

# 个人信息

姓名：付文轩

学号：1911410

专业：信息安全

学院：网络空间安全学院

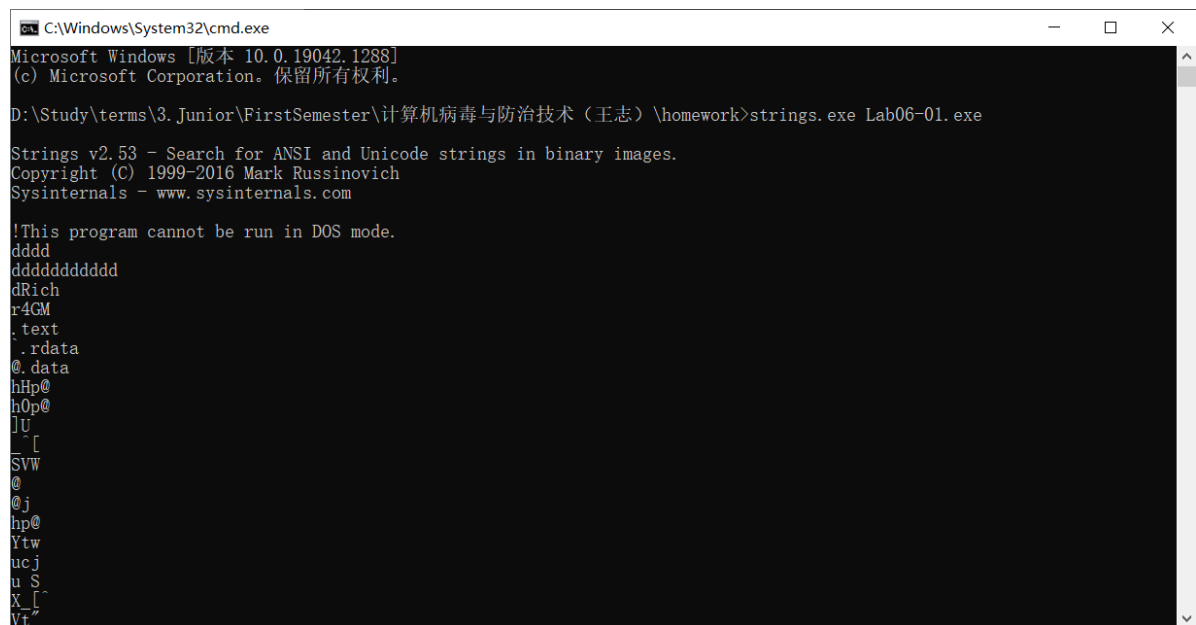
## lab 06-01

### 问题

1. What is the major code construct found in the only subroutine called by main?
2. What is the subroutine located at 0x40105F?
3. What is the purpose of this program?

### 实验过程

首先进行简单的静态分析：使用strings工具可以看到如下一些结果



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1288]
(c) Microsoft Corporation。保留所有权利。

D:\Study\terms\3. Junior\FirstSemester\计算机病毒与防治技术（王志）\homework>strings.exe Lab06-01.exe

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

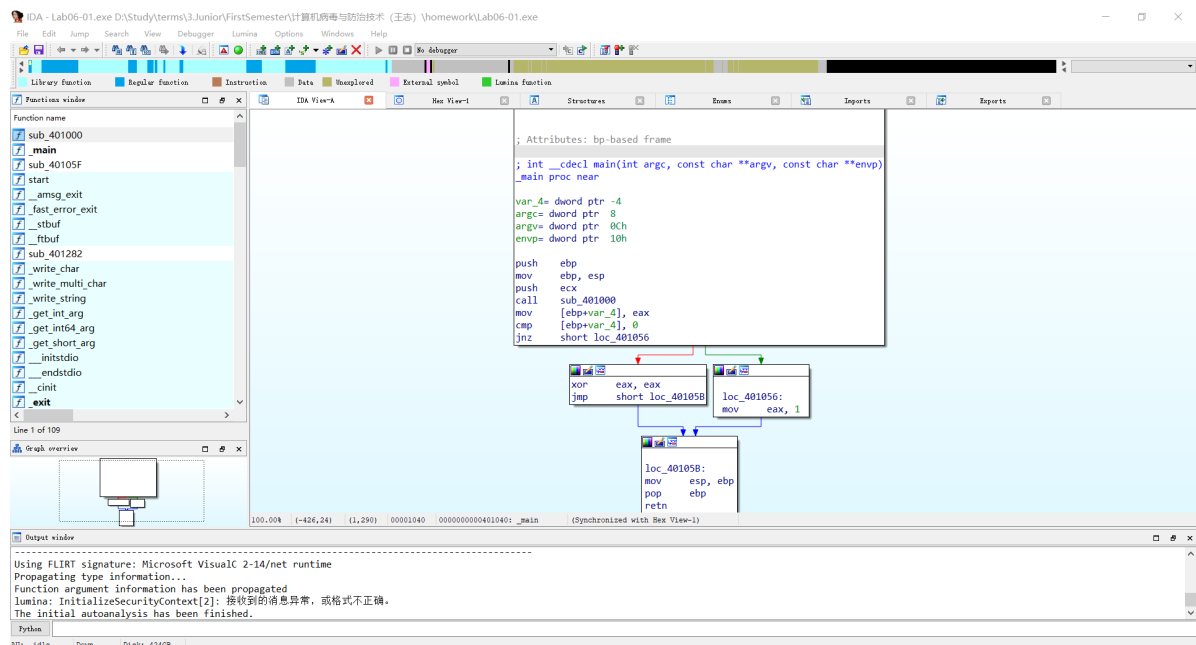
!This program cannot be run in DOS mode.
dddd
dddddddddd
dRich
r4GM
.text
.rdata
@.data
hHp@
hOp@
JU
SVW
@
@j
hp@
Ytw
ucj
u S
X_[-
Vt
```

```
C:\Windows\System32\cmd.exe
GetVersionExA
HeapDestroy
HeapCreate
VirtualFree
HeapFree
RtlUnwind
WriteFile
HeapAlloc
GetCPInfo
GetACP
GetOEMCP
VirtualAlloc
HeapReAlloc
GetProcAddress
LoadLibraryA
GetLastError
FlushFileBuffers
SetFilePointer
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle
KERNEL32.dll
D4@
Error 1.1: No Internet
Success: Internet Connection
0a@
```

```
C:\Windows\System32\cmd.exe
TLOSS error
SING error
DOMAIN error
R6028
- unable to initialize heap
R6027
- not enough space for lowio initialization
R6026
- not enough space for stdio initialization
R6025
- pure virtual function call
R6024
- not enough space for _onexit/atexit table
R6019
- unable to open console device
R6018
- unexpected heap error
R6017
- unexpected multithread lock error
R6016
- not enough space for thread data
abnormal program termination
R6009
- not enough space for environment
R6008
- not enough space for arguments
R6002
- floating point not loaded
Microsoft Visual C++ Runtime Library
Runtime Error!
```

在strings的结果中可以发现一些比较有用的信息：Error 1.1: No Internet Success: Internet Connection，根据这两个字符串可以猜测到次样本应当是有网络访问。同时在第一行有一个string为：!This program cannot be run in DOS mode.，可以知道这个程序是不能在DOS模式下运行的。

使用IDA对本次实验样本进行分析，进入到IDA以后的页面如下



从图形化界面可以看出这个程序比较简单，main函数中只有很少的功能，并且功能存在有分支，也就是说这个程序的主要结构只有一个if结构。

跳转到地址为0040105F处

```
.text:0040105F
.text:0040105F ; ===== SUBROUTINE =====
.text:0040105F
.text:0040105F sub_40105F      proc near          ; CODE XREF: sub_401000+1C1p
.text:0040105F                                     ; sub_401000+301p
.text:0040105F arg_0          = dword ptr 4
.text:0040105F arg_4          = dword ptr 8
.text:0040105F
.text:0040105F      push    ebx
.text:00401060      push    esi
.text:00401061      mov     esi, offset File
.text:00401066      push    edi
.text:00401067      push    esi
.text:00401068      call   __stbuf
.text:0040106D      mov     edi, eax
.text:0040106F      lea     eax, [esp+10h+arg_4]
.text:00401073      push    eax          ; int
.text:00401074      push    [esp+14h+arg_0] ; int
.text:00401078      push    esi          ; File
.text:00401079      call   sub_401282
.text:0040107E      push    esi
.text:0040107F      push    edi
.text:00401080      mov     ebx, eax
.text:00401082      call   __ftbuf
.text:00401087      add     esp, 18h
.text:0040108A      mov     eax, ebx
.text:0040108C      pop     edi
.text:0040108D      pop     esi
.text:0040108E      pop     ebx
.text:0040108F      retn
```

不难发现在40105F这个位置是一个函数，并且从右边的注释可以看出在上方有两处调用

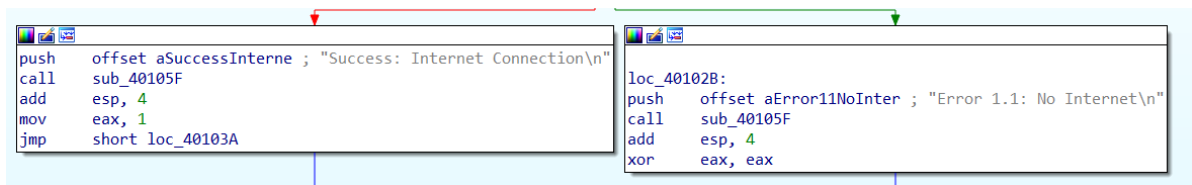
定位到两个调用位置：

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40105F

push    offset aError11NoInter ; "Error 1.1: No Internet\n"
call    sub_40105F
```

会发现在调用sub\_40105F之前都压入了一个参数，并且这个参数都是字符串类型，从字符串的内容能大概猜到是要输出或者是打印一个提示内容。

再次回到之前图形化界面



仔细观察不难发现，左右两个分支都对这个函数进行了调用

查找了一下printf函数的内容：

```
1  int __cdecl printf (  
2      const char *format,  
3      ...  
4  )  
5  /*  
6   * stdout 'PRINT', 'F'ormatted  
7   */  
8  {  
9      va_list arglist;  
10     int buffering;  
11     int retval;  
12  
13     va_start(arglist, format);  
14  
15     _ASSERTE(format != NULL);  
16  
17     _lock_str2(1, stdout);  
18  
19     buffering = _stbuf(stdout); // 注意这里  
20  
21     retval = _output(stdout, format, arglist); // 还有记住这里调用了一次外部函  
22     数  
23     _ftbuf(buffering, stdout); // 注意这里  
24  
25     _unlock_str2(1, stdout);  
26  
27     return(retval);  
28 }
```

结合之前我们看到的这个sub\_40105F函数中也有先 call \_\_stbuf，之后 call sub\_401282，最后 call \_\_ftbuf，和printf内标注出来的调用顺序相同，更加验证了这个函数的功能是进行打印的猜想。

之后在xp虚拟机中尝试运行了一下这个程序，也确实是输出了 Error 1.1: No Internet，由此可见猜测应当是正确的。

## 问题回答

### Q1

从图形化界面可以清楚的看出，这个程序的结构是if

## Q2

就是一个printf函数的功能

## Q3

检查是否能够联网，并显示相应的提示信息

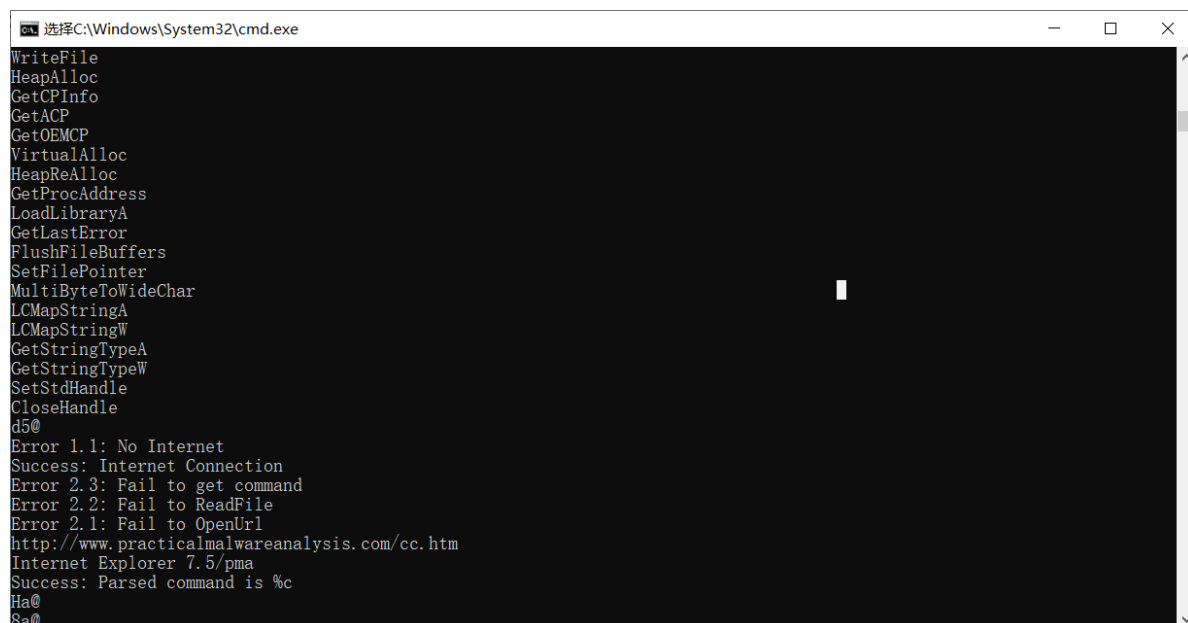
# lab 06-02

## 问题

1. What operation does the first subroutine called by main perform?
2. What is the subroutine located at 0x40117F?
3. What does the second subroutine called by main do?
4. What type of code construct is used in this subroutine?
5. Are there any network-based indicators for this program?
6. What is the purpose of this malware?

## 实验过程

首先进行简单的静态分析，使用strings工具检测结果如下



```
选择C:\Windows\System32\cmd.exe
WriteFile
HeapAlloc
GetCPInfo
GetACP
GetOEMCP
VirtualAlloc
HeapReAlloc
GetProcAddress
LoadLibraryA
GetLastError
FlushFileBuffers
SetFilePointer
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle
d5@
Error 1.1: No Internet
Success: Internet Connection
Error 2.3: Fail to get command
Error 2.2: Fail to ReadFile
Error 2.1: Fail to OpenUrl
http://www.practicalmalwareanalysis.com/cc.htm
Internet Explorer 7.5/pma
Success: Parsed command is %c
Ha@
Sa@
```

可以发现这个程序有一部分和lab06-01.exe中的字符串是一样的，比如那两个提示信息。同时能发现在这里面有更多的错误提示信息，其中有了关于获取命令行的、读取文件的、打开URL的，结合下面得到的strings的信息，猜测这个程序应当也是会先检查网络连接状态，之后访问相关网址并进行命令行和文件操作。同时有一个要注意的字符串是：Internet Explorer 7.5/pma，这里是指的浏览器的版本，猜测可能连浏览器的版本都进行了检测。

使用IDA进行分析

```

_main proc near
var_8= byte ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 8
call    sub_401000
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz     short loc_401148

```

仔细观察main函数上来执行的内容，不难发现这个和之前lab06-01.exe中的内容是相同的，也就验证了之前的猜想：程序在刚执行后，会先检查网络状态

接下来定位到40117f位置

```

.text:0040117F ; ===== S U B R O U T I N E =====
.text:0040117F
.text:0040117F
.text:0040117F sub_40117F      proc near                ; CODE XREF: sub_401000+1C↑p
.text:0040117F                                     ; sub_401000+30↑p ...
.text:0040117F
.text:0040117F arg_0        = dword ptr 4
.text:0040117F arg_4        = dword ptr 8
.text:0040117F
.text:0040117F      push    ebx
.text:00401180      push    esi
.text:00401181      mov     esi, offset File
.text:00401186      push    edi
.text:00401187      push    esi
.text:00401188      call    __stbuf
.text:0040118D      mov     edi, eax
.text:0040118F      lea     eax, [esp+10h+arg_4]
.text:00401193      push    eax                ; int
.text:00401194      push    [esp+14h+arg_0]    ; int
.text:00401198      push    esi                ; File
.text:00401199      call    sub_4013A2
.text:0040119E      push    esi
.text:0040119F      push    edi
.text:004011A0      mov     ebx, eax
.text:004011A2      call    __ftbuf
.text:004011A7      add     esp, 18h
.text:004011AA      mov     eax, ebx
.text:004011AC      pop     edi
.text:004011AD      pop     esi
.text:004011AE      pop     ebx
.text:004011AF      retn

```

发现这里依旧是一个函数，内容格式同之前的printf非常相似，查看一下对这个函数调用的环境

```

push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4

push    offset aError11NoInter ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4

```

不难发现还是将两个字符串作为参数，结合之前的内容，不难想到这个函数是一个printf

回到main函数，可以看见第二个子过程是在call sub\_401040，双击查看相关内容

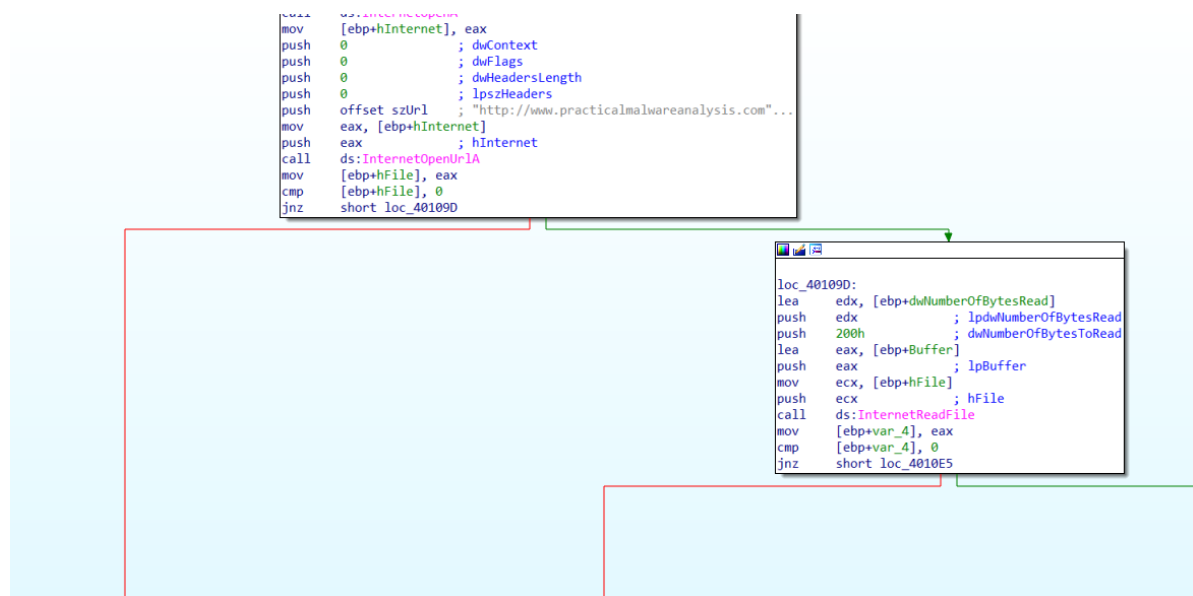
```

.text:00401040 var_4 = dword ptr -4
.text:00401040
.text:00401040 push ebp
.text:00401041 mov ebp, esp
.text:00401043 sub esp, 210h
.text:00401049 push 0 ; dwFlags
.text:0040104B push 0 ; lpszProxyBypass
.text:0040104D push 0 ; lpszProxy
.text:0040104F push 0 ; dwAccessType
.text:00401051 push offset szAgent ; "Internet Explorer 7.5/pma"
.text:00401056 call ds:InternetOpenA
.text:0040105C mov [ebp+hInternet], eax
.text:0040105F push 0 ; dwContext
.text:00401061 push 0 ; dwFlags
.text:00401063 push 0 ; dwHeadersLength
.text:00401065 push 0 ; lpszHeaders
.text:00401067 push offset szUrl ; "http://www.practicalmalwareanalysis.com"
.text:0040106C mov eax, [ebp+hInternet]
.text:0040106F push eax ; hInternet
.text:00401070 call ds:InternetOpenUrlA
.text:00401076 mov [ebp+hFile], eax
.text:00401079 cmp [ebp+hFile], 0
.text:0040107D jnz short loc_40109D
.text:0040107F push offset aError21FailTo0 ; "Error 2.1: Fail to OpenUrl\n"
.text:00401084 call sub_40117F
.text:00401089 add esp, 4
.text:0040108C mov ecx, [ebp+hInternet]
.text:0040108F push ecx ; hInternet
.text:00401090 call ds:InternetCloseHandle
.text:00401096 xor al, al
.text:00401098 jmp loc_40112C

```

可以看见这一部分的流程就是先后调用了三个函数 `InternetOpenA` `InternetOpenUrlA` `InternetCloseHandle`，同时结合边上的注释不难看出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

查看这一部分的图形化界面



可以很清楚的看出这一段就是if的一个结构，在打开了网页以后会将网页放到一个Buffer中并打开（也就是将这个网页下载了下来），如果打开失败就显示提示信息，如果打开成功就进行其他的操作；之后的内容是对读取出来的前4个字节进行匹配，需要前四个字节的内容为 `<!--`，可以看出这是html中注释部分的开头

## 问题回答

## Q1

和lab06-01.exe一样，检查网络连接状态

## Q2

结合程序中的内容以及调用时压入的字符串参数，认为这个函数是一个printf

## Q3

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

## Q4

主要结构是if结构

## Q5

访问 `http://www.practicalmalwareanalysis.com` 并将网页下载下来，同时在访问的时候使用 `Internet Explorer 7.5/pma` 作为usr-agent

## Q6

检查当前是否有网络连接，如果有就访问 `http://www.practicalmalwareanalysis.com` 并将网页下载下来，之后对下载下来的内容进行逐字比对

# lab 06-03

---

## 问题

---

1. Compare the calls in main to Lab 6-2's main method. What is the new function called from main?
2. What parameters does this new function take?
3. What major code construct does this function contain?
4. What can this function do?
5. Are there any host-based indicators for this malware?
6. What is the purpose of this malware?

## 实验过程

---

首先依旧是进行简单的静态分析

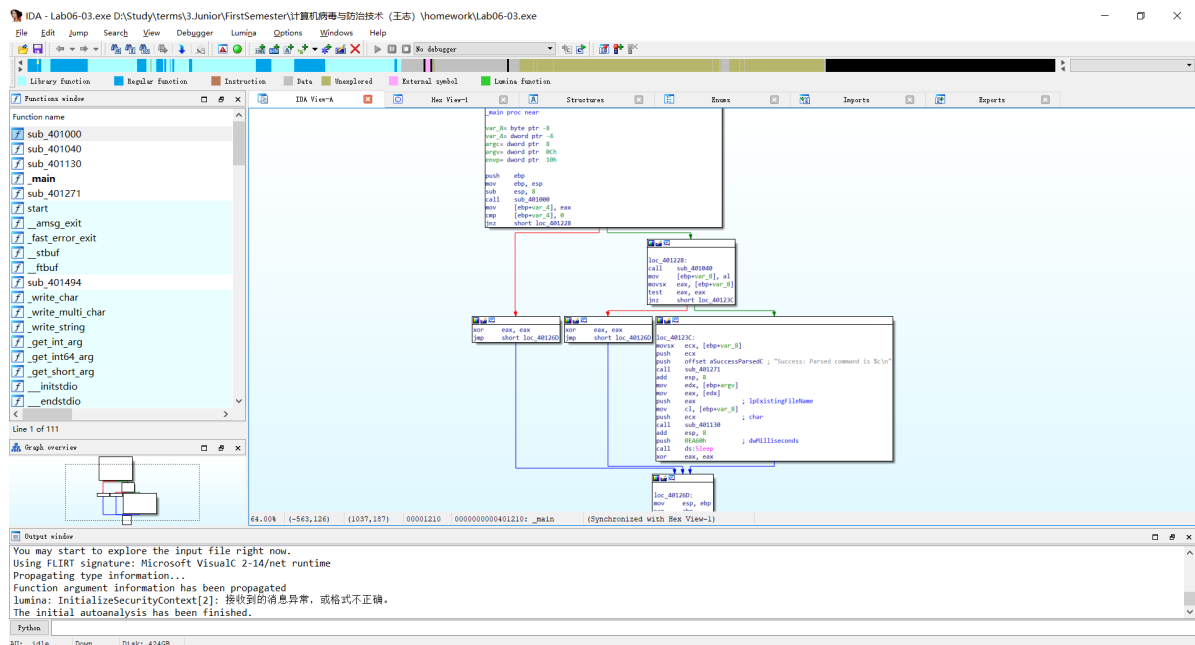


```
C:\Windows\System32\cmd.exe
HeapReAlloc
GetProcAddress
LoadLibraryA
GetLastError
FlushFileBuffers
SetFilePointer
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle
T6@
Error 1.1: No Internet
Success: Internet Connection
Error 2.3: Fail to get command
Error 2.2: Fail to ReadFile
Error 2.1: Fail to OpenUrl
http://www.practicalmalwareanalysis.com/cc.htm
Internet Explorer 7.5/pma
Error 3.2: Not a valid command provided
Error 3.1: Could not set Registry value
Malware
Software\Microsoft\Windows\CurrentVersion\Run
C:\Temp\cc.exe
C:\Temp
Success: Parsed command is %c
a@
Pa@
```

- ```
1 Error 3.2: Not a valid command provided
2 Error 3.1: Could not set Registry value
3 Malware
4 Software\Microsoft\Windows\CurrentVersion\Run
5 C:\Temp\cc.exe
6 C:\Temp
```

可以发现在多出来的内容里有注册表的内容，有对C:\Temp目录下的cc.exe的操作， 综上猜测应该是多出来了一个在C盘目录下创建一个exe程序，并将其加入到自启动的注册表中

使用IDA进行分析，得到图形化界面如下



可以看出前面的部分依旧是检查网络连接，但是在输出了Success的提示消息之后，这里又多调用了一个函数：sub 401130

回到调用这个函数的部分

```

add     esp, 8
mov     edx, [ebp+argv]
mov     eax, [edx]
push    eax                ; lpExistingFileName
mov     cl, [ebp+var_8]
push    ecx                ; char
call    sub_401130

```

可以看见在这个函数调用之前一共有两个参数进行了入栈，结合右边的注释可以看见有一个是 lpExistingFileName，另一个是一个char。在之前的程序中可以发现有一条为：

```

mov     [ebp+var_8], al

```

这里其实存入的就是之前读取到那个网页中内容时的第一个字符。而[ebp+argv]就是argv[0]里的内容，也就是当前这个可执行程序的名字

进入到新调用的这个函数体内部

```

.text:00401130 phkResult      = dword ptr -4
.text:00401130 arg_0         = byte ptr 8
.text:00401130 lpExistingFileName= dword ptr 0Ch
.text:00401130
.text:00401130             push    ebp
.text:00401131             mov     ebp, esp
.text:00401133             sub     esp, 8
.text:00401136             movsx   eax, [ebp+arg_0]
.text:0040113A             mov     [ebp+var_8], eax
.text:0040113D             mov     ecx, [ebp+var_8]
.text:00401140             sub     ecx, 61h ; 'a' ; switch 5 cases
.text:00401143             mov     [ebp+var_8], ecx
.text:00401146             cmp     [ebp+var_8], 4
.text:0040114A             ja      def_401153 ; jumtable 00401153 default case
.text:00401150             mov     edx, [ebp+var_8]
.text:00401153             jmp     ds:jpt_401153[edx*4] ; switch jump

```

可以发现有一个关键字：switch jump，这里也就提示了我们这个函数使用的是switch结构进行跳转

之后往下看

```

.text:0040115A loc_40115A: ; CODE XREF: sub_401130+23↑j
.text:0040115A             ; DATA XREF: .text:jpt_401153↓o
.text:0040115A             push    0 ; jumtable 00401153 case 97
.text:0040115C             push    offset PathName ; "C:\\Temp"
.text:00401161             call   ds:CreateDirectoryA
.text:00401167             jmp     loc_4011EE
.text:0040116C ; -----
.text:0040116C loc_40116C: ; CODE XREF: sub_401130+23↑j
.text:0040116C             ; DATA XREF: .text:jpt_401153↓o
.text:0040116C             push    1 ; jumtable 00401153 case 98
.text:0040116E             push    offset Data ; "C:\\Temp\\cc.exe"
.text:00401173             mov     eax, [ebp+lpExistingFileName]
.text:00401176             push    eax ; lpExistingFileName
.text:00401177             call   ds:CopyFileA
.text:0040117D             jmp     short loc_4011EE
.text:0040117F ; -----
.text:0040117F loc_40117F: ; CODE XREF: sub_401130+23↑j
.text:0040117F             ; DATA XREF: .text:jpt_401153↓o
.text:0040117F             push    offset Data ; jumtable 00401153 case 99
.text:00401184             call   ds>DeleteFileA
.text:0040118A             jmp     short loc_4011EE
.text:0040118C ; -----
.text:0040118C loc_40118C: ; CODE XREF: sub_401130+23↑j
.text:0040118C             ; DATA XREF: .text:jpt_401153↓o
.text:0040118C             lea     ecx, [ebp+phkResult] ; jumtable 00401153 case 100
.text:0040118F             push    ecx ; phkResult
.text:00401190             push    0F003Fh ; samDesired
.text:00401195             push    0 ; ulOptions
.text:00401197             push    offset SubKey ; "Software\\Microsoft\\Windows\\CurrentVe"

```

```

.text:0040119C      push     80000002h          ; hKey
.text:004011A1      call     ds:RegOpenKeyExA
.text:004011A7      push     0Fh                ; cbData
.text:004011A9      push     offset Data         ; "C:\\Temp\\cc.exe"
.text:004011AE      push     1                   ; dwType
.text:004011B0      push     0                   ; Reserved
.text:004011B2      push     offset ValueName    ; "Malware"
.text:004011B7      mov      edx, [ebp+phkResult]
.text:004011BA      push     edx                 ; hKey
.text:004011BB      call     ds:RegSetValueExA
.text:004011C1      test     eax, eax
.text:004011C3      jz       short loc_4011D2
.text:004011C5      push     offset aError31CouldNo ; "Error 3.1: Could not set Registry value"...
.text:004011CA      call     sub_401271
.text:004011CF      add      esp, 4
.text:004011D2      loc_4011D2:                jmp      short loc_4011EE ; CODE XREF: sub_401130+93↑j

```

当这个函数进行顺序执行的时候，可以发现这个函数先后执行了：在C盘下创建一个Temp目录->复制cc.exe到Temp目录下->删除可执行程序->打开注册表使得程序能够自启->Sleep

## 问题回答

### Q1

和之前的结果进行对比以后可以发现这里新调用了函数是：sub\_401130

### Q2

一共有两个参数：一个是读取的网页中的第一个字符，一个是当前程序的名字

### Q3

使用了switch结构

### Q4

在C盘下创建一个Temp目录->复制cc.exe到Temp目录下->删除可执行程序->打开注册表使得程序能够自启->Sleep

### Q5

根据之前的操作，这个程序会在C盘下创建一个Temp目录，并复制一个可执行程序cc.exe到目录下；

修改注册表信息，使得cc.exe能够自启动

### Q6

会访问一个网址并进行解析，同时将一个可执行文件复制到C盘目录下并修改注册表使得其能够自启动

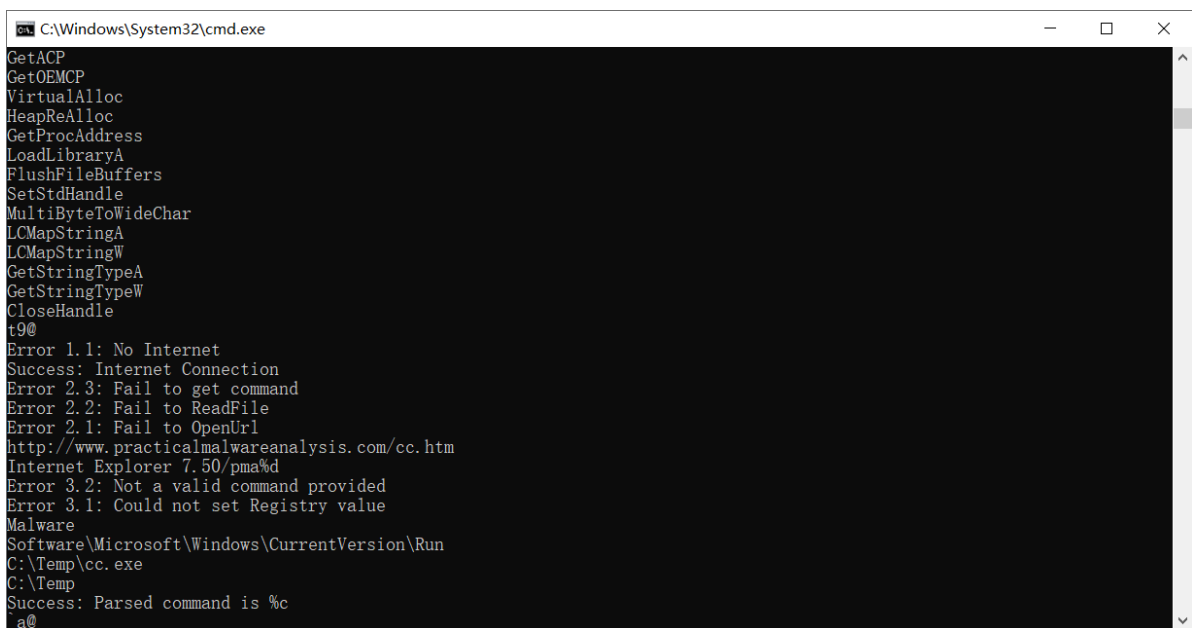
## lab 06-04

## 问题

1. What is the difference between the calls made from the main method in Labs 6-3 and 6-4?
2. What new code construct has been added to main?
3. What is the difference between this lab's parse HTML function and those of the previous labs?
4. How long will this program run? (Assume that it is connected to the Internet.)
5. Are there any new network-based indicators for this malware?
6. What is the purpose of this malware?

## 实验过程

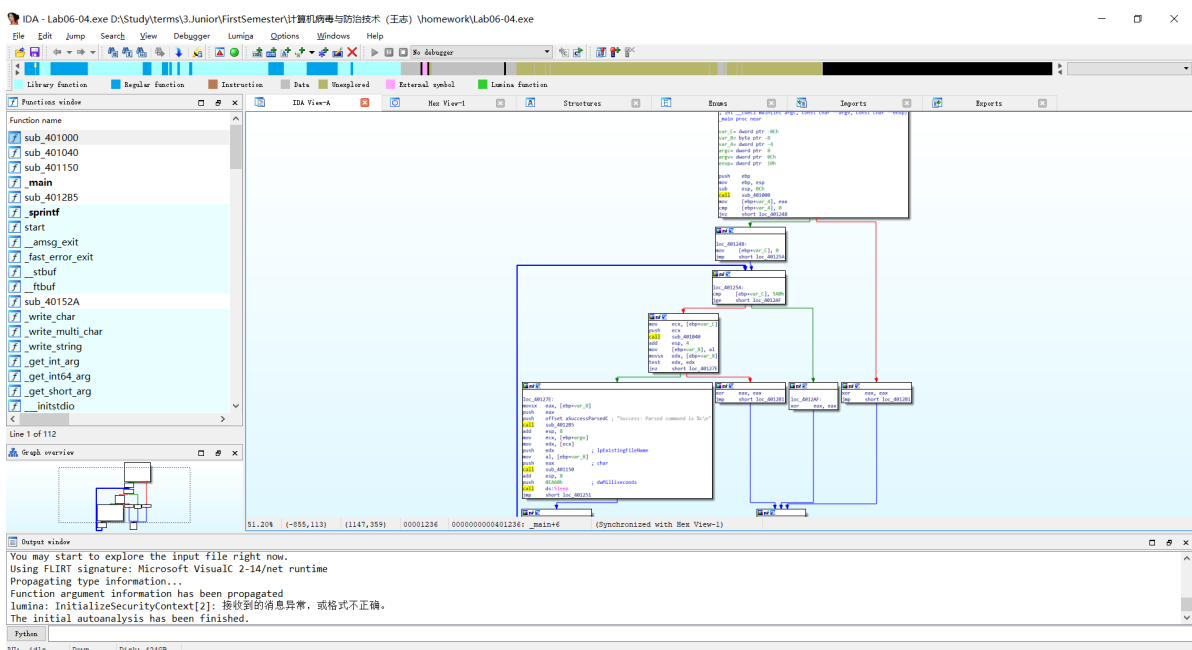
先使用strings工具进行简单的静态分析，得到的检测结果如下



```
C:\Windows\System32\cmd.exe
GetACP
GetOEMCP
VirtualAlloc
HeapReAlloc
GetProcAddress
LoadLibraryA
FlushFileBuffers
SetStdHandle
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
CloseHandle
t9@
Error 1.1: No Internet
Success: Internet Connection
Error 2.3: Fail to get command
Error 2.2: Fail to ReadFile
Error 2.1: Fail to OpenUrl
http://www.practicalmalwareanalysis.com/cc.htm
Internet Explorer 7.50/pma%d
Error 3.2: Not a valid command provided
Error 3.1: Could not set Registry value
Malware
Software\Microsoft\Windows\CurrentVersion\Run
C:\Temp\cc.exe
C:\Temp
Success: Parsed command is %c
a@
```

从strings的结果来看感觉和之前的lab06-03.exe基本上没有什么区别，唯一一处是在 Internet Explorer 7.50/pma 后多出了一个%d，根据这里猜测可能是usr-agent会比之前多出一些变化。

使用IDA进行查看，得到图形化界面如下



从main函数看起，可以发现main函数依旧是先调用了sub\_401000（判断是否有网络连接，并输出提示），然后调用sub\_401040（解析获取的HTML），在解析过程中调用了之前的sub\_4012B5（printf）。

在main函数中发现一个比较有趣的部分：loc\_40125A

```
.text:00401251 loc_401251: ; CODE XREF: _main+7D↓j
.text:00401251 mov     eax, [ebp+var_C]
.text:00401254 add     eax, 1
.text:00401257 mov     [ebp+var_C], eax
.text:0040125A loc_40125A: ; CODE XREF: _main+1F↑j
.text:0040125A cmp     [ebp+var_C], 5A0h
.text:00401261 jge     short loc_4012AF
.text:00401263 mov     ecx, [ebp+var_C]
.text:00401266 push    ecx
.text:00401267 call    sub_401040
.text:0040126C add     esp, 4
.text:0040126F mov     [ebp+var_8], al
.text:00401272 movsx   edx, [ebp+var_8]
.text:00401276 test    edx, edx
.text:00401278 jnz     short loc_40127E
.text:0040127A xor     eax, eax
.text:0040127C jmp     short loc_4012B1
.text:0040127F : -----
```

当main执行到40125A这里时，会把var\_C里的值和5A0h比较；根据下面的代码可以看出来，这里是进行了一个循环，循环体内部是一个sleep，并且sleep的参数是0EA60h，也就是60秒，总共循环1440次（5A0h），总共也就是需要睡眠24h

之后在调用subsub\_401040之前，向其中压入了ecx，也就是5A0h，接下来查看函数体内部

```
.text:00401040 dwNumberOfBytesRead= dword ptr -8
.text:00401040 var_4 = dword ptr -4
.text:00401040 arg_0 = dword ptr 8
.text:00401040
.text:00401040 push    ebp
.text:00401041 mov     ebp, esp
.text:00401043 sub     esp, 230h
.text:00401049 mov     eax, [ebp+arg_0]
.text:0040104C push    eax
.text:0040104D push    offset Format ; "Internet Explorer 7.50/pma%d"
.text:00401052 lea     ecx, [ebp+szAgent]
.text:00401055 push    ecx ; Buffer
.text:00401056 call    _sprintf
.text:00401058 add     esp, 0Ch
.text:0040105E push    0 ; dwFlags
.text:00401060 push    0 ; lpszProxyBypass
.text:00401062 push    0 ; lpszProxy
.text:00401064 push    0 ; dwAccessType
.text:00401066 lea     edx, [ebp+szAgent]
.text:00401069 push    edx ; lpszAgent
.text:0040106A call    ds:InternetOpenA
.text:00401070 mov     [ebp+hInternet], eax
.text:00401073 push    0 ; dwContext
.text:00401075 push    0 ; dwFlags
.text:00401077 push    0 ; dwHeadersLength
.text:00401079 push    0 ; lpszHeaders
.text:0040107B push    offset szUrl ; "http://www.practicalmalwareanalysis.com"...
.text:00401080 mov     [ebp+hInternet], eax
```

可以发现在调用之前压入了[ebp+arg\_0]、字符串和[ebp+szAgentn]，其中[ebp+szAgentn]是InternetOpenA这个函数的参数，而arg\_0其实就是刚刚main函数中的计数器，同时可以看见[ebp+szAgentn]被传入了buffer中。之后其他的功能和上一个样本相同。也就是说，这里是usr-agent是随着计数器的变化而发生改变的

## 问题回答

## Q1

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

## Q2

for循环

## Q3

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

## Q4

如果仅考虑for循环体带来的影响，本样本将会运行至少24h

## Q5

本次的usr-agent会发生变化

## Q6

先检查是否有网络连接，之后使用一个会发生变化的usr-agent从

<http://www.practicalmalwareanalysis.com> 下载HTML并进行解析，之后根据解析结果进行创建目录、复制exe文件、删除exe文件、修改注册表达到自启动、睡眠这一系列的操作。总共循环次数为1440次，预计程序执行的时间不少于24h

# yara扫描

本次实验涉及到的4个样本是一种逐渐完善形式的，每个后面的样本都比之前要多出一点结构和功能，其主体功能基本都是判断网络连接、下载一个HTML、解析，在后续多出来了对exe文件的操作。

所以在这里只写一套规则进行扫描，编写yara规则如下

```
1  import "pe"
2
3  rule Message {
4      strings:
5          $ErrorM = "Error"
6          $SuccessM = "Success"
7          $Internet = "Internet"
8      condition:
9          $ErrorM or $SuccessM or $Internet
10 }
11
12 rule MalURLRequest {
13     strings:
14         $Mal = "practicalmalwareanalysis"
15         $Http = "http"
16     condition:
17         $Mal and $Http
18 }
```

```
19
20 rule EXE {
21     strings:
22         $exe = /[a-zA-Z0-9_]+.exe/
23     condition:
24         $exe
25 }
26
27 rule Regedit {
28     strings:
29         $run = "Software\\Microsoft\\windows\\CurrentVersion\\Run"
30     condition:
31         $run
32 }
```

扫描结果如下:

```
D:\Study\terms\3. Junior\FirstSemester\计算机病毒与防治技术 (王志) \homework>yara64.exe -r lab6_yar lab6
Message lab6\Lab06-01.exe
Message lab6\Lab06-02.exe
MalURLRequest lab6\Lab06-02.exe
Message lab6\Lab06-04.exe
MalURLRequest lab6\Lab06-04.exe
EXE lab6\Lab06-04.exe
Regedit lab6\Lab06-04.exe
Message lab6\Lab06-03.exe
MalURLRequest lab6\Lab06-03.exe
EXE lab6\Lab06-03.exe
Regedit lab6\Lab06-03.exe
```

检测结果正确