

**恶意代码分析与防治课程实验报告**

**实验五：ida动态分析**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2111033

姓 名 艾明旭

班 级 信息安全一班

1. **实验目的**

**巩固复习ida的使用技巧，重新将之前学习过的ida使用方法使用得到解决新的出现的问题。**

**练习跟随Ida流程图进行相关恶意代码的流程分析，识别和解决恶意代码对于普通的关于计算机进程的操控等等的方法和策略。**

**对于ida相关应用程序的分析，可以考虑到计算机网络连接的一些内容，我们在相关的应用实践当中，可以将应用程序对网络的依赖进行很好的分析，学习了解恶意代码依赖和控制网络的方法。**

**同时，本次实验的四种恶意代码十分相近，其中有许多的以及**

1. **实验原理**
   1. **实验环境**

Windows xp，VMWARE，Windows11 ,win8.1

* 1. **实验工具**

**Ida pro 6.6 ida python,yara**

* 1. **原理**

**IDA Python是基于IDA Pro的Python扩展，它允许用户通过编写Python脚本与IDA Pro进行交互和自动化操作。**

**扩展性：IDA Pro是一款反汇编和静态分析工具，IDA Python充分利用了Python的灵活性和强大的标准库，提供了一组API和对象来与IDA Pro进行交互。用户可以通过编写Python脚本来扩展IDA Pro的功能，实现自定义的反汇编、分析、导出等操作。**

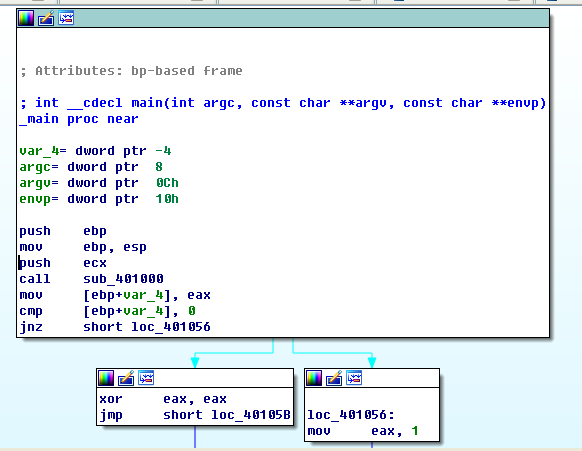
**提供API：IDA Python提供了一组完整的API，用于操作和访问IDA Pro的各种特性和数据结构。这些API包括函数、变量、指令、图形界面、数据库查询、导出和导入等功能。用户可以通过调用这些API来实现各种自动化任务，例如自动识别函数、修改指令、导出数据等。**

**事件驱动：IDA Python还支持事件处理机制，允许用户注册和处理特定的IDA事件。当IDA Pro发生某些事件时，例如载入二进制文件、分析完成、用户交互等，用户可以编写回调函数来响应这些事件，执行相应的操作。**

**脚本界面：IDA Python提供了一个交互式的Python解释器，可以在IDA Pro中直接编写和执行Python代码。用户可以通过这个Python解释器与IDA Pro进行实时的交互，进行数据查询、运行脚本等操作。**

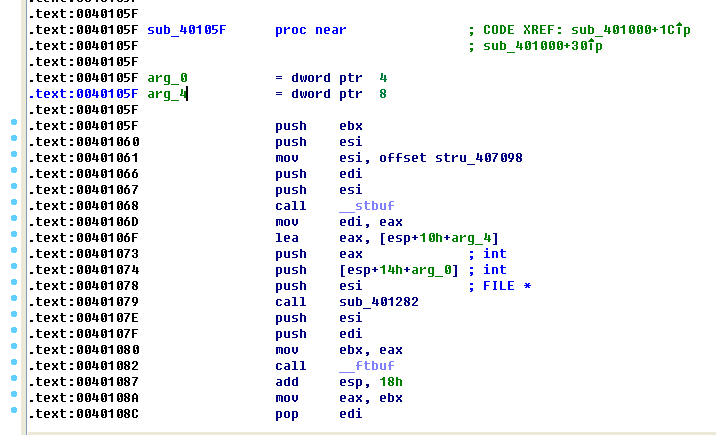
1. **实验过程**

****(一)Lab06-01.exe****  
****1、由main函数调用的唯一子过程中发现的主要代码结构是什么？****

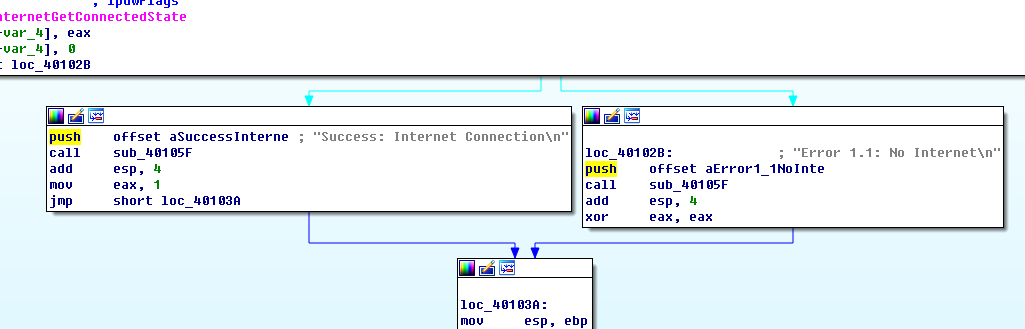


先声明了一个局部变量var\_4，然后调用了InternetGetConnectedState函数，接下来明显是一个if语句的分支结构，如果InternetGetConnectedState函数的返回值是0，则跳转loc\_10102B处，如果InternetGetConnectedState函数返回值为1则不跳转，接着执行下面的指令。

1. ****位于0x40105F的子过程是什么？****  
   双击0x40105F来到函数中，发现它顺序调用了三个函数，还有一个数据offset stru\_407098,并且在该偏移量中还存在FILE的字符，然后将这个数据存在了esi中，并且在这三个函数的调用中均用到了esi指向的FILE。



还发现sub\_40105F这个函数的调用处就是在main函数的那个if判断语句中，而且在调用sub\_40105F这个函数之前均先在栈中压入了一段字符串，所以猜测sub\_40105F这个函数就是一个printf函数

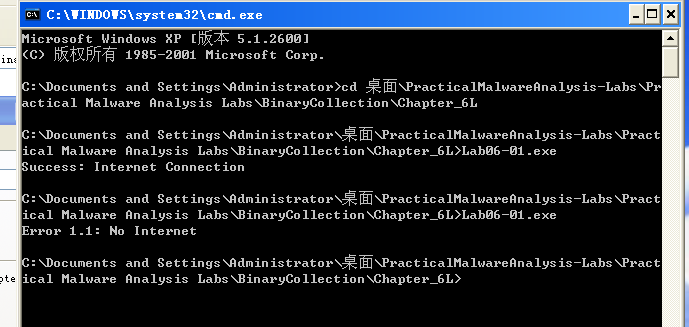


通过查找资料以及Printf相关的函数的认定，最终发现该函数就是Printf函数

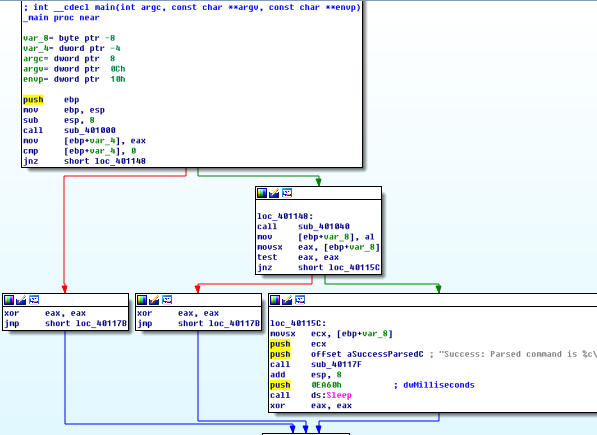
3、这个程序的目的是什么

由以上的分析可以猜测，这个程序的目的是检查是否有可用的Internet连接，如果有则打印“Success: Internet Connection”，如果没有则打印“Error 1.1: No Internet”。

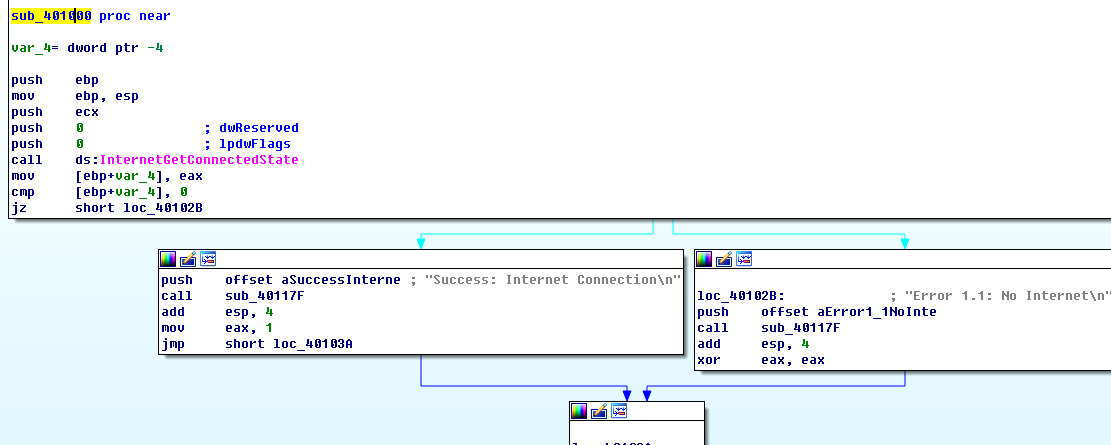
为了证明猜测，分别在联网状态和网络连接断开状态运行一下Lab06-01.exe，发现猜想正确。



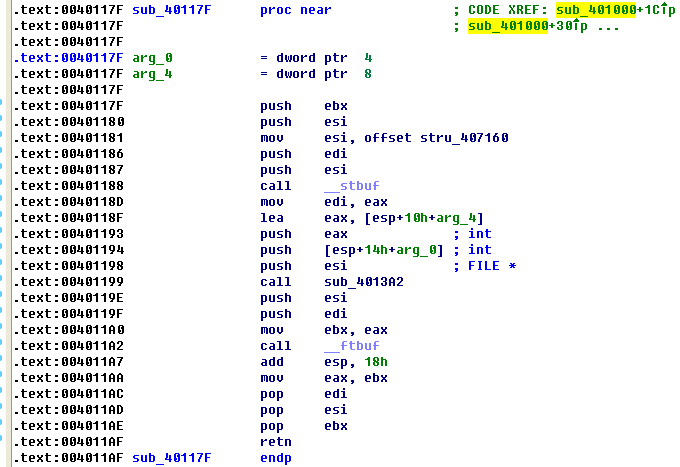
****(二)Lab06-02.exe  
1、main函数调用的第一个子程序执行了什么操作？****  
Main函数调用的第一个子程序是sub\_401000函数，双击进入函数中查看代码



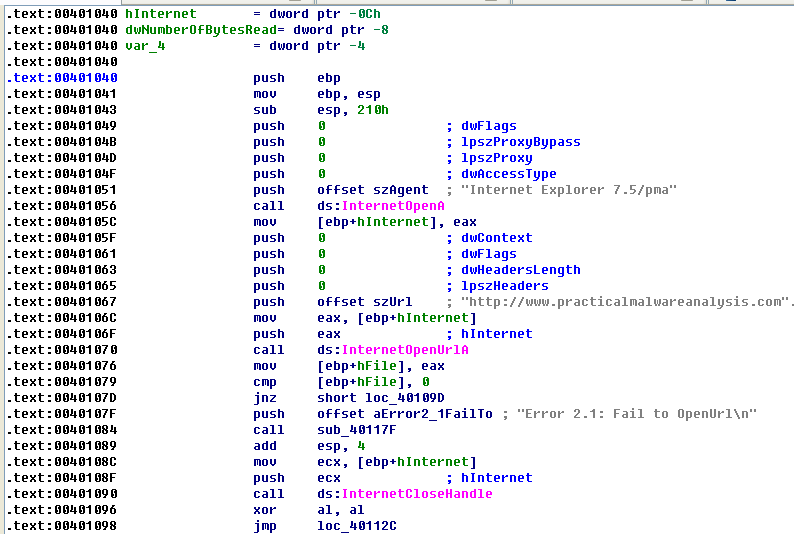
发现同Lab06-01.exe一样，查看是否存在可用的网络连接，如果是联网状态则打印字符串“Success: Internet Connection”，如果没有联网，则打印字符串“Error 1.1: No Internet”。

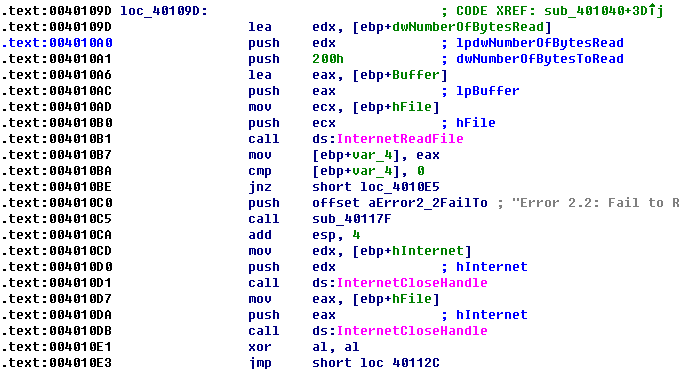


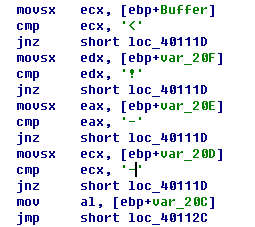
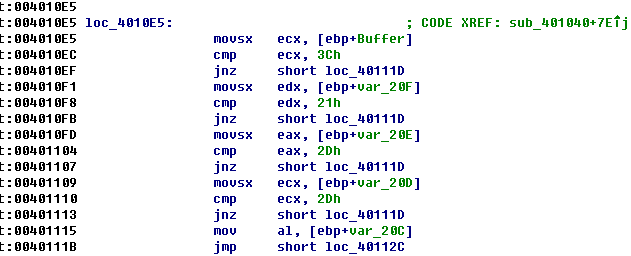
****2、位于0x40117F的子过程是什么？****  
进入sub\_40117F函数中查看，发现同Lab06-01.exe中的sub\_40105F函数一样，就是个printf函数



1. ****被main函数调用的第二个子过程做了什么？****  
   Main函数调用的第二个子过程是sub\_401040函数，双击进入函数内部首先看到了以下几个函数的调用，以及cmp和jnz的比较跳转



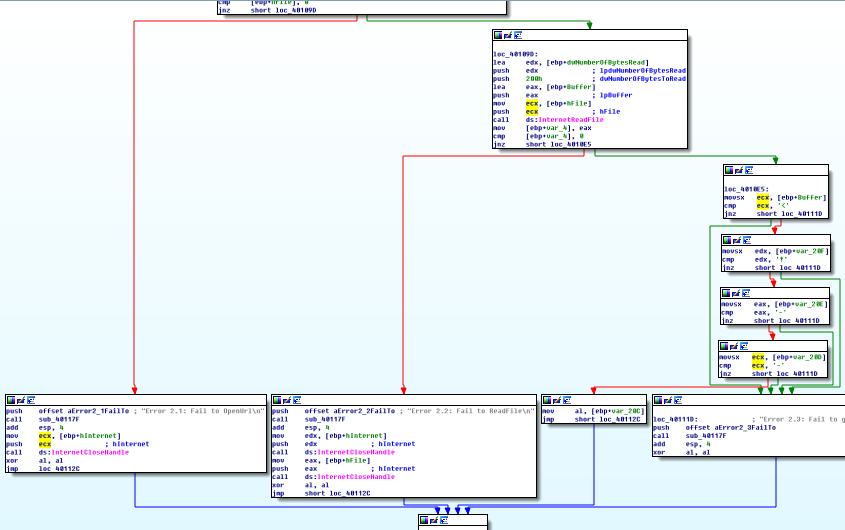




还发现了对于一些字符的比较

用R键将这些字符转化后，发现就是<!–，这个字符串一般位HTLM文档的注释开始部分

为了更直观的查看这些函数之间的逻辑结构，按空格键查看图形界面，放大后可用很清楚的看到这些函数之间的调用关系



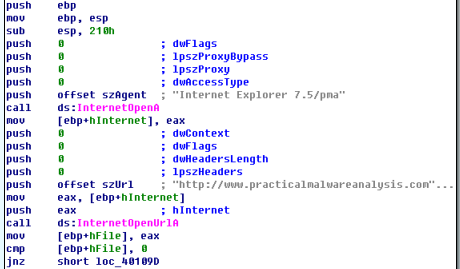
通过以上分析可用得出结论，sub\_401040首先调用InternetOpenA函数打开网络，然后调用InternetOpenUrlA函数从http://www.practicalmalwareanalysis.com下载HTML网页，然后cmp指令做出一个判读，如果下载不成功，则打印字符串Error 2.1: Fail to OpenUrl并调用InternetCloseHandle函数关闭连接，如果成功下载，则跳转到loc\_40109D地址处，然后调用InternetReadFile读取下载的HTML文件，紧接着又是一个cmp和jnz的比较跳转，如果读取不成功则打印Error 2.2: Fail to ReadFile并且调用InternetCloseHandle函数关闭连接。如果读取成功则跳转loc\_4010E5地址处执行代码解析网页，而我们看到这个解析规则，先比较前四个字符是否为<!–，如果均比较成功，则将第五个字符存放到al中作为返回值，如果不是<!–这四个字符中任何一个比较失败，则打印字符串“Error 2.3: Fail to get command”。所以就是找到网页注释正文开始的地方，并返回首地址。

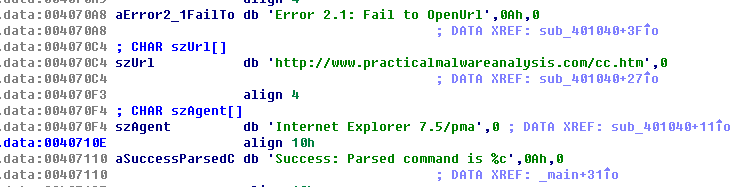
4、在这个子过程中使用了什么类型的代码结构？

很显然这个子过程使用了一个多层的if结构

5、在这个程序中有任何基于网络的指示吗？

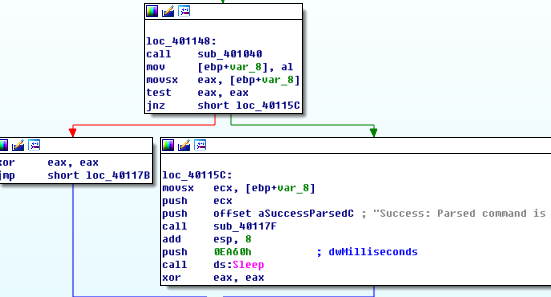
返回来到InternetOpenUrlA函数调用处，在它之前先将数据offset szUrl压入了栈中，我们双击szUrl可用明显的看到基于网络的指示





6、这个恶意代码的目的是什么？

返回main函数，在sub\_401040函数调用后，将解析出来的字符首地址在al返回并最后存放到ecx中压入栈作为参数，然后打印字符串“Success: Parsed command is %c”，而%c就是sub\_401040函数解析出来的字符串。紧接着休眠0EA60h=60000ms也就是60秒

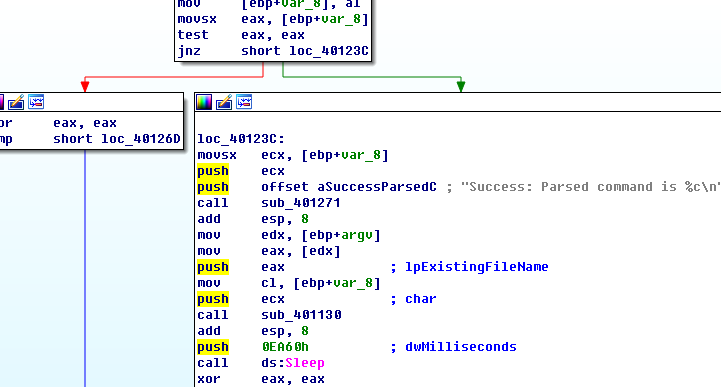


所以综合以上分析可以得出结论，Lab06-02.exe先判断是否有可用的网络，如果没有则停止运行，如果已连接网络则在http://www.practicalmalwareanalysis.com网址下载并解析HTML文件，然后打印字符串Success : Parsed command is加上注释正文字符，最后程序休眠60s然后终止运行。

(三)Lab06-03.exe

1、比较在main函数与实验6-2的main函数的调用。从main中调用的新的函数是什么？

对比发现新的函数是sub\_401130



1. ****这个新的函数使用的参数是什么****

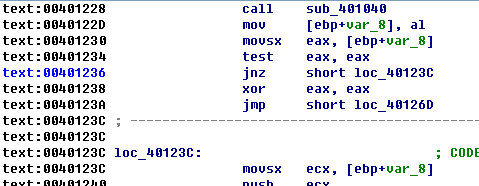


**看到在调用该函数时，先后push了eax和ecx，所以该函数有两个参数。**

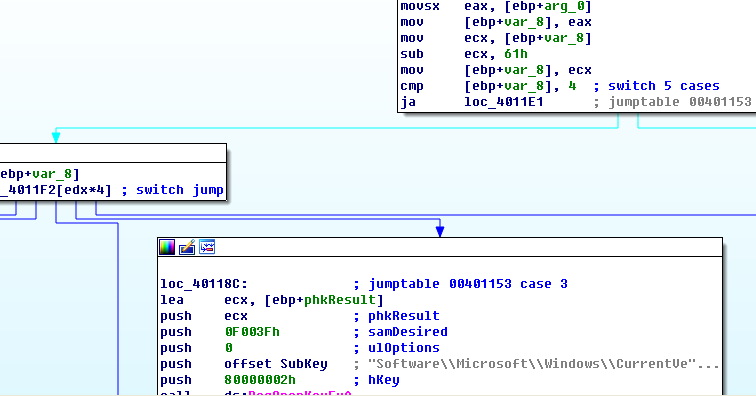
**从代码中可以看到，就是将argv的为地址的内容存放到了eax中，而argv的地址就是argv[0]的首地址，而argv[0]在main函数中作为一个指针，指向程序的路径及名称。也就是说，eax中存放的就是程序的名称即Lab06-03.exe的字符串。**

**对于ecx，是将var\_8的内容存放在了其中，而偏移地址var\_8的内容又是al也就是sub\_401040的返回值，前面提到sub\_401040函数的返回值就是所下载的HTML文件的注释正文的首地址。**

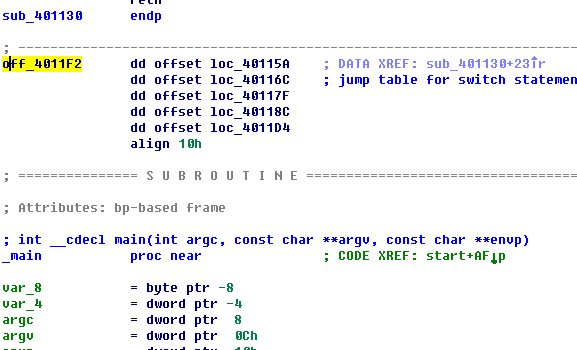
**所以，这两个参数一个是程序的名称，一个是注释正文字符。**



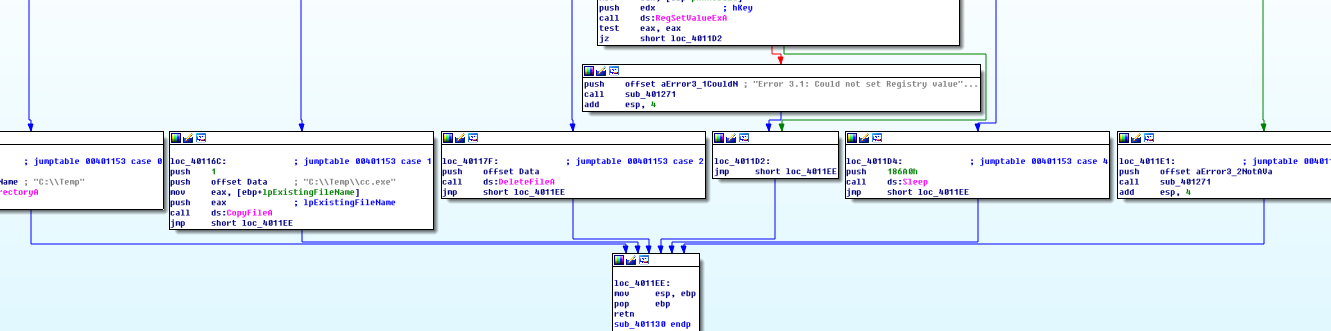
1. ****这个函数包含的主要代码结构是什么？****



从图中可以看到，ecx先减了一个61h也就是‘a’，然后在和4比较，如果大于4，则跳转到loc\_4011E1处，否则向下执行一个无条件跳转。  
双击off\_4011F2入无条件跳转处



可以看到它正好有5个跳转，很明显这就是一个swich结构



根据参数var\_8,其地址处的内容：

如果是a则跳转到loc\_40115A处；

如果是b则跳转到loc\_40116C处；

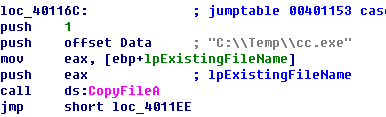
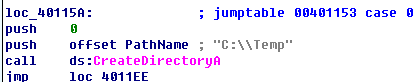
如果是c则跳转到loc\_40117F处；

如果是d则跳转到loc\_40118C处；

如果是e则跳转到loc\_4011D4处；

如果是其它数据则跳转到loc\_4011E1处。

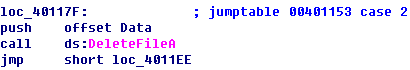
1. ****这个函数能够做什么？****  
   参数为‘a’跳转到loc\_40115A时，它创建了C:\Temp的目录



参数为‘b’跳转到loc\_40116C时，它调用了CopyFileA函数，并在之前压入了两个参数，一个是“C:\Temp\cc.exe”,一个是lpExistingFileName的内容。而lpExistingFileName是 sub\_401130的第二个参数

栈是先进后出的，在sub\_401130函数调用前先压入的argv也就是程序本身的名称，所以lpExistingFileName就是程序本身的名字。所以跳转到loc\_40116C时，就是将程序本身的名字复制到C:\Temp下并改名为cc.exe。

参数为‘c’跳转到loc\_40117F处时，它删除了C:\Temp\cc.exe。



参数为‘d’跳转到loc\_40118C处时，调用函数RegOpenKeyExA打开Software\Microsoft\Windows\CurrentVersion\Run，然后调用RegSetValueExA将C:\Temp\cc.exe写进入，设为开启自启动

参数为‘e’跳转到loc\_4011D4时，它让程序休眠186A0h毫秒，也就是100秒

5、在这个恶意代码中有什么本地特征吗？

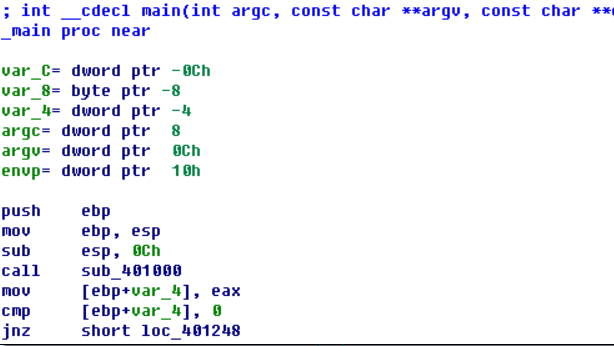
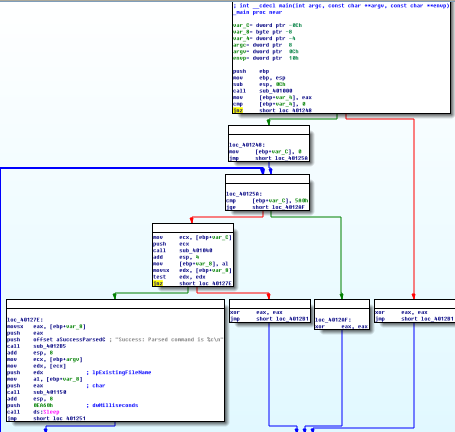
通过以上的分析可以知道该恶意代码的本地特征创建了目录C:\Temp，并在该目录下创建了cc.exe程序，然后修改了注册表将C:\Temp\cc.exe设为了开机自启动

6、这个恶意代码的目的是什么？

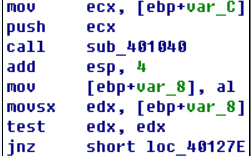
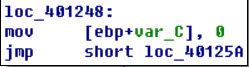
由以上分析可知，该恶意代码先进行一个联网检测，如果未联网则停止运行，如果已联网则会下载HTML文件，并解析。根据HTML文件注释的第一个字母“a,b,c,d,e”进行在C:\Temp路径下的自我复制和开机自启。

(四)Lab06-04.exe

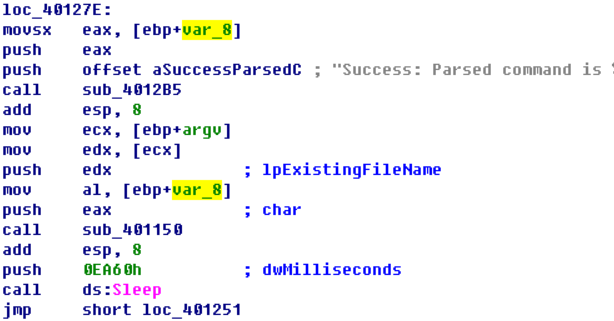
1、在实验6-3和6-4的main函数中调用之间的区别是什么？



首先在main函数先调用了sub\_401000函数，和之前的实验一样，这个函数就是检测是否联网如果是联网状态则跳转到loc\_401248处。来到loc\_401248处，发现这个代码块是比之前的实验多的，它将var\_C偏移地址处的内容赋值0，然后无条件跳转到loc\_40125A。接下来的代码调用了sub\_401040函数，和之前一样就是下载HTML文件并返回注释正文的首地址，如果返回首地址成功则跳转到loc\_40127E

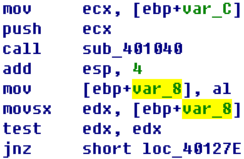
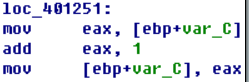


来到loc\_40127E可以看到和之前一样，就是打印出注释正文的的第一个字符然后调用sub\_401150函数，最后休眠60s，然后无条件跳转到loc\_401251处。

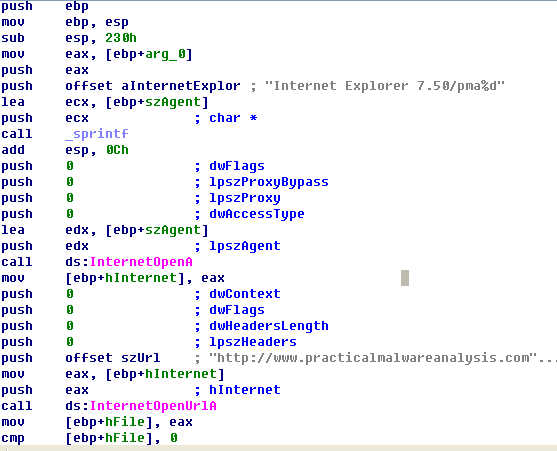


而sub\_401150函数和之前的sub\_401130函数一样，就是一个swich语句根据HTML文件注释正文的第一个字符实现程序自我复制和开机自启等功能。

来到loc\_401251处，发现它就是对变量var\_C进行了加1操作，类似于c代码的自增，然后就又回到了loc\_40125A处再接着与5A0h做计较，很明显这是一个循环结构，一直到变量大于等于5A0h则跳出循环结束程序。



1. ****什么新的代码结构已经被添加到main中？****  
   由以上分析可知，一个循环结构被添加到了main中  
   ****3、这个实验的解析HTML的函数和前面实验中的那些有什么区别？****  
   看到在调用解析函数之前先把表示循环次数的变量压入栈作为了sub\_401040函数的参数



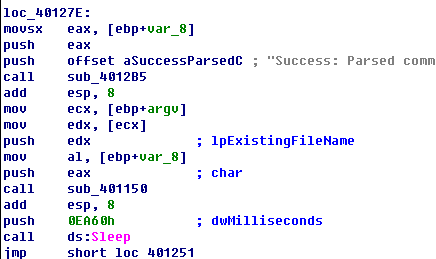
进入函数sub\_401040中，可以看到相比之前字符串Internet Explorer 7.50/pma%d发生了变化，多了一个%d型的参数，对应变量就是arg\_0也就是函数调用之前传入的参数，即循环的次数。接下来还调用了一个sprintf函数，对传入的字符串进行了格式化，然后传给了InternetOpenA函数。

4、这个程序会运行多久？

主要是找到程序中的sleep，看程序睡眠的时间

首先在函数sub\_401150调用完后会有一个0EA60h毫秒也就是60s的睡眠，而这是循环一次的睡眠时间，前面已经分析，一共要循环5A0h

也就是1440次。所以一次循环一分钟，整个程序运行完毕一共要1440分钟也就是24小时。



而在sun\_401150函数中，如果swich结构的参数是e的话，同样会使程序休眠186A0h毫秒也就是100s，所以程序运行一次至少要24小时。

****5、在这个恶意代码中有什么新的基于网络的迹象吗？****

新的网络迹象就是增加了一个计数器，记录循环的次数，使我们可以知道程序运行的时间。

6、这个恶意代码的目的是什么？

首先检测网络连接，如果没有网络连接则停止运行；

如果网络已连接则下载HTML网页，该网页包含了注释，解析该网页，然后根据注释正文的第一个字符做出swich语句的跳转操作：

参数为‘a’时创建C:\Temp的目录；

参数为‘b’时，进行了程序的自我复制；

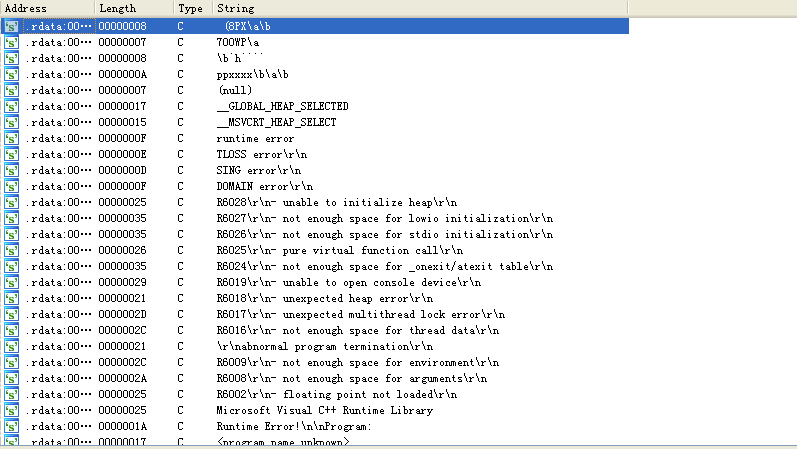
参数为‘c’时，删除了C:\Temp\cc.exe；

参数为‘d’时，将程序设为开机自启动；

参数为‘e’时，程序休眠100秒。

并且将检测到网络连接后的操作循环运行1440次。

Yara规则的编写



查看四个程序的字符串列表之后，我们编写了下列的yara规则

import "pe"

rule Message {

strings:

$ErrorM = "Error"

$SuccessM = "Success"

$Internet = "Internet"

condition:

$ErrorM or $SuccessM or $Internet

}

rule MalURLRequest {

strings:

$Mal = "practicalmalwareanalysis"

$Http = "http"

condition:

$Mal and $Http

}

rule EXE {

strings:

$exe = /[a-zA-Z0-9\_]+.exe/

condition:

$exe

}

rule Regedit {

strings:

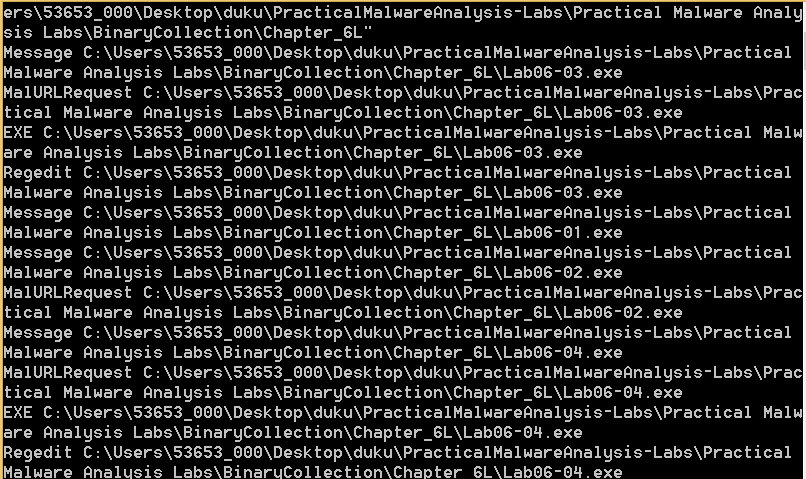
$run = "Software\\Microsoft\\Windows\\CurrentVersion\\Run"

condition:

$run

}

扫描结果如下：



Ida python 的编写：

用ida pro打开lab06-01.exe的函数列表，之后设计如下的ida python脚本

from idaapi import \*

# 设置颜色

def judgeAduit(addr):

not safe function handler

MakeComm(addr,"### AUDIT HERE ###")

SetColor(addr,CIC\_ITEM,0x0000ff) #set backgroud to red

pass

# 函数标识

def flagCalls(danger\_funcs):

not safe function finder

count = 0

for func in danger\_funcs:

faddr = LocByName( func )

if faddr != BADADDR:

# Grab the cross-references to this address

cross\_refs = CodeRefsTo( faddr, 0 )

for addr in cross\_refs:

count += 1

Message("%s[%d] calls 0x%08x\n"%(func,count,addr))

judgeAduit(addr)

if \_\_name\_\_ == '\_\_main\_\_':

handle all not safe functions

# 列表存储需要识别的函数

danger\_funcs = ["\_fflush","\_wctomb","\_\_lseek","\_fclose","RtlUnwind"]

flagCalls(danger\_funcs)

**同理我们设计其他应用程序的ida python 代码**

**Lab06-02.exe**

from idaapi import \*

# 设置颜色

def judgeAduit(addr):

not safe function handler

MakeComm(addr,"### AUDIT HERE ###")

SetColor(addr,CIC\_ITEM,0x0000ff) #set backgroud to red

Pass

# 函数标识

def flagCalls(danger\_funcs):

not safe function finder

count = 0

for func in danger\_funcs:

faddr = LocByName( func )

if faddr != BADADDR:

# Grab the cross-references to this address

cross\_refs = CodeRefsTo( faddr, 0 )

for addr in cross\_refs:

count += 1

Message("%s[%d] calls 0x%08x\n"%(func,count,addr))

judgeAduit(addr)

if \_\_name\_\_ == '\_\_main\_\_':

handle all not safe functions

print "-------------------------------"

# 列表存储需要识别的函数

danger\_funcs = ["\_\_abnormal\_termination","\_\_isatty","\_wctomb","\_\_fcloseall","RtlUnwind"]

flagCalls(danger\_funcs)

**Lab06-03.exe**

from idaapi import \*

# 设置颜色

def judgeAduit(addr):

not safe function handler

MakeComm(addr,"### AUDIT HERE ###")

SetColor(addr,CIC\_ITEM,0x0000ff) #set backgroud to red

pass

# 函数标识

def flagCalls(danger\_funcs):

not safe function finder

count = 0

for func in danger\_funcs:

faddr = LocByName( func )

if faddr != BADADDR:

# Grab the cross-references to this address

cross\_refs = CodeRefsTo( faddr, 0 )

for addr in cross\_refs:

count += 1

Message("%s[%d] calls 0x%08x\n"%(func,count,addr))

judgeAduit(addr)

if \_\_name\_\_ == '\_\_main\_\_':

handle all not safe functions

# 列表存储需要识别的函数

danger\_funcs = ["\_\_XcptFilter","\_\_initterm","\_\_\_crtGetEnvironmentStringsA","\_\_abnormal\_termination","\_\_get\_osfhandle"]

flagCalls(danger\_funcs)

**Lab06-04.exe**

from idaapi import \*

# 设置颜色

def judgeAduit(addr):

not safe function handler

MakeComm(addr,"### AUDIT HERE ###")

SetColor(addr,CIC\_ITEM,0x0000ff) #set backgroud to red

pass

# 函数标识

def flagCalls(danger\_funcs):

not safe function finder

count = 0

for func in danger\_funcs:

faddr = LocByName( func )

if faddr != BADADDR:

# Grab the cross-references to this address

cross\_refs = CodeRefsTo( faddr, 0 )

for addr in cross\_refs:

count += 1

Message("%s[%d] calls 0x%08x\n"%(func,count,addr))

judgeAduit(addr)

if \_\_name\_\_ == '\_\_main\_\_':

handle all not safe functions

# 列表存储需要识别的函数

danger\_funcs = ["\_\_get\_osfhandle","\_\_\_crtLCMapStringA","\_\_alloca\_probe","\_\_abnormal\_termination","\_\_\_crtGetEnvironmentStringsA"]

flagCalls(danger\_funcs)

1. **实验结论及心得体会**

这一次的实验是恶意代码与防治分析的Lab6实验，对理论课上讲的IDA Python编写技术有了一定的了解，也对IDA Pro的使用比如说交叉引用、语句跳转、反汇编分析等更加的熟练。

在本次实验中，也对所检测程序编写了相应的yara规则，对于yara规则的编写也更加的熟练。

这次实验当中，我们对于ida python的编写有了更加熟练的掌握，了解到我们更多的通过函数列表了解进程，从而了解函数的运行方式，最终得到一个比较好的检验方式，给我后续的恶意代码分析提供了新的思路。