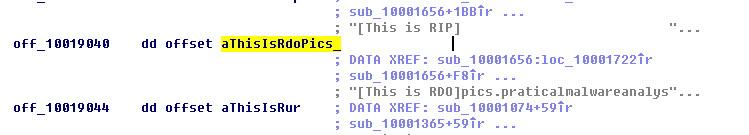
1. **有多少函数调用了gethostbyname?**

**使用`CTRL + X`查看其交叉引用，可以看到有18行记录。仔细查看该记录，可以看到IDA将p（被调用的引用）与r（被读取的引用）都予以计算，故而是9次引用。而地址栏中的`+`与`.`都是表示地址偏移，故属于同一函数，故而共计有5个函数调用了该函数。**

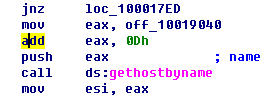


1. **将精力集中在位于0x10001757处的对gethostbyname的调用，你能找出哪个DNS 请求将被触发吗?**

**跳转到该地址，可以看到，该函数使用了一个参数。双击查看，可以发现，该地址存储了字符串[This is PDO\\]pics.practicalmalwareanalysis.com。**

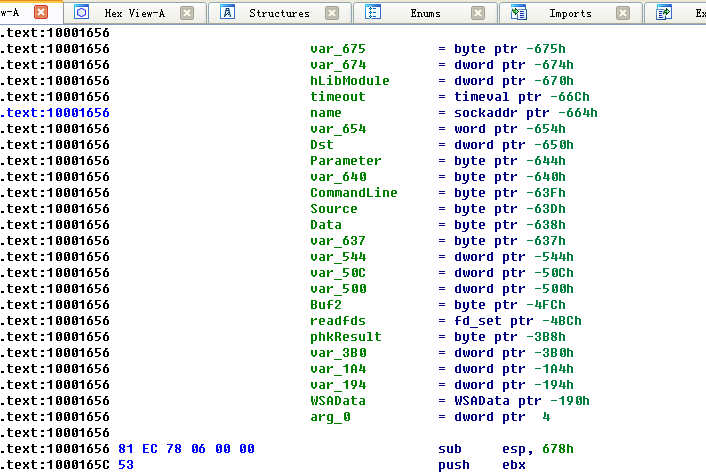


**其放入eax寄存器后，又增加了0Dh，经过计算可以发现，增加之后，该地址正好指向前面字符串中的`p`，即eax此时存储的为该url。**



1. **IDA Pro识别了在0x10001656处的子过程中的多少个局部变量?**

**跳转到该地址后，可以看到许多绿色高亮的代码，这些即为局部变量，经过计算以及删除`arg\_0`（参数），可以知道，总共有23个。**

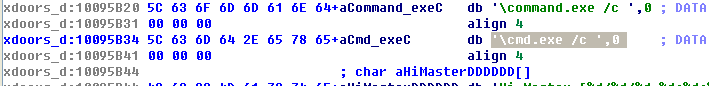


1. **IDA Pro 识别了在0x10001656处的子过程中的多少个参数?**

**由上面图中分析可知，其识别了1个参数arg\_0。**

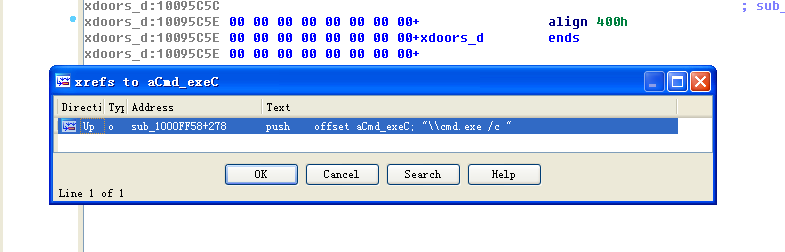
1. **使用Strings窗口，在反汇编中定位字符串\cmd.exe /c。它位于哪?**

**在Strings窗口中搜索然后双击，即可看到其相关情况。可以看到，其位于xdoors\_d 0x10095B34处。**

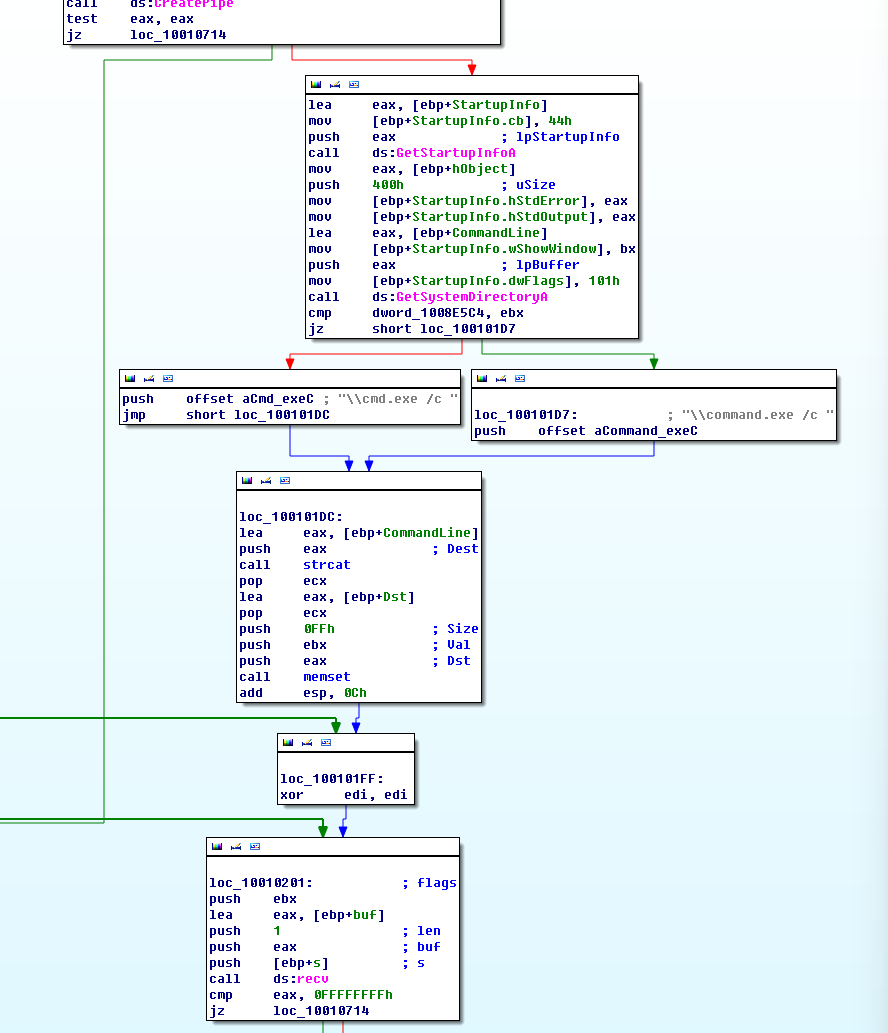


1. **在引用\cmd.exe /c 的代码所在区域发生了什么?**

**查看交叉引用，可以看到该字符串被压栈。**

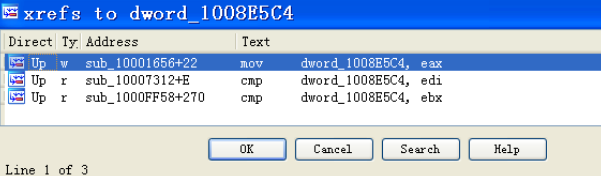


**点击`OK`跳转到其被引用位置并切换视图，可以看到后面会有诸如recv、quit、exit、cd等指令，以及”This Remote Shell Session“字符串，推测是一个远程会话函数。**

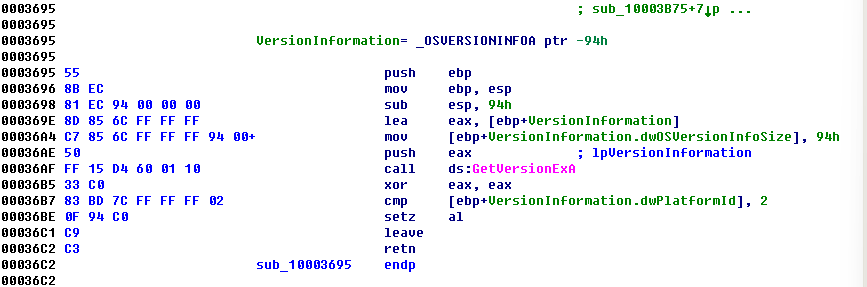


1. **在同样的区域，在0x100101C8处，看起来好像dword\_1008E5C4是一个全局变量，它帮助决定走哪条路径。那恶意代码是如何设置dword\_1008E5C4的呢?（提示：使用dword\_1008E5C4的交叉引用）**

**跳转到该地址，可以看到`cmp dword\_1008E5C4, ebx`的指令，即将ebx与该全局变量进行比较。查看其交叉引用，可以看到，只有`mov`改变了其值。**

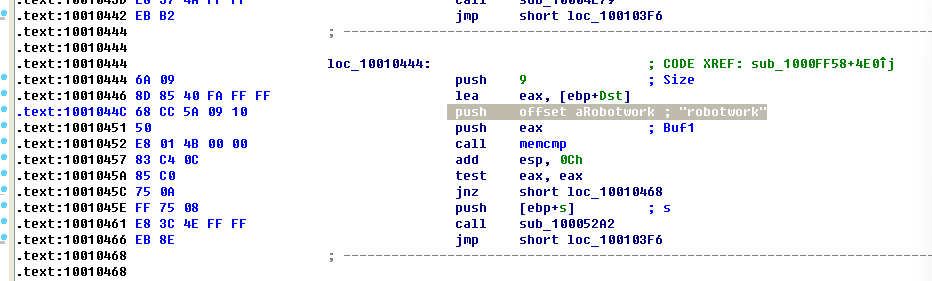


**跳转到该位置可以看到，eax是上面调用函数的返回值。双击查看该函数，可以看到其调用了GetVersionEx，获取当前操作系统的信息。xor eax, eax语句则将eax置0，并且cmp [ebp+VersionInformation.dwPlatformId],2 语句将平台类型同2相比。这里只是简单的判断当前操作系统是否为Windows 2000或更高版本，根据微软的文档，我们得知通常情况下dwPlatformId 的值为2。**

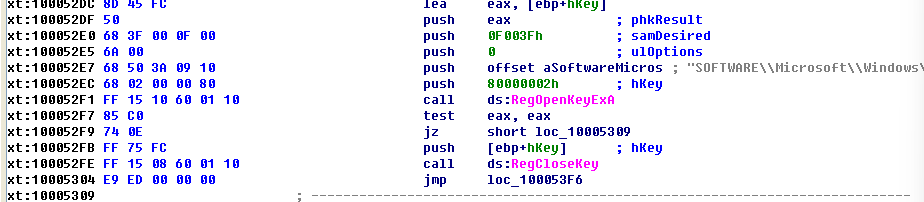


1. **在位于0x1000FF58处的子过程中的几百行指令中，有一系列使用memcmp 来比较字符串的指令。如果对rotbotwork的字符串比较是成功的（memcmp返回0），会发生什么?**

**定位到该位置，可以看到与robotwork比较的memcmp，如果eax和robotwork相同，则memcmp的结果为0，即eax为0。test的作用和and类似，只是不修改寄存器操作数，只修改标志寄存器，因此test eax,eax语句的含义是，若eax为0，那么test的结果为ZF=1。而jnz检验的标志位就是ZF , 若ZF=1，则不会跳转，继续向下执行，直到call sub\_100052A2。**

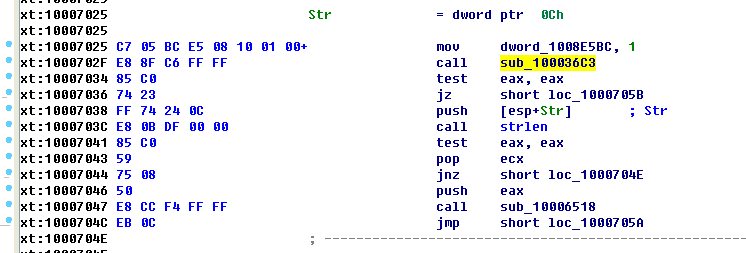


**双击查看该代码，可以看到其参数为socket类型，即上面push的[ebp+s]。继续阅读可以发现，后面aSoftWareMicros处的值为”SOFTWARE\Microsoft\Windows\CurrentVersion“，然后调用RegOpenKeyEx函数读取该注册表值。**



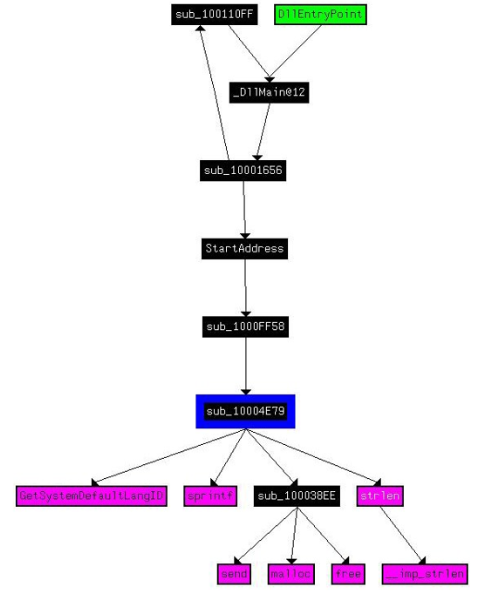
1. **PSLIST导出函数做了什么?**

**在Exports窗口找到PSLIST，双击查看其情况。可以看到首先调用sub\_100036C，这个函数检查操作系统的版本是Windows Vista/7 或是Windows XP/2003/2000。这两条代码都是用CreateToolhelp32Snapshot函数，从相关字符串和API调用来看，用于获得一个进程列表，这两条代码都通过send将进程列表通过socket发送。**



1. **使用图模式来绘制出对sub\_10004E79的交叉引用图。当进入这个函数时，哪个API函数可能被调用? 仅仅基于这些API函数，你会如何重命名这个函数?**

**主要调用的API为GetSystemDefaultLangID和send。因此推测可能是通过socket发送语言标志，因而可以直接在函数名处右键Rename 重命名为send\_languageID。**



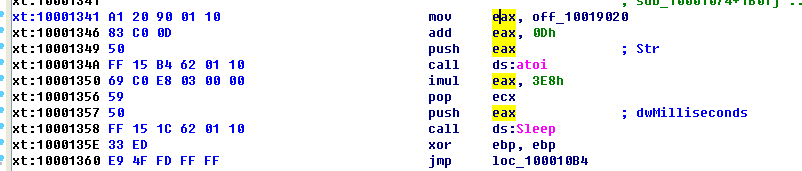
1. **DllMain直接调用了多少个Windows API? 多少个在深度为2时被调用?**

**使用Xref可以知道DllMain调用了非常多函数。第二层调用了如\_\_ imp \_ strlen、memcpy、gethostbyname等许多函数。**

1. **在0x10001358处，有一个对Sleep（一个使用包含要睡眠的毫秒数的参数的API函数）的调用。顺着代码向后看，如果这段代码执行，这个程序会睡眠多久?**

**调用的sleep的参数为上一行push的eax，而eax 的值又来自imul eax,3E8h的运算结果。再往上看，可以看到，eax是由atoi函数对Str运算得到的，也即字符串转整数。**

**继续回溯，可以看到，Str由off\_10019020+0Dh位置的字符串得到，最终转换成数字30。所以睡眠的时间应为30\*1000 = 30000（毫秒），即30 秒。**



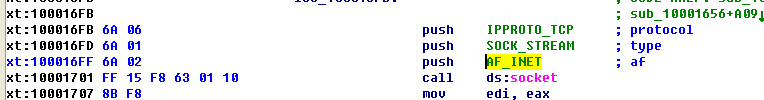
1. **在0x10001701处是一个对socket的调用。它的3个参数是什么?**

**跳转到该地址，可以看到三个参数名：af、type、protocol。**



1. **使用MSDN页面的socket和IDA Pro中的命名符号常量，你能使参数更加有意义吗? 在你应用了修改以后，参数是什么?**

**查阅socket的官方文档，可以确认，输入的参数含义为建立基于IPv4的TCP连接的socket，通常在HTTP中使用。在数字上右键，Use standard symbolic constant，分别替换成如图所示的实际的常量名。**

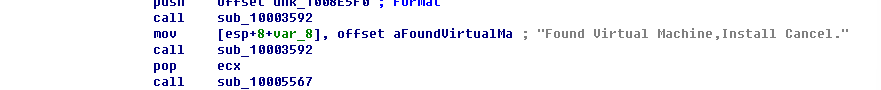


1. **搜索in指令（opcode 0xED）的使用。这个指令和一个魔术字符串VMXh用来进行VMware检测。这在这个恶意代码中被使用了吗? 使用对执行in指令函数的交叉引用，能发现进一步检测VMware的证据吗?**

**搜索in指令，可以发现，该指令只在.text 0x100061DB处的in eax, dx处进行使用。**

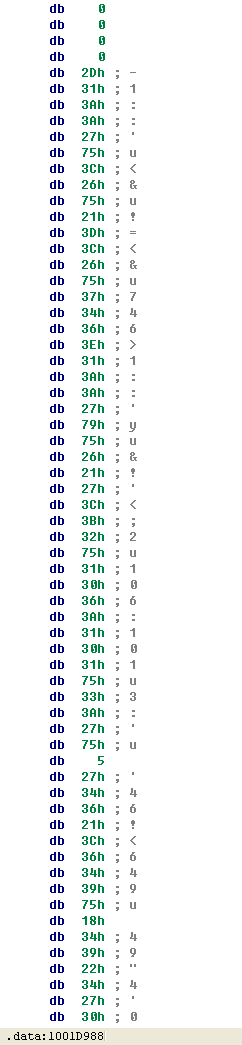


**双击查看其相关内容，可以看到，eax中存储了字符串”VMXh“，即反虚拟机技术。继续查找，可以看到其入口，查看交叉引用，可以看到字符串”Found Virtual Machine,Install Cancel.“，确认其使用反虚拟机技术。**



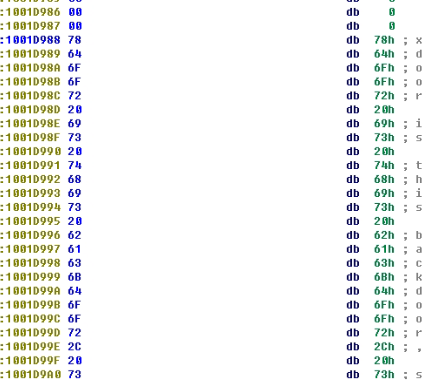
1. **将你的光标跳转到0x1001D988处，你发现了什么?**

**看到一串乱码。**



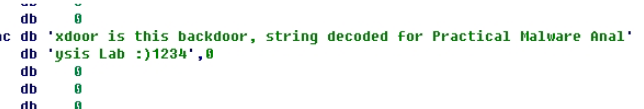
1. **如果你安装了IDA Python 插件（ 包括IDA Pro 的商业版本的插件） ， 运行Lab05-01.py，一个本书中随恶意代码提供的IDA Pro Python 脚本，（确定光标是在0x1001D988 处）在你运行这个脚本后发生了什么?**

**可以看到文件被解密。**



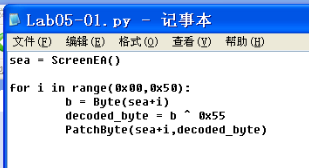
1. **将光标放在同一位置，你如何将这个数据转成一个单一的ASCII字符串?**

**按下键盘上的`A`键，即可转换成ASCII字符串，得到”xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234“。**



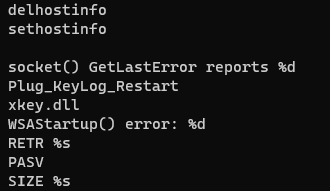
1. **使用一个文本编辑器打开这个脚本。它是如何工作的?**

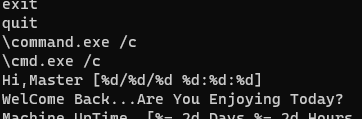
**打开后看到该.py文件是对长度为0x50字节的数据，用0x55分别与其进行异或，然后用PatchByte函数在IDA Pro中修改这些字节。**



1. **Yara规则**

**使用Strings工具分析该dll文件，有如下字符串。**

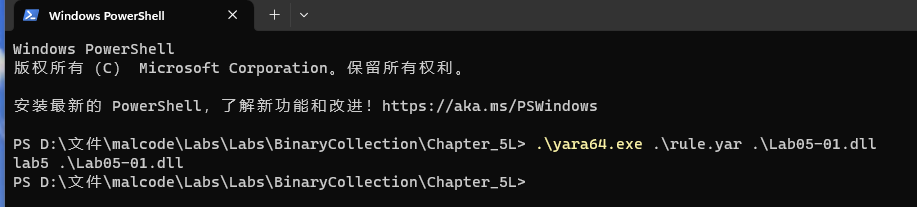




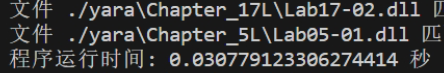
**根据以上分析结果，得到Yara规则如下。**

1. rule lab5
2. {
3. strings:
4. $string1 = "socket() GetLastError reports %d"
5. $string2 = "WSAStartup() error: %d"
6. $string3 = "xkey.dll"
7. $string4 = "\command.exe /c"
8. $string5 = "\cmd.exe /c"
9. condition:
10. filesize < 150KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
11. 0x00004550 and all of them
12. }

**运行结果如下，可看到成功匹配该文件。**



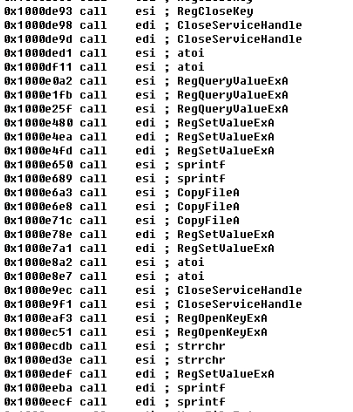
**在存放所有样例的文件夹下测试其检测效率，结果如下。**



1. **IDA python脚本编写**

import idautils  
for func in idautils.Functions():  
 flags = idc.GetFunctionFlags(func)  
 if flags & FUNC\_LIB or flags & FUNC\_THUNK:  
 continue  
 dism\_addr = list(idautils.FuncItems(func))  
 for line in dism\_addr:  
 m = idc.GetMnem(line)  
 if m == 'call' or m == 'jmp':  
 op = idc.GetOpType(line,0)  
 if op == o\_reg:  
 print '0x%x %s' %

**遍历所有函数，排除库函数或简单跳转函数，当反汇编的助记符为call或者jmp且操作数为寄存器类型时，输出该行反汇编指令。**



1. **实验结论及心得体会**
2. **掌握了使用IDA分析二进制文件的能力。**
3. **了解了IDA python等插件的使用方法。**