

**恶意代码课程实验报告**

**实验九：Windows恶意代码**

****

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**

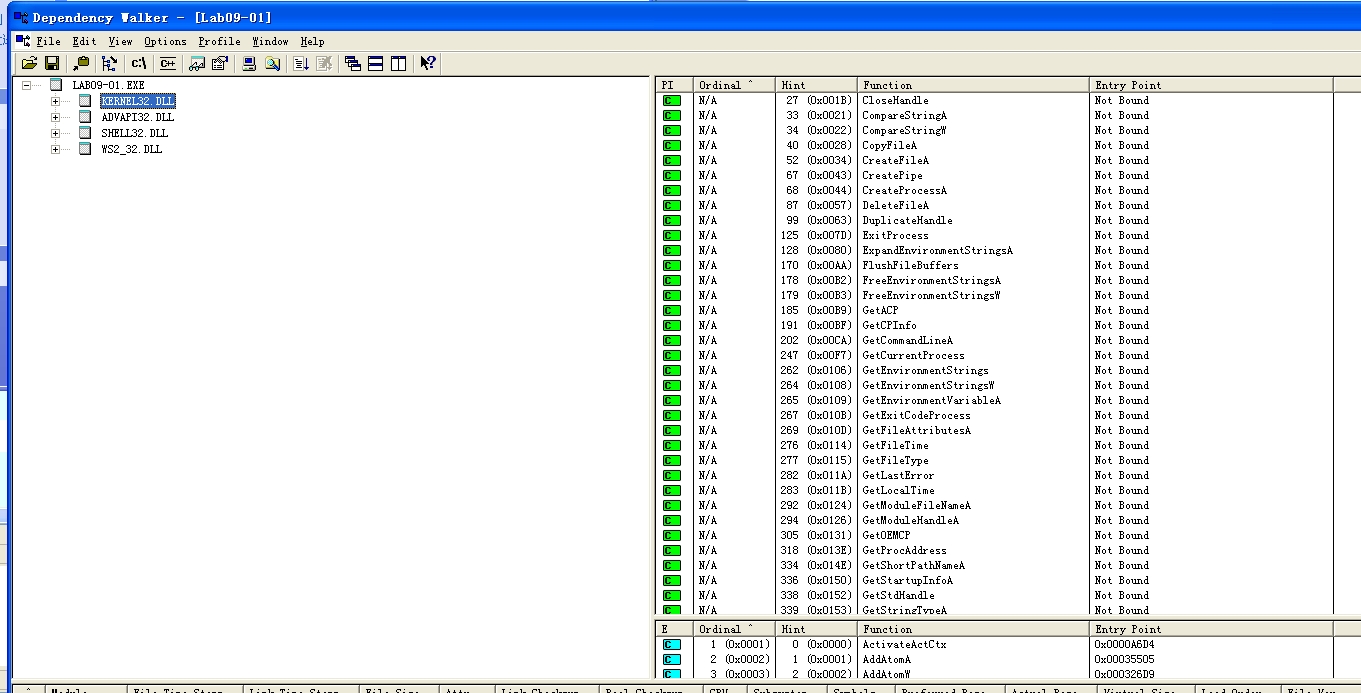
**完成课本Lab7的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。**

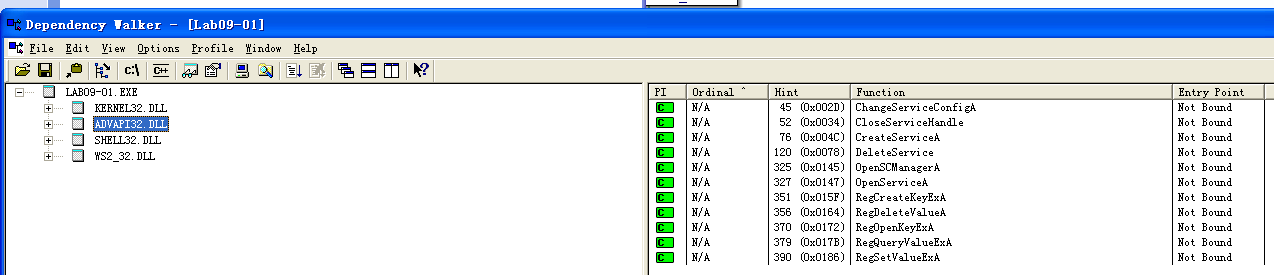
1. **实验原理**

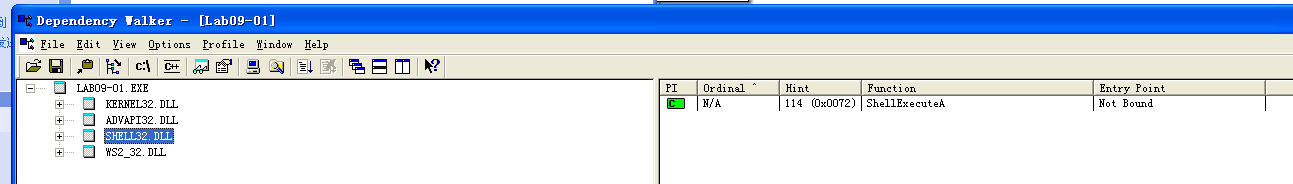
**配置XP虚拟机环境，安装动态、静态分析工具，使用静态和动态分析工具。**

1. **实验过程**
2. **Lab09-01**

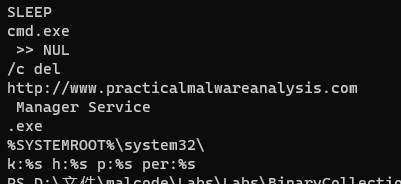
**查看导入函数，看到调用了KERNEL32.DLL和ADVAPI.DLL中许多关于文件、服务的函数，还有SHELL32.DLL中的ShellExecuteA函数，查找资料后知道该函数可用于运行可执行文件或用浏览器链接一个url。除此之外还调用WS3\_32.DLL，进行网络连接和数据传输。**



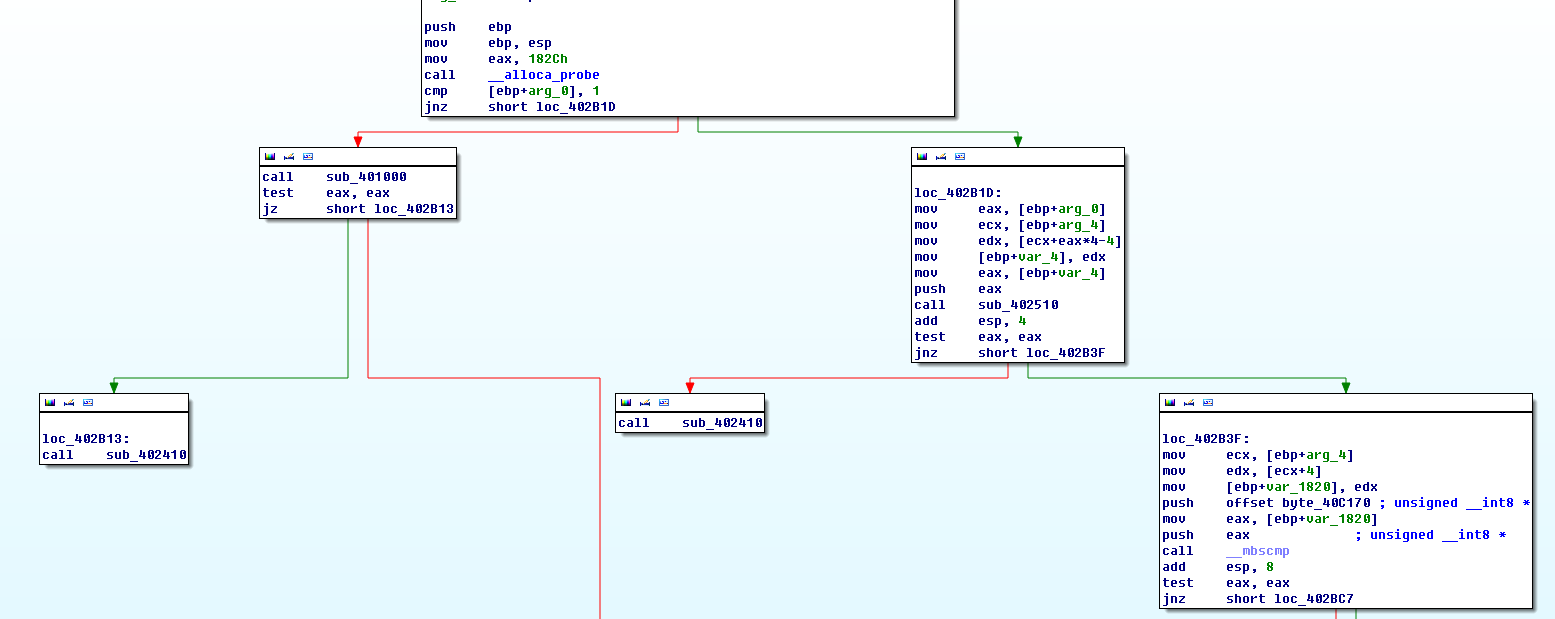




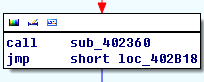
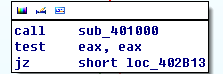
**查看字符串，看到一个可能会连接的网址，一个系统路径可能用于存放其他恶意文件。**



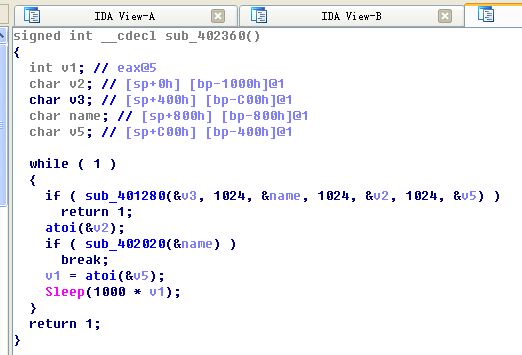
**打开IDA分析，观察整体的流程图，发现会根据程序的参数进行跳转，发现当仅仅只有默认参数（即双击程序）时程序的逻辑会非常简单，不会有特别的代码执行；另一个多参数的分支则逻辑更加就复杂。**



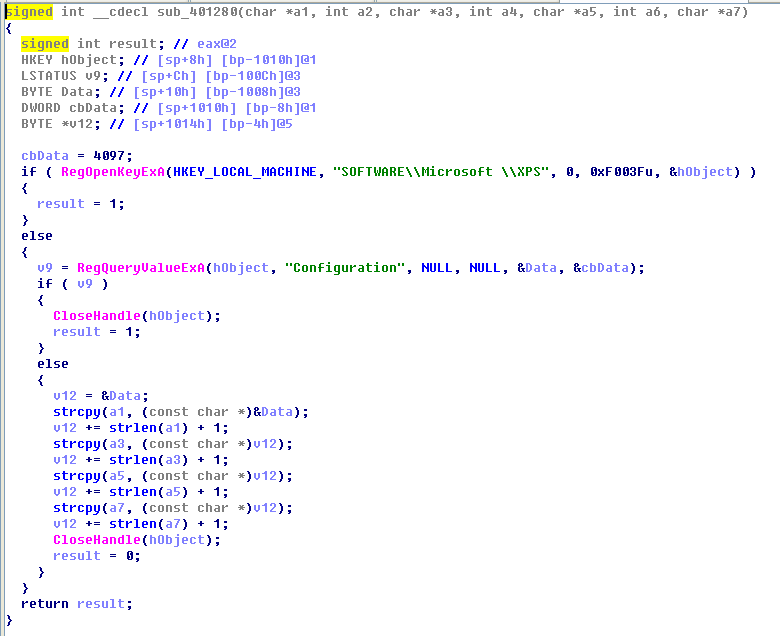
**左边的逻辑分支会进行指定注册表的获取和分析，如果该键值不存在则直接会调用一个函数进行自我删除，如果存在的话，则会调用sub\_402360 函数，然后结束。**



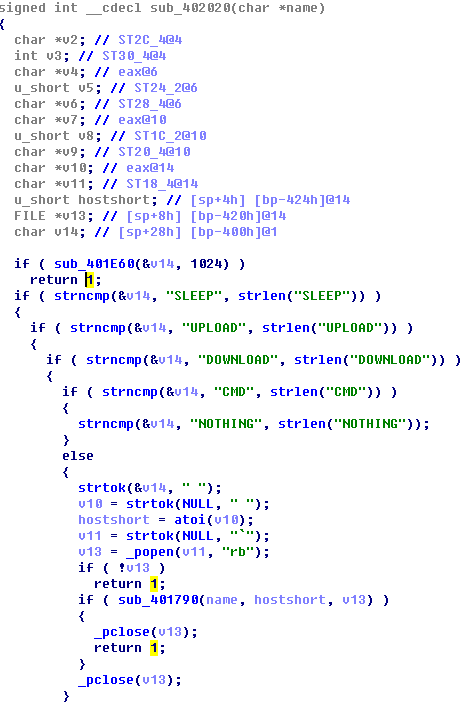
**跟进sub\_402360 函数进行分析，发现存在一个死循环，其中仅调用了两个函数sub\_401280和sub\_402020函数。**



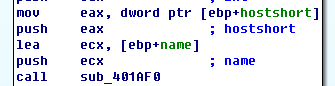
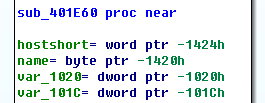
**先分析sub\_401280，发现该函数会获取指定注册表下的键值对，并且放入指定的缓存中。**



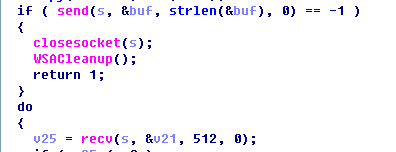
**再分析sub\_402020函数，发现该函数是一个后门执行的函数，其中包括 “sleep”, “upload”, “download”, “cmd” 等指令；其中获取的指令是通过函数sub\_401E60获得的。**



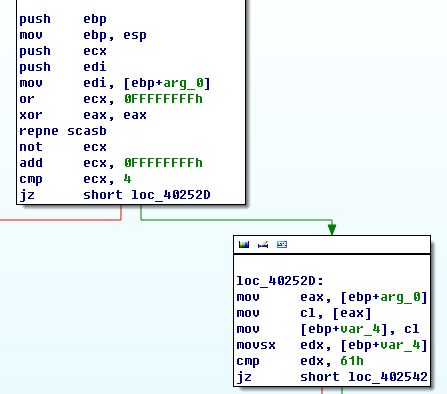
**再跟入函数sub\_401E60进行分析，发现其中的sub\_401AF0函数非常可疑，传给它的参数带有hosts和name等关键词，猜测该函数可能涉及网络数据发送。**

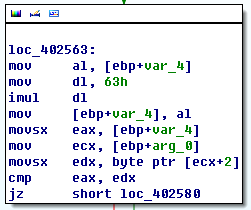
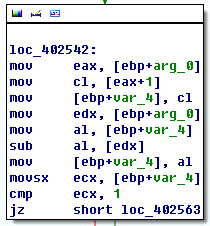


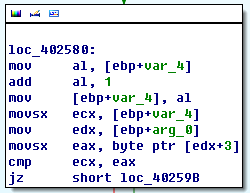
**跟入该函数，发现该函数为一个网络数据发送和接受函数；其中网络数据包的相关参数来着前面函数sub\_401470 和 sub\_401D80从指定注册表的键值中获取。**



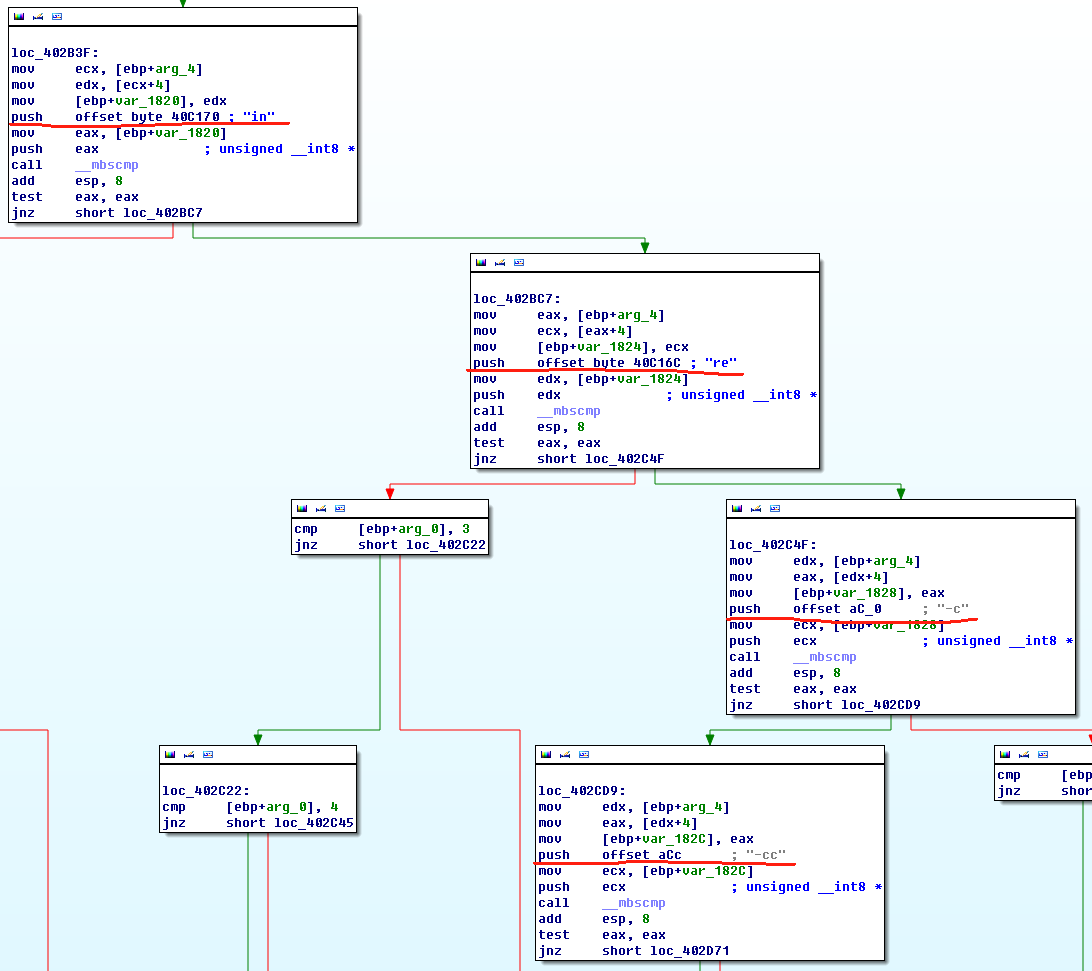
**右边分支发现一个函数sub\_402510调用，此函数的调用直接影响下面的逻辑跳转，跟进去分析，发现该函数是一个程序执行密码的检测函数，通过参数输入的字符串，挨个比较，从而决定下一步的跳转逻辑：先检查密码的长度是否为4，再依次检查第一位是否为a，第二位是否是b，第三位是否是c，第四位是否是d。**



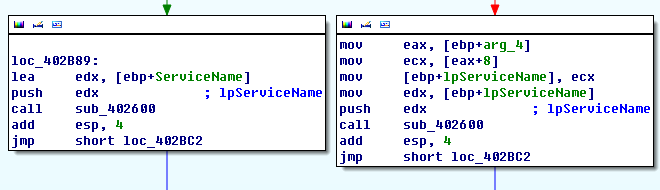




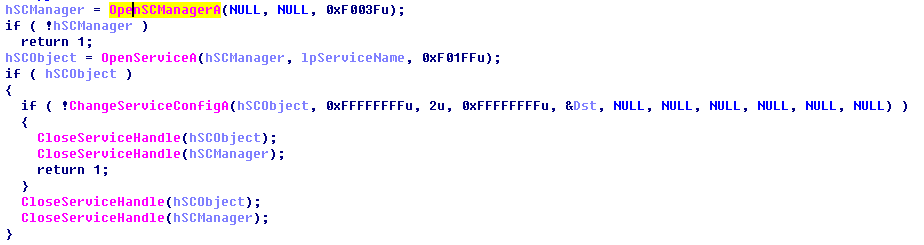
**再根据逻辑往下分析，发现程序将根据命令行输入的另一个参数进行不同的代码执行，分别有几种模式 ‘-in’, ‘-re’, ‘-c’, ‘-cc’。**



**分析 “-in” 的模块，该模块又会进行参数个数的判断，但是在参数个数为3和为4的两个分支中都会调用同一个函数，经过分析，我们可以确定该函数为恶意程序的安装代码部分。**

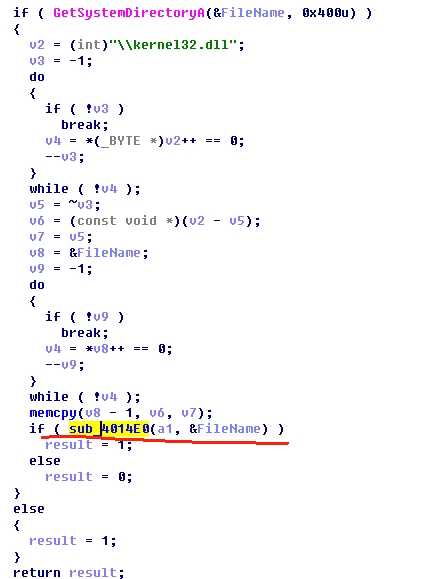


**跳转到调用的过程，发现这段代码会进行服务的创建和程序的自我复制到系统目录文件下。**





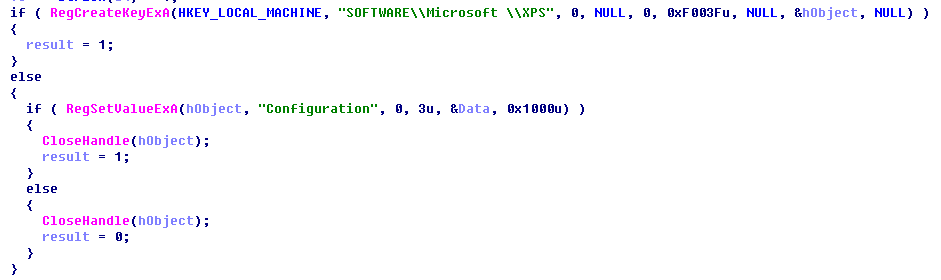
**该段代码的一个分支调用了一个函数sub\_4015B0，跟进该函数后发现该函数修改该新复制的文件的时间戳与系统文件一样。**



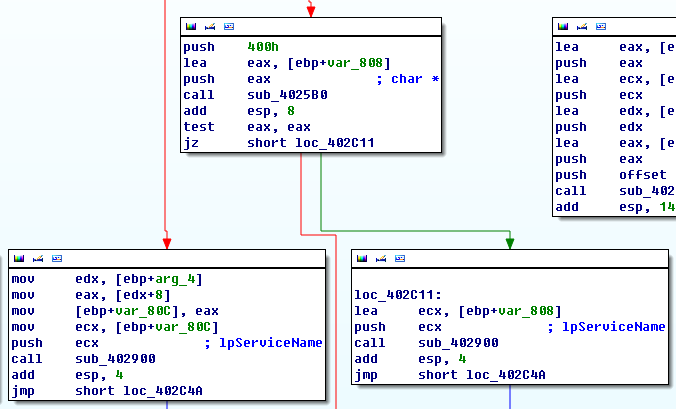
**同时该段代码还会跳转到下一个模块调用sub\_401070函数，同时出现了可疑的字符串“60”，“80” 和 “http://www.practicalmalwareanalyss.com” 被作为参数传入到该函数。**



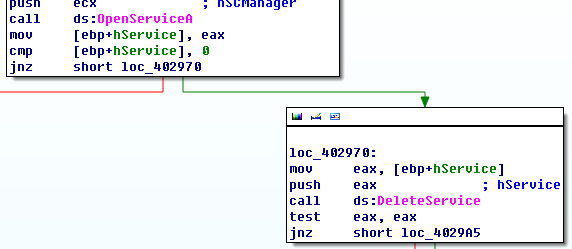
**跟入该函数进行分析，发现该函数会创建一个注册表项 “HKLM\SOFTWARE\Microsoft\XPS” 的 “configuration”键，并且修改相应的键值为输入的参数即上面的字符串。**

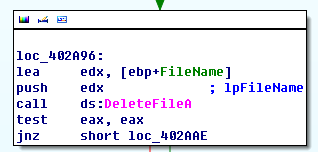


**分析 ‘-re‘ 模块，与 ’-in‘ 模块一样，该模块同样会判断参数的个数，三个参数和四个参数会跳转到不同的逻辑，但最终也会调用同一个函数sub\_402900，右边函数在调用sub\_402900之前还调用了sub\_4025B0函数，该函数非常简单，用于获取当前程序的执行的绝对路径从而得到函数sub\_402900的输入参数；而在左边的部分，该参数是由命令行输入提供的，也就是说两个分支会根据命令行输入参数的个数来获取调用函数sub\_402900的参数。**

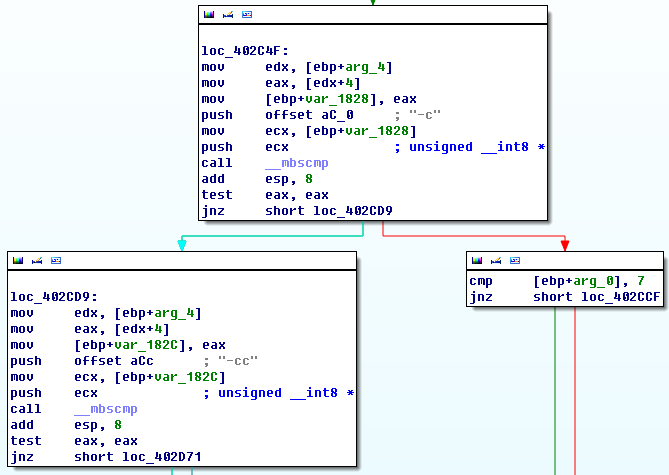


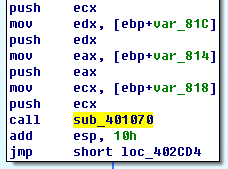
**跟进函数sub\_402900 进行分析：发现该函数会根据传入的服务名删除程序创建的服务，同时根据指定路径进行程序的自我删除。**



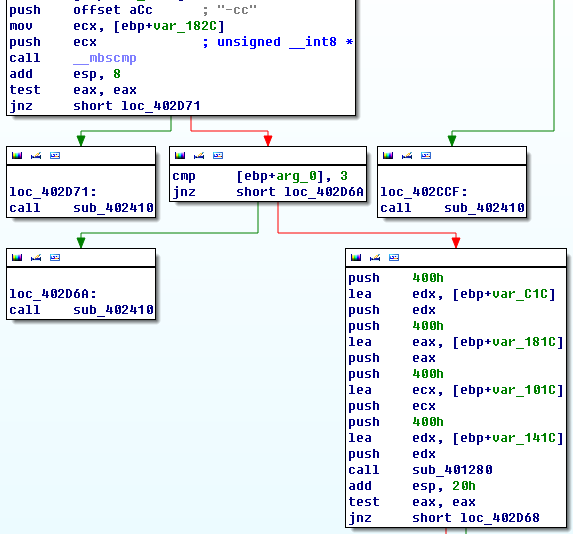


**分析 “-c” 模块，该模块有两个逻辑分支，一个指向 “-cc” 模块，此处分析另一个模块，该模块需要7个参数，同时该函数还调用了sub\_401070 函数来修改指定的键值。**





**再分析 “-cc” 模块，除了一些自我删除的函数调用逻辑模块外，另一个逻辑分支调用了函数sub\_401280和sub\_402E7E。**



**其中sub\_401280会获取指定注册项下的键值对，并且放入指定的缓存区中。**

**另一个函数sub\_402E7E输入参数为一个字符串和四个变量，这个与printf函数的格式非常相似，该函数为字符串打印函数。**



**综上分析，**

* **该恶意程序需要命令行 ‘-in’ 进行安装，同时安装需要密码 ‘abcd’。**
* **该恶意程序还能能执行多种指令 ‘-in’ 进行安装，‘-re’ 进行恶意程序的彻底清楚，‘-cc’ 进行注册表打印，‘-c’ 进行注册表配置。**
* **此恶意程序在成功安装后自我复制到指定系统目录下，并且创建服务进行自启动。**
* **另外，该程序还会创建后门与C&C通信，执行任意命令。**

**问题：**

1. **如何让代码安装自身？**

**答：利用命令行输入参数 ‘-in’ 和密码 ‘abcd’**

1. **这个恶意代码的命令行选项是什么？它要求的密码是什么？**

**答： 该恶意程序还能能执行多种指令 ‘-in’ 进行安装，‘-re’ 进行恶意程序的彻底清楚，‘-cc’ 进行注册表打印，‘-c’ 进行恶意代码更新配置；要求的密码是 ‘abcd’**

1. **如何利用OllyDbg永久修补这个恶意代码，使其不需要指定的命令行密码？**

**答： 直接修改检验部分，跳过检验步骤即可**

1. **这个恶意代码基于系统的特征是什么？**

**答：会创建一个指定注册表项 “HKLM\SOFTWARE\Microsoft \XPS” 的 “configuration”键”；同时会创建一个服务，该服务的名称由安转时传入的参数决定；另外，该程序还会自我赋值到Windows系统目录下。**

1. **这个恶意代码通过网络命令执行了哪些不同的操作？**

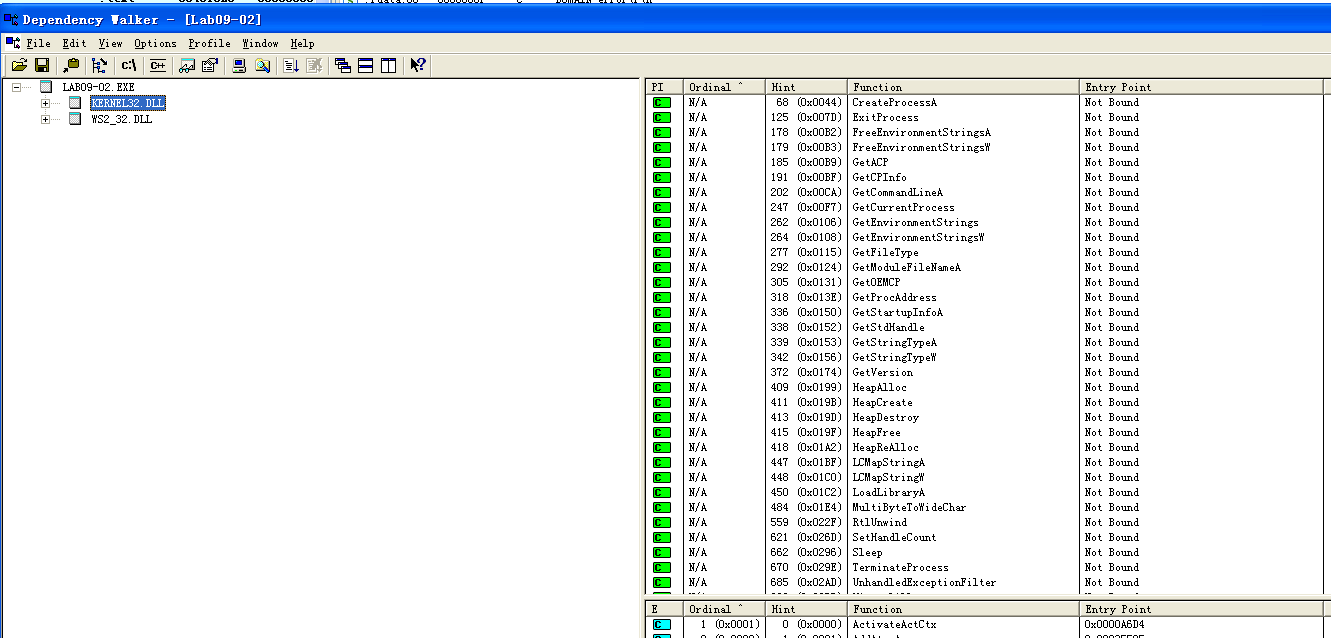
**答：通过网络命令可以执行 “sleep”, “upload”, “download”, “cmd” 和 “nothing”；**

1. **这个恶意代码是否有网络特征？**

**答：会向指定的站点发出http (HTTP/1.0 GET) 请求，默认的站点是“http://www.practicaolmalwareanalysis.com”**

1. **Lab09-02**

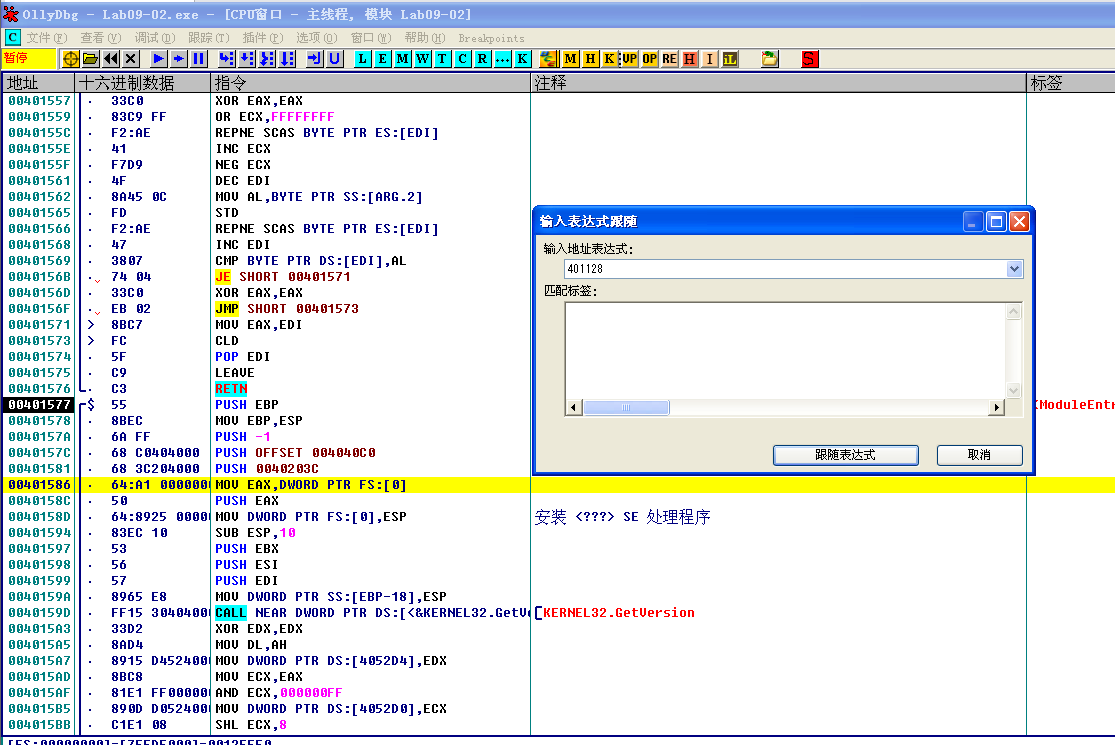
**查看导入函数，看到KERNEL32.DLL中许多重要的系统函数，并调用WS3\_32.DLL，进行网络连接和数据传输。**



**查看字符串，看到这些导入函数与cmd的字符串。**

****

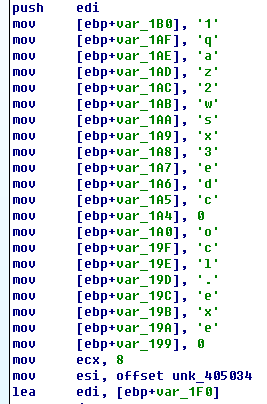
**在Ollydbg中跳转到IDA中显示的main位置401128。**



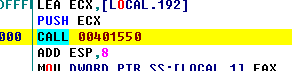
**看到一系列的mov指令给一个字符数组赋值。**



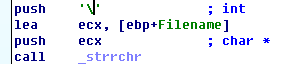
**在IDA中转换后看到这是两个字符串，分别为1qaz2wsx3edc和ocl.exe。**

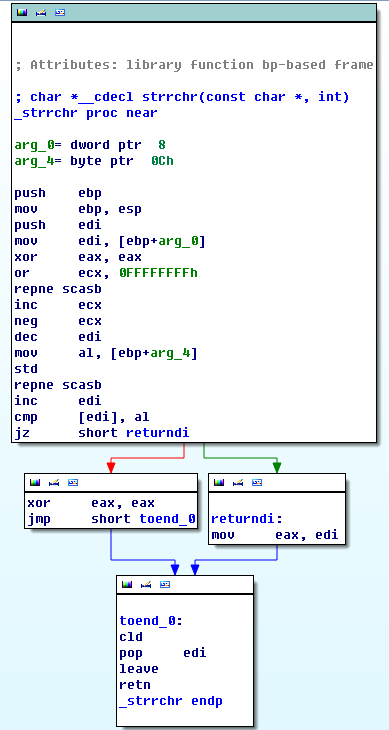


**从主函数入口开始单步执行，看到调用了401550函数。**

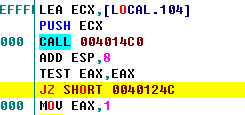


**在IDA中转到该函数，看到传入了参数‘\’和Filename，这个函数在这里就是用于查找最后一次出现\的位置，并返回从\开始往后的所有字符，换句话说，就是用于获取当前文件的名称。**

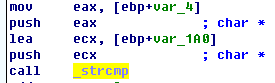
****



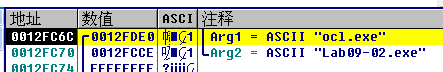
**继续执行，发现调用了4014C0。**



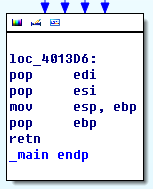
**在IDA中看到是\_strcmp。**



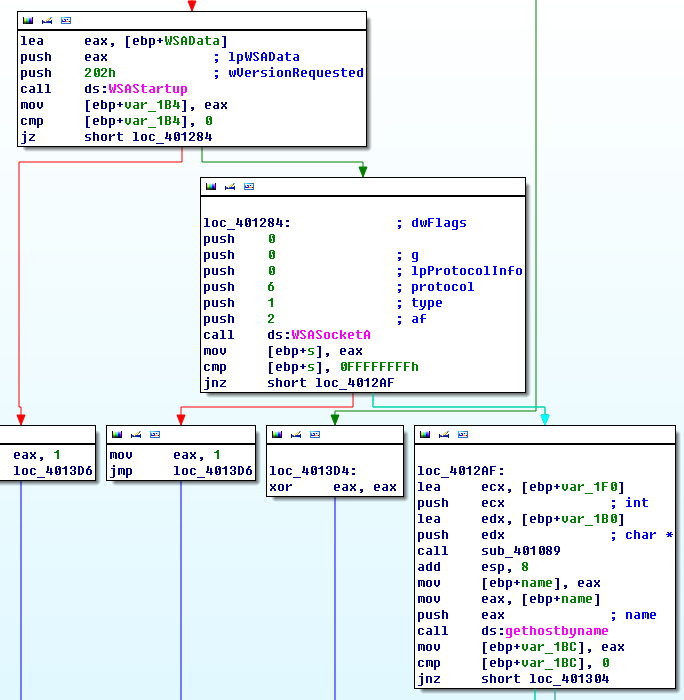
**而传入的两个参数一个是文件名，即Lab09-02.exe，另一个是ocl.exe，不相等。**



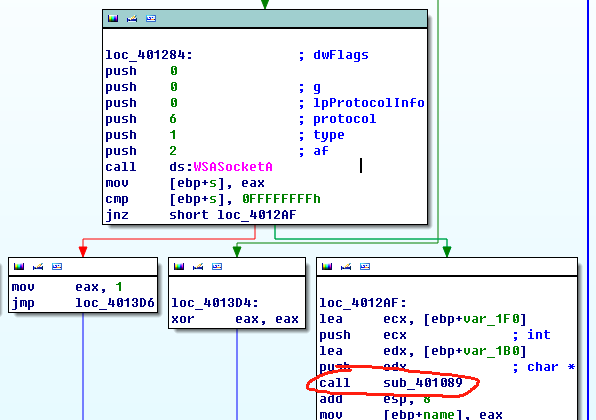
**到IDA中查看，发现如果结果不相等，则会直接结束程序。**



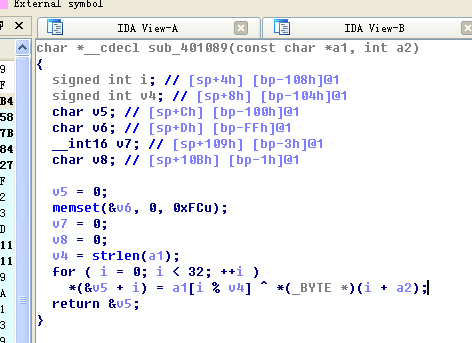
**在IDA中查看另外的分支，首先调用WSAStartup，准备通过网络传输。**



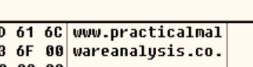
**之后调用WSASocketA，结果若非全1，则会将得到的缓冲区结果和字符串1qaz2wsx3edc传入sub\_401089函数。**



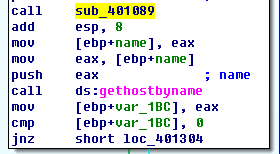
**查看该函数，看到该函数的作用是将传入的两个字符串逐个进行异或。**



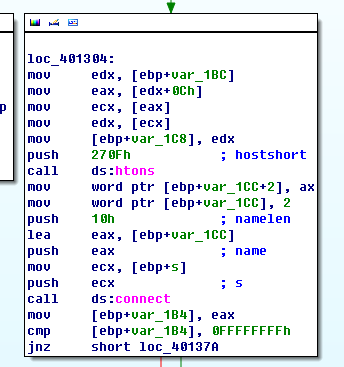
**下断点执行，看到结果为一个网址。**



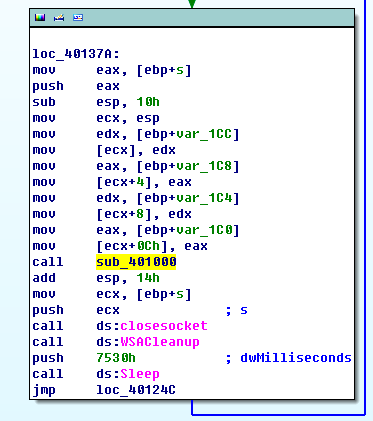
**该网址被用在gethostbyname中作为name参数，而结果会返回一个ip地址。**



**获取成功后会调用connet进行连接。**



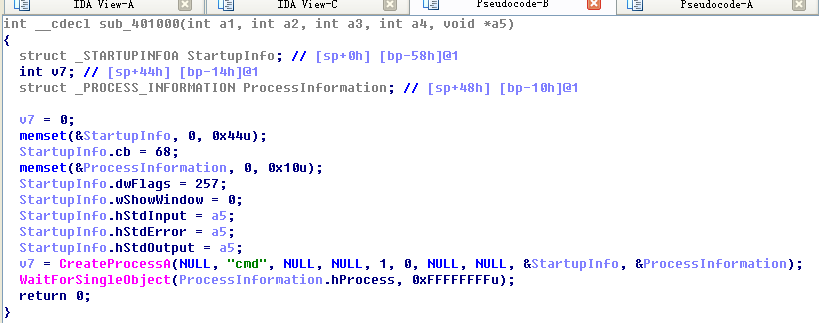
**连接后会将cmd.exe为参数调用sub\_401000。**



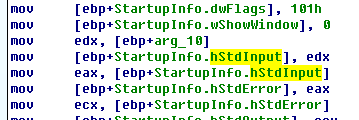
**在createprocess之前的信息可以看出，这是一个反向shell。**

**传递给CreateProcessA的STARTUPINTO结构被修改了。**

**可以看到cmd.exe作为参数被传递，CreateProcesA将要运行cmd.exe。这个结构的成员wShowWindow被设置为0，意思就是说会以窗口隐藏的方式运行。**



**注意到标准输入输出等都被设置为套接字，这样就绑定了套接字和cmd的标准流，也就是说cmd启动之后，所有经过套接字的数据都将发送给cmd，cmd产生的输出会通过套接字发出。**



1. **在二进制文件中，看到的字符串是什么？**

**可以看到一些导入函数和字符串cmd。**

1. **当运行这个二进制文件时，会发生什么？**

**它没有做什么就退出了。**

1. **怎样让恶意代码的攻击负载（payload）获得运行？**

**运行它之前将它命名为ocl.exe。**

1. **在地址0x00401133发生了什么？**

**一个字符串在栈上被创建，攻击者利用它来混淆静态分析技术和字符串工具中的字符串。**

1. **传递给sub\_401089的参数是什么？**

**是字符串1qaz2wsx3edc和一个数据缓冲区。**

1. **恶意代码使用的域名是什么？**

**practicalmalwareanalysis.com。**

1. **恶意代码使用什么编码函数来混淆域名？**

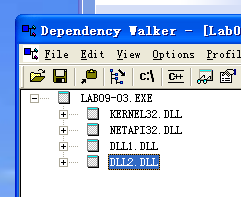
**用字符串1qaz2wsx3edc异或加密的DNS名来解密域名。**

1. **恶意代码在0x0040106e处调用CreateProcessA函数的意义是什么？**

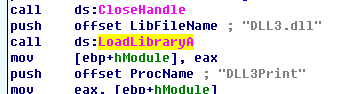
**设置stdout. stderr和stdin的句柄到socket(被用在CreateProcessA的STARTUPINFO结构中)。由于用cmd作为CreateProcessA的参数调用CreateProcessA，因此通过绑定一一个套接字与命令shell来创建逆向shell。**

1. **Lab09-03**
2. **lab09-03.exe导入了哪些dll？**

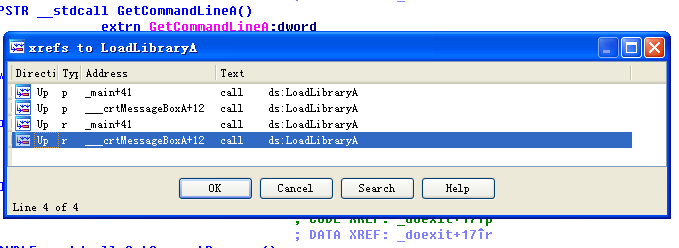
**查看导入函数，看到导入了4个dll。**



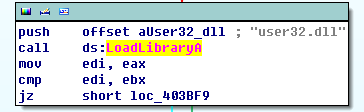
**再用IDA打开，发现还使用LoadLibraryA导入了DLL3.dll。**



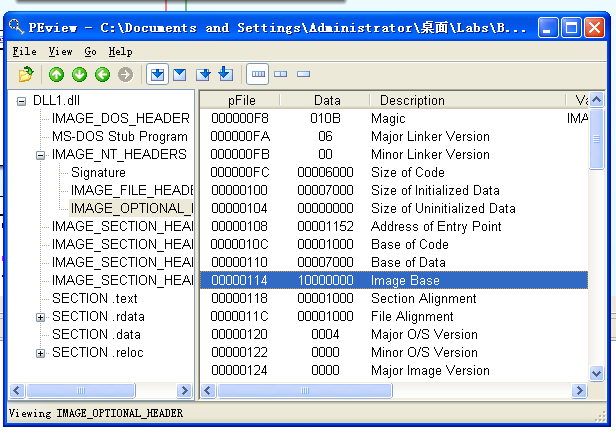
**查看LoadLibraryA的Xrefs，看到还有一个位置。**



**跳转到该位置，看到调用了user32.dll。**

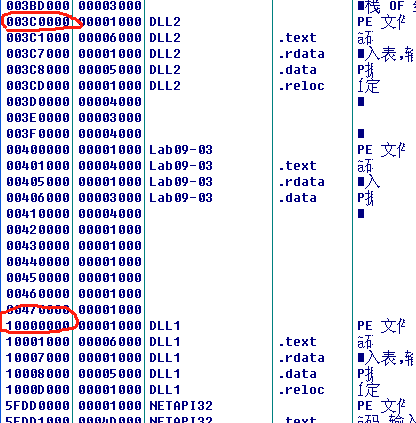


1. **dll1.dll,dll2.dll,dll3.dll要求的基地址是多少？**

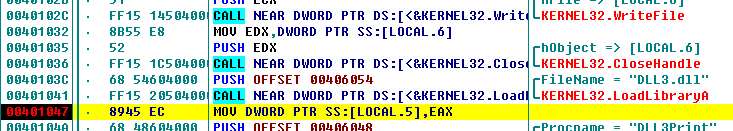
**用PEView查看，看到都是0x10000000.**

1. **当使用Ollydbg调试Lab09-03.exe时，为dll1.dll,dll2.dll,dll3.dll分配的基地址是什么？**

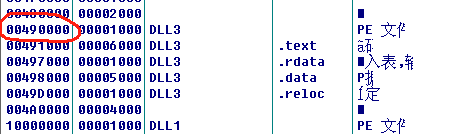
**用Ollydbg调试，切换到内存窗口，看到DLL1的是0x10000000，DLL2的是0x003C0000。**



**因为DLL3是在程序中动态加载的，因此找到LoadLibrary函数，在下一行下断点。**

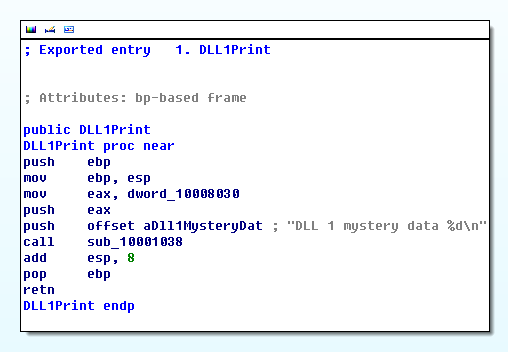


**运行后打开内存窗口，看到DLL3基地址是0x00490000.**

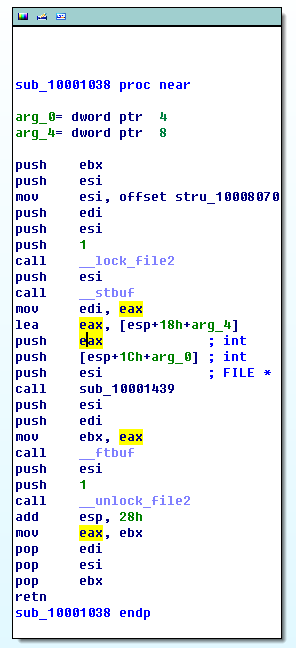
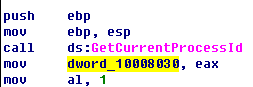


1. **当lab09-03.exe调用dll1.dll中的一个导入函数时，这个导入函数都做了什么？**

**第一个导入函数为DLL1PRINT，用IDA载入DLL1，搜索该函数，看到将一个字符串“DLL 1 mystery data”，和一个全局变量作为参数，调用了sub\_10001038。**



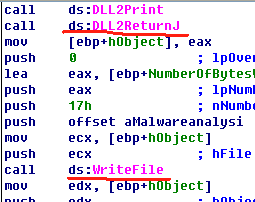
**跳转到该函数，第一个参数是字符串，第二个参数在eax中，来自dword\_10008030。查看交叉引用，看到其中的内容是GetCurrentProcessId的返回值。**



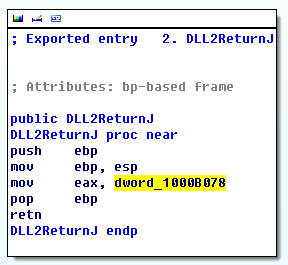
**综上，该函数会打印“DLL 1 mystery data %d”，%d代表当前进程的id.**

1. **当lab09-03.exe调用WriteFile函数时，写入的文件名是什么？**

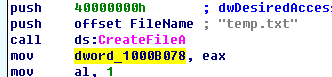
**回到.exe文件，看到传入WriteFile的参数hObject来自DLL2RETURNJ的返回值。**



**在DLL2中打开该函数，发现返回值是dword\_1000B078.**

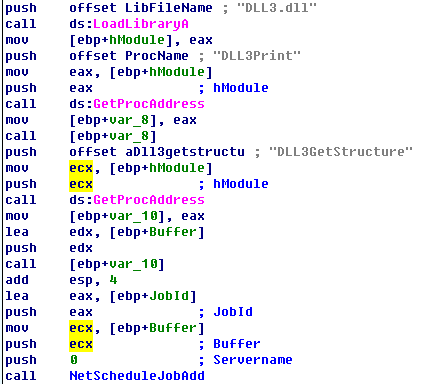


**查看其Xref，发现在主函数中将temp.txt的句柄赋给了它。也就是说，写入的文件名就是temp.txt。**

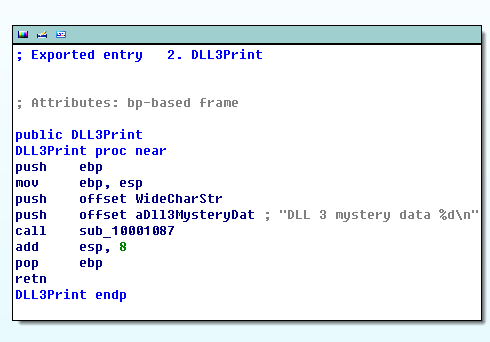


1. **当lab09-03.exe调用NetScheduleJobAdd创建一个job时，从哪里获取第二个参数的数据？**

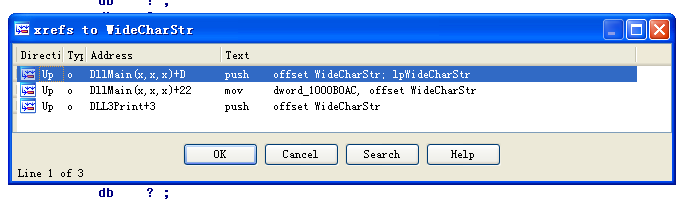
**先是加载了dll3.dll到内存中，之后是调用getprocaddress获取dll3.dll中的dll3print的地址，之后call [ebp+var\_8]来执行该函数，接着调用getprocaddress获取dll3getstructure的地址，然后call [ebp+var\_10]执行该函数。**

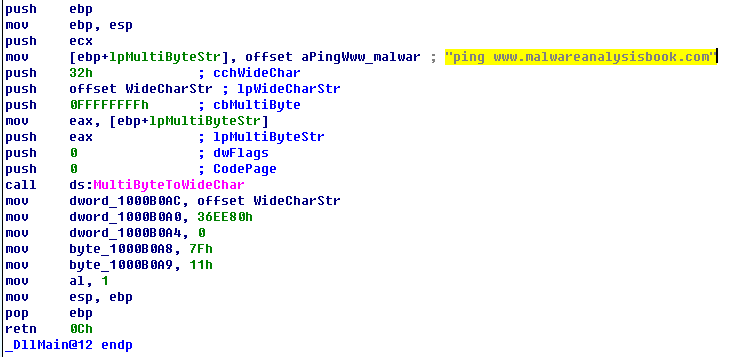


**使用ida载入dll3.dll，先看dll3print。**

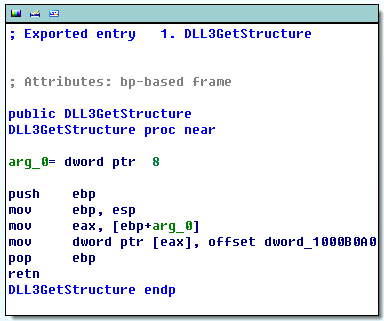


**我们知道它要打印的数据为widecharstr，双击跟进后查看交叉引用。**

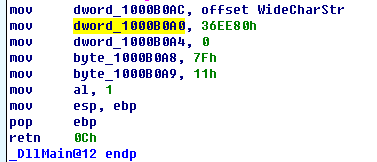
**可以看到在它之前调用了Multibyteyowidechar，这个函数的作用就是将多字节的形式转换成宽字符的形式。它要转换的内容是lpmultibytestr，其实就是上面的"ping www.malwareanalysisbook.com"字符串。将其转换成宽字符的形式后将其保存到widecharstr。**



**接着分析dll3getstructure。可以看到这个函数有一个参数，将其赋给eax，该函数调用时传入的参数是edx，而edx的值是buffer的内容。也就是说在调用dll3strcture时其参数就是局部变量的地址。把dword的值赋给eax指向的地址中，也就是前面说的buffer。**



**查看交叉引用，可以看到是一系列的move赋值的操作。这些数据的地址会保存在buffer里，而buffer保存在ecx，而之后ecx是作为netschedulejobadd的第二个参数。这些数据其实就是at\_info结构体的数据**



**综上，Lab09-03.exe 从DLL3GetStructure中获取NetScheduleJobAdd调用的缓冲区，它动态地解析获得第二个参数的数据。**

1. **运行或调试lab09-03.exe时，会看到打印出三块神秘数据。dll1的神秘数据，dll2的神秘数据，dll3的神秘数据分别是什么？**

**神秘数据1是当前进程的标识。神秘数据2是打开temp.xt文件的句柄，神秘数据3是字符串ping www.malwareanalysisbook. com在内存中的位置。**

1. **如何将dll2.dll加载到IDA中，使得它与ollydbg使用的加载地址匹配？**

**当使用IDAPro加载DLL时选择手动加载，当提示时，输入新的映像基准地址。本例中，地址是0x003C0000。**

1. **Yara规则**

**根据以上分析结果，得到Yara规则如下。**

rule lab0701

{

strings:

$string1 = "Malservice"

$string2 = "[**http://www.malwareanalysisbook.com**](http://www.malwareanalysisbook.com)"

$string3 = " Internet Explorer 8.0"

condition:

filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) == 0x00004550 and all of them

}

rule lab0702

{

strings:

$string1 = "http://www.practicalmalwareanalysis.com/cc.html"

$string2 = "CoCreateInstance"

condition:

filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) == 0x00004550 and all of them

}

rule lab0703

{

strings:

$string1 = "127.26.15.13"

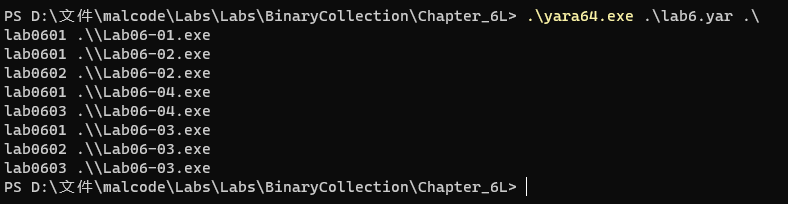
$string2 = "WARNING\_THIS\_WILL\_DESTORY\_YOUR\_MACHINE"

$string3 = "Lab07-03.dll"

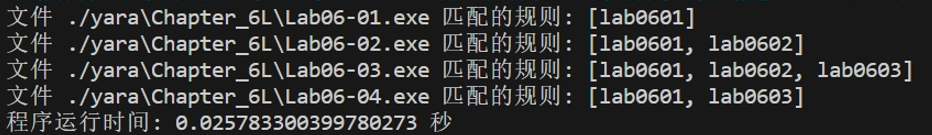
condition:

filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) == 0x00004550 and all of them

**运行结果如下，可看到成功匹配所有文件。**



**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**

**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。**

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)

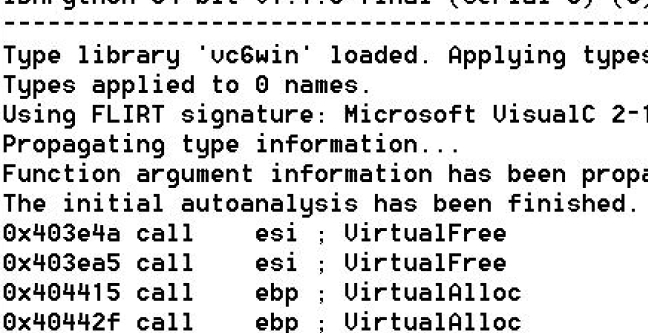
if m == 'call' or m == 'jmp':

op = idc.GetOpType(line,0)

if op == o\_reg:

print '0x%x %s' % (line,idc.GetDisasm(line))

**结果如下**

****

1. **实验结论及心得体会**
2. **对程序结合IDA和Ollydbg的分析有了进一步了解。**
3. **了解了更多Windows恶意代码的形式和特征。**