

计算机病毒及其防治技术

Lab5

沙璇 1911562

Lab 06-01

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？

输入表查看器					
DLL名称	OriginalFirstThunk	时间/日期标志	ForwarderChain	名称	FirstThunk
WININET.dll	000065B0	00000000	00000000	000065D4	000060B0
KERNEL32.dll	00006500	00000000	00000000	000068CE	00006000

Thunk RVA	Thunk 偏移	Thunk 值	提示/序号	API 名称
000060B0	000060B0	000065B8	0066	InternetGetConnectedState

输入表查看器					
DLL名称	OriginalFirstThunk	时间/日期标志	ForwarderChain	名称	FirstThunk
WININET.dll	000065B0	00000000	00000000	000065D4	000060B0
KERNEL32.dll	00006500	00000000	00000000	000068CE	00006000

Thunk RVA	Thunk 偏移	Thunk 值	提示/序号	API 名称
00006060	00006060	0000679A	022F	RtlUnwind
00006064	00006064	000067A6	02DF	WriteFile
00006068	00006068	000067E2	0199	HeapAlloc
0000606C	0000606C	000067BE	00BF	GetCPInfo
00006070	00006070	000067CA	00B9	GetACP
00006074	00006074	000067D4	0131	GetOEMCP
00006078	00006078	000067E0	02BB	VirtualAlloc
0000607C	0000607C	000067F0	01A2	HeapReAlloc

查阅资料可知,

InternetGetConnectState 函数用于检查本地系统的网络连接状态。

GetACP 获取当前系统的代码页编码, 如简体中文是 936。

GetCPinfo 取得与指定代码页有关的信息。

使用 strings 得到如下字符串:

```
InternetGetConnectedState
WININET.dll
GetCommandLineA
GetVersion
ExitProcess
TerminateProcess
GetCurrentProcess
UnhandledExceptionFilter
GetModuleFileNameA
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
```

GetCommandLineA 获取命令行输入参数:

```
Error 1.1: No Internet
Success: Internet Connection
0a0
 a0
000
tc0
Pc0
$00
```

得到以下输出：

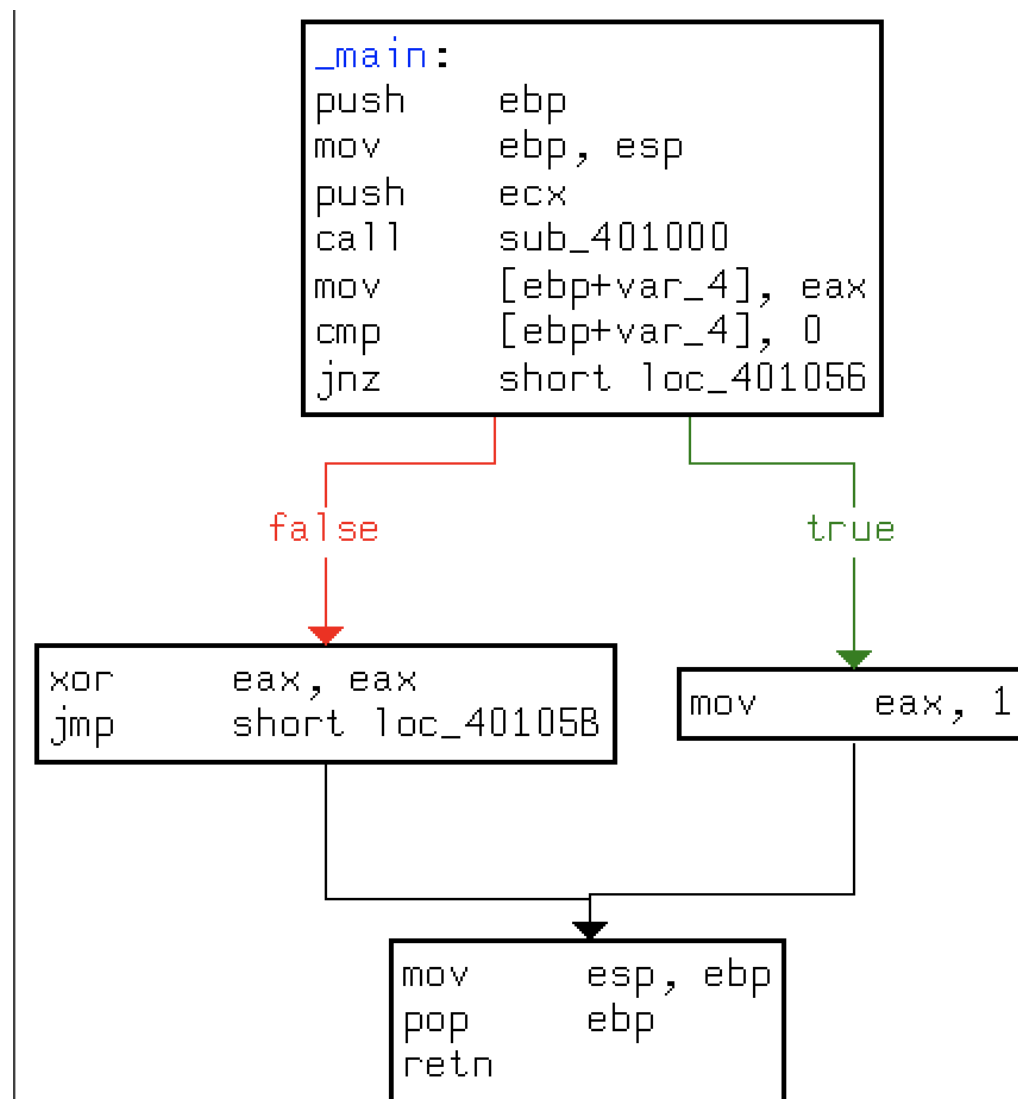
Error 1.1: No Internet

Success: Internet Connection

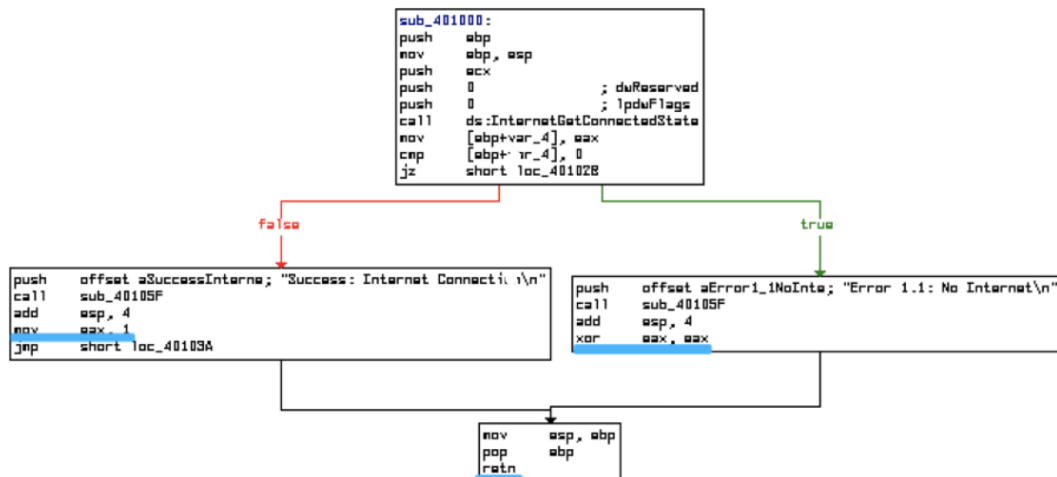
该程序可能检查系统中是否存在可用的网络连接。

动态运行后发现，只打印了 Success: Internet Connection 即退出。

使用 ida pro 加载，从 main 函数(.text:00401040)开始。



sub_401000 比较关键，进入查看：



使用 `cmp` 对保存结果的 `eax` 寄存器与 0 比较, 使用 `jz` 指令控制执行流。

`InternetGetConnectedState` 用于检查本地系统的网络连接状态, 存在可用连接时, 函数返回 1, 否则返回 0。返回 1 时, 零标志位 (ZF) 会被清除, `jz` 指令进入 `false` 分支。

`mov eax, 1` 说明连接成功; `xor eax, eax` 说明连接失败。

此处代码结构为 `if` 语句。

2. 位于 0x40105F 的子过程是什么？

`Printf` 函数：

```

int __cdecl printf(
    const char *format,
    ...
)
/*
 * stdout 'PRINT', 'F'ormatted
 */
{
    va_list arglist;
    int buffering;
    int retval;
    va_start(arglist, format);
    _ASSERTE(format != NULL);
    _lock_str2(1, stdout);
    buffering = _stbuf(stdout);
    retval = _output(stdout, format, arglist);
    _ftbuf(buffering, stdout);
    _unlock_str2(1, stdout);
    return(retval);
}

```

样本中的汇编代码：

```

.text:0040105F sub_40105F      proc near                ; CODE XREF: sub_401000+1C1p
.text:0040105F                                     ; sub_401000+301p
.text:0040105F arg_0          = dword ptr 4
.text:0040105F arg_4          = dword ptr 8
.text:0040105F
.text:0040105F      push     ebx
.text:00401060      push     esi
.text:00401061      mov     esi, offset 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

```

对比发现题目输入的是 offset stru_407098, 而不是像 printf 的内部实现那里的 stdout, 然后查看一下这个汇编里的文件描述符:

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

查阅资料得知, 对应关系如下:

```

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

```

typedef struct _iobuf FILE;

这里的 _file 代表了打开的文件在系统中的编号。

一般有三个文件的文件描述符是固定写死的, 在系统中对应的值如下:

```

stdin -> 0
stdout -> 1
stderr -> 2

```

文件描述符指向的是 stdout

根据前面的贴出来的 printf 函数的内部实现和对照汇编代码, 以及文件描述符是 stdout, 基本可以确定这个函数就是 printf。

3.这个程序的目的是什么?

检查是否存在可用的 Internet 连接, 如果存在, 打印结果并返回 1, 不存在则返回 0。

Lab 06-02

1. 由 main 函数调用的第一个子过程执行了什么操作？

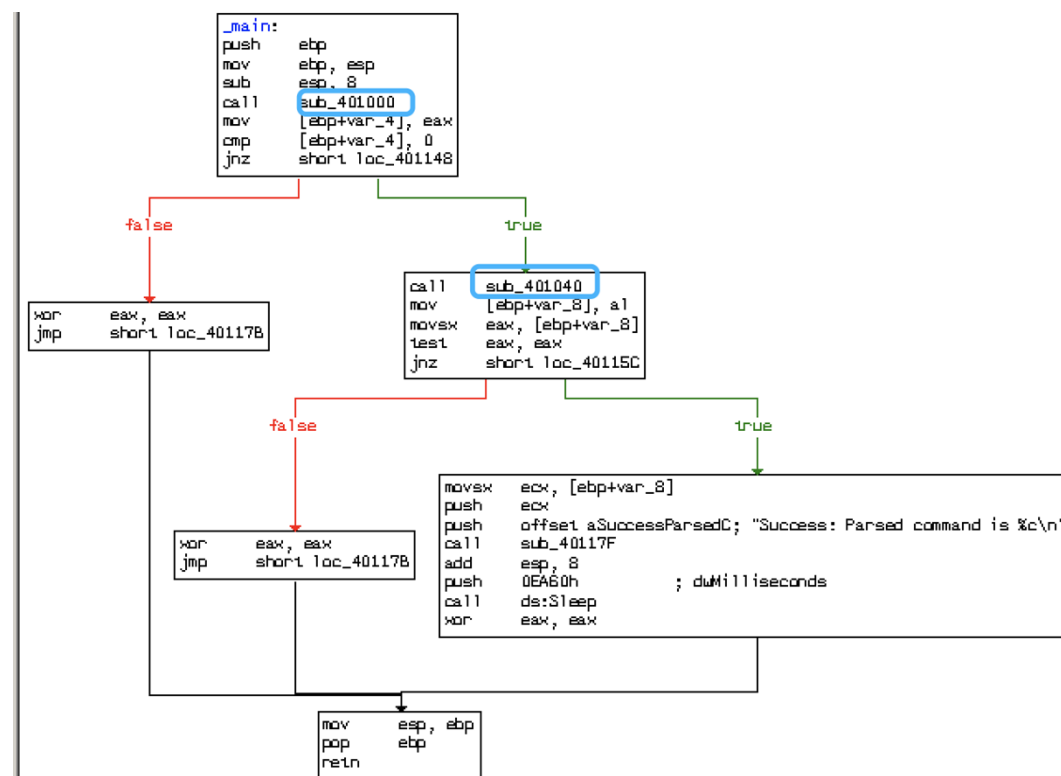
同 Lab 6-1, 是一个 if 语句, 用于检查是否存在可用的网络连接。

2. 位于 0x40117F 的子过程是什么？

同 Lab 6-1, 是 printf 函数。

```
text:0040117F sub_40117F      proc near          ; CODE XREF: sub_401000+1C1p
text:0040117F                                     ; sub_401000+301p ...
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          ; FILE stru_407160
text:00401188          call    __stbuf     stru_407160      FILE <0, 0, 0
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
```

3. 被 main 函数调用的第二个子过程做了什么？



位于 0x401040, 下载网页, 从页面开始处解析 HTML 注释。

进入 sub_401040, 和动态分析一致。

```

.text:00401040      push     ebp
.text:00401041      mov     ebp, esp
.text:00401043      sub     esp, 210h
.text:00401049      push     0                ; dwFlags
.text:0040104B      push     0                ; lpszProxyBypass
.text:0040104D      push     0                ; lpszProxy
.text:0040104F      push     0                ; dwAccessType
.text:00401051      push     offset szAgent    ; "Internet Explorer 7.5/pma"
.text:00401056      call    ds:InternetOpenA
.text:0040105C      mov     [ebp+hInternet], eax
.text:0040105F      push     0                ; dwContext
.text:00401061      push     0                ; dwFlags
.text:00401063      push     0                ; dwHeadersLength
.text:00401065      push     0                ; lpszHeaders
.text:00401067      push     offset szUrl      ; "http://www.practicalmalwareanalysis.com"
.text:0040106C      mov     eax, [ebp+hInternet]
.text:0040106F      push     eax                ; hInternet
.text:00401070      call    ds:InternetOpenUrlA
.text:00401076      mov     [ebp+hFile], eax
.text:00401079      cmp     [ebp+hFile], 0
.text:0040107D      jnz     short loc_40109D
.text:0040107F      push     offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
.text:00401084      call    sub_40117F
.text:00401089      add     esp, 4
.text:0040108C      mov     ecx, [ebp+hInternet]
.text:0040108F      push     ecx                ; hInternet
.text:00401090      call    ds:InternetCloseHandle
.text:00401096      xor     al, al
.text:00401098      jmp     loc_40112C

```

首先调用 InternetOpenA 初始化; InternetOpenUrlA 用于打开 URL 的静态网页, 引发在动态分析中看到的 DNS 请求。

```

.text:00401070      call     ds:InternetOpenUrlA
.text:00401076      mov     [ebp+hFile], eax
.text:00401079      cmp     [ebp+hFile], 0
.text:0040107D      jnz     short loc_40109D

```

InternetOpenUrlA 结果赋给了 hFile, 并在 1 处与 0 进行比较。相等返回, 否则传给下一个函数 InternetReadFile。hFile 实际上是一个句柄, 一种访问已经打开的 URL 的途径。

```

.text:0040109D      loc_40109D:
.text:0040109D      lea     edx, [ebp+dwNumberOfBytesRead] ; CODE XREF: sub_40109D+3
.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:004010BF      jnz     short loc_4010F5

```

InternetReadFile 从 InternetOpenUrlA 打开的网页中读取内容。

//MSDN

```

BOOLAPI InternetReadFile(
    HINTERNET hFile,
    LPVOID lpBuffer,
    DWORD dwNumberOfBytesToRead,
    LPDWORD lpdwNumberOfBytesRead
);

```

第二个参数也即是第 2 处, Buffer 是保存数据的数组, 最多保存 0x200 字节的数据, 即 3 处的 NumberOfBytesToRead 参数的值。

```

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

```

cmp ecx, 3Ch 用于检查第一个字符是否等于 0x3C, 对应的字符为<, 依次右击更改显示后发现<!--, 是 HTML 注释的开始部分。

movsx edx, [ebp+var_20F]这个位置的 var_20F, 是由于 IDA pro 没有识别出 Buffer 这个局部变量的大小是 512 字节, 因此显示为一个局部变量。

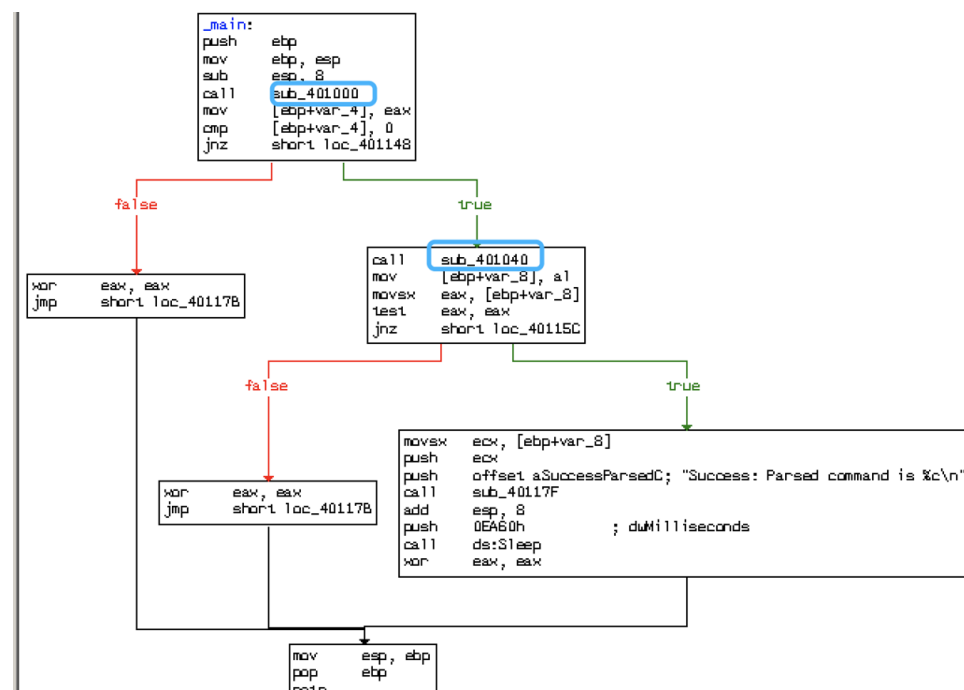
实质上相当于一个 Buffer+1。填充函数栈后:

```

.text:004010E5 ; -----
.text:004010E5
.text:004010E5 loc_4010E5: ; CODE XREF: sub_401040+7E↑j
.text:004010E5 movsx | ecx, [ebp+Buffer]
.text:004010EC cmp ecx, '<'
.text:004010EF jnz short loc_40111D
.text:004010F1 movsx edx, [ebp+Buffer+1]
.text:004010F8 cmp edx, '!'
.text:004010FB jnz short loc_40111D
.text:004010FD movsx eax, [ebp+Buffer+2]
.text:00401104 cmp eax, '-'
.text:00401107 jnz short loc_40111D
.text:00401109 movsx ecx, [ebp+Buffer+3]
.text:00401110 cmp ecx, '-'
.text:00401113 jnz short loc_40111D
.text:00401115 mov al, [ebp+Buffer+4]
.text:0040111B imd short loc_40112C

```

4. 在这个子过程中使用了什么类型的代码结构？



sub_401040 函数返回一个非 0 值, 打印"Success: Parsed command is %c\n", 其中%c 是格式字

字符串, 是从 HTML 中解析出来的字符。

紧接着调用了 sub_40117F 函数, 也就是 printf 函数
再调用 sleep 函数, 0EA60h 表示一分钟 60000 毫秒

5. 在这个程序中有任何基于网络的特征的指示吗?

两条网络特征。

使用 Internet Explorer 7.5/pma 作为 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?

首先判断是否在可用的网络连接, 不存在就终止运行。存在即使用独特的用户代理下载一个网页, 该网页包含一段由<!--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 注释解析出来的字符, 解析成功, 程序会休眠一分钟, 然后终止运行。

Lab 06-03

1. 比较在 main 函数于实验 6-2 的 mian 函数的调用。从 main 中调用的新的函数是什么?

```
.text:00401241      push     offset aSuccessParsedC ; "Success: Parsed command is %
.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
```



sub_401000(检查网络连接)和 sub_401040(下载网页并解析 HTML 注释)两个函数相同,

sub_401271 函数是 printf。新函数是 sub_401130。

2. 这个新的函数使用的参数是什么?

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

arg_0 是标准 main 函数参数的 argv[0], 程序名本身。

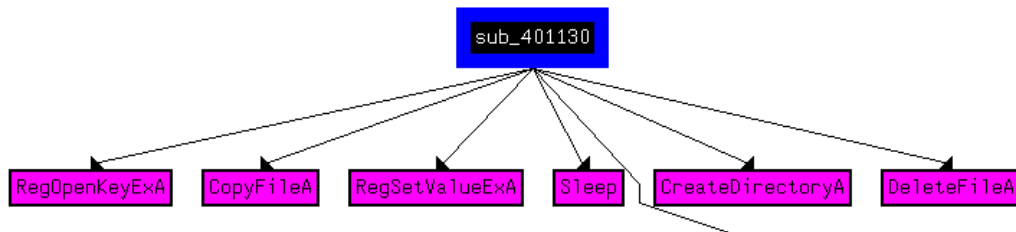
var_8 用 AL 设置。eax 是上一个函数(sub_401040 用于下载 HTML 网页解析注释)调用的返回结果, 而 AL 包含在 eax 中, 因此 var_8 包含容 HTML 注释中解析出的指令字符。

3. 这个函数包含的主要代码结构是什么?

```
mov     edx, [ebp+var_8]
jmp     ds:off_4011F2[edx*4] ; switch jump
```

Switch 结构

4. 这个函数能够做什么?



分别是打印出错信息、删除一个文件、创建一个文件、设置一个注册表项值、复制一个文件，或者休眠 100 秒。

再加上一个 printf。

5. 在这个恶意代码中有什么本地特征？

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

文件路径: C:\Temp\cc.exe

6. 这个恶意代码的目的是什么？

首先检查是否存在网络连接。没有则终止，有则继续尝试下载网页，包含以<!--开头的 HTML 注释，该注释的第一个字符用于 switch 语句决定程序在本地系统中的下一步行为，包括删除一个文件、创建一个文件、设置一个注册表项值、复制一个文件，或者休眠 100 秒。

Lab 06-04

1. 在实验 6-3 和 6-4 的 main 函数中的调用之间的关系是什么？

sub_401000: 检查网络连接

sub_401040: 解析 HTML

sub_401150: switch 语句

sub_4012B5: printf 函数

2. 什么新的代码结构已经被添加到 main 中？

一个 for 循环。

```

.text:00401248 ; -----
.text:00401248
.text:00401248 loc_401248: ; CODE XREF: _main+12↑j
.text:00401248 mov [ebp+var_C], 0
.text:0040124F jmp short loc_40125A
.text:00401251 ; -----
.text:00401251 loc_401251: ; CODE XREF: _main+7D↑j
.text:00401251 mov eax, [ebp+var_C]
.text:00401254 add eax, 1
.text:00401257 mov [ebp+var_C], eax
.text:0040125A loc_40125A: ; CODE XREF: _main+1F↑j
.text:0040125A cmp [ebp+var_C], 5A0h
.text:00401261 jge short loc_4012AF
.text:00401263 mov ecx, [ebp+var_C]
.text:00401266 push ecx
.text:00401267 call sub_401040
.text:0040126C add esp, 4
.text:0040126F mov [ebp+var_8], al
.text:00401272 movsx edx, [ebp+var_8]
.text:00401276 test edx, edx
.text:00401278 jnz short loc_40127E
.text:0040127A xor eax, eax
.text:0040127C jmp short loc_401281

```

```

text:004012A2      push    0EA60h          ; dwMilliseconds
text:004012A7      call    ds:Sleep
text:004012AD      jmp     short loc_401251 5
text:004012AF

```

3. 这个实验的解析 HTML 的函数和前面实验中的那些有什么区别？

```

.text:00401040      push    ebp
.text:00401040      mov     ebp, esp
.text:00401041      sub     esp, 230h
.text:00401043      mov     eax, [ebp+arg_0]
.text:00401049      push    eax
.text:0040104C      push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
.text:00401052      lea     ecx, [ebp+szAgent]
.text:00401055      push    ecx              ; char *
.text:00401056      call    _sprintf
.text:00401058      add     esp, 0Ch
.text:0040105E      push    0                ; dwFlags
.text:00401060      push    0                ; lpszProxyBypass
.text:00401062      push    0                ; lpszProxy
.text:00401064      push    0                ; dwAccessType
.text:00401066      lea     edx, [ebp+szAgent]
.text:00401069      push    edx              ; lpszAgent
.text:0040106A      call    ds:InternetOpenA
.text:00401070      mov     [ebp+hInternet], eax
.text:00401073      push    0                ; dwContext
.text:00401075      push    0                ; dwFlags
.text:00401077      push    0                ; dwHeadersLength
.text:00401079      push    0                ; lpszHeaders
.text:0040107B      push    offset szUrl      ; "http://www.practicalmalwareanalysis.com"...
.text:00401080      mov     eax, [ebp+hInternet]
.text:00401083      push    eax              ; hInternet
.text:00401084      call    ds:InternetOpenUrlA
.text:0040108A      mov     [ebp+hFile], eax
.text:0040108D      cmp     [ebp+hFile], 0
.text:00401091      jnz     short loc_4010B1
.text:00401093      push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
.text:00401098      call    sub_4012B5
.text:0040109D      add     esp, 4
.text:004010A0      mov     ecx, [ebp+hInternet]
.text:004010A3      push    ecx              ; hInternet
.text:004010A4      call    ds:InternetCloseHandle
.text:004010AA      xor     al, al
.text:004010AC      jmp     loc_401140

```

很明显在开头多了一个 sprintf 的调用,这个函数把 Internet Explorer 7.50/pma%d 写到了 szagent,对比 lab06-03 发现原来 user-agent 是固定不变的 Internet Explorer 7.5/pma,现在是可变的。

```

.text:00401040      push    ebp
.text:00401041      mov     ebp, esp
.text:00401043      sub     esp, 230h
.text:00401049      mov     eax, [ebp+arg_0]
.text:0040104C      push    eax
.text:0040104D      push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
.text:00401052      lea     ecx, [ebp+szAgent]
.text:00401055      push    ecx              ; char *
.text:00401056      call    _sprintf
.text:00401058      add     esp, 0Ch
.text:0040105E      push    0                ; dwFlags
.text:00401060      push    0                ; lpszProxyBypass
.text:00401062      push    0                ; lpszProxy
.text:00401064      push    0                ; dwAccessType
.text:00401066      lea     edx, [ebp+szAgent]
.text:00401069      push    edx              ; lpszAgent
.text:0040106A      call    ds:InternetOpenA
.text:00401070      mov     [ebp+hInternet], eax
.text:00401073      push    0                ; dwContext
.text:00401075      push    0                ; dwFlags
.text:00401077      push    0                ; dwHeadersLength
.text:00401079      push    0                ; lpszHeaders
.text:0040107B      push    offset szUrl      ; "http://www.practicalmalwareanalysis.com"...
.text:00401080      mov     eax, [ebp+hInternet]
.text:00401083      push    eax              ; hInternet
.text:00401084      call    ds:InternetOpenUrlA
.text:0040108A      mov     [ebp+hFile], eax
.text:0040108D      cmp     [ebp+hFile], 0
.text:00401091      jnz     short loc_4010B1
.text:00401093      push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
.text:00401098      call    sub_4012B5
.text:0040109D      add     esp, 4
.text:004010A0      mov     ecx, [ebp+hInternet]
.text:004010A3      push    ecx              ; hInternet
.text:004010A4      call    ds:InternetCloseHandle
.text:004010AA      xor     al, al
.text:004010AC      jmp     loc_401140

```

传入字符串参数经过 sprintf->InternetOpenA 最终赋值给了 InternetOpenUrlA 的 hInternet 字段。

4. 这个程序会运行多久?(假设它已经链接到互联网)

```
push    0EA60h          ; dwMilliseconds  
call    ds:Sleep
```

0EA60h,10 进制是 24h

5. 在这个恶意代码中有什么新的基于网络的迹象吗?

```
.data:00401088 0000001D C Error 2.2: Fail to ReadFile\n  
.data:004010A8 0000001C C Error 2.1: Fail to OpenUrl\n  
.data:004010C4 0000002F C http://www.practicalmalwareanalysis.com/cc.htm  
.data:004010F4 0000001D C Internet Explorer 7.50/pma%d  
.data:00401114 00000029 C Error 3.2: Not a valid command provided\n  
.data:00401140 00000029 C Error 3.1: Could not set Registry value\n  
.data:0040116C 00000008 C Malware  
.data:00401174 0000002E C Software\\Microsoft\\Windows\\CurrentVersion\\Run  
.data:004011A4 0000000F C C:\\Temp\\cc.exe  
.data:004011B4 00000008 C C:\\Temp
```

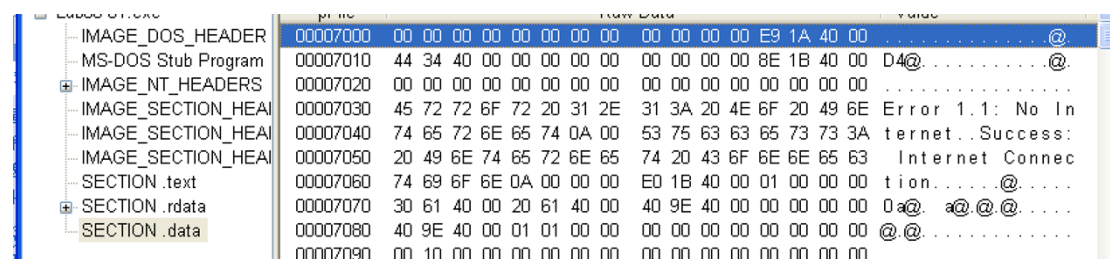
6. 这个恶意代码的目的是什么?

目的跟 lab6-3 是一样的。

多了一个使用可变的特殊 user-agent 来下载 html, 并且运行 24 小时后会停止。

YARA 规则编写

Lab06-01



Strings:

Success: Internet Connection

Lab06-02

Strings:

<http://www.practicalmalwareanalysis.com/cc.htm>

Internet Explorer 7.5/pma

Lab06-03

Strings:

Software\\Microsoft\\Windows\\CurrentVersion\\Run

Lab06-04

Strings:

Internet Explorer 7.50/pma%d

规则编写如下:

```
rule lab06{  
  meta:  
    description="lab6rule"  
  strings:
```

\$a="http://www.practicalmalwareanalysis.com/cc.htm"wide ascii

\$b="Software Microsoft Windows CurrentVersion Run"wide ascii

\$c="Internet Explorer 7.5/pma"wide ascii

\$d="Internet Explorer 7.50/pma%d"wide ascii

condition:

\$a or \$b or \$c or \$d

}