

**《恶意代码分析与防治技术》课程实验报告**

**实验六**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

班 级 信息安全1班

《恶意代码分析与防治技术》可成Lab6实验报告

[一、 实验目的 3](#_Toc19844)

[二、 实验原理 3](#_Toc12698)

[三、 实验过程 3](#_Toc11319)

[（一） Lab6 3](#_Toc12338)

[1. Lab6-1 3](#_Toc20348)

[2. Lab6-2 4](#_Toc7135)

[3. Lab6-3 10](#_Toc29730)

[4. Lab6-4 15](#_Toc16003)

[（二） Yara 19](#_Toc910)

[（三） IDA Python 20](#_Toc11991)

[四、 实验结论及心得体会 21](#_Toc18672)

[（一） 实验结论 21](#_Toc17719)

[（二） 心得体会 22](#_Toc21853)

1. **实验目的**

本次实验的主要目的是分析和理解一系列恶意代码的实验样本，包括Lab6-1、Lab6-2、Lab6-3、Lab6-4，以及编写相应的Yara规则来检测这些恶意代码的存在。通过分析这些实验样本，实验者可以了解恶意代码的行为、结构和特征，以及如何使用Yara规则来检测类似的恶意代码。

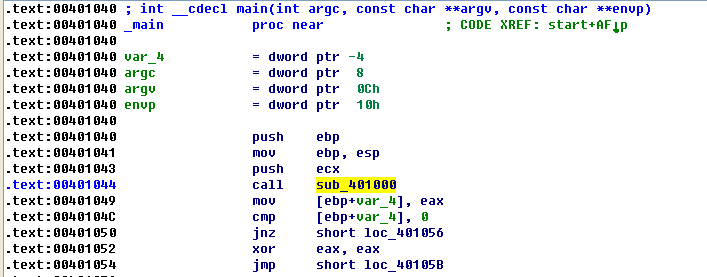
1. **实验原理**

实验基于静态分析和反汇编技术，通过使用IDA Pro等工具来分析恶意代码的汇编指令和逻辑结构。

1. **实验过程**
2. **Lab6**
3. **Lab6-1**
4. **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?**

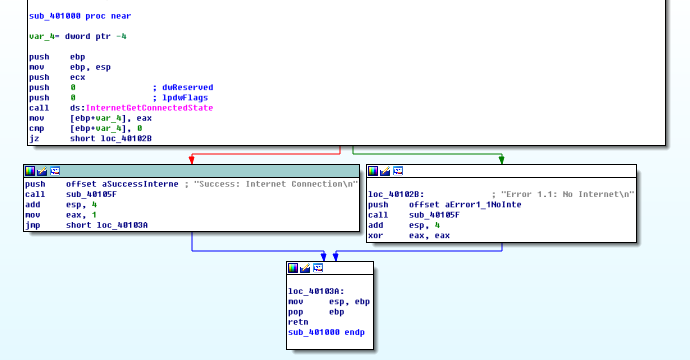
使用IDA Pro对Lab6-1进行分析。

main函数位于0x00401040，调用了位于0x00401000处的函数。



图形窗口和反汇编窗口查看sub\_401000。

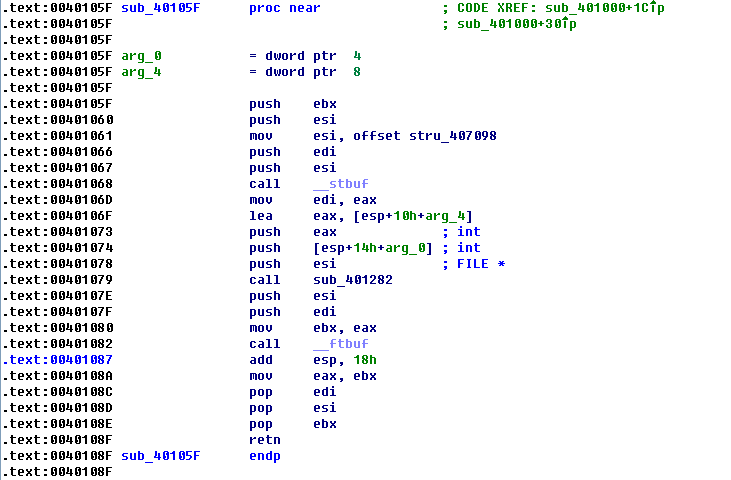




根据对InternetGetConnectedState函数调用的结果，出现了两条不同的代码路径。当存在一个可用的Internet连接时，InternetGetConnectedState函数返回1,否则返回0。如果存在,不跳转继续进入0x00401017所示的分支，不存在跳转loc\_0040102B所示的分支。

综上所述，main函数调用的唯一子过程中发现的主要代码结构是位于0x00401000处的if语句。

1. **位于0x40105F的子过程是什么?**



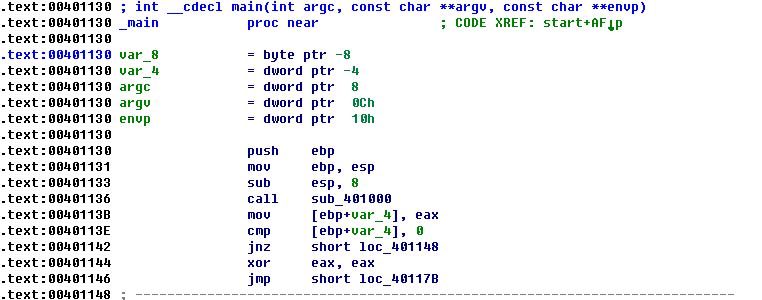
它顺序调用了三个函数，还有一个数据offset stru\_407098,并且在该偏移量中还存在FILE的字符，然后将这个数据存在了esi中，并且在这三个函数的调用中均用到了esi指向的FILE。找到调用子例程前被压到栈上的参数。一共有两处，都有一个格式化字符串被压栈（“Success: Internet Connection\n”，“Error 1.1:No Internet\n”），并且字符串结尾是\n这个换行符。推断该函数是printf。

1. **这个程序的目的是什么?**

该程序检查是否有一个可用的Internet连接。如果有就打印“Success:Internet Connection”,否则，打印“Error 1.1:No Internet”。恶意代码在连接Internet之前，可以使用该程序来检查是否存在一个连接。

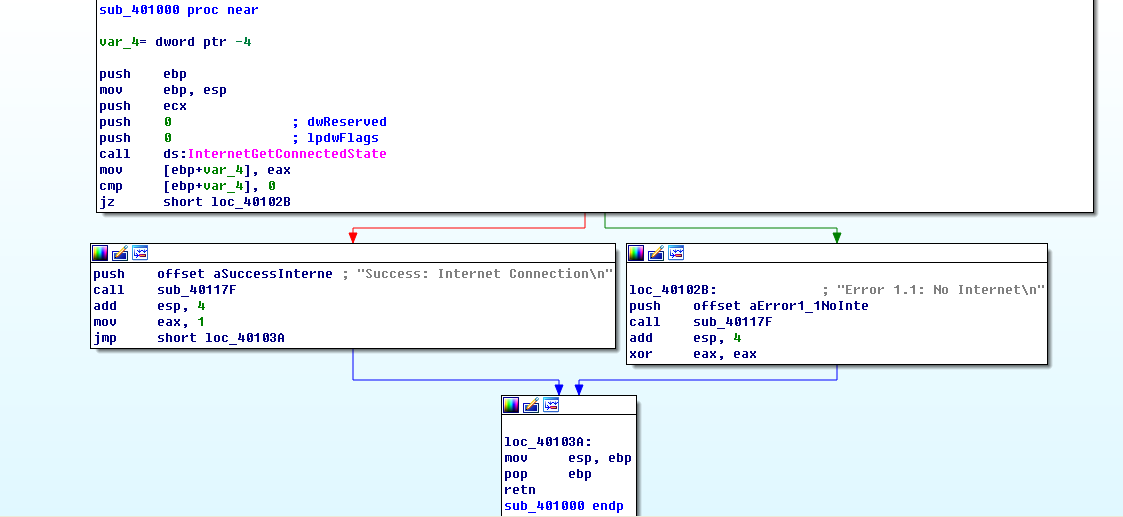
1. **Lab6-2**
2. **main函数调用的第一个子过程执行了什么操作?**

main函数位于0x00401130，调用了位于0x00401000处的函数子过程。



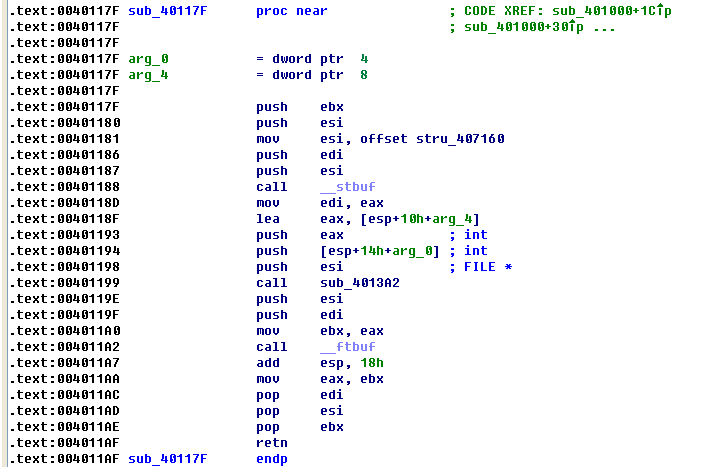
图形窗口和反汇编窗口查看sub\_401000。





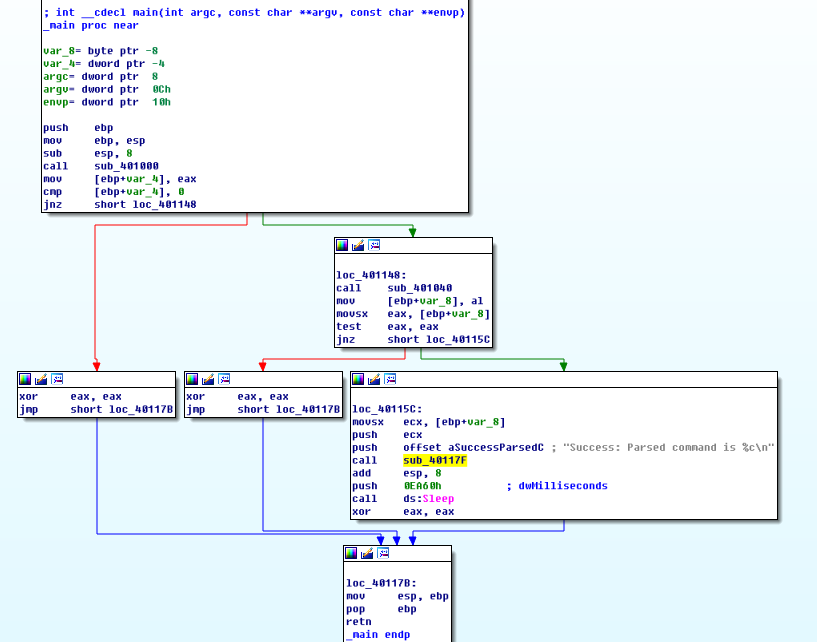
和Lab6-1的0x401000一样，位于0x401000的第一个子例程是一个if语句，检查是否存在可用的Internet连接。如果是联网状态则打印字符串“Success: Internet Connection”，如果没有联网，则打印字符串“Error 1.1: No Internet”。

1. **位于0x40117F的子过程是什么?**



查看main函数

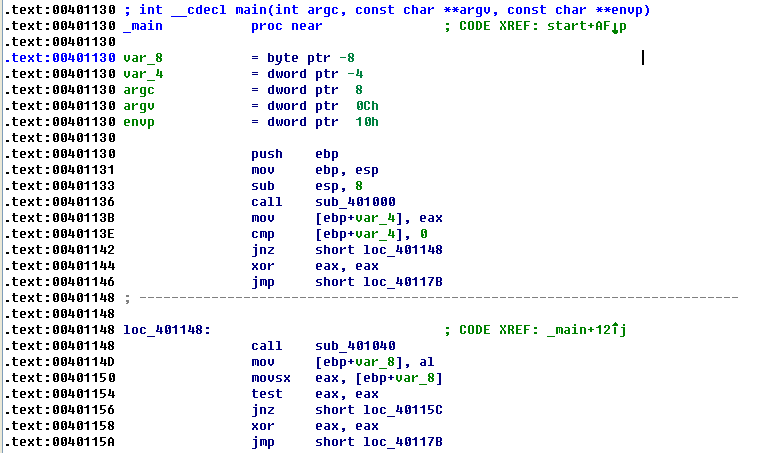




sub\_40117F被调用前，它顺序调用了三个函数，还有一个数据offset stru\_407160,并且在该偏移量中还存在FILE的字符，然后将这个数据存在了esi中，并且在这三个函数的调用中均用到了esi指向的FILE。有两个参数被压入了栈，其中之一是一个格式化字符串“Success:Parsed command is ?”另一个是从前面对loc\_401148（sub\_401040）的调用返回的字符。%c和%d这样的格式化字符表示是一个格式化字符串，推断在0x40117F处的子例程是printf，会打印该字符串，并把其中的替换成另一个被压入栈的参数。

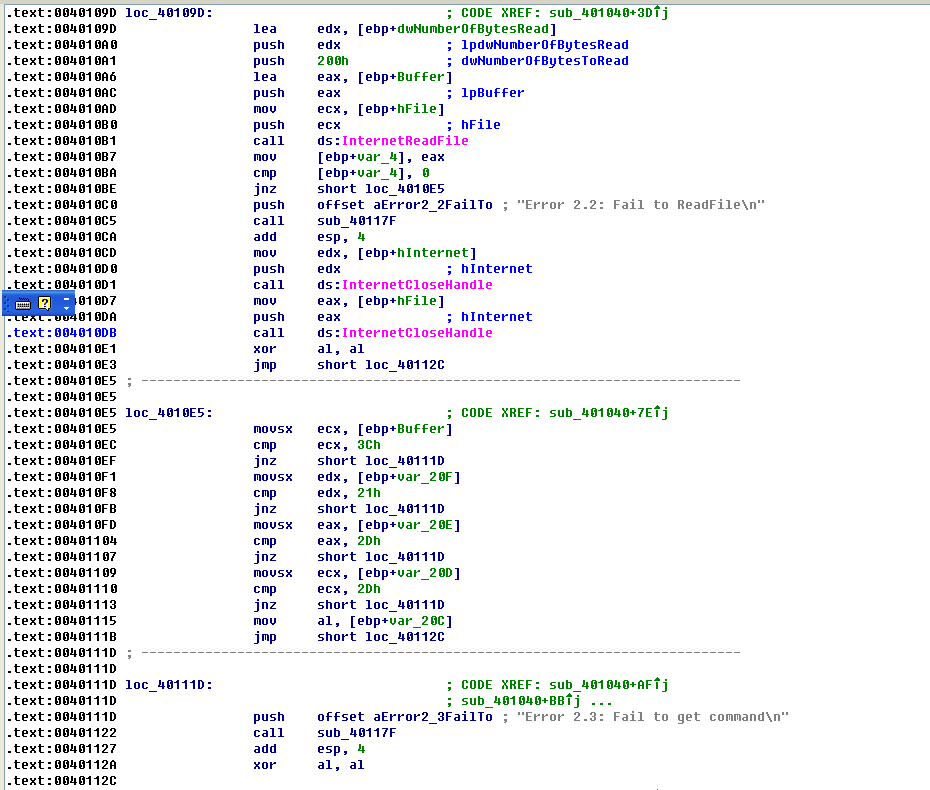
1. **被main函数调用的第二个子过程做了什么?**

main调用的第二个函数位于0x401040

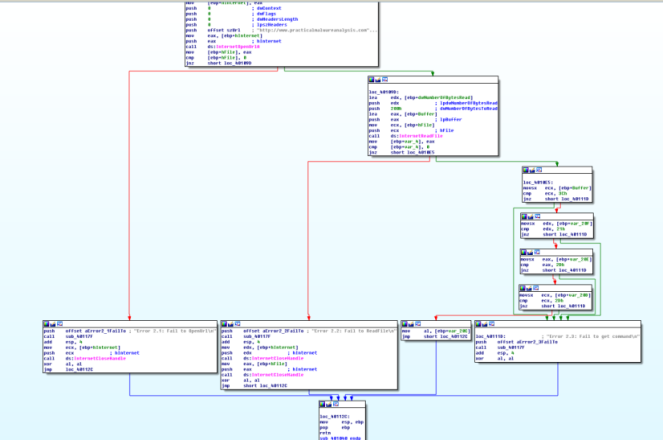


查看loc\_401148，调用了sub\_401040，查看sub\_401040。

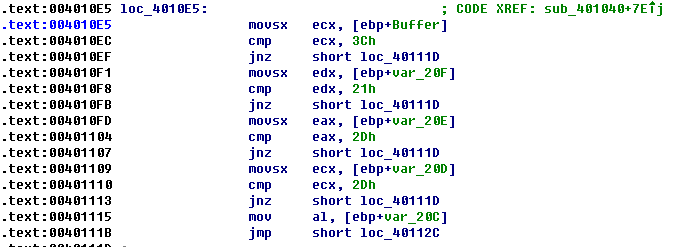








该函数存在网络相关的函数调用。首先调用了InternetopenA，用于初始化对WinINet库的使用，并设置用于HTTP通信的User-Agent字段（Internet Explorer 7.5/pma被压入栈）。接下来调用InternetopenUrlA，从http://www.practicalmalwareanalysis.com下载HTML网页。如果下载不成功，则打印字符串“Error 2.1: Fail to OpenUrl”并调用InternetCloseHandle函数关闭连接，如果成功下载，则跳转到loc\_40109D地址处，然后调用InternetReadFile读取下载的HTML文件。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。

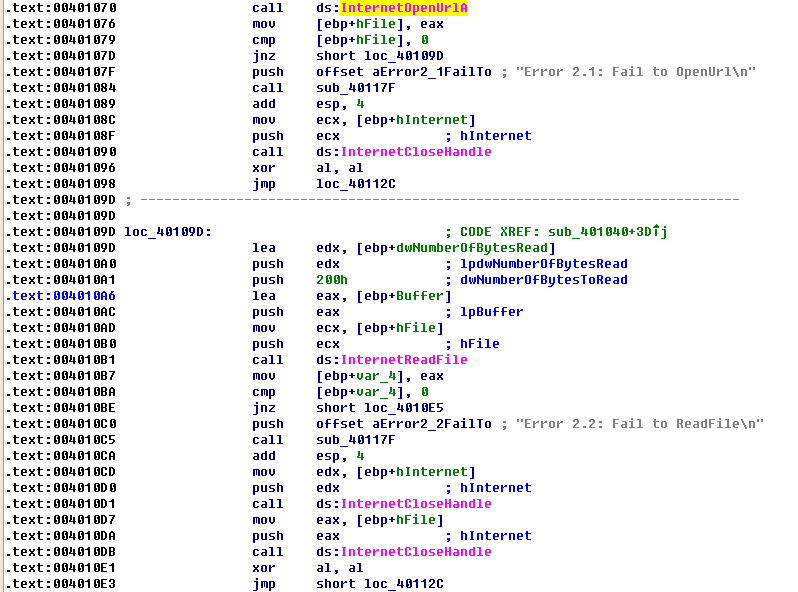


如果读取成功则跳转loc\_4010E5地址处执行代码解析网页。0x004010E5处的cmp指令用来检查第一个字符是否等于0x3C,对应的ASCII字符是<，后面的cmp指令中的21h、2Dh对应的ASCII字符是！和-，连起来得到字符串<！--，是HTML中注释的开始部分。如果不是<!--这四个字符中任何一个比较失败，则打印字符串“Error 2.3: Fail to get command”。

综上所述，此处会下载位于http://www.practicalmalwareanalysis.com/cc.htm

的网页，并从页面开始处解析HTML注释。

1. **在这个子过程中使用了什么类型的代码结构?**



InternetopenUrlA的返回结果被赋给了hFile，cmp [ebp+hFile]，与0进行比较。如果为0，该函数会返回，否则，hFile变量会被传给下一个函数，也就是InternetReadFile. hFile变量是一个用于访问URL的句柄——一种访问已经打开的东西的途径。

InternetReadFile用于从InternetopenUrlA打开的网页中读取内容。第二个参数是被标记为Buffer的一个保存数据的数组，如0x004010A6处（lea eax,[ebp+Buffer]）所示，读取最多0x200字节的数据，如0x004010A1处（push 200h ;dwNumberOfBytesToRead）NumberOfBytesToRead参数的值，是用来读取一个HTML网页的。

接下来调用InternetReadFile,在0x004010BA处的代码（cmp [ebp+var\_4],0）检查其返回值是否为0。如果为0该函数关闭句柄并终止，否则代码会将Buffer逐一地每次与一个字符进行比较。每次取出内容到一个寄存器时，对Buffer的索引值都会增加1,然后取出来再比较。

综上所述，该子例程是一个多层的if分支结构，调用InternetReadFile,将返回的数据填充到一个字符数组中，然后每次一个字节地对这个数组进行比较，以解析一个HTML注释。

1. **在这个程序中有任何基于网络的指示吗?**



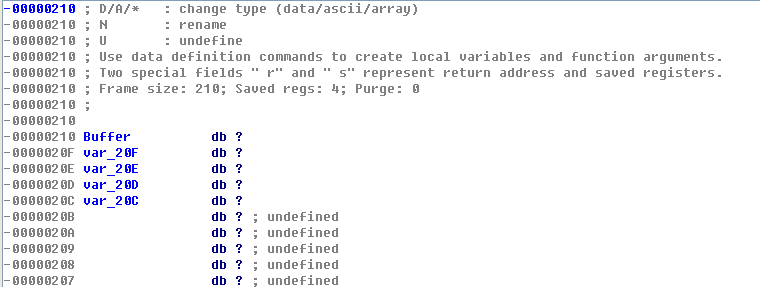
存在一些网址，可以作为指示。该恶意代码使用Internet Explorer 7.5/pma作为HTTP的User-Agent字段，并从http://www.practicalmalwareanalysis.com/cc.htm下载了网页。

1. **这个恶意代码的目的是什么?**

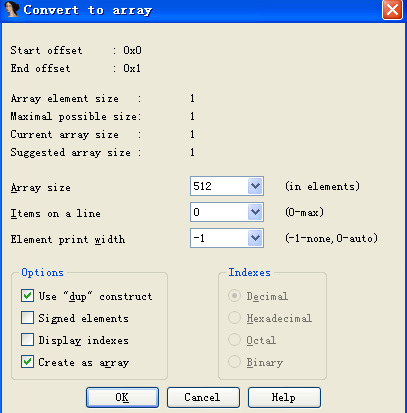
0x004010F1处，Buffer+1被赋值给edx后，再与0x21比较。假设Buffer是通过InternetReadFile下载到网页的字符数组，Buffer指向了网页的起始处，四条cmp指令的用处是检查网页最开始处的一条HTML注释，如果所有的比较都成功了，该网页就是由一条注释开头的，0x00401115处的代码会被执行。

填充sub\_401040的栈，显示一个512字节的数组，使Buffer在整个函数中被正确标记。

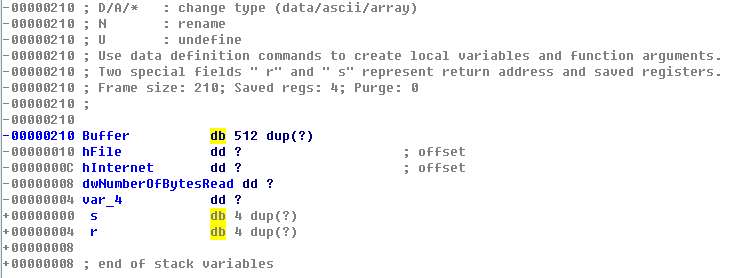
在sub\_401040函数中ctrl+K，



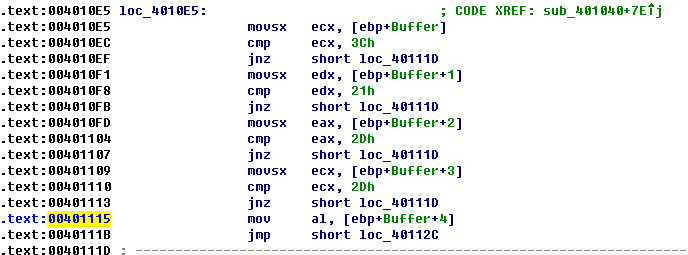
Buffer的第一个字节处右击，定义一个512字节的数组，每个元素1字节宽。



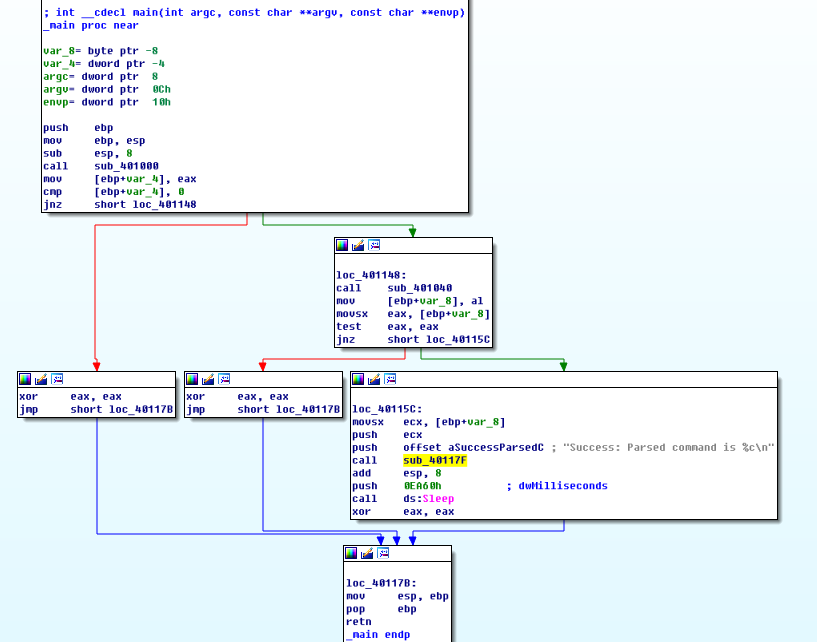
显示了正确栈看起来的样子。



看到0x00401115处的指令显示为[ebp+Buffer+4]。如果前四个字节(Buffer[0]-Buffer[3])与<!--匹配上了，第5个字符就会被移到AL中并从这个函数返回。



回到main函数，sub\_401040函数返回后,如果是一个非零的值，main函数就会打印“Success: Parsed command is X”,X是从HTML注释中解析出的字符。接下来，在0x00401173处调用了Sleep函数。Sleep函数只有一个参数，用于说明要休眠的毫秒数。这里将0xEA60作为参数压入栈，就相当于要休眠一分钟(60000毫秒)。

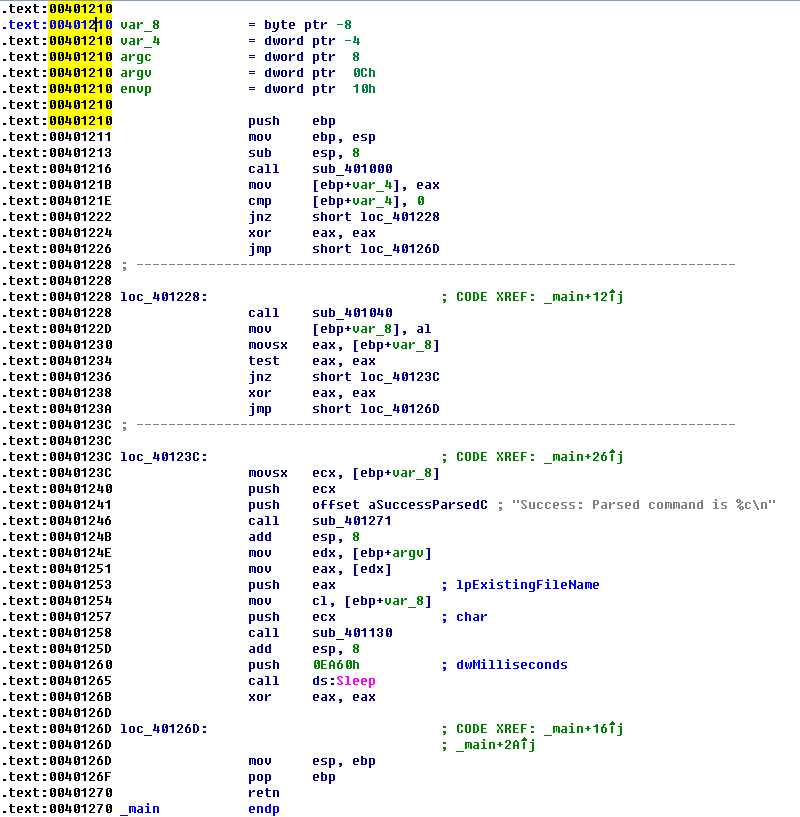


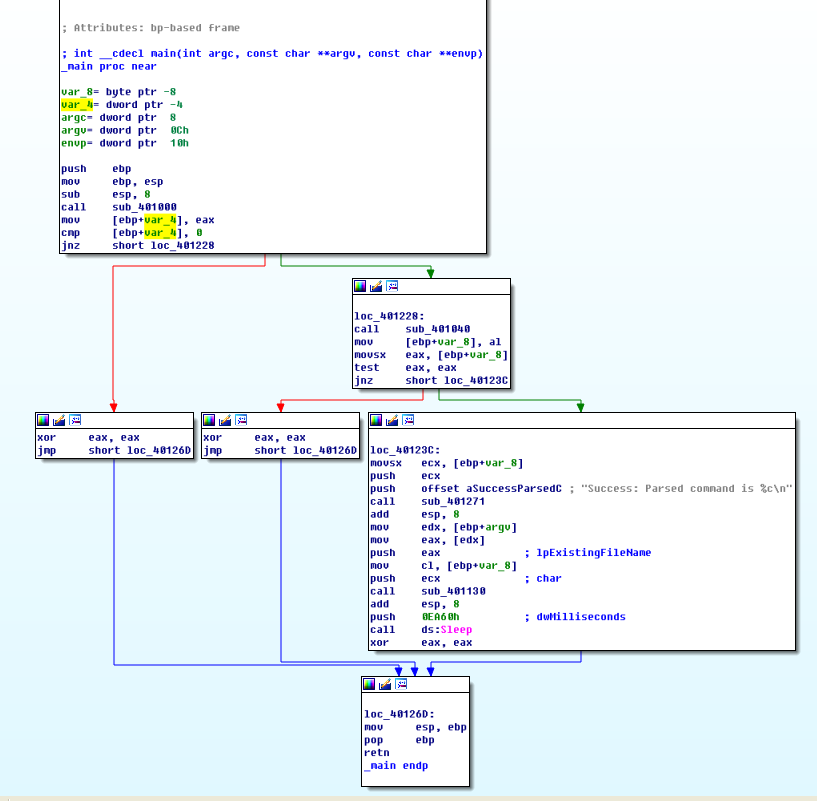


综上所述，程序首先判断是否存在一个可用的Internet连接，如果不存在就终止运行。否则，程序使用一个独特的用户代理尝试下载一个网页。该网页包含了一段由<!--开始的HTML注释，程序解析其后的那个字符并输出到屏幕，输出格式是"Success: Parsed command is X",其中X就是从该HTML注释中解析出来的字符。如果解析成功，程序会休眠1分钟，然后终止运行。

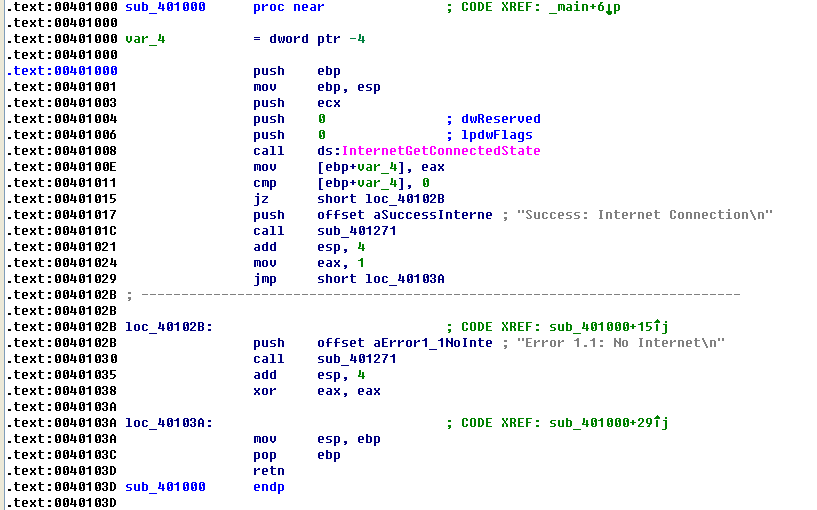
1. **Lab6-3**
2. **比较在main函数与实验6-2的main函数的调用。从main中调用的新的函数是什么?**

查看main函数，

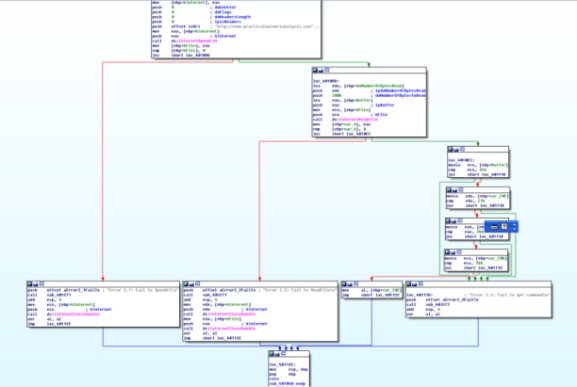




sub\_401000函数：



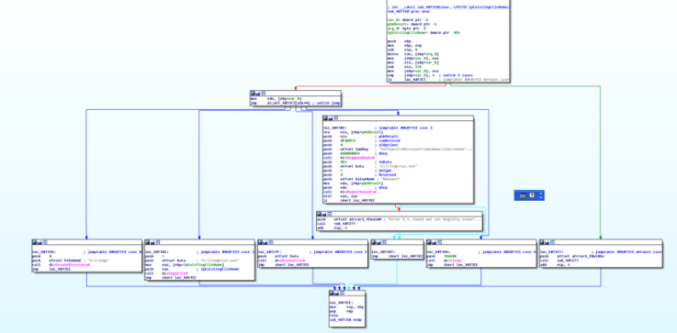
sub\_401040函数：



sub\_401130函数：







在0x401000和0x401040处的函数与Lab 6-2的一样。在0x401271处的是printf。0x401130处的函数则是本实验中新出现的。

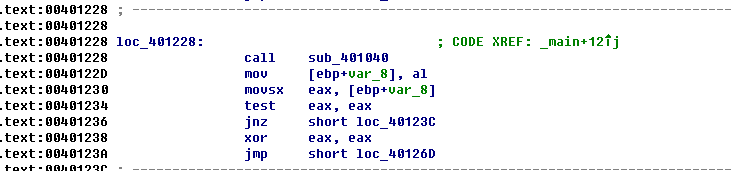
1. **这个新的函数使用的参数是什么?**





在调用前，argv和var\_8被压入了栈中。在调用sub\_401030时，先后push了eax和ecx，根据函数栈帧结构，该函数有两个参数。argv的地址的内容存放到了eax中，argv的地址就是argv[0]的首地址，argv[0]在main函数中作为一个指针，指向程序的路径及名称。eax中存放的就是程序的名称即Lab06-03.exe的字符串。

ecx将var\_8的内容存放在了ecx里。var\_8在0x0040122D处被用AL设置。EAX是上一个函数调用的返回结果，而AL包含在EAX中，而上一个函数调用是0x401040(下载网页并解析HTML注释),因此，传递给0x401130的var\_8包含了从HTML注释中解析出来的指令字符。

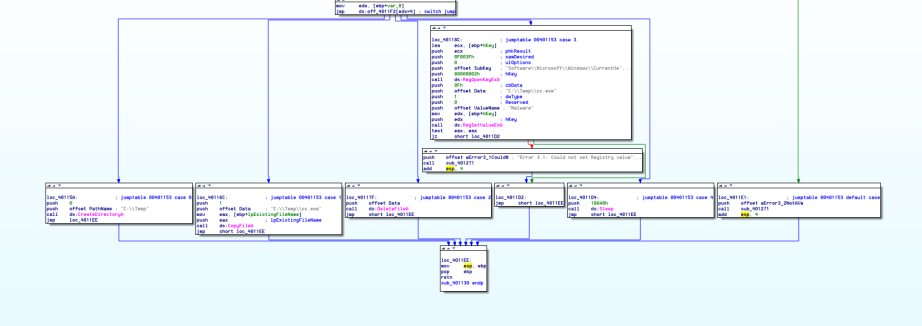


这个新的函数有两个参数，第一个是从HTML注释中解析来的指令字符，第二个参数是标准main函数的参数中的argv[0],也就是程序名本身。

1. **这个函数包含的主要代码结构是什么？**







Internet获取并解析得到的指令字符。这个指令字符被赋给var\_8,最后在0x0040113D处加载到ECX中。下一条指令是从ECX中减掉0x61(也就是ASCII中的字母a)。因此，如果arg\_0等于a,这条指令被执行后，ECX变为0。

在0x00401146处将ECX与4进行比较，以判断该指令字符是否a、b、c、d或e。如果是其他结果，就会引发ja指令跳转离开这段代码；否则，可以看到这个指令字符被用作跳转表的索引。0x00401153处，EDX被乘以4,这是因为跳转表是一组指向不同执行路径的内存地址，而每个地址的大小是4字节，0x00401153处的跳转表有五条记录。根据参数,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。

综上所述，新的函数包含了一条switch语句和一个跳转表。

1. **这个函数能够做什么?**

参数为‘a’跳转到loc\_40115A时，创建了C:\Temp的目录；

参数为‘b’跳转到loc\_40116C时，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。lpExistingFileName是 sub\_401130的第二个参数，根据栈结构的特点，在sub\_401130函数调用前先压入的argv也就是程序本身的名称，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。

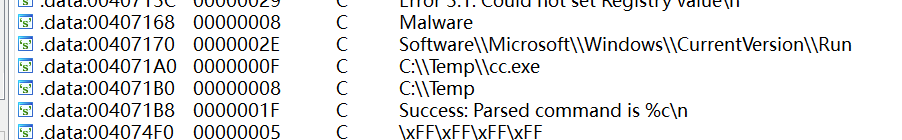
参数为‘c’跳转到loc\_40117F处时，删除了C:\Temp\cc.exe；

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，程序休眠186A0h毫秒，也就是100秒。

综上所述，新的函数可以打印出错信息、删除一个文件、创建一个文件夹、设置一个注册表项的值、复制一个文件，或者休眠100秒等。

1. **在这个恶意代码中有什么本地特征吗?**



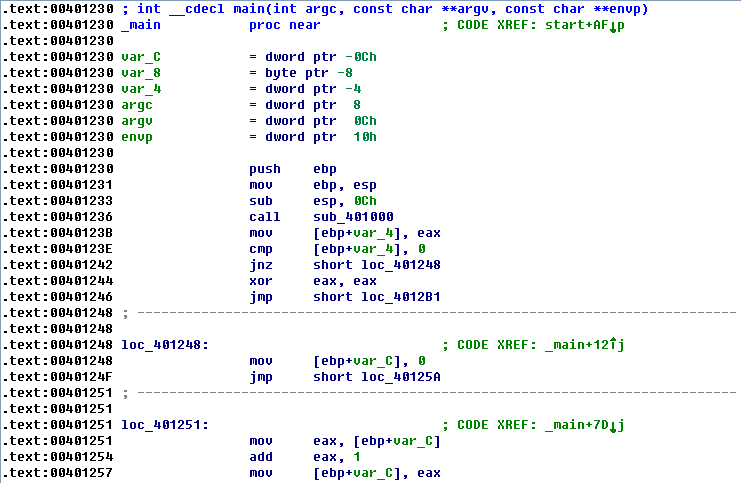
在字符串窗口中可以看到本地文件路径，并且通过之前的分析可以知道该恶意代码的本地特征创建了目录C:\Temp，并在该目录下创建了cc.exe程序，然后修改了注册表将C:\Temp\cc.exe设为了开机自启动。

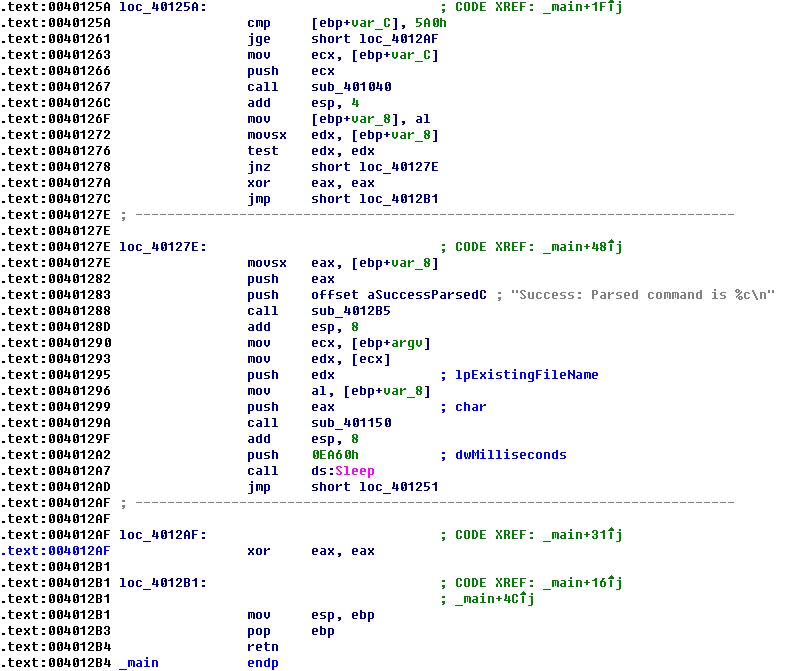
综上所述，注册表键Software\Microsoft\Windows\CurrentVersion\Run\Malware和文件路径C:\Temp\cc.exe都可以当作本地特征。

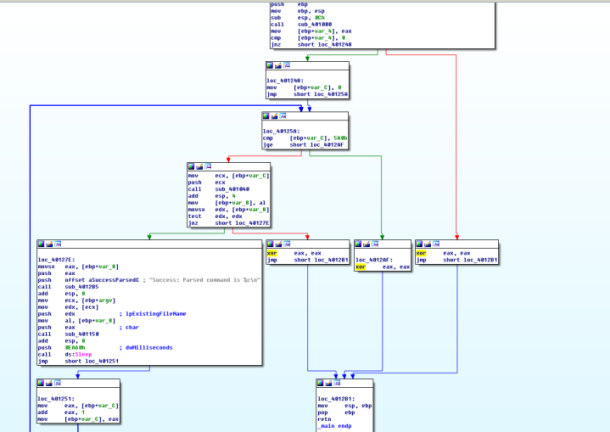
1. **这个恶意代码的目的是什么?**

该程序先检查是否存在有效的Internet连接。如果找不到，程序直接终止。否则，该程序会尝试下载一个网页，该网页包含了一段以<!--开头的HTML注释。该注释的第一个字符被用于switch语句来决定程序在本地系统运行的下一步行为，包括是否删除一个文件、创建一个目录、设置一个注册表run键、复制一个文件或者休眠100秒。

1. **Lab6-4**
2. **在实验6-3和6-4的main函数中的调用之间的区别是什么?**

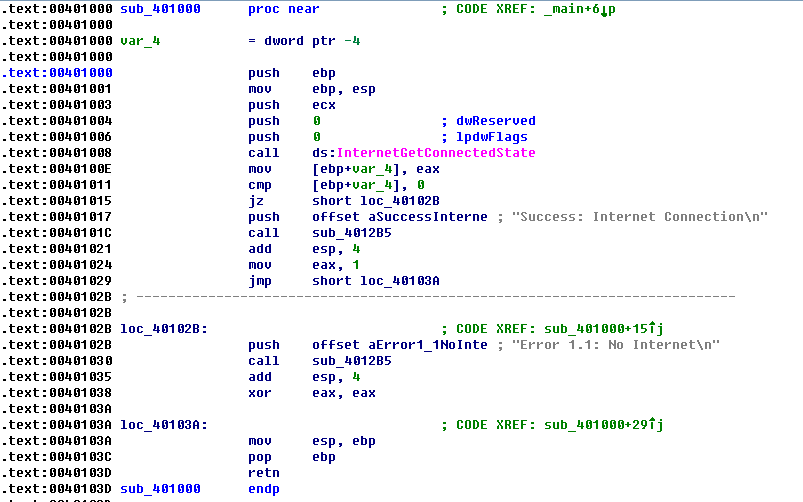






看到main中调用了，sub\_401000、sub\_401040、sub\_4012B5、sub\_401150。

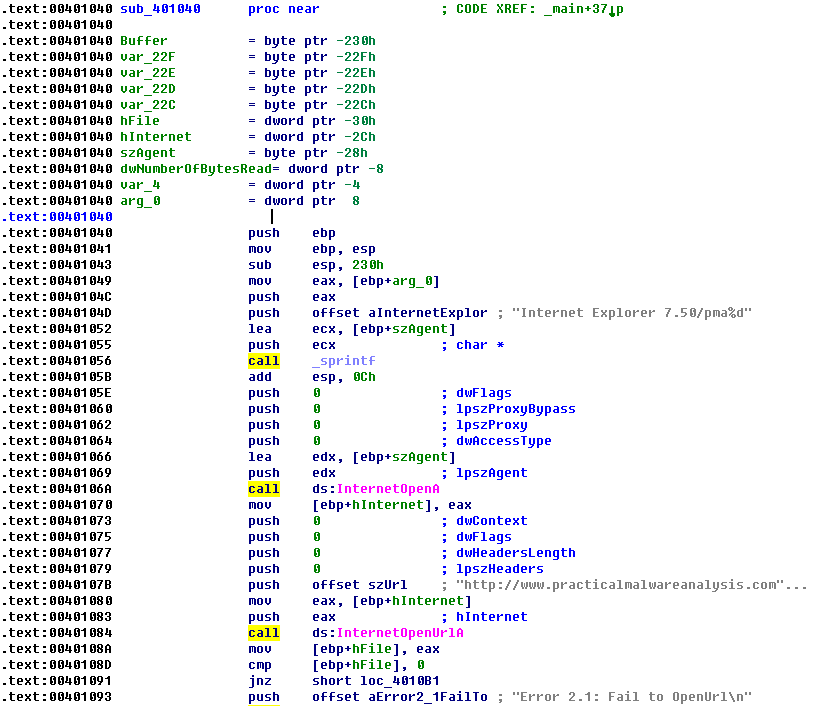
sub\_401000：



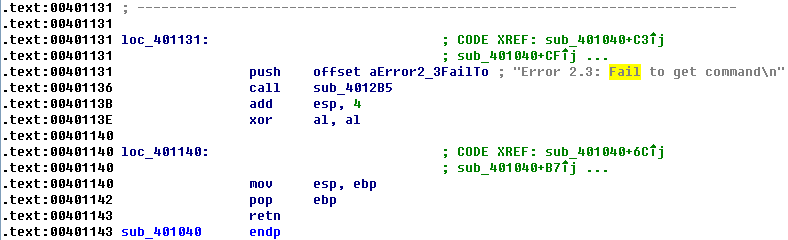
分析得知0x401000处的函数是检查Internet连接，如果是联网状态则跳转到loc\_401248处。

loc\_401248处将var\_C偏移地址处的内容赋值0，然后无条件跳转到loc\_40125A。在loc\_40125A处，与5A0作比较，jge指令是如果大于等于则跳转，而之前的变量var\_C被赋值0很显然不能跳转，继续向下执行代码。

sub\_401040：





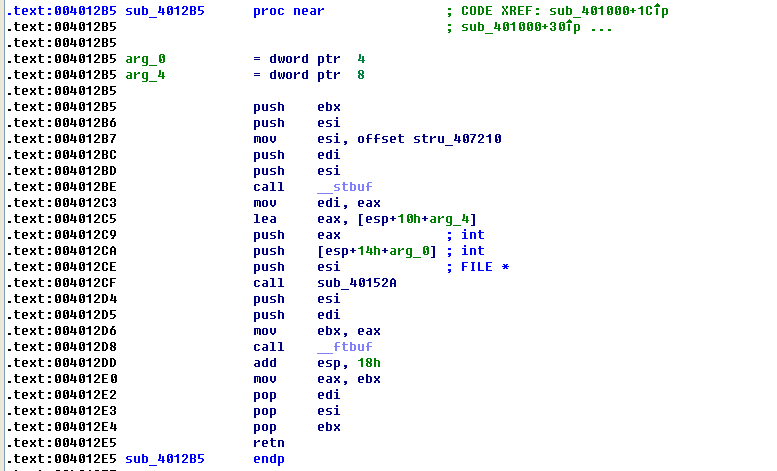


arg\_0是唯一的参数，只有main函数调用了sub\_401040，推断arg\_0始终是从main函数中传入的计数器(var\_C)。在0x0040104D处，arg\_0与一个格式化字符串及一个目标地址一起被压入栈。sub\_401040下载HTML文件并返回注释正文的首地址，如果返回首地址成功则跳转到loc\_40127E。而loc\_40127E就是打印出注释正文的的第一个字符然后调用sub\_401150函数，最后休眠60s，然后无条件跳转到loc\_401251处。然后看到sprintf被调用，创建了一个字符串，并将其存储在目的缓冲区，也就是被标记为szAgent的局部变量。在0x00401066处，szAgent被传给了InternetopenA，也就是说每次计数器递增，User-Agent也会随之改变。

这个机制可以被管理和监控web服务器的攻击者跟踪恶意代码运行了多长时间。

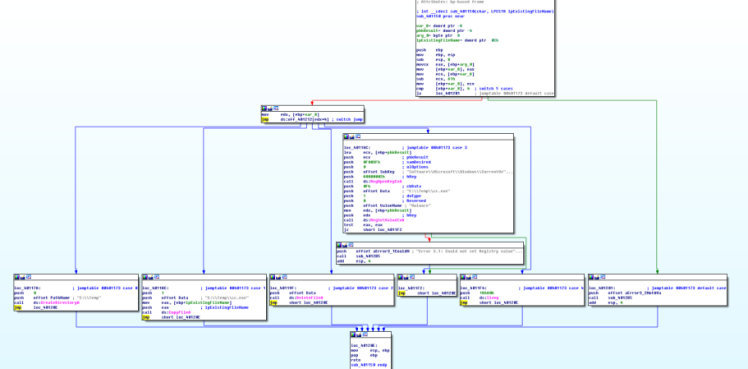
分析得知sub\_401040是解析HTML的方法。

sub\_4012B5：



分析得知，0x004012B5处的函数是printf。

sub\_401150：



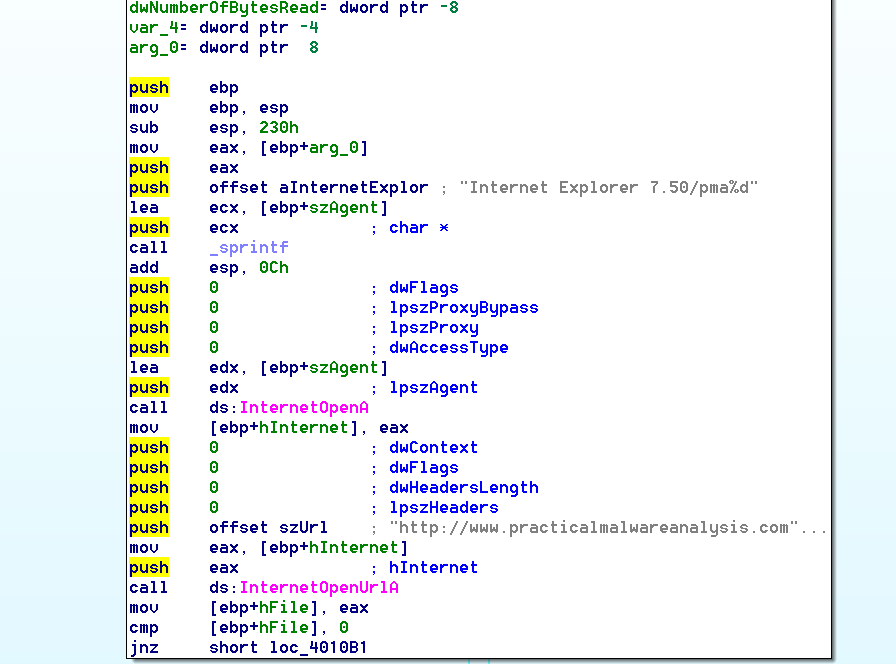
分析得知0x401150处是switch语句。

1. **什么新的代码结构已经被添加到main中?**

在main函数中加了一个for循环语句。

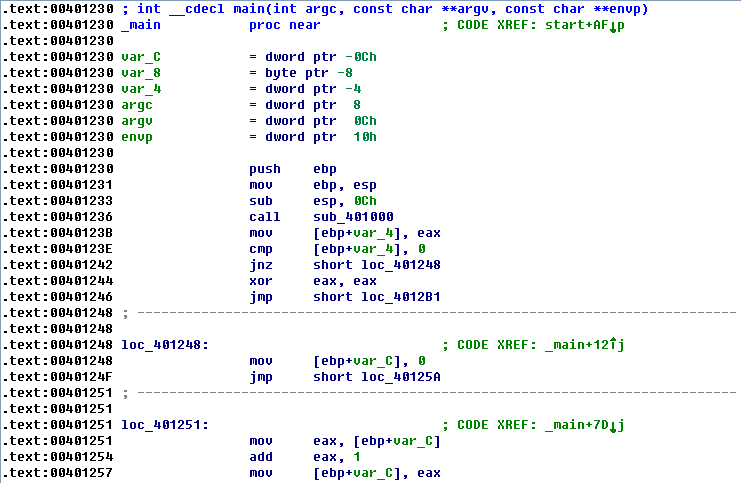
1. **这个实验的解析HTML的函数和前面实验中的那些有什么区别?**

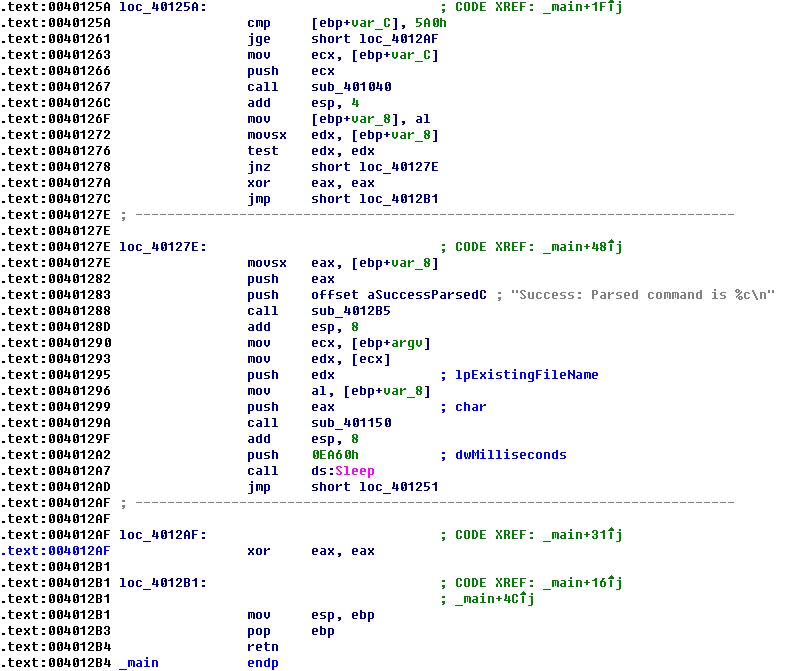
进入函数sub\_401040，可以看到相比之前字符串Internet Explorer 7.50/pma%d发生了变化，多了一个%d型的参数，对应变量就是arg\_0也就是函数调用之前传入的参数，即循环的次数。接下来还调用了一个sprintf函数，对传入的字符串进行了格式化，然后传给了InternetOpenA函数。



综上所述，sub\_401040使用格式化字符串Internet Explorer 7.50/pma%d来调用sprintf函数，从而使用传入的参数创建用于HTTP通信的User-Agent字段。

1. **这个程序会运行多久?(假设它已经连接到互联网。)**





局部变量var\_C用于循环计数。在0x401248处，这个计数器被初始化为0,每次跳回0x401254处递增，在0x40125A处进行检查，每次到了0x4012AD就跳回去递增。这0x4012AD处代码的出现告诉是一个循环结构。如果计数器var\_C大于或等于0x5A0(1440),循环就会终止：否则，在0x401263处的代码会被执行。调用0x401040之前，这里的代码会将var\_C压入栈上，然后循环在执行到0x4012AD之前会休眠1分钟，最后将计数器加1。因此，这个过程会持续1440分钟，也就是24小时。

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。

综上所述该程序一共运行1440分钟(24小时)。

1. **在这个恶意代码中有什么新的基于网络的迹象吗?**

综上所述，该恶意代码使用了一个新的User-Agent。它的形式是Internet Explorer 7.50/pma%d其中%d程序已经运行的分钟数。

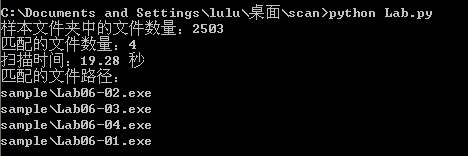
1. **这个恶意代码的目的是什么?**

首先，程序会检查是否有可用的Internet连接。如果找不到，程序就终止运行。否则，程序使用一个独特的User-Agent来下载一个网页，这个User-Agent包含了一个计数器，用于说明程序已经运行了多少分钟。下载下来的网页中包含了以<!--开头的HTML注释代码，这段注释代码中接下来的第一个字符被用于一个switch语句，以决定接下来在本地系统的行为。其中包含了一些硬编码的行为，包括删除一个文件、创建一个目录、设置一个注册表run键、复制一个文件、休眠100秒等。该程序会运行24小时后终止。

1. **Yara**

使用一些特征字符串编写yara规则。

|  |
| --- |
| rule Lab06{ |
| meta: |
| description = "Lab06-01.exe" |
| strings: |
| $s1 = "Success: Internet Connection" fullword ascii |
| $s2 = "Error 1.1: No Internet" fullword ascii |
| $s3 = "InternetGetConnectedState" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 100KB and |
| all of them |
| } |

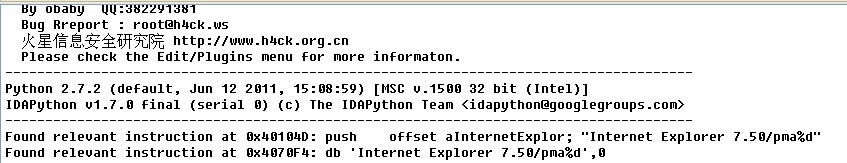


1. **IDA Python**

|  |
| --- |
| import idc |
| import idautils |
|  |
| # 目标注释 |
| target\_string = "Internet Explorer 7.50/pma%d" |
|  |
| # 获取函数的起始和结束地址 |
| start\_address = 0x00401000 |
| end\_address = 0x0040AFC8 |
|  |
| # 初始化一个列表来存储包含特定注释的汇编语句 |
| relevant\_instructions = [] |
|  |

|  |
| --- |
| # 遍历指令，查找包含指定字符串的汇编语句 |
| current\_address = start\_address |
|  |
| while current\_address <= end\_address: |
| # 获取当前指令的反汇编文本 |
| disasm = idc.GetDisasm(current\_address) |
|  |
| # 检查是否包含指定字符串 |
| if target\_string in disasm: |
| relevant\_instructions.append((current\_address, disasm)) |
|  |
| # 移动到下一条指令 |
| current\_address = idc.NextHead(current\_address) |
|  |
| # 输出找到的相关汇编语句 |
| for instruction in relevant\_instructions: |
| print("Found relevant instruction at 0x%X: %s" % (instruction[0], instruction[1])) |

上面这段代码可用于在Lab06-04.exe中查找“Internet Explorer 7.50/pma%d”字符串



1. **实验结论及心得体会**
2. 实验结论

通过对四个实验样本的分析，可以得出以下结论：

这些恶意代码样本都涉及与Internet通信，用于检查网络连接和下载HTML网页。

恶意代码样本中包含了不同的行为，如删除文件、创建目录、修改注册表等，这些行为受到特定的控制字符的触发。

恶意代码样本使用特定的User-Agent标识，以及特定的URL来下载HTML网页。

通过静态分析和反汇编，可以深入了解恶意代码的内部结构和行为，包括函数调用、参数传递和控制流。

编写Yara规则是一种有效的方法，用于检测和识别这些恶意代码的存在。

1. 心得体会

静态分析的重要性：实验中使用IDA Pro等工具进行静态分析是深入了解恶意代码的关键。通过反汇编和代码分析，可以揭示恶意代码的内部结构、功能和行为。

恶意代码多样性：实验中的不同样本展示了恶意代码的多样性。它们采用不同的策略和行为，包括检查网络连接、下载网页、修改文件系统和注册表等。了解这些多样性有助于更好地理解恶意代码的复杂性。

Yara规则的应用：编写Yara规则是一种强大的手段，用于检测恶意代码的存在。通过识别特定的特征字符串和行为模式，可以有效地检测和分类恶意代码样本。