

# 恶意代码分析与防治技术实验报告

---

## Lab6

---

### 网络空间安全学院 信息安全专业

---

2112492 刘修铭 1063

---

<https://github.com/lxmliu2002/Malware Analysis and Prevention Techniques>

## 一、实验目的

---

1. 识别汇编代码中的C代码结构；
2. 进一步熟悉动态分析的过程。

## 二、实验环境

---

为了保护本机免受恶意代码攻击，本次实验主体在虚拟机上完成，以下为相关环境：

1. 已关闭病毒防护的Windows11
2. 在VMware上部署的Windows XP虚拟机
  - 在进行动态分析时，需对虚拟机做如下处理：
    - 对VMware进行快照，便于恢复到运行前的状态
    - 启动ApateDNS，将DNS Reply IP设置为127.0.0.1
    - 启动Process Monitor，并按照实验要求设置过滤条件
    - 启动Process Explorer
    - 启动netcat：nc-l -p XXX
    - 启动wireShark抓取数据包

## 三、实验工具

---

1. 待分析病毒样本（解压缩于XP虚拟机）
2. 相关病毒分析工具，如PETools、PEiD、Strings等
3. Yara检测引擎

## 四、实验过程

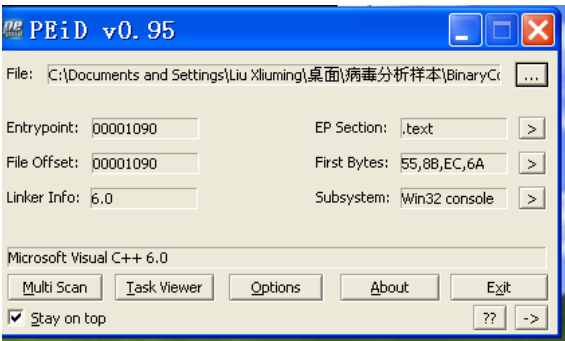
---

本次实验主要基于动态分析，但在动态分析之前需要先对其进行静态分析，掌握文件的整体情况。故而整体的实验思想为，先静态再动态。

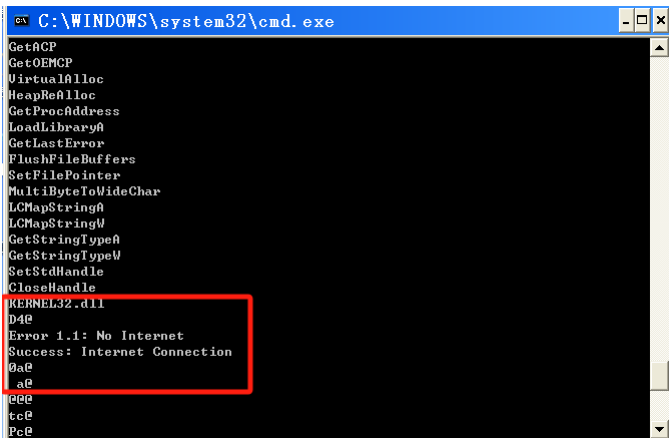
## (一) Lab6-1

### 1.静态分析

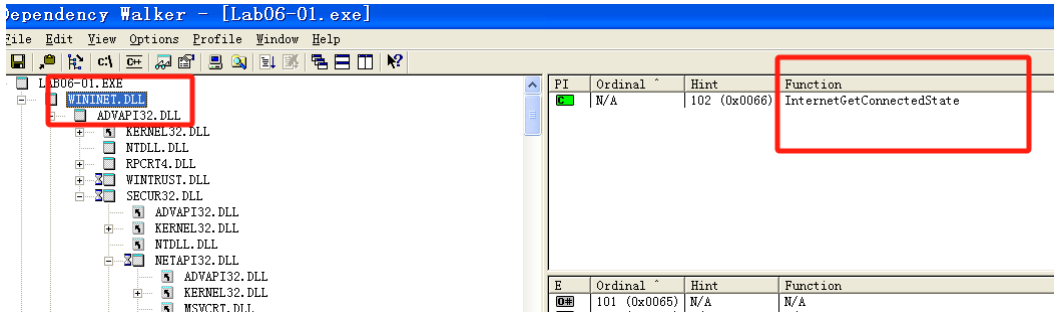
使用PEiD打开文件，可以看到文件未加壳。



接着使用Strings查看文件的字符串，可以看到“Error 1.1L No Internet”、“Success: Internet Connection”等，推测该恶意代码会检测系统中是否存在可用的Internet 连接。



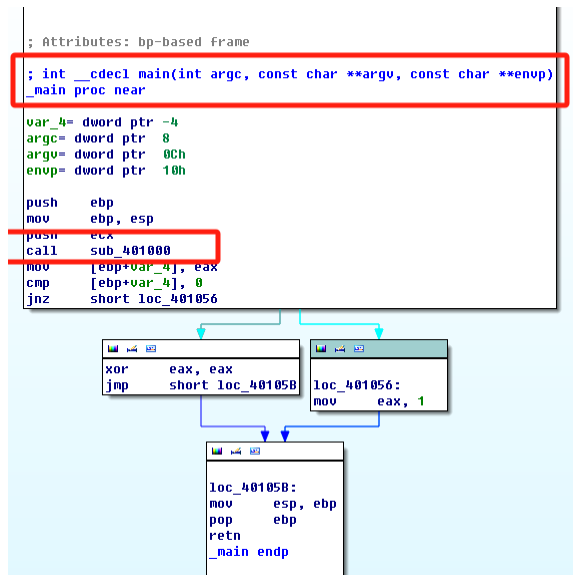
下面使用Dependency Walker查看导入表，可以看到wininet.dll中的InternetGetConectedState，经查询可知，该函数的作用是获得本地系统的网络连接状态。



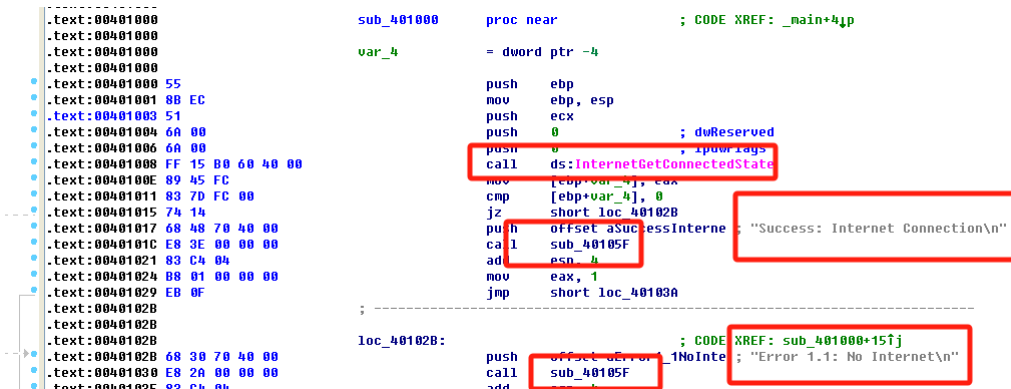
### 2.动态分析

基于上述分析，我们将着重研究已经发现的相关问题。

使用IDA打开文件，切换视图，可以看到main函数首先调用了sub\_401000函数，eax中保存着该函数的返回地址，根据该返回地址进行判断跳转，故而重点分析该函数。



双击该函数可以看到，若存在网络连接，则将字符串“Sucess: Internet Connection\n”作为参数传给sub\_40105F函数；若不存在，则将字符串“Error: 1.1 No Internet\n”作为参数传给sub\_40105F函数，推断函数sub\_0x40105F的作用是打印字符串。



### 3.问题解答

(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？

位于0x00401000处的if语句。

(2) 位于0x40105F的子过程是什么？

推测是printf打印过程。

(3) 这个程序的目的是什么？

该函数会检查是否存在一个可用的Internet连接：如果存在，打印相应字符串结果并返回1，否则返回0，从而确定设备是否可以联网。

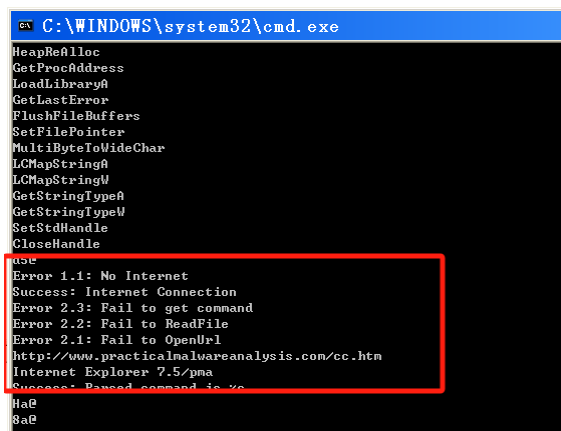
## (二) Lab6-2

# 1.静态分析

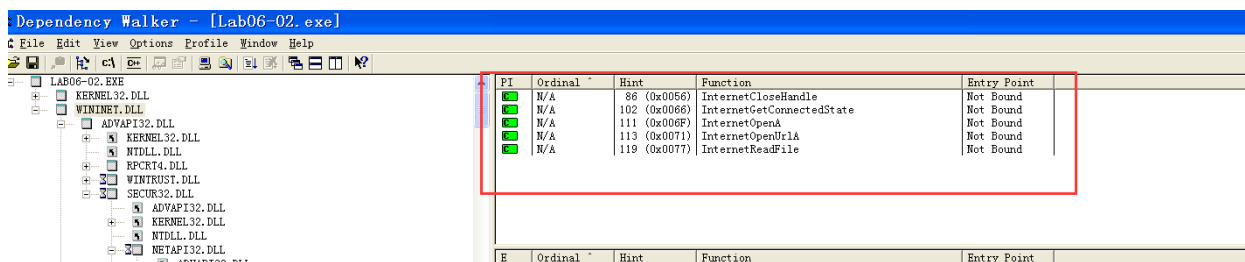
使用PEiD打开文件，可以看到文件未加壳。



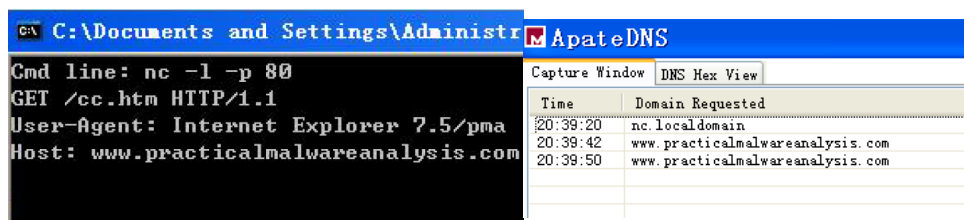
接着使用Strings查看恶意代码的字符串列表，发现<http://www.practicalmalwareanalysis.com/cc.htm>可以作为检测特征；同时，还看到三条错误信息，推测该恶意代码会打开网页并解析指令。



使用Dependency Walker打开文件，查看其导入函数。可以看到其调用了wininet.dll，导入的函数均为联网相关的操作。



下面利用netcat与ApateDNS进行监测，可以看到恶意代码向上面提到的url发送请求。



## 2.动态分析

使用IDA打开文件，可以看到main函数的起始地址是0x00401040。调用的第一个子过程为sub\_401000函数，main函数还调用了两个没有在Lab6-1中出现的方法：0x00401040 和0x0040117F，在0x0040117F这个新的调用前，有两个参数被压入栈，其中之一是一个格式化字符串“Sucess: Pased command is %c\n”，另一个参数是从前面对0x00401040 返回字符，像%c 和%d 这样的格式化字符串，可以推断在0x0040117F处调用了printf，printf 会打印该字符串，并把其中的%c替换成另一个被压入栈的参数。

```
.text:00401130
.text:00401130 55
.text:00401131 8B EC
.text:00401133 83 EC 08
.text:00401136 E8 C5 FE FF FF
.text:0040113B 89 45 FC
.text:0040113E 83 7D FC 00
.text:00401142 75 04
.text:00401144 33 C0
.text:00401146 EB 33
.text:00401148
.text:00401148
.text:00401148
.text:00401148 E8 F3 FE FF FF
.text:0040114D 88 45 F8
.text:00401150 0F BE 45 F8
.text:00401154 85 C0
.text:00401156 75 04
.text:00401158 33 C0
.text:0040115A EB 1F
.text:0040115C
.text:0040115C
.text:0040115C
.text:0040115C 0F BE 4D F8
.text:00401161 68 10 71 40 00
.text:00401166 E8 14 00 00 00
.text:0040116B 83 C4 08
.text:0040116E 68 60 EA 00 00
.text:00401173 FF 15 00 60 40 00
.text:00401179 33 C0
.text:0040117B
.text:0040117B
.text:0040117B

push ebp
mov ebp, esp
sub esp, 8
call sub_401000
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jnz short loc_401148
xor eax, eax
jmp short loc_40117B

loc_401148:
call sub_401040 ; CODE XREF: _main+127j
mov [ebp+var_0], eax
movsx eax, [ebp+var_8]
test eax, eax
jnz short loc_40115C
xor eax, eax
jmp short loc_40117B

loc_40115C:
movsx ecx, [ebp+var_8] ; CODE XREF: _main+267j
push ecx
push offset aSucessPasedC ; "Sucess: Pased
call sub_40117F
add esp, 8
push 0EA60h ; dwMilliseconds
call ds:Sleep
xor eax, eax

loc_40117B:
; CODE XREF: _main+167j
; _main+2A7j
```

接下来观察对0x00401040的调用，该函数包含了对我们在静态分析中发现的所有WinINet API的调用：首先调用了InternetOpen，以初始化对WinNet库的使用，接下来调用InternetOpenUrl，来打开位于压入栈参数的静态网页，这个函数会引发在动态分析时看到的DNS请求，而InternetCloseHandle函数作用是关闭一个网络句柄。

```
.text:00401040
.text:00401041 8B EC
.text:00401043 81 EC 10 02 00 00
.text:00401049 6A 00
.text:0040104B 6A 00
.text:0040104D 6A 00
.text:0040104F 6A 00
.text:00401051 68 F4 70 40 00
.text:00401056 FF 15 C4 60 40 00
.text:0040105C 89 45 F4
.text:0040105F 6A 00
.text:00401061 6A 00
.text:00401063 6A 00
.text:00401065 6A 00
.text:00401067 68 C4 70 40 00
.text:0040106C 8B 45 F4
.text:0040106F 50
.text:00401070 FF 15 B4 60 40 00
.text:00401076 89 45 F0
.text:00401079 83 7D F0 00
.text:0040107D 75 1E
.text:0040107F 68 A8 70 40 00
.text:00401084 E8 F6 00 00 00
.text:00401089 83 C4 04
.text:0040108C 8B 4D F4
.text:0040108F 51
.text:00401090 FF 15 B8 60 40 00
.text:00401096 32 C0
.text:00401098 E9 8F 00 00 00

push ebp
mov ebp, esp
sub esp, 210h
push 0 ; dwFlags
push 0 ; lpszProxyBypass
push 0 ; lpszProxy
push 0 ; dwAccessType
push offset szAgent ; "Internet Explorer 7.5/pna"
call ds:InternetOpen
mov [ebp+hInternet], eax
push 0 ; dwContext
push 0 ; dwFlags
push 0 ; dwHeadersLength
push 0 ; lpszHeaders
push offset szUrl ; "http://www.practicalmalwareanalysis.
mov [ebp+hInternet], eax
push 0 ; dwInternet
call ds:InternetOpenUrlA
[ebp+hFile], eax
cmp [ebp+hFile], 0
jnz short loc_40109D
push offset aError2_FailTo ; "Error 2.1: Fail to OpenUrl\n"
call sub_40117F
add esp, 4
mov ecx, [ebp+hInternet]
push ecx ; hInternet
call ds:InternetCloseHandle
xor esi, esi
jmp loc_40112C
```

上面InternetOpenUrlA的返回结果被赋给了hFile，并与0进行比较，如果不为0，hFile变量会被传给下一个函数，也就是InternetReadFile，hFile变量实际上是一个句柄，而这个句柄是用于访问URL。

InternetReadFile用于从InternetOpenUrlA打开网页中读取内容，第二个参数buffer是一个保存数据的数组，最多读取0x200字节的数据。而已经知道这个函数是用来读取一个HTML网页的，故可以认为Buffer是一个字符数组，调用该函数之后检查返回值是否为0，如果为0则关闭该函数句柄并终止，否则，代码会马上将buffer逐一地每次与一个字符进行比较，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。

```

.text:0040109D
.text:0040109D
.text:0040109D 8D 55 F8
.text:004010A0 52
.text:004010A1 68 00 02 00 00
.text:004010A6 8D 85 F0 FD FF FF
.text:004010AC 50
.text:004010AD 8B 4D F0
.text:004010B0 51
.text:004010B1 FF 15 BC 60 40 00
.text:004010B7 89 45 FC
.text:004010BA 83 7D FC 00
.text:004010BE 75 25
.text:004010C0 68 88 70 40 00
.text:004010C5 E8 B5 00 00 00
.text:004010CA 83 C4 04
.text:004010CD 8B 55 F4
.text:004010D0 52
.text:004010D1 FF 15 B8 60 40 00
.text:004010D7 8B 45 F0
.text:004010DA 50
.text:004010DB FF 15 B8 60 40 00
.text:004010E1 32 C0
.text:004010E3 EB 47

loc_40109D:
lea     edx, [ebp+dwNumberOFBytesRead]
push    edx
push    200h
lea     eax, [ebp+Buffer]
push    eax
mov     ecx, [ebp+hFile]
push    ecx
call    ds:InternetReadFile
mov     [ebp+var_4], ecx
cmp     [ebp+var_4], 0
jnz     short loc_4010E5
push    offset aError2_2FailTo ; "Error 2.2: Fail to
call    sub_40117F
add     esp, 4
mov     edx, [ebp+hInternet]
push    edx
call    ds:InternetCloseHandle
mov     eax, [ebp+hFile]
push    eax
call    ds:InternetCloseHandle
xor     al, al
jmp     short loc_40112C

```

有一条cmp指令来检查第一字符是否等于0x3C，对应的ASCII 字符是<，类似的后面的21h、2Dh 和2Dh，将这些字符合并起来就是<!--，它是HTML 中注释开始的部分。同时注意到buffer以及后面的几个var\_\*，事实上var\_\*应当是一个偏移量，但是IDA Pro没有识别出来Buffer是512字节。

```

.text:004010E5
.text:004010E5
.text:004010E5 0F BE 8D F0 FD FF FF
.text:004010EC 83 F9 3C
.text:004010EF 75 2C
.text:004010F1 0F BE 95 F1 FD FF FF
.text:004010F8 83 FA 21
.text:004010FB 75 20
.text:004010FD 0F BE 85 F2 FD FF FF
.text:00401104 83 F8 2D
.text:00401107 75 14
.text:00401109 0F BE 8D F3 FD FF FF
.text:00401110 83 F9 2D
.text:00401113 75 08
.text:00401115 8A 85 F4 FD FF FF
.text:0040111B EB 0F

loc_4010E5:
movsx   ecx, [ebp+Buffer]
cmp     ecx, 3Ch
jnz     short loc_40111D
movsx   edx, [ebp+var_20F]
cmp     edx, 21h
jnz     short loc_40111D
movsx   eax, [ebp+var_20E]
cmp     eax, 2Dh
jnz     short loc_40111D
movsx   ecx, [ebp+var_20D]
cmp     ecx, 2Dh
jnz     short loc_40111D
mov     al, [ebp+var_20C]
jmp     short loc_40112C

```

接着按下Ctrl+K，可以看到栈变量中的数据。

```

00000210
-00000210 Buffer db ?
-0000020F var_20F db ?
-0000020E var_20E db ?
-0000020D var_20D db ?
-0000020C var_20C db ?
-0000020B db ?
-0000020A db ?

```

右键设置Array size为512，可以得到参数信息。

```

-00000210
-00000210 Buffer db 512 dup(?)
-00000210 hFile dd ? ; offset
-0000020C hInternet dd ? ; offset
-00000208 dwNumberOFBytesRead dd ?
-00000204 var_4 dd ?
+00000200 5 db 4 dup(?)
+00000204 r db 4 dup(?)
+00000208 ; end of stack variables

```

原来的var\_\*也自动变成了buffer加上一个偏移量。所以图中这段内容就是比较buffer[0:3] 的内容是否为注释开头“<!--”，如果是，则将Buffer[4]的内容写入al。

```

0010E5:
movsx   ecx, [ebp+Buffer]
cmp     ecx, 3Ch
jnz     short loc_40111D
movsx   edx, [ebp+Buffer+1]
cmp     edx, 21h
jnz     short loc_40111D
movsx   eax, [ebp+Buffer+2]
cmp     eax, 2Dh
jnz     short loc_40111D
movsx   ecx, [ebp+Buffer+3]
cmp     ecx, 2Dh
jnz     short loc_40111D
mov     al, [ebp+Buffer+4]
jmp     short loc_40112C

-00000210 ; Use data definition commands to create local variables and function arguments.
-00000210 ; Two special fields "r" and "s" represent return address and saved registers.
-00000210 ; Frame size: 210; Saved regs: 4; Purge: 0
-00000210
-00000210 Buffer db 512 dup(?)
-00000210 hFile dd ? ; offset
-0000020C hInternet dd ? ; offset
-00000208 dwNumberOFBytesRead dd ?

```

接着回到主函数中sub\_401040处，继续向下执行，可以看到al的值赋给了eax，并判断eax是否为0，若非0（也即buffer[4]的字符有意义），则跳到loc\_40115C，然后打印eax对应的字符“Success: Parsed command is %c\n”，其中“%c”就是Buffer[4]转换得到的字符。最后休眠（Sleep），传入的参数0EA60h = 60000毫秒，即60秒。

```
.text:00401148  
.text:00401148  
.text:00401148  
.text:00401148 E8 F3 FE FF FF  
.text:0040114D 88 45 F8  
.text:00401150 0F BE 45 F8  
.text:00401154 85 C0  
.text:00401156 75 04  
.text:00401158 33 C0  
.text:0040115A EB 1F  
.text:0040115C  
.text:0040115C  
.text:0040115C  
.text:0040115C 0F BE 4D F8  
.text:00401160 51  
.text:00401161 68 10 71 40 00  
.text:00401166 E8 14 00 00 00  
.text:0040116B 83 C4 08  
.text:0040116E 68 60 EA 00 00  
.text:00401173 FF 15 00 60 40 00  
.text:00401179 33 C0  
; -----  
loc_401148: ; CODE XREF: _main+12↑  
call sub_401040  
mov [ebp+var_8], al  
movzx ecx, [ebp+var_8]  
test eax, eax  
jnz short loc_40115C  
xor eax, eax  
jmp short loc_40117B  
; -----  
loc_40115C: ; CODE XREF: _main+26↑  
movsx ecx, [ebp+var_8]  
push ecx  
push offset aSuccessParsedC ; "Success: Par  
call sub_40117F  
add esp, 8  
push 0EA60h ; dwMilliseconds  
call ds:Sleep  
xor eax, eax
```

### 3.问题解答

(1) main函数调用的第一个子过程执行了什么操作？位于0x40105F的子过程是什么？

与Lab6-1一样，是一个if语句，检查是否存在可用的Internet连接。

(2) 位于0x40117F的子过程是什么？

printf。

(3) 被main函数调用的第二个子过程做了什么？

这一部分的内容应该是尝试使用7.5版本的浏览器打开<http://www.practicalmalwareanalysis.com>这个url，然后查看是否打开成功：如果不成功就返回一条错误信息；之后不论成功与否都把刚刚打开的网络连接给关闭。

(4) 在这个子过程中使用了什么类型的代码结构？

字符数组和if结构。

(5) 在这程序中有任何基于网络的指示吗？

有。

InternetOpen 中使用User-Agent: “Internet Explorer 7.5/pma”，InternetOpenUrl从远程主机下载文件：<http://www.practicalmalwareanalysis.com/cc.htm>。

(6) 这个恶意代码的目的是什么？

恶意代码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；如果存在，则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，程序解析之后的那个字符，进行逐字比对，并打印“Success: Parsedcommandis %c\n”，其中%c就是从该字符。如果解析成功，程序会休眠60秒，然后终止运行。



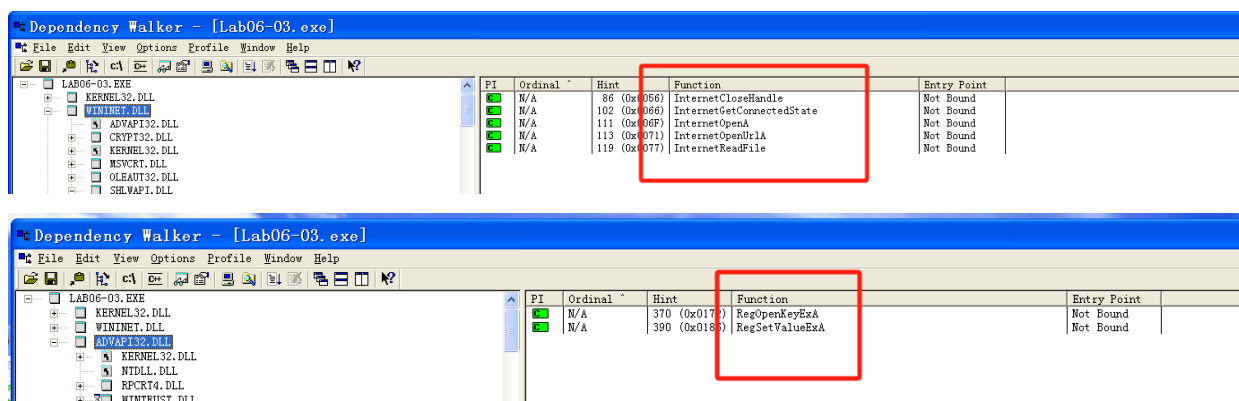
## (三) Lab6-3

### 1.静态分析

使用Strings查看文件字符串，可以看到除了Lab 6-2中出现的和网络请求相关的部分，还多出了注册表和命令Software\Microsoft\Windows\CurrentVersion\Run是注册表中一个常用的autorun位置，C:\Temp\cc.exe则是一个目录和文件名，也许是一个有效特征，推测可能要读写注册表，并执行远程下载的恶意程序。

```
C:\WINDOWS\system32\cmd.exe
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle
T6E
Error 1.1: No Internet
Success: Internet Connection
Error 2.3: Fail to get command
Error 2.2: Fail to ReadFile
Error 2.1: Fail to OpenUrl
http://www.practicalmalwareanalysis.com/cc.htm
Internet Explorer 7.5/pma
Error 3.2: Not a valid command provided
Error 3.1: Could not set Registry value
Malware
Software\Microsoft\Windows\CurrentVersion\Run
C:\Temp\cc.exe
C:\Temp
Success: Parsed command is %c
'aE
PaE
aE
```

接着使用Dependency Walker查看导入函数，wininet.dll中InternetGetConnectedState、InternetOpen、InternetOpenUrl、InternetReadFile 和InternetCloseHandle，同Lab6-2类似。advapi32.dll中有RegOpenKeyEx 和RegSetValueEx，一起用于向注册表插入信息，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。



### 2.动态分析与问题解答

(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？

用IDA来加载这个可执行文件，其main函数看起来与Lab6-2很像，但多了一个0x401130的调用，其他部分，包括0x401000（检查Internet 连接）、0x401040（下载网页并解析HTML 注释的调用则与Lab6-2中的一致。



```

.text:00401228
.text:00401228
.text:00401228
.text:00401228 E8 13 FE FF FF
.text:0040122D 88 45 F8
.text:00401230 0F BE 45 F8
.text:00401234 95 C0
.text:00401236 75 04
.text:00401238 33 C0
.text:0040123A EB 31
.text:0040123C
.text:0040123C
.text:0040123C
.text:0040123C 0F BE 4D F8
.text:00401240 51
.text:00401241 68 88 71 40 00
.text:00401246 E8 26 00 00 00
.text:0040124B 83 C4 08
.text:0040124E 80 55 0C
.text:00401251 80 02
.text:00401253 50
.text:00401254 8A 4D F8
.text:00401257 51
.text:00401258 E8 D3 FE FF FF
.text:0040125D 83 C4 08
.text:00401260 68 60 EA 00 00
.text:00401265 FF 15 30 60 40 00
.text:0040126B 33 C0

loc_401228:
call sub_401040 ; CODE XREF: _main+12fj
mov [ebp+var_8], al
movsx eax, [ebp+var_8]
test eax, eax
jnz short loc_40123C
xor eax, eax
jmp short loc_40126D

loc_40123C:
movsx ecx, [ebp+var_8] ; CODE XREF: _main+26fj
push ecx
push offset aSuccessParsedC ; "Success: Parsed c
call sub_401271
add esp, 8
mov edx, [ebp+argv]
mov eax, [edx]
push eax ; lpExistingFileName
mov cl, [ebp+var_8] ; char
call sub_401130 ; char
add esp, 8
push 0EA60h ; dwMilliseconds
call ds:Sleep
xor eax, eax

```

## (2) 这个新的函数使用的参数是什么？

传入的第一个参数是char类型，即此前读出的HTML字符。第二个参数是指向文件名字符串的指针（实际上是标准main函数的argv[0]，即该程序自己的文件名）。

```

.text:00401130
.text:00401130 55
.text:00401131 8B EC
.text:00401133 83 EC 08
.text:00401136 0F BE 45 08
.text:0040113A 89 45 F8
.text:0040113D 8B 4D F8
.text:00401140 83 E9 61
.text:00401143 89 4D F8
.text:00401146 83 7D F8 04
.text:0040114A 0F 87 91 00 00 00
.text:00401150 8B 55 F8
.text:00401153 FF 24 95 F2 11 40 00
.text:0040115A

push ebp
mov ebp, esp
sub esp, 8
movsx eax, [ebp+arg_0]
mov [ebp+var_8], eax
mov ecx, [ebp+var_8]
sub ecx, 61h
mov [ebp+var_8], ecx
cmp [ebp+var_8], 4 ; switch 5 cases
ja loc_4011E1 ; jumtable 00401153 default case
mov edx, [ebp+var_8]
jmp ds:off_4011F2[edx*4] ; switch jump

```

## (3) 这个函数包含的主要代码结构是什么？

双击进入函数，进一步查看。arg\_0是IDA自动生成的标签，表示第一个参数（最后一个压入栈中的参数），将arg\_0的值赋给var\_8，将var\_8自减61h（对应ASCII字符'a'），若该字符减'a'大于4（非'a'、'b'、'c'、'd'、'e'），则跳到loc\_4011E1，否则，将该值赋给edx，进入switch语句。下图所示为switch判断结构：



off\_4011F2对应一张跳转表('a'~'e'的分支)上loc\_4011E1（default分支），共有6个分支。

```

.text:004011F1 ;
.text:004011F2 5A 11 40 00 6C 11 40 00+off_4011F2 dd offset loc_40115A ; DATA XREF:
.text:004011F2 7F 11 40 00 8C 11 40 00+ dd offset loc_40116C ; jump table
.text:004011F2 04 11 40 00 dd offset loc_40117F
.text:004011F2 dd offset loc_40118C
.text:004011F2 dd offset loc_4011D4
.text:00401206 CC CC CC CC CC CC CC CC+ align 10h

```

#### (4) 这个函数能够做什么？

由上述分析可知，该函数最重要的部分即为switch部分，下面就该部分进行重点分析。

- case 0，即字符为'a'。调用CreateDirectory创建了一个文件夹"C:\Temp"。

```

.text:0040115A
.text:0040115A
.text:0040115A loc_40115A: ; CODE XREF: sub_401130+231j
.text:0040115A 6A 00 push 0 ; DATA XREF: .text:off_4011F2↓0
.text:0040115C 68 00 71 40 00 push offset PathName ; "C:\\Temp"
.text:00401161 FF 15 0C 60 40 00 call ds:CreateDirectoryA
.text:00401167 E9 82 00 00 00 jmp loc_4011EE

```

- case 1，即字符串为'b'。调用CopyFile复制文件：源文件是lpExistingFileName，前文提过是argv[0]，也即该程序自己的文件名“Lab06-03.exe”；目标文件是“C:\Temp\cc.exe”。即该分支将Lab06-03.exe复制到C:\Temp\cc.exe。

```

.text:0040116C
.text:0040116C loc_40116C: ; CODE XREF: sub_401130+231j
.text:0040116C 6A 01 push 1 ; DATA XREF: .text:off_4011F2↓0
.text:0040116E 68 00 71 40 00 push offset Data ; "C:\\Temp\\cc.exe"
.text:00401173 8B 45 0C mov eax, [ebp+lpExistingFileName]
.text:00401176 50 push eax ; lpExistingFileName
.text:00401177 FF 15 14 60 40 00 call ds:CopyFileA
.text:0040117D EB 6F jmp short loc_4011EE

```

- case 3,即字符串为'd'，首先调用RegOpenKeyEx打开注册表键“Software\Microsoft\Windows\CurrentVersion\Run”，然后再在该键下创建一个新的键“...\Malware”，其值为“C:\Temp\cc.exe”。这样系统启动时，如果C:\Temp\cc.exe存在，则也会跟随系统启动，自动运行。

```

.text:0040118C
.text:0040118C loc_40118C: ; CODE XREF: sub_401130+231j
.text:0040118C 8D 40 FC lea ecx, [ebp+hKey] ; DATA XREF: .text:off_4011F2↓0
.text:0040118F 51 push ecx ; jump table 00401153 case 3
.text:00401190 68 3F 00 0F 00 push ecx ; phkResult
.text:00401195 6A 00 push 0 ; sanDesired
.text:00401197 68 70 71 40 00 push offset SubKey ; "Software\Microsoft\Windows\CurrentVe"
.text:0040119C 68 02 00 00 80 push 80000002h ; hKey
.text:004011A1 FF 15 04 60 40 00 call ds:RegOpenKeyExA
.text:004011A7 6A 0F push 0Fh ; cbData
.text:004011A9 68 00 71 40 00 push offset Data ; "C:\\Temp\\cc.exe"
.text:004011AE 6A 01 push 1 ; dwType
.text:004011B0 6A 00 push 0 ; Reserved
.text:004011B2 68 68 71 40 00 push offset ValueName ; "Malware"
.text:004011B7 8B 55 FC mov edx, [ebp+hKey]
.text:004011BA 52 push edx ; hKey
.text:004011BB FF 15 00 60 40 00 call ds:RegSetValueExA
.text:004011C1 85 C0 test eax, eax
.text:004011C3 74 00 jz short loc_4011D2
.text:004011C5 68 3C 71 40 00 push offset aError3_1CouldN ; "Error 3.1: Could not set Registry
.text:004011CA E8 A2 00 00 00 call sub_401271
.text:004011CF 83 C4 04 add esp, 4
.text:004011D2 EB 1A jmp short loc_4011EE

```

- case 4，即字符串为'e'。调用Sleep休眠186A0h=100000毫秒，也即100秒。

```

.text:0040117D EB 0F jmp short loc_4011EE
.text:0040117F
.text:0040117F loc_40117F: ; CODE XREF: sub_401130+231j
.text:0040117F 68 A0 71 40 00 push offset Data ; DATA XREF: .text:off_4011F2↓0
.text:00401184 FF 15 28 60 40 00 call ds>DeleteFileA
.text:0040118A EB 62 jmp short loc_4011EE

```

- default case，即字符串非'a'~'d'。则调用sub\_401271，打印字符串“Error 3.2: Not a valid command provided”。

```

00000000 .text:00A0118C ; CODE XREF: sub_401130+23↑j
00000001 .text:00A0118C ; DATA XREF: .text:off_4011F2↓o
00000002 .text:00A0118C loc_40118C:
00000003 .text:00A0118C 8D 4D FC lea     ecx, [ebp+hKey] ; jumpTable 00A01153 case 3
00000004 .text:00A0118F 51 push    ecx                    ; phkResult
00000005 .text:00A01190 68 3F 00 0F 00 push   0F003Fh                ; samDesired
00000006 .text:00A01195 6A 00 push    0                     ; u1Options
00000007 .text:00A01197 68 70 71 40 00 push   offset SubKey          ; "Software\Microsoft\Windows\CurrentUser...
00000008 .text:00A0119C 68 02 00 00 80 push   80000002h              ; hkey
00000009 .text:00A011A1 FF 15 04 60 40 00 call    ds:RegOpenKeyExh      ; cbData
0000000A .text:00A011A7 6A 0F push    0Fh                   ; cbData
0000000B .text:00A011A9 68 A0 71 40 00 push   offset Data            ; "C:\Temp\cc.exe"
0000000C .text:00A011AE 6A 01 push    1                      ; dwType
0000000D .text:00A011B0 6A 00 push    0                      ; Reserved
0000000E .text:00A011B2 68 68 71 40 00 push   offset ValueName       ; "Malware"
0000000F .text:00A011B7 8B 55 FC mov     edx, [ebp+hKey]        ; hkey
00000010 .text:00A011BA 52 push    edx                    ; hkey
00000011 .text:00A011BB FF 15 00 60 40 00 call    ds:RegSetValueExh
00000012 .text:00A011C1 85 C0 test    eax, eax
00000013 .text:00A011C3 74 00 jz      short loc_4011D2
00000014 .text:00A011C5 68 3C 71 40 00 push   offset aError3_1CouldN ; "Error 3.1: Could not set Registry value"...
00000015 .text:00A011CA E8 A2 00 00 00 call    sub_401271
00000016 .text:00A011CF 83 C4 04 add     esp, 4
00000017 .text:00A011D2
00000018 .text:00A011D2 loc_4011D2: ; CODE XREF: sub_401130+93↑j
00000019 .text:00A011D4 EB 1A jmp     short loc_4011EE

```

### (5) 这个恶意代码中有什么本地特征吗?

注册表键Software\Microsoft\Windows\CurrentVersion\Run\Malware和本地文件C:\Temp\cc.exe都可以作为其本地特征。

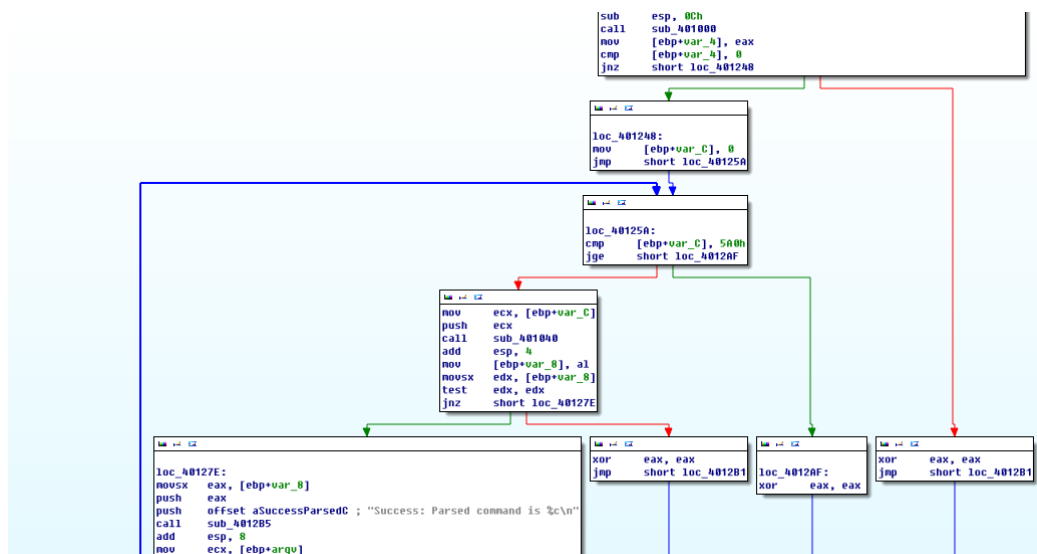
### (6) 这个恶意代码的目的是什么？

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。

#### (四) Lab6-4

## 1. 动态分析

用IDA打开文件，使用图视图打开，可以看到有一个明显的循环结构。



局部变量var\_C用于循环计数，这个计数器初始化为0，每次跳回0x401251处递增，在0x40125A处检查，如果计数器var\_C大于或等于5A0h=1440时，跳出循环，否则继续执行，将var\_C压入栈，调用0x401040，然后执行Sleep休眠1分钟，最后计数器加1。上述过程会执行1440分钟，也就是24小时。

.text:00401248	loc_401248:	; CODE XREF: _main+127j
.text:00401248 C7 45 F4 00 00 00	mov	[ebp+var_C], 0
.text:0040124F EB 09	jmp	short loc_40125A
.text:00401251		
.text:00401251	loc_401251:	; CODE XREF: _main+7D4j
.text:00401251 8B 45 F4	mov	eax, [ebp+var_C]
.text:00401254 83 C0 01	add	eax, 1
.text:00401257 89 45 F4	mov	[ebp+var_C], eax
.text:0040125A	loc_40125A:	; CODE XREF: _main+1F7j
.text:0040125A 81 7D F4 A0 05 00 00	cmp	[ebp+var_C], 5A0h
.text:00401261 7D 4C	jge	short loc_4012AF
.text:00401263 8B 4D F4	mov	ecx, [ebp+var_C]
.text:00401266 51	push	ecx
.text:00401267 E8 D4 FD FF FF	call	sub_401040
.text:0040126C 83 C4 04	add	esp, 4
.text:0040126F 8B 45 F8	mov	[ebp+var_8], al
.text:00401272 0F BE 55 F8	movsx	edx, [ebp+var_8]
.text:00401276 85 D2	test	edx, edx
.text:00401278 75 04	jnz	short loc_40127E
.text:0040127A 33 C0	xor	eax, eax
.text:0040127C EB 33	jmp	short loc_4012B1
.text:0040127E		
.text:0040127E	loc_40127E:	; CODE XREF: _main+487j
.text:0040127E 0F BE 45 F8	movsx	eax, [ebp+var_8]
.text:00401282 50	push	eax
.text:00401283 68 BC 71 40 00	push	offset aSuccessParsedC ; "Success: Parsed command is %c\r
.text:00401288 E8 28 00 00 00	call	sub_4012B5
.text:0040128D 83 C4 08	add	esp, 8
.text:00401290 8B 4D 0C	mov	ecx, [ebp+argv]
.text:00401293 8B 11	mov	edx, [ecx]
.text:00401295 52	push	edx ; lpExistingFileName
.text:00401296 8A 45 F8	mov	al, [ebp+var_8]
.text:00401299 50	push	eax ; char
.text:0040129A E8 B1 FE FF FF	call	sub_401150
.text:0040129F 83 C4 08	add	esp, 8
.text:004012A2 68 60 EA 00 00	push	0EA60h ; dwMiliiseconds

在这里，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（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，然后可以看到sprintf被调用了，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。

.text:00401040	push	ebp
.text:00401040 55	mov	ebp, esp
.text:00401041 8B EC	sub	esp, 230h
.text:00401043 81 EC 30 02 00 00	mov	eax, [ebp+arg_0]
.text:00401049 8B 45 08	push	eax
.text:0040104C 50	push	offset aInternetExplor ; "Internet Explorer 7.50/pna%c
.text:0040104D 68 F4 70 40 00	lea	ecx, [ebp+szAgent]
.text:00401052 8D 4D 08	push	ecx ; char *
.text:00401055 51	call	_sprintf
.text:00401056 E8 88 02 00 00	add	esp, 0Ch
.text:00401058 83 C4 0C	push	0 ; dwFlags
.text:0040105E 6A 00	push	0 ; lpszProxyBypass
.text:00401060 6A 00	push	0 ; lpszProxy
.text:00401062 6A 00	push	0 ; dwAccessType
.text:00401064 6A 00	lea	edx, [ebp+szAgent]
.text:00401066 8D 55 D8	push	edx ; lpszAgent
.text:00401069 52	call	ds:InternetOpenA
.text:0040106A FF 15 DC 60 40 00	mov	[ebp+hInternet], eax
.text:00401070 89 45 D4	push	0 ; dwContext
.text:00401073 6A 00	push	0 ; dwFlags
.text:00401075 6A 00	push	0 ; dwHeadersLength
.text:00401077 6A 00	push	0 ; lpszHeaders
.text:00401079 6A 00	push	offset szUrl ; "http://www.practicalmalwareanalysis
.text:0040107B 68 C4 70 40 00	mov	eax, [ebp+hInternet]
.text:00401080 8B 45 D4	push	eax ; hInternet
.text:00401083 50	call	ds:InternetOpenUrlA
.text:00401084 FF 15 CC 60 40 00	mov	[ebp+hFile], eax
.text:0040108A 89 45 D0	mov	[ebp+hFile], 0
.text:0040108D 83 7D D0 00	cmp	[ebp+hFile], 0
.text:00401091 75 1E	jnz	short loc_4010B1
.text:00401093 68 A8 70 40 00	push	offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\r
.text:00401098 E8 18 02 00 00	call	sub_4012B5
.text:0040109D 83 C4 04	add	esp, 4
.text:004010A0 8B 4D D4	mov	ecx, [ebp+hInternet]
.text:004010A3 51	push	ecx ; hInternet
.text:004010A4 FF 15 D0 60 40 00	call	ds:InternetCloseHandle
.text:004010AA 32 C0	xor	al, al
.text:004010AC E9 8F 00 00 00	jmp	loc_401140

## 2.问题解答

### (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？

在main函数中增加了一个循环，这个循环总共循环1440次，循环体内部是睡眠60秒，共计睡眠24h；同时本次的usr-agent会随着循环体计数器的变化而变化。

## (2) 什么新的代码结构已经被添加到 main 中?

for循环。

## (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?

本次对HTML进行解析时，usr-agent不像之前那样是固定的，而是会随着加入的for循环的计数器发生变化。

## (4) 这个程序会运行多久? (假设它已经连接到互联网。)

至少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?

本次的usr-agent会发生变化。

## (6) 这个恶意代码的目的是什么?

该恶意代码会使用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，会被读到一个字符数组里，并与“<!--”——进行比较，然后从注释中抽取下一个字符，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100s。该程序会运行1400分钟后终止。

## (五) yara规则

基于上述分析，借助yargen工具，编写得到如下yara规则：

```
1 rule lab0601
2 {
3   strings:
4     $string1 = "Error 1.1: No Internet"
5     $string2 = "Success: Internet Connection"
6     $string3 = "InternetGetConnectedState"
7   condition:
8     filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
9     0x00004550 and all of them
10  }
11 rule lab0602
12 {
13   strings:
14     $string1 = "http://www.practicalmalwareanalysis.com/cc.htm"
15     $string2 = "Error 2.3: Fail to get command"
16     $string3 = "Internet Explorer 7.5/pma"
17   condition:
18     filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
19     0x00004550 and all of them
20  }
21 rule lab0603
22 {
23   strings:
```

```

22     $string1 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
23     $string2 = "C:\\Temp\\cc.exe"
24     $string3 = "C:\\Temp"
25     condition:
26         filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
27 }
28 rule lab0604
29 {
30     strings:
31         $string1 = "Success: Parsed command is %c"
32         $string2 = "DDDDDDDDDDDDDD"
33     condition:
34         filesize < 100KB and uint16(0) == 0x5A4D and uint16(uint16(0x3C)) ==
0x00004550 and all of them
35 }

```

下面是运行结果图。

```

PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara> .\yara64.exe .\lab5.y .\Lab06-01.exe
lab0601 .\Lab06-01.exe
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara> .\yara64.exe .\lab5.y .\Lab06-02.exe
lab0601 .\Lab06-02.exe
lab0602 .\Lab06-02.exe
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara> .\yara64.exe .\lab5.y .\Lab06-03.exe
lab0601 .\Lab06-03.exe
lab0602 .\Lab06-03.exe
lab0603 .\Lab06-03.exe
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara> .\yara64.exe .\lab5.y .\Lab06-04.exe
lab0601 .\Lab06-04.exe
lab0603 .\Lab06-04.exe

```

下面测试其运行效率，得到如下运行结果。

```

PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara> .\yara64.exe .\lab5.y .\Lab06-01.exe
文件 .\yara\Chapter_6L\Lab06-01.exe 匹配的规则: [lab0601]
文件 .\yara\Chapter_6L\Lab06-02.exe 匹配的规则: [lab0601, lab0602]
文件 .\yara\Chapter_6L\Lab06-03.exe 匹配的规则: [lab0601, lab0602, lab0603]
文件 .\yara\Chapter_6L\Lab06-04.exe 匹配的规则: [lab0601, lab0603]
程序运行时间: 0.025783300399780273 秒
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\恶意代码分析与防治技术 王志, 邓琮弋\Malware_Analysis_and_Prevention_Techniques\lab5\yara>

```

## (六) IDA Python脚本编写

遍历所有函数，排除库函数或简单跳转函数，当反汇编的助记符为call或者jmp且操作数为寄存器类型时，输出该行反汇编指令。

```

1 import idutils
2 for func in idutils.Functions():
3     flags = idc.GetFunctionFlags(func)
4     if flags & FUNC_LIB or flags & FUNC_THUNK:
5         continue
6     dism_addr = list(idutils.FuncItems(func))
7     for line in dism_addr:
8         m = idc.GetMnem(line)
9         if m == 'call' or m == 'jmp':
10             op = idc.GetOpType(line,0)
11             if op == o_reg:
12                 print '0x%x %s' % (line,idc.GetDisasm(line))

```

得到如下结果：

```
-----
Type library 'uc6win' loaded. Applying types
Types applied to 0 names.
Using FLIRT signature: Microsoft VisualC 2-1
Propagating type information...
Function argument information has been propa
The initial autoanalysis has been finished.
0x403e4a call    esi ; VirtualFree
0x403ea5 call    esi ; VirtualFree
0x404415 call    ebp ; VirtualAlloc
0x40442f call    ebp ; VirtualAlloc
```

## 五、实验结论及心得

---

1. 熟悉了静态与动态结合分析病毒的方法；
2. 更加熟悉了yara规则的编写。