

# 个人信息

---

姓名：付文轩

学号：1911410

专业：信息安全

## Lab 11-1

---

### 问题1

---

这个恶意代码向磁盘释放了什么？

首先依旧是对这个样本进行一个简单的静态分析，查看一下字符串

C:\WINDOWS\system32\cmd.exe

```
HeapCreate
VirtualFree
RtlUnwind
HeapAlloc
HeapReAlloc
SetStdHandle
FlushFileBuffers
SetFilePointer
CreateFileA
GetCPInfo
GetACP
GetOEMCP
GetProcAddress
LoadLibraryA
SetEndOfFile
ReadFile
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
dTe
TGAD
BINARY
GinaDLL
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
msgina32.dll
\msgina32.dll
Xq@
Hq@
@
xs@
Ls@
lr@
4r@
@
@
<<<<<
H
@
'y!
@~
[
Q^ _j2
BINARY
TGAD
!This program cannot be run in DOS mode.
>
Rich<
.text
.rdata
.data
.reloc
```

Bluetooth 网络连接

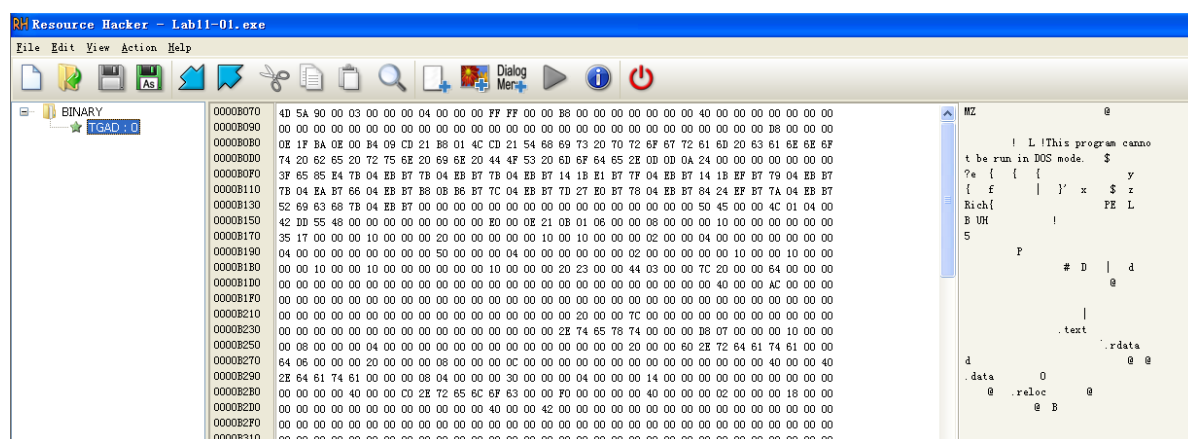
```
C:\WINDOWS\system32\cmd.exe

WlxGetConsoleSwitchCredentials
WlxGetStatusMessage
WlxInitialize
WlxIsLockOk
WlxIsLogoffOk
WlxLoggedOnSAS
WlxLoggedOutSAS
WlxLogoff
WlxNegotiate
WlxNetworkProviderLoad
WlxReconnectNotify
WlxRemoveStatusMessage
WlxScreenSaverNotify
WlxShutdown
WlxStartApplication
WlxWkstaLockedSAS
\MSGina
ShellShutdownDialog
WlxActivateUserShell
WlxDisconnectNotify
WlxDisplayLockedNotice
WlxDisplaySASNotice
WlxDisplayStatusMessage
WlxGetConsoleSwitchCredentials
WlxGetStatusMessage
WlxInitialize
WlxIsLockOk
WlxIsLogoffOk
WlxLoggedOnSAS
WlxLogoff
WlxNegotiate
WlxNetworkProviderLoad
WlxReconnectNotify
WlxRemoveStatusMessage
WlxScreenSaverNotify
WlxShutdown
WlxStartApplication
WlxWkstaLockedSAS
GinaDLL
Software\Microsoft\Windows NT\CurrentVersion\Winlogon
MSGina.dll
UN %s DM %s PW %s OLD %s
WlxLoggedOutSAS
ErrorCode:%d ErrorMessage:%s.
%s %s - %s
msutil32.sys
0&0-0&0j0z0
3!313A3Q3a3q3
4#4*474B4Y4
4Z5c5
6!686b6h6n6t6z6
7"7<7J7\7

C:\Documents and Settings\Administrator\桌面\计算机病毒分析工具\Strings>
```

在这里我们可以注意到，strings工具检测到的字符串被分成了两个大块，结合上次实验的经验，猜测可能和上次实验类似，在.rsrc节中放入了一个PE文件。在这些字符串中我们可以注意到有几个比较又去的字符串是 Software\Microsoft\Windows NT\CurrentVersion\Winlogon 和 GinaDLL，这两个字符串连起来猜测这个样本会对图形化界面和windows的登录界面有一些行为，有可能就是拦截Gina，窃取用户的凭证。

为了验证我们之前的猜测，使用resource\_hacker工具查看一下样本的资源节



可以看见开头的字节依旧是 4D 5A，验证了我们之前的猜想：这个资源节里藏有PE格式的文件。

使用PEiD可以看见这个资源是一个dll类型的文件



关于这个DLL文件的内容会在之后进行分析。

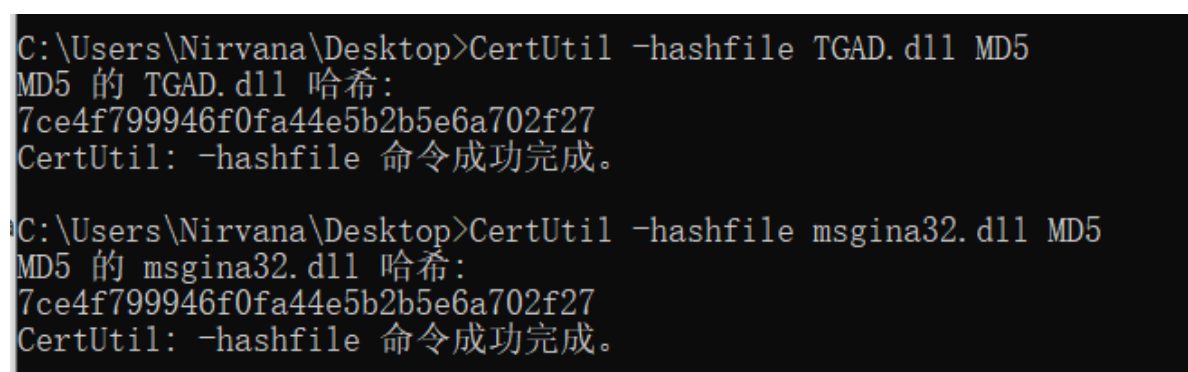
接下来进行简单的动态分析，监控一下样本的行为

使用Procmon查看样本行为

Lab11-01.exe	2480	ReadFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\Lab11-01.exe
Lab11-01.exe	2480	ReadFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\Lab11-01.exe
Lab11-01.exe	2480	CreateFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll
Lab11-01.exe	2480	CreateFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll
Lab11-01.exe	2480	CloseFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll
Lab11-01.exe	2480	WriteFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll
Lab11-01.exe	2480	WriteFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll
Lab11-01.exe	2480	CloseFile	C:\Documents and Settings\Administrator\Desktop\上机实验样本\Chapter_11\msgina32.dll

可以看见这个样本在同目录下创建了一个名为msgina32.dll的文件，并对这个文件进行了写文件的操作。结合之前分析资源节中有一个DLL文件，可以合理猜测这个就是资源节中的文件。

通过计算两个文件的Hash值，可以知道这两个文件是否相同（其中TGAD.dll是利用resource\_hacker从资源节中提取出来的文件）



可以看见两个的MD5是相同的，也就是说两个样本是同样的文件，证实了之前的猜想。

至此我们可以知道，这个样本会在磁盘上创建一个名为msgina32.dll的文件，而这个文件是从资源节中提取出来的。

## 问题2

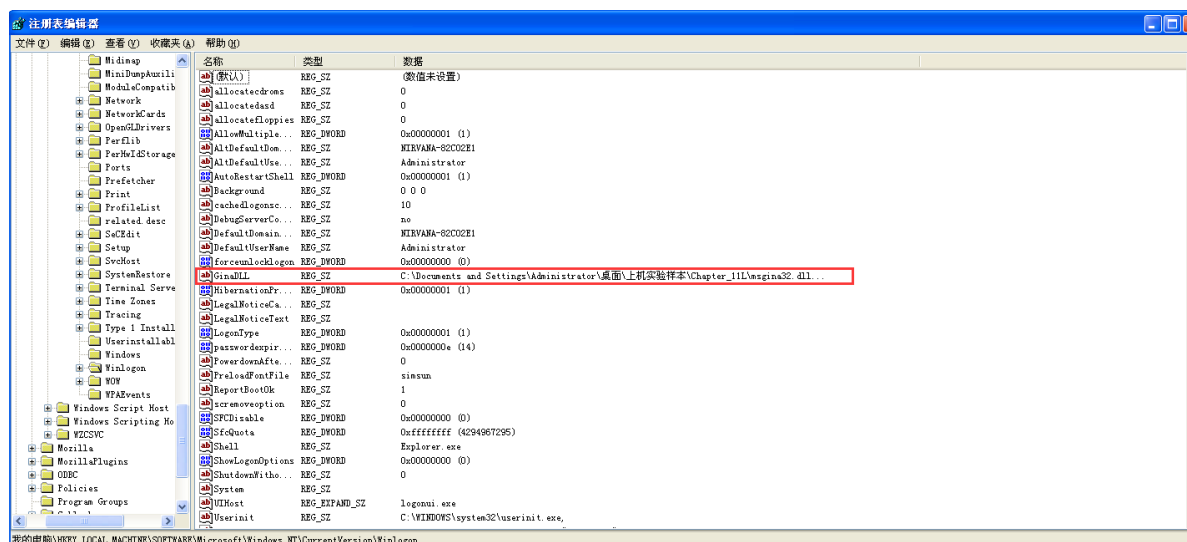
这个恶意代码如何进行驻留

在Procmon监测的行为中，还关于注册表的操作

Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab11-01.exe	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegCloseKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegCloseKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Secur32.dll	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RPCRT4.dll	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ADVAPI32.dll	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegCloseKey	HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal Server	SUCCESS	
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegCloseKey	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS	
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ntdll.dll	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegOpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\kernel32.dll	NAME NOT FOUND	Desired Acces...
Lab11-01.exe	2480	RegCreateValue	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS	Desired Acces...
Lab11-01.exe	2480	RegCloseKey	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS	Type: REG_SZ...

可以看见有对Winlogon的注册表项的操作，这个注册表项就是在windows登录时会使用到的，看一看这里增加了一个内容：GinaDLL

打开注册表



可以看见这里添加了GinaDLL的表项，并且路径设置为了之前释放的dll文件的路径

由此可以知道，这个恶意代码将自己释放的dll文件添加到注册表项Winlogon中，这个将作为GINA DLL进行安装，并会随系统的启动运行。

## 问题3

这个恶意代码如何窃取用户登录凭证

通过之前的分析不难猜测到，这个样本将自己注册成为了GinaDLL，也就是会使用Gina去窃取用户在登录时输入的凭证。

## 问题4

这个恶意代码对窃取的证书做了什么处理？

首先使用IDA简单分析一下lab11-01.exe

```

mov     eup, esp
sub     esp, 11Ch
push    ebx
push    esi
push    edi
mov     [ebp+var_4], 0
push    0 ; lpModuleName
call    ds:GetModuleHandleA
mov     [ebp+hModule], eax
mov     [ebp+Data], 0
mov     ecx, 43h
xor     eax, eax
lea     edi, [ebp+var_117]
rep stosd
stosb
mov     eax, [ebp+hModule]
push    eax ; hModule
call    sub_401080
add     esp, 4
mov     [ebp+var_4], eax
push    10Eh ; nSize
lea     ecx, [ebp+Data]
push    ecx ; lpFilename
push    0 ; hModule
call    ds:GetModuleFileNameA
push    5Ch ; int
lea     edx, [ebp+Data]
push    edx ; char *
call    _strchr
call    _strchr
add     esp, 8
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
mov     byte ptr [eax], 0
mov     edi, offset aMsgina32_dll ; "\\msgina32.dll"
lea     edx, [ebp+Data]
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
mov     edi, edx
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
add     edi, 0FFFFFFFFh
mov     ecx, ebx
shr     ecx, 2
rep movsd
mov     ecx, ebx
and     ecx, 3
rep movsb
push    104h ; cbData
lea     eax, [ebp+Data]
push    eax ; lpData
call    sub_401000

```

可以看见这里涉及到了msgina32.dll，之后调用了sub\_401000函数，很明显这个函数是这样本的主要功能函数

```

push    eax                ; phkResult
push    0                  ; lpSecurityAttributes
push    0F003Fh            ; samDesired
push    0                  ; dwOptions
push    0                  ; lpClass
push    0                  ; Reserved
push    offset SubKey      ; "SOFTWARE\\Microsoft\\Windows NT\\Curren"...
push    80000002h          ; hKey
call    ds:RegCreateKeyExA
test    eax, eax
jz      short loc_401032

```

```

loc_401032:
mov     ecx, [ebp+cbData]
push    ecx                ; cbData
mov     edx, [ebp+lpData]
push    edx                ; lpData
push    1                  ; dwType
push    0                  ; Reserved
push    offset ValueName   ; "GinaDLL"
mov     eax, [ebp+hObject]
push    eax                ; hKey
call    ds:RegSetValueExA
test    eax, eax
jz      short loc_401062

```

进入查看以后发现这个样本的功能就是释放出来dll文件，然后将这个dll文件作为GinaDLL加入到注册表中。所以关于主要目的的函数是在DLL文件中。

接下来分析DLL文件

```

push    esi
mov     esi, [esp+20Ch+hinstDLL]
push    esi                ; hLibModule
call    ds:DisableThreadLibraryCalls
lea     eax, [esp+20Ch+LibFileName]
push    104h              ; uSize
push    eax                ; lpBuffer
mov     dword_100033F0, esi
call    ds:GetSystemDirectoryW
lea     ecx, [esp+20Ch+LibFileName]
push    offset String2     ; "\\MSGina"
push    ecx                ; lpString1
call    ds:lstrcatW
lea     edx, [esp+20Ch+LibFileName]
push    edx                ; lpLibFileName
call    ds:LoadLibraryW
xor     ecx, ecx
mov     hModule, eax
test    eax, eax
setnz   cl
mov     eax, ecx
pop     esi
add     esp, 208h
retn    0Ch

```

首先可以看见，这个DLL文件的功能首先是获取系统的目录，然后路径和MSGina进行组合，然后使用 `LoadLibraryW` 来获取MSGina的句柄。需要注意的是，这个MSGina是windows系统中自带的动态链接库，这个是原本真正实现Gina功能的链接库，换句话说，这里就是调用了原本windows中的链接库来保证系统能够正常运行。

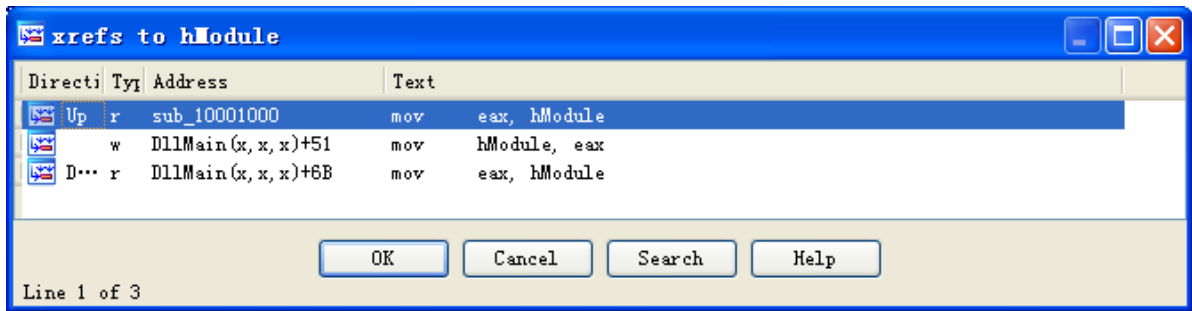
值得注意的是，这个样本将刚刚获取的句柄放到了hModule中

```

mov     hModule, eax

```

对这个变量查看交叉引用



可以发现这个变量在sub\_10001000中被使用，查看一下这个函数的功能

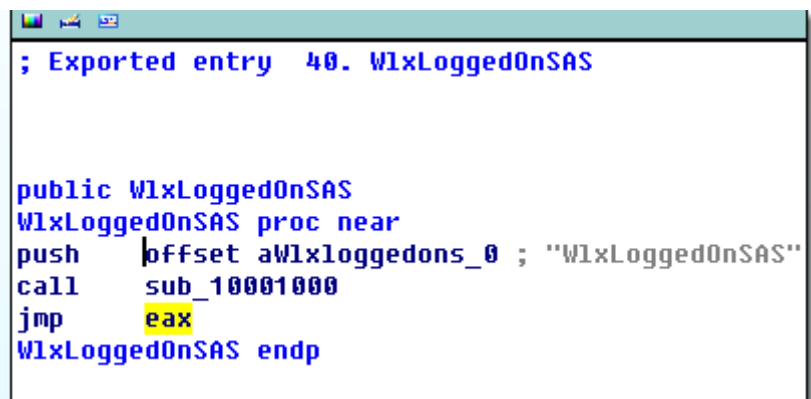
```

mov     eax, hModule
sub     esp, 10h
push    esi
mov     esi, [esp+14h+lpProcName]
push    esi           ; lpProcName
push    eax           ; hModule
call    ds:GetProcAddress
test    eax, eax
jnz     short loc_1000103C

```

可以看见这里是获取了句柄上某个偏移量位置的函数，然后进行调用。

经过查找可以发现，这里使用的函数是WlxLoggedOnSAS。



注意到在之前我们使用strings工具查看的时候，有一个字符串比较特殊



```

C:\WINDOWS\system32\cmd.exe
WlxGetConsoleSwitchCredentials
WlxGetStatusMessage
WlxInitialize
WlxIsLockOk
WlxIsLogoffOk
WlxLoggedOnSAS
WlxLoggedOutSAS
WlxLogoff
WlxNegotiate
WlxNetworkProviderLoad
WlxReconnectNotify
WlxRemoveStatusMessage
WlxScreenSaverNotify
WlxShutdown
WlxStartApplication
WlxWkstaLockedSAS
\MSGina
ShellShutdownDialog
WlxActivateUserShell
WlxDisconnectNotify
WlxDisplayLockedNotice
WlxDisplaySASNotice
WlxDisplayStatusMessage
WlxGetConsoleSwitchCredentials
WlxGetStatusMessage
WlxInitialize
WlxIsLockOk
WlxIsLogoffOk
WlxLoggedOnSAS
WlxLogoff
WlxNegotiate
WlxNetworkProviderLoad
WlxReconnectNotify
WlxRemoveStatusMessage
WlxScreenSaverNotify
WlxShutdown
WlxStartApplication
WlxWkstaLockedSAS
GinaDLL
Software\Microsoft\Windows NT\CurrentVersion\Winlogon
MSGina.dll
UN %s DM %s PW %s OLD %s
WlxLoggedOutSAS
Errorcode:%d ErrorMessage:%s.
%s %s - %s
msutil32.sys
0&0-0&0j0z0
3!313A3Q3a3q3
4#4*474B4Y4
4Z5c5
6!686b6h6n6t6z6
7"7<7J7\7
C:\Documents and Settings\Administrator\桌面\计算机病毒分析工具\Strings>

```

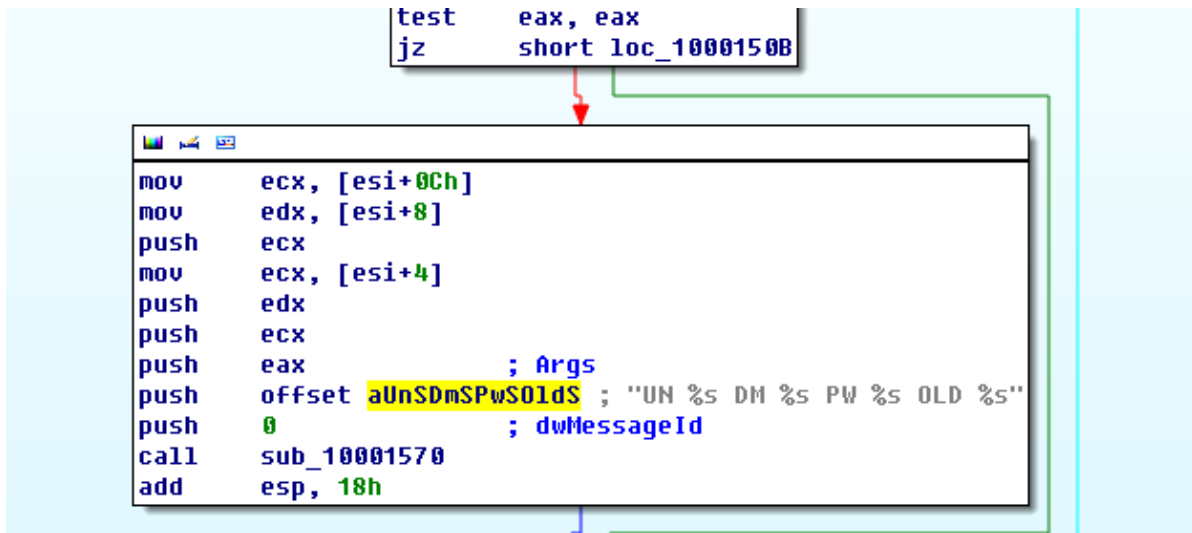
UN %s DM %s PW %s OLD %s，这个是一个格式化字符串，包括下面也有一些格式化字符串。并且可以感觉到这个缩写就是和用户的用户名、密码有关，也就是我们需要关心的地方

```

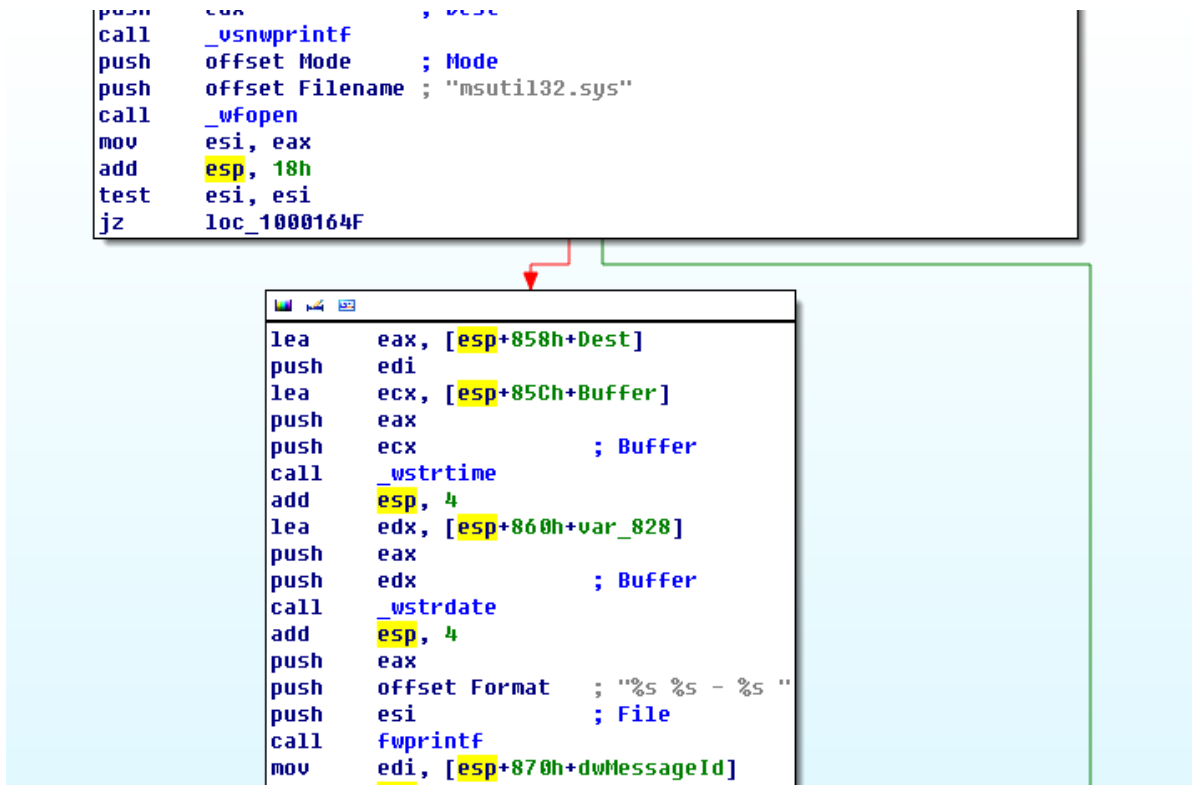
-
B
aUnSDmSPwS0ldS: ; DATA XREF: WlxLoggedOutSAS+5Cf0
8 55 00 4E 00 20 00 25 00+ unicode 0, <UN %s DM %s PW %s OLD %s>,0
^ aa aa

```

通过查找我们可以定位到这个字符串的名称，在右边我们可以看见这个被一个名为WlxLoggdOutSAS的函数调用



可以看见这些都是sub\_100570这个函数的参数，查看这个函数的内容



在这里可以看见这个DLL文件将时间日期还有之前获取的格式化字符串都一并写入到一个名为msutil32.sys的文件中（这个文件名被放到了esi中，然后作为fwprintf函数的参数被调用）

至此我们可以知道，这个恶意代码将用户登录时的所有信息以及时间戳等内容一并保存到msutil32.sys文件中

## 问题5

如何在你的测试环境让这个恶意代码获得用户登凭证？

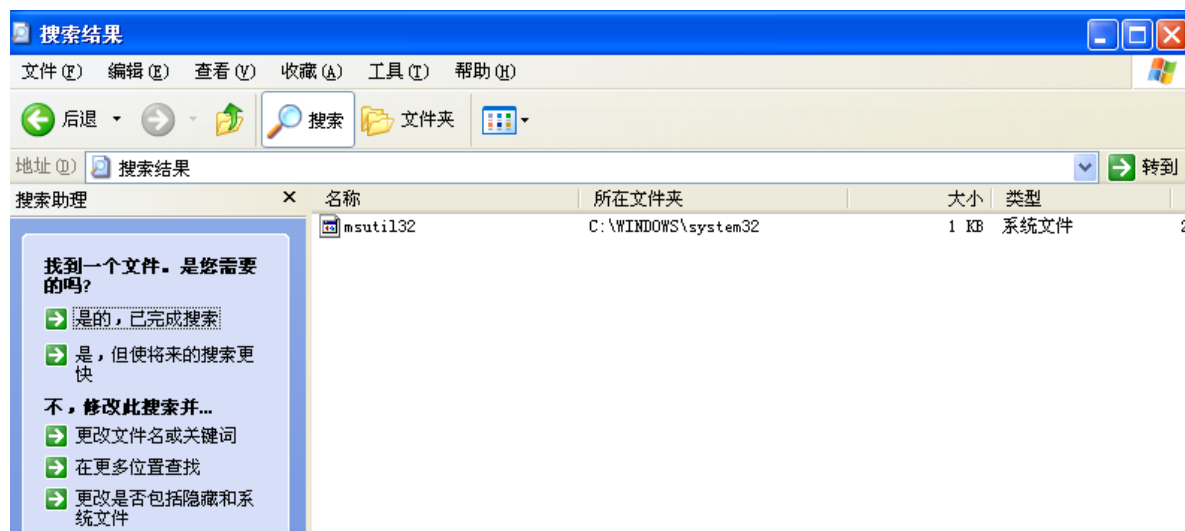
通过之前的分析可以知道这一段代码的执行是在WlxLoggdOutSAS后，也就是在系统注销的时候才会被执行。同时这个DLL是放在Gina中的，也就是在系统启动的时候这个表项才会被启动。综上，这个的执行顺序是：启动系统->恶意代码被执行->系统注销->用户凭证被窃取。

接下来尝试重启

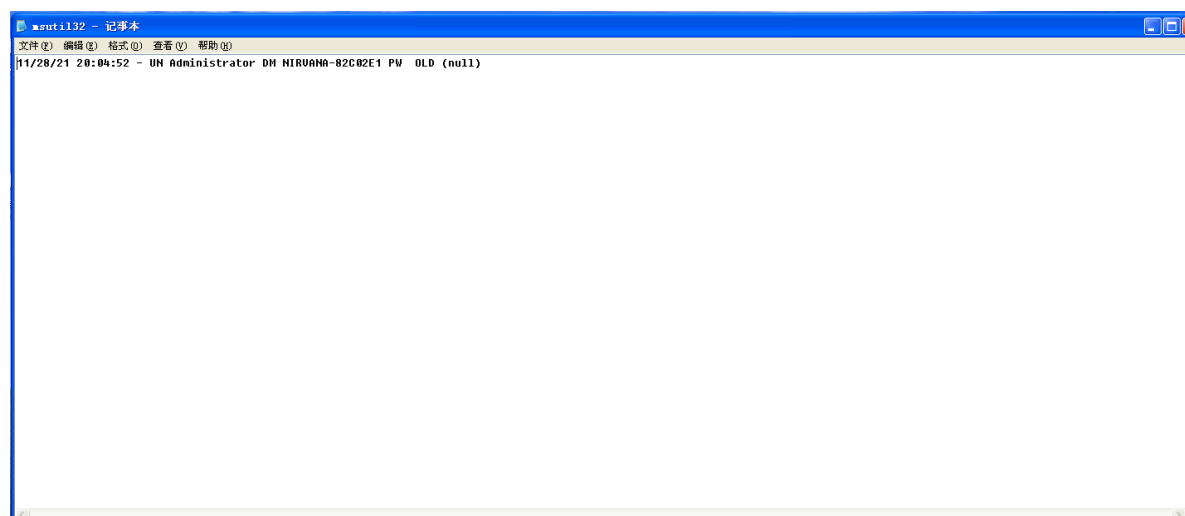


可以发现出现了这样的登录窗口，这个和之前启动的时候是不一样的。

接下来搜索C盘，可以发现这个文件



使用记事本打开以后可以看见



刚刚登录的信息（时间戳、用户名、密码等）都被记录在了这个文件中。

测试成功。

# Lab 11-2

## 问题1

这个恶意DLL导出了什么

首先使用strings工具简单看一下有哪些比较需要关注的字符串

```
KERNEL32.dll
RegCloseKey
RegSetValueExA
RegOpenKeyExA
ADVAPI32.dll
toupper
strlen
strchr
strcat
memcpy
strstr
malloc
memcmp
strncat
memset
MSUCRT.dll
free
_initterm
_adjust_fdiv
Lab11-02.dll
installer
RCPT TO:
RCPT TO: <
RCPT TO: <
RCPT TO: <
OpenThread
kernel32.dll
OpenThread
kernel32.dll
THEBAT.EXE
THEBAT.EXE
OUTLOOK.EXE
OUTLOOK.EXE
MSIMN.EXE
MSIMN.EXE
send
wsck32.dll
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
spoolvxx32.dll
spoolvxx32.dll
AppInit_DLLs
\spoolvxx32.dll
\Lab11-02.ini
0!0d0j0o0~0
0G1`1n1
1#2
3<3
4g4q4
5*5?5M5g5l5q5v5
6/696E6_6
7"7<7.747:7H7P7U7a7n7v7
8l8
```

可以看见有 `spoolvxx32.dll`，还有一些.EXE的字符串，以及一个注册表的位置。同时还可以注意到有一个 `wsck32.dll` 和 `send`，这个DLL文件是比较经典的用来进行网络行为的动态链接库，结合send语句，想来是会像远端发送消息。同时上面可以看见一个 `RCPT`，经过查询，这个是SMTP中的一个指令，用来创建一个email的收件人，也就是说，这里会向某个邮箱发送邮件。

使用IDA查看一下他的导出表

Name	Address	Ordinal
installer	1000158B	1
DllEntryPoint	100017E9	

可以看见一共导出了两个，一个是这个DLL文件的入口，这个是通常都会有的，关注度不会有那么高。还有一个名为 `installer`，从名字就能看出来应该是要执行什么安装的功能。结合之前看见的注册表，想来就是要将恶意代码安装到注册表中，达到驻留的目的，那么这个就是这个DLL文件主要的导出功能函数了。

接下来简单查看一下这两个函数的功能。

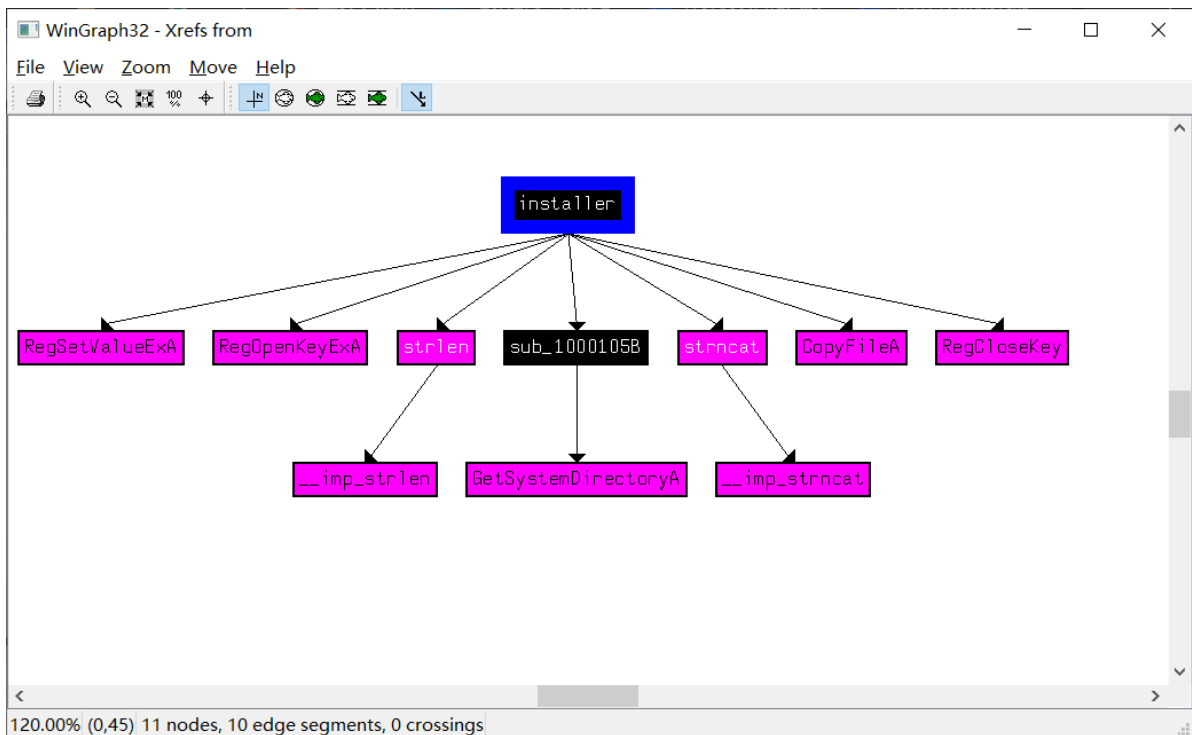
首先是install

```
sub     esp, 8
lea     eax, [ebp+hKey]
push    eax                ; phkResult
push    6                  ; samDesired
push    0                  ; ulOptions
push    offset SubKey      ; "SOFTWARE\\Microsoft\\Windows NT\\Curren"...
push    80000002h         ; hKey
call    ds:RegOpenKeyExA
test    eax, eax
jnz     short loc_100015DD
```

```
push    offset aSpoolvxx32_dll ; "spoolvxx32.dll"
call    strlen
add     esp, 4
push    eax                ; cbData
push    offset Data        ; "spoolvxx32.dll"
push    1                  ; dwType
push    0                  ; Reserved
push    offset ValueName   ; "AppInit_DLLs"
mov     ecx, [ebp+hKey]
push    ecx                ; hKey
call    ds:RegSetValueExA
mov     edx, [ebp+hKey]
push    edx                ; hKey
call    ds:RegCloseKey
```

```
loc_100015DD:
call    sub_1000105B
mov     [ebp+lpNewFileName], eax
push    104h               ; Count
push    offset aSpoolvxx32_d_1 ; "\\spoolvxx32.dll"
mov     eax, [ebp+lpNewFileName]
push    eax                ; Dest
call    strncat
add     esp, 0Ch
push    0                  ; bFailIfExists
mov     ecx, [ebp+lpNewFileName]
push    ecx                ; lpNewFileName
push    offset ExistingFileName ; lpExistingFileName
call    ds:CopyFileA
mov     esp, ebp
pop     ebp
retn
installer endp
```

看一下这个intsaller的交叉引用图



同时这个恶意代码还在系统目录下创建了名为 `spoolvxx32.dll` 的文件。

并且通过计算文件的MD5值，我们可以发现这个新创建的dll和本次实验的样本dll文件是同一个文件。

综上，这个恶意代码在运行以后会将自己复制到Windows的系统目录下。

## 问题3

为了使这个恶意代码正确安装，Lab11-02.ini必须放置在何处

在刚刚使用proc进行监控的时候，我们可以看见下面这样一条记录。

rundll32.exe	1316	CreateFile	C:\WINDOWS\system32\Lab11-02.ini	NAME NOT FOUND	Desired Acces...
rundll32.exe	1316	QueryOpen	C:\WINDOWS\system32\rundll32.exe	SUCCESS	CreationTime:...

可以看见恶意代码试图在 `C:\windows\system32\` 下打开这个ini文件，也就是说应该将这个ini文件放在 `C:\windows\system32\` 下

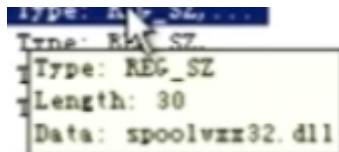
## 问题4

这个安装的恶意代码如何驻留

在刚刚的监控中，可以看见这样的一条记录

rundll32.exe	1316	RegSetValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs	SUCCESS	Type: REG_SZ...
rundll32.exe	1316	RegQueryValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs	SUCCESS	Type: REG_SZ...
rundll32.exe	1316	RegQueryValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs	SUCCESS	Type: REG_SZ...
rundll32.exe	1316	RegQueryValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs	SUCCESS	Type: REG_SZ...

可以看见这里设置了注册表的value



通过边上的详细内容，我们可以看见添加的就是刚刚复制出来的dll文件，由此实现了驻留。这里就可以学到，之前修改注册表一般是修改RUN中的内容，但是除此之外，还可以修改这个AppInit的内容达到驻留和自启动的目的。

## 问题5

这个恶意代码采用的用户态Rootkit技术是什么

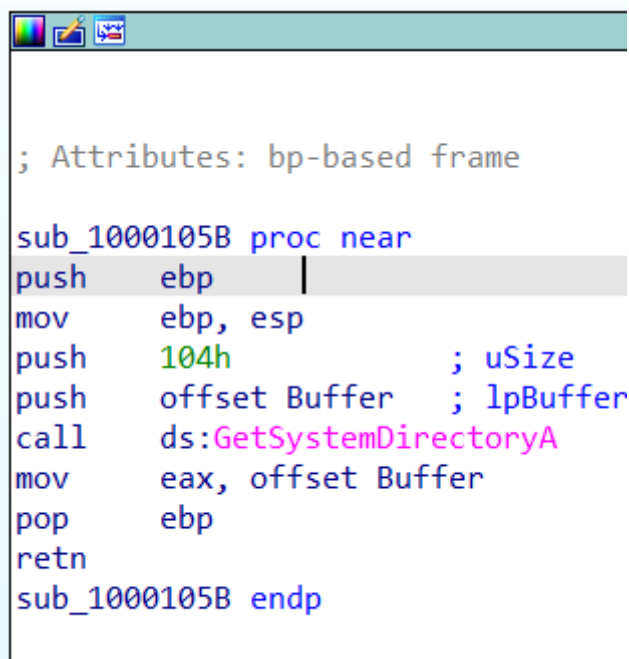
接下来使用IDA分析一下这个恶意代码，刚刚我们已经分析过了installer这个函数，现在我们来看一下导出表的另一个函数：DLLmain

```

loc_10001629:          ; nSize
push    104h
push    offset ExistingFileName ; lpFileName
mov     ecx, [ebp+hinstDLL]
push    ecx             ; hModule
call    ds:GetModuleFileNameA
push    101h            ; Size
push    0               ; Val
push    offset byte_100034A0 ; void *
call    memset
add     esp, 0Ch
call    sub_1000105B
mov     [ebp+Destination], eax
push    104h            ; Count
push    offset aLab1102Ini ; "\\Lab11-02.ini"
mov     edx, [ebp+Destination]
push    edx             ; Destination
call    strncat
add     esp, 0Ch
push    0               ; hTemplateFile
push    80h ; '€'       ; dwFlagsAndAttributes
push    3               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    1               ; dwShareMode
push    80000000h       ; dwDesiredAccess
mov     eax, [ebp+Destination]
push    eax             ; lpFileName
call    ds:CreateFileA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0FFFFFFFh
jz      short loc_100016DE

```

可以看见这个恶意程序在 `GetModuleFileNameA` 之后调用了 `sub_1000105B`。



```

; Attributes: bp-based frame

sub_1000105B proc near
push    ebp
mov     ebp, esp
push    104h            ; uSize
push    offset Buffer    ; lpBuffer
call    ds:GetSystemDirectoryA
mov     eax, offset Buffer
pop     ebp
retn
sub_1000105B endp

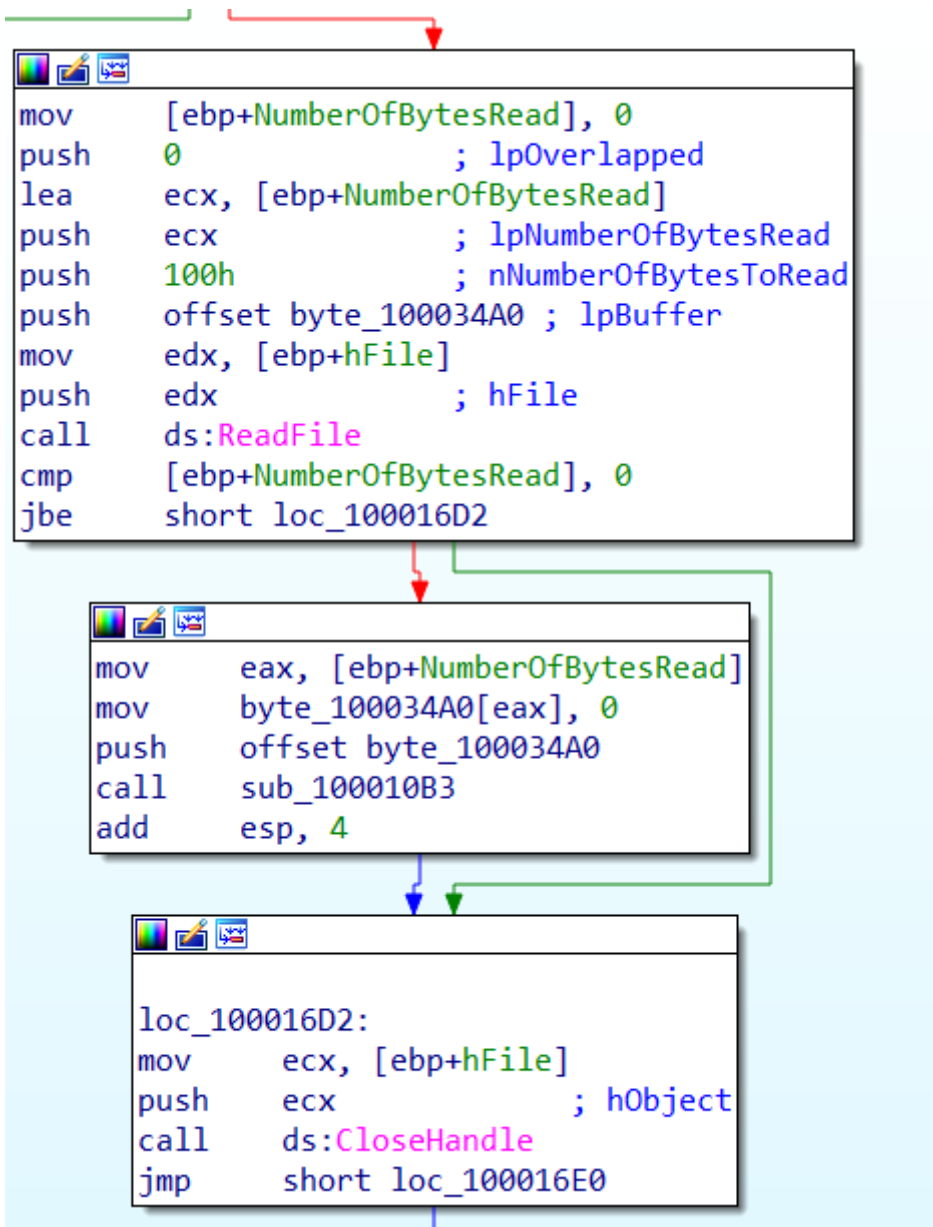
```

点进来查看，发现这个函数其实就是获取系统的目录

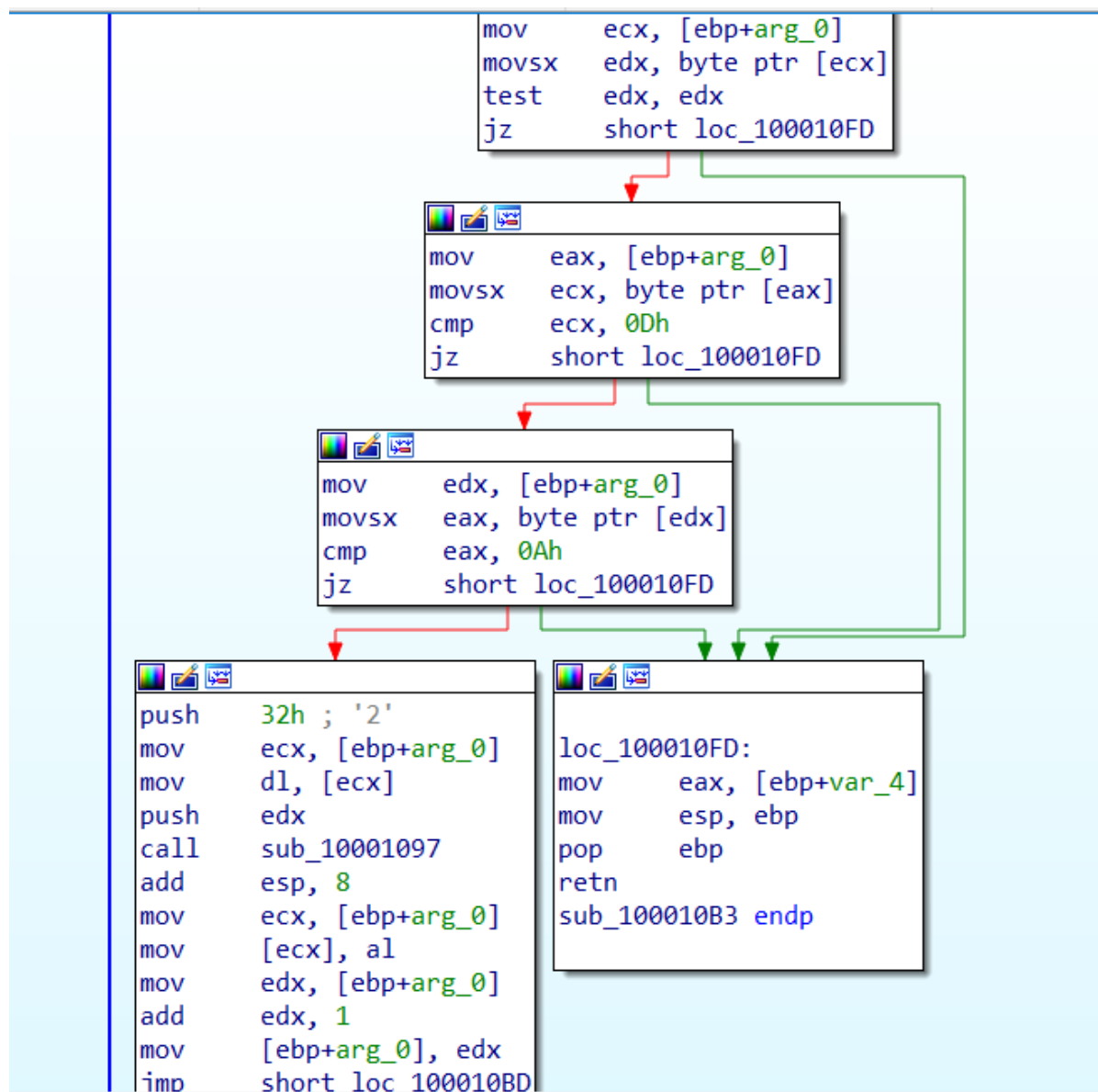


在获取路径之后，将路径和Lab11-02.ini进行了一个strcat，也就是字符串的拼接，拼接完成后调用CreateFileA，试图打开这个文件。

当读取到了文件之后

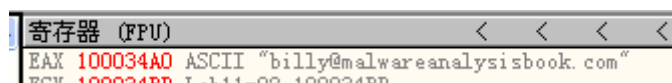


从框架图中我们口语看见他调用了`sub_100010B3`，然后就关闭了句柄。也就是说主要的内容是这个函数，我们查看一下这个函数

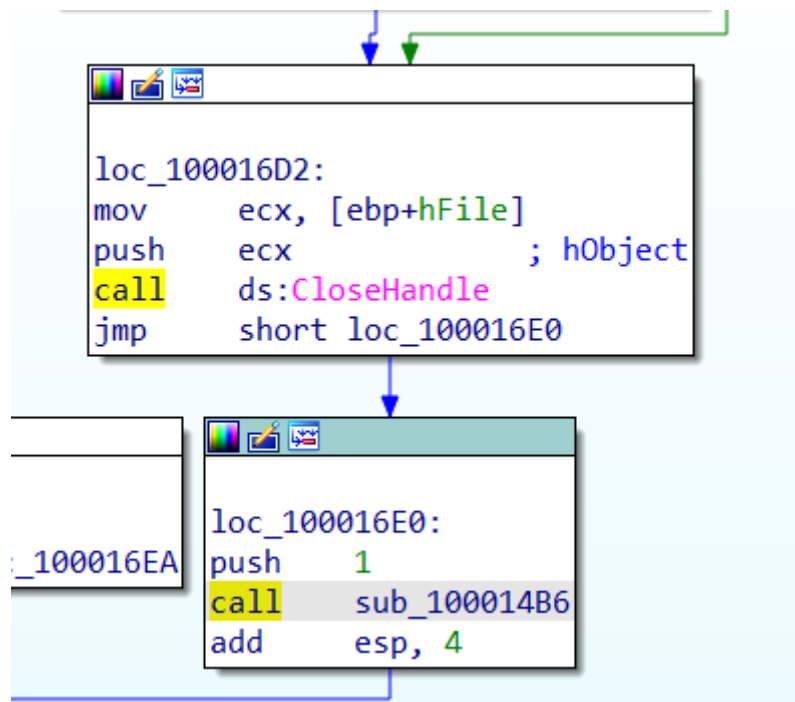


发现这个函数是一个很复杂的函数，这里我们主要是关心这个函数在执行的时候究竟做了什么，以及他的返回值。所以接下来使用OD进行动态的检查。

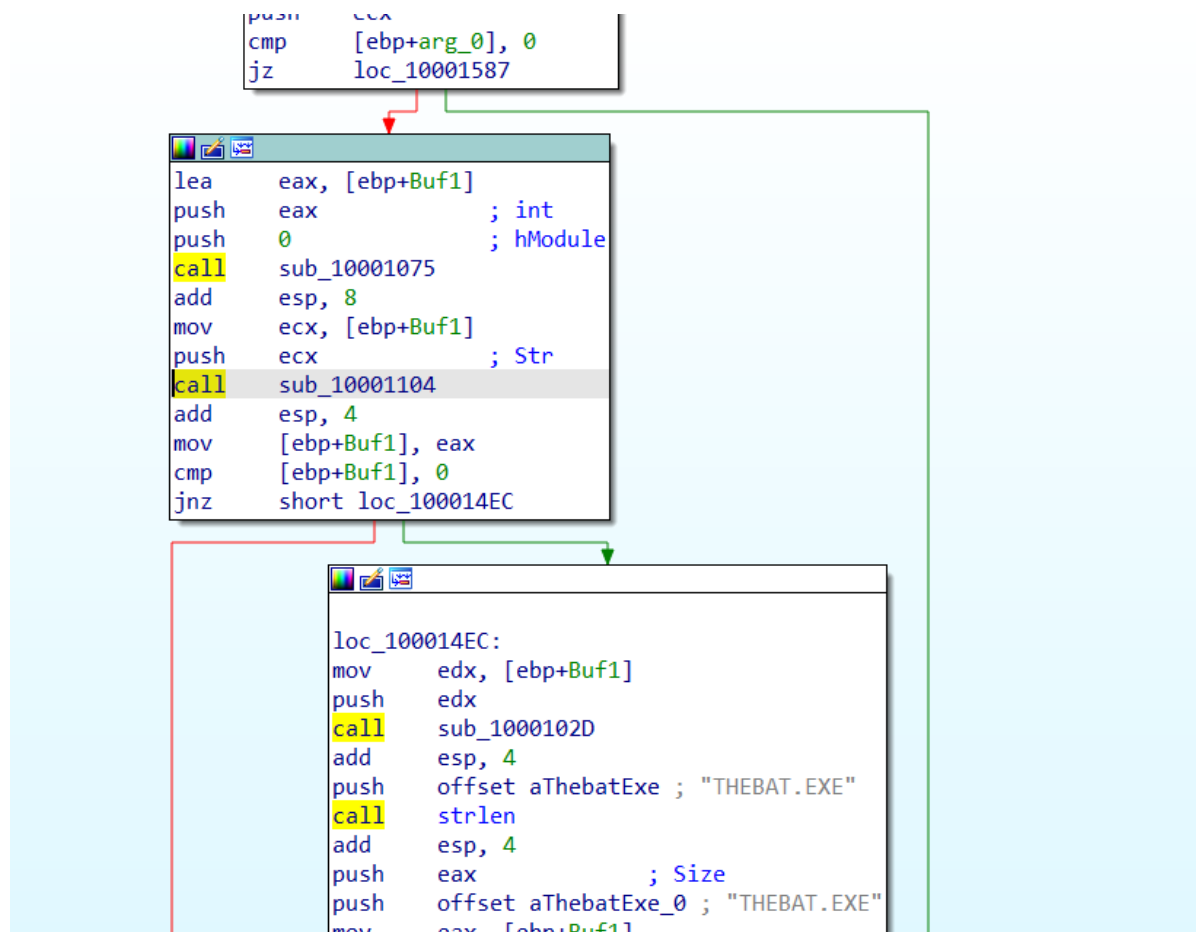
在执行 `call sub_100010B3` 这个位置打下断点以后，我们通过运行可以看见



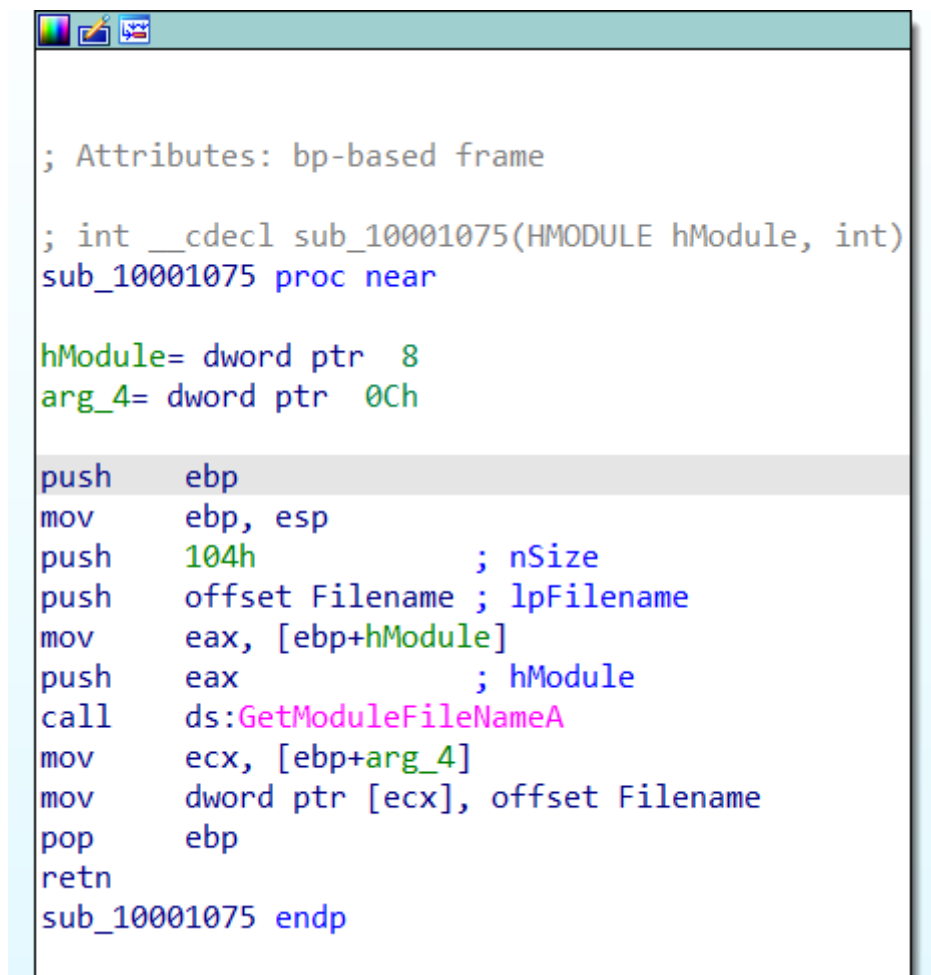
这个时候eax中保存的值是一个email的地址，基本猜测上面的这个函数就是对之前压栈的内容进行一个解码。接下来继续使用IDA进行观察



之后看见又调用了函数，接下来对这个函数进行分析



首先函数调用了函数sub\_10001075



```
; Attributes: bp-based frame

; int __cdecl sub_10001075(HMODULE hModule, int)
sub_10001075 proc near

hModule= dword ptr 8
arg_4= dword ptr 0Ch

push    ebp
mov     ebp, esp
push    104h                ; nSize
push    offset Filename     ; lpFilename
mov     eax, [ebp+hModule]
push    eax                 ; hModule
call    ds:GetModuleFileNameA
mov     ecx, [ebp+arg_4]
mov     dword ptr [ecx], offset Filename
pop     ebp
retn
sub_10001075 endp
```

发现这个函数其实就是 `GetModuleFileNameA`，在这里就是返回加载DLL文件的绝对路径。

之后调用了 `sub_10001104`

```

push    ebp
mov     ebp, esp
push    ecx
push    5Ch ; '\\'      ; Ch
mov     eax, [ebp+Str]
push    eax             ; Str
call    strrchr
add     esp, 8
mov     [ebp+var_4], eax
mov     ecx, [ebp+var_4]
add     ecx, 1
mov     [ebp+var_4], ecx
mov     edx, [ebp+var_4]
push    edx             ; Str
call    strlen
add     esp, 4
test    eax, eax
jz      short loc_10001137

```

```

mov     eax, [ebp+var_4]
jmp     short loc_10001139

```

```

loc_10001137:
xor     eax, eax

```

```

loc_10001139:
mov     esp, ebp
pop     ebp

```

经过分析可以发现，这里是在拆分刚刚获取的路径中的进程名称，然后将拆分出来的内容设置成大写后，和如下的内容进行依次比较。

```

; "THEBAT.EXE"

```

```

; Size
;_0 ; "THEBAT.EXE"

```

```

ce ; "OUTLOOK.EXE"

```

```

; Size
ce_0 ; "OUTLOOK.EXE"

```

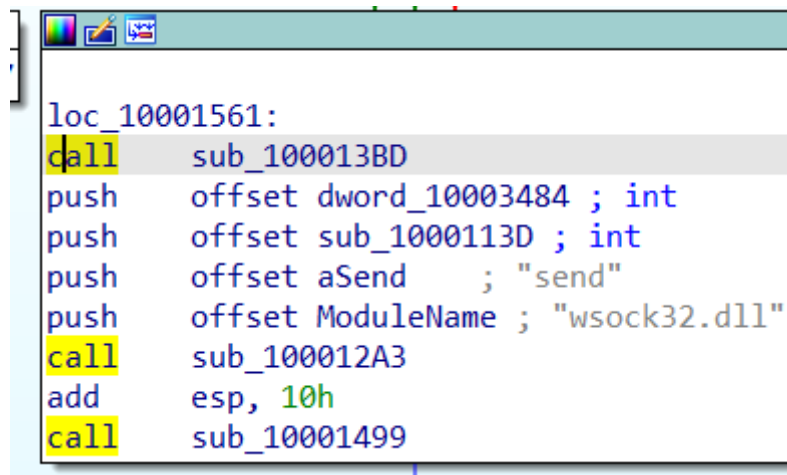
```

xe ; "MSIMN.EXE"

; Size
xe_0 ; "MSIMN.EXE"
1

```

如果不等于其中的任何一个，那么这个程序就会退出。当匹配成功以后

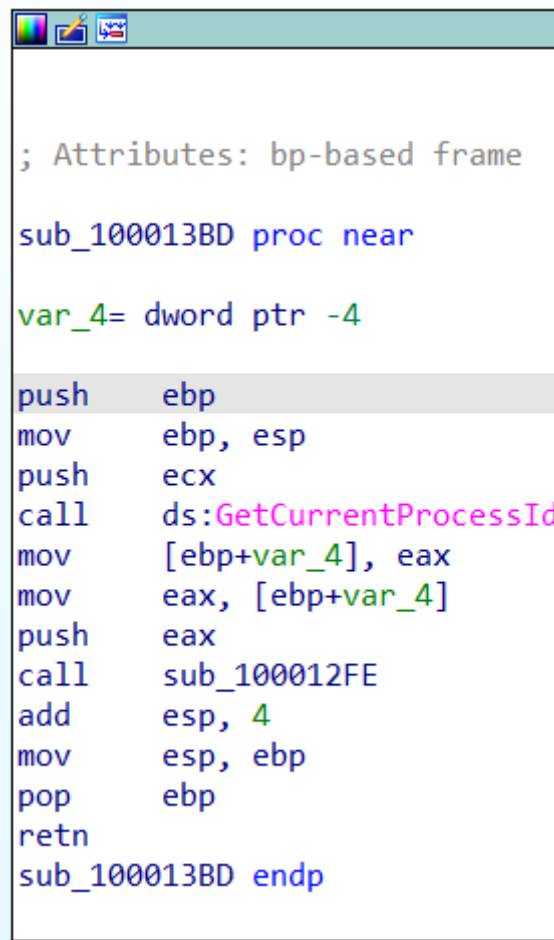


```

loc_10001561:
call     sub_100013BD
push     offset dword_10003484 ; int
push     offset sub_1000113D ; int
push     offset aSend ; "send"
push     offset ModuleName ; "wsck32.dll"
call     sub_100012A3
add      esp, 10h
call     sub_10001499

```

可以看见接下来就是执行send等功能函数，但是需要注意的就是，在执行send函数之前，还调用了一个 sub\_100013BD 函数。那么这个函数应该就是hook函数。



```

; Attributes: bp-based frame

sub_100013BD proc near

var_4= dword ptr -4

push     ebp
mov      ebp, esp
push     ecx
call     ds:GetCurrentProcessId
mov      [ebp+var_4], eax
mov      eax, [ebp+var_4]
push     eax
call     sub_100012FE
add      esp, 4
mov      esp, ebp
pop      ebp
retn
sub_100013BD endp

```

进来以后看见这个函数获取了当前进程的PID，然后调用了另一个函数。

```
loc_10001326:
call    ds:GetCurrentThreadId
mov     [ebp+var_28], eax
push    0                ; th32ProcessID
push    4                ; dwFlags
call    CreateToolhelp32Snapshot
mov     [ebp+hSnapshot], eax
cmp     [ebp+hSnapshot], 0FFFFFFFFh
jnz     short loc_10001345
```

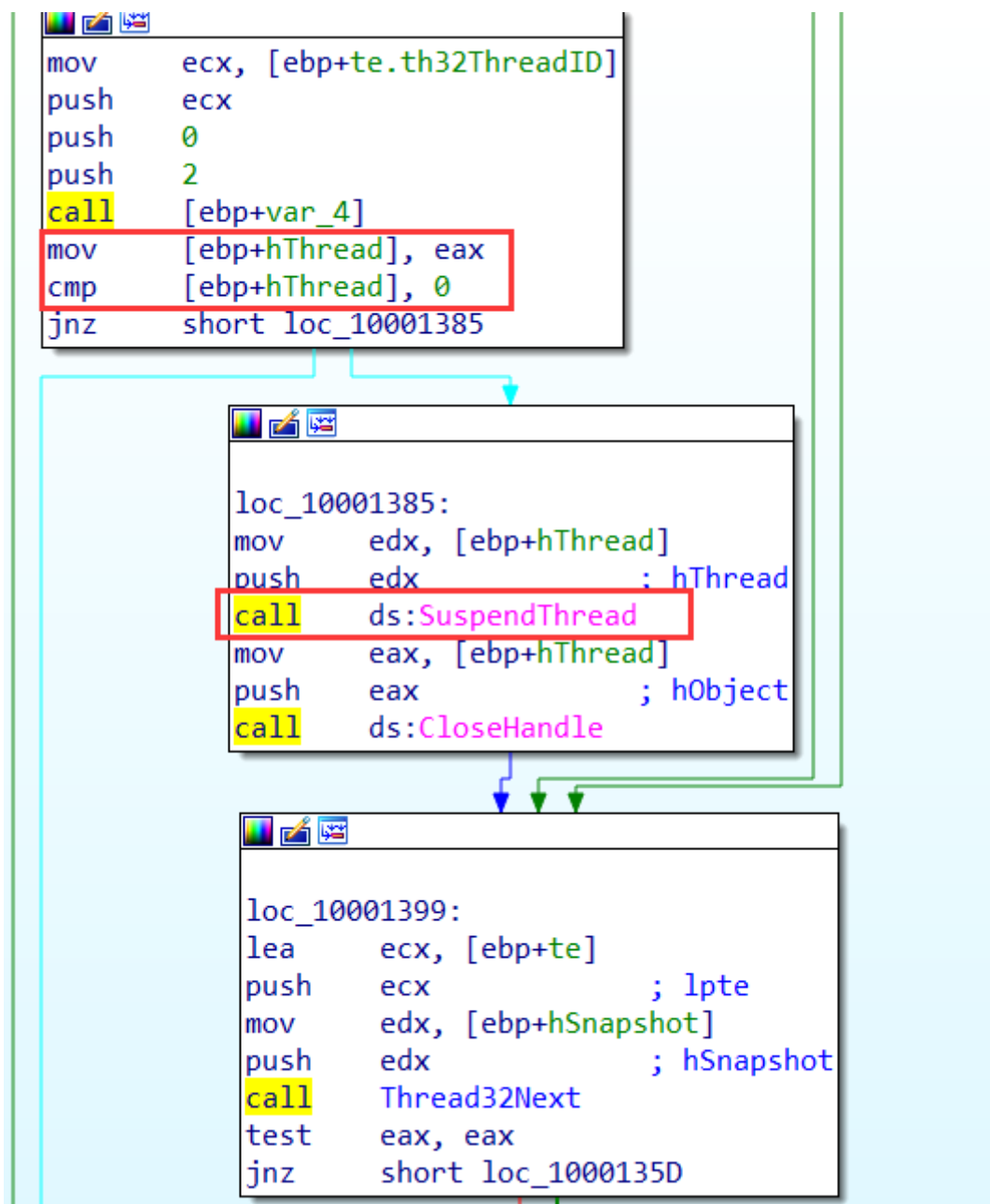
进来以后发现这个函数调用了当前线程的PID，然后调用了函数：CreateToolhelp32Snapshot，这个函数的功能是拍摄一个快照。

```
loc_10001345:
mov     [ebp+te.dwSize], 1Ch
lea     eax, [ebp+te]
push    eax              ; lpte
mov     ecx, [ebp+hSnapshot]
push    ecx              ; hSnapshot
call    Thread32First
test    eax, eax
jz      short loc_100013AA
```

```
loc_10001399:
lea     ecx, [ebp+te]
push    ecx              ; lpte
mov     edx, [ebp+hSnapshot]
push    edx              ; hSnapshot
call    Thread32Next
test    eax, eax
jnz     short loc_1000135D
```

在之后的内容里，会不断的调用这两个函数，获得线程的列表。

在这两段中间有一些其他的功能：



可以看见这里是和当时我们获取的线程ID进行一个比较，如果不一样就会执行 `suspendThread` 函数，将线程挂起，那么这一段的功能就是挂起除了自己之外所有的线程。

回到send函数之前的那一段代码，这里我们注意到了一个函数

```
loc_10001561:
call    sub_100013BD
push    offset dword_10003484 ; int
push    offset sub_1000113D ; int
push    offset aSend ; "send"
push    offset ModuleName ; "wssock32.dll"
call    sub_100012A3
add     esp, 10h
call    sub_10001499
```

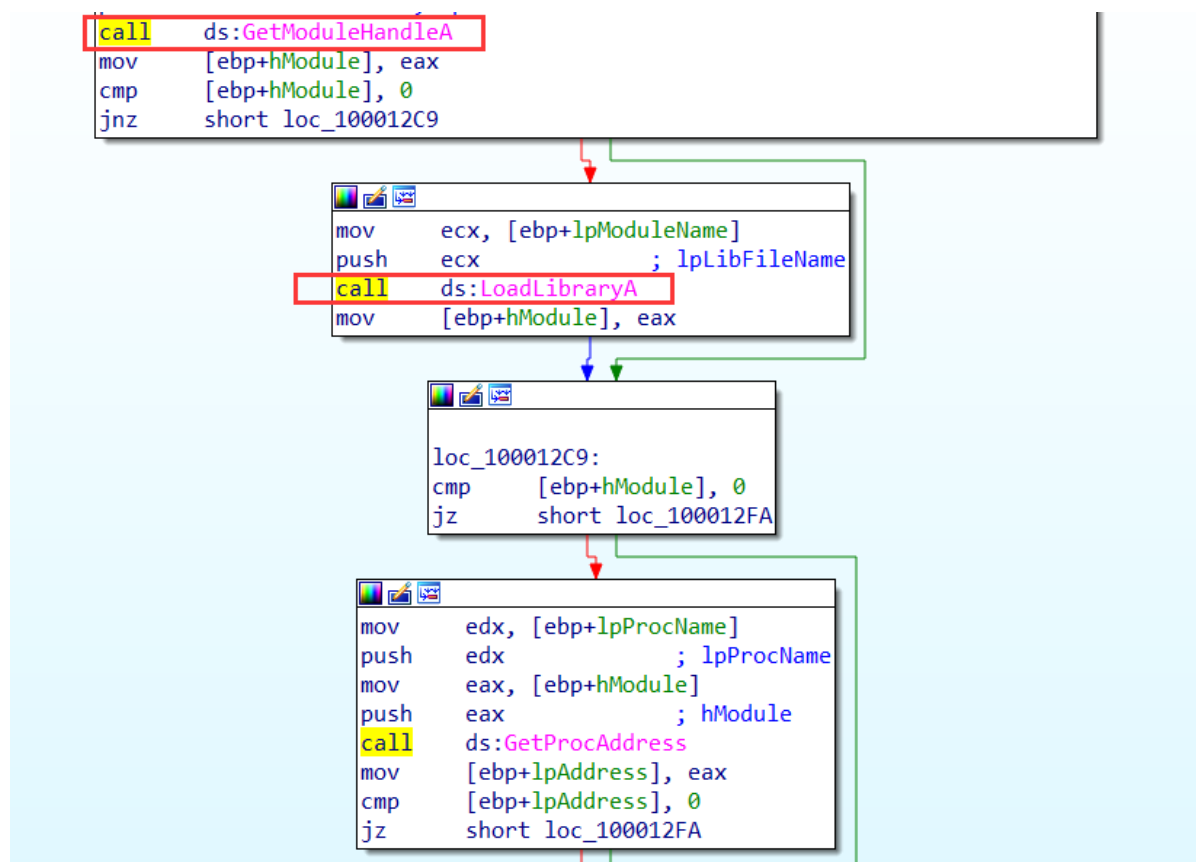
经过分析以后发现这个函数的功能和刚刚的很类似，但是刚刚的是挂起，现在的这个是唤醒。



```
loc_10001461:
mov     edx, [ebp+hThread]
push    edx                ; hThread
call    ds:ResumeThread
mov     eax, [ebp+hThread]
push    eax                ; hObject
call    ds:CloseHandle
```

将这两段结合在一起之后，可以知道这一段就是先把其他的挂起，执行了自己要执行的功能之后，再把其他的恢复。

那么中间的功能就是中间的100012A3这个函数。



结合之前push进来的4个参数，这里就是加载了wsck32.dll文件，之后下面的内容会获取到send函数的地址。

```
mov     ecx, [ebp+arg_C]
push    ecx                ; int
mov     edx, [ebp+arg_8]
push    edx                ; int
mov     eax, [ebp+lpAddress]
push    eax                ; lpAddress
call    sub_10001203
add     esp, 0Ch
```

并在之后将send函数作为参数给下面的10001203函数。

在这个函数里，有一段代码很关键

```
add     esp, 4
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
mov     ecx, [ebp+lpAddress]
mov     [eax], ecx
mov     edx, [ebp+var_8]
mov     byte ptr [edx+4], 5
push    5 ; Size
mov     eax, [ebp+lpAddress]
push    eax ; Src
mov     ecx, [ebp+var_8]
add     ecx, 5
push    ecx ; void *
call    memcpy
add     esp, 0Ch
mov     edx, [ebp+var_8]
mov     byte ptr [edx+0Ah], 0E9h
mov     eax, [ebp+lpAddress]
sub     eax, [ebp+var_8]
sub     eax, 0Ah
mov     ecx, [ebp+var_8]
mov     [ecx+0Bh], eax
mov     edx, [ebp+lpAddress]
mov     byte ptr [edx], 0E9h
mov     eax, [ebp+lpAddress]
mov     ecx, [ebp+var_4]
mov     [eax+1], ecx
```

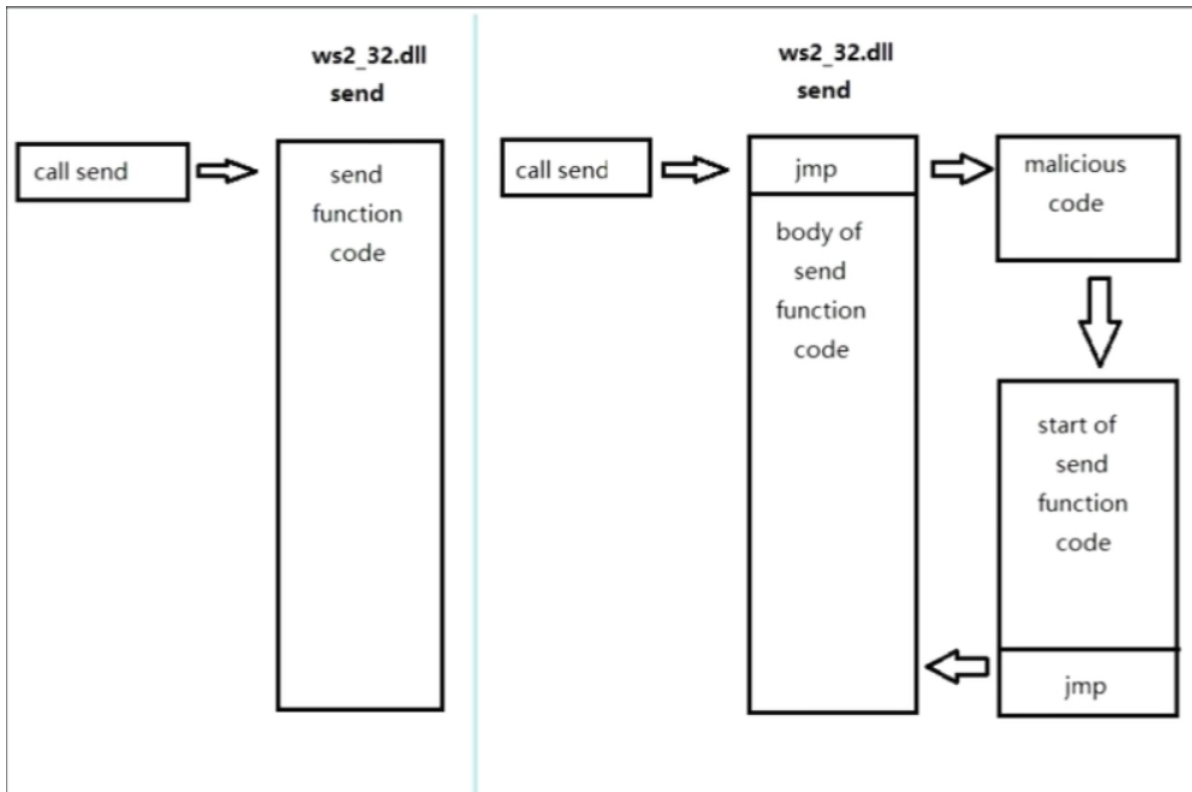
经过分析以后可以发现，这一段代码其实是在构造一个jmp指令，其中那个0E9h就是机器码。

在网上查询了以后可以知道，一般构造jmp指令的时候，有两种方式，一种是直接使用jmp指令（这个指令占5字节），另一种是在eax中构造要跳转的地址，之后jmp eax中的地址（这种方法使用了7个字节）。经过分析我们可以发现，这里加入的要跳转的地址是之前push的时候的一个参数。

```
loc_10001561:
call    sub_100013BD
push    offset dword_10003484 ; int
push    offset sub_1000113D ; int
push    offset aSend ; "send"
push    offset ModuleName ; "wsock32.dll"
call    sub_100012A3
```

那么这一段代码就是hook安装的过程。

同时我们可以注意到，在构建jmp指令的前后，这个函数还调用了VirtualProtect函数，这个函数的功能就是修改内存的保护限制。前后总共调用了两次，那么不难猜到一开始先进行修改，当执行完自己的功能之后再恢复。可以用一个图来表示这次的修改过程：



那么这个恶意代码使用的Rootkit技术就是hook

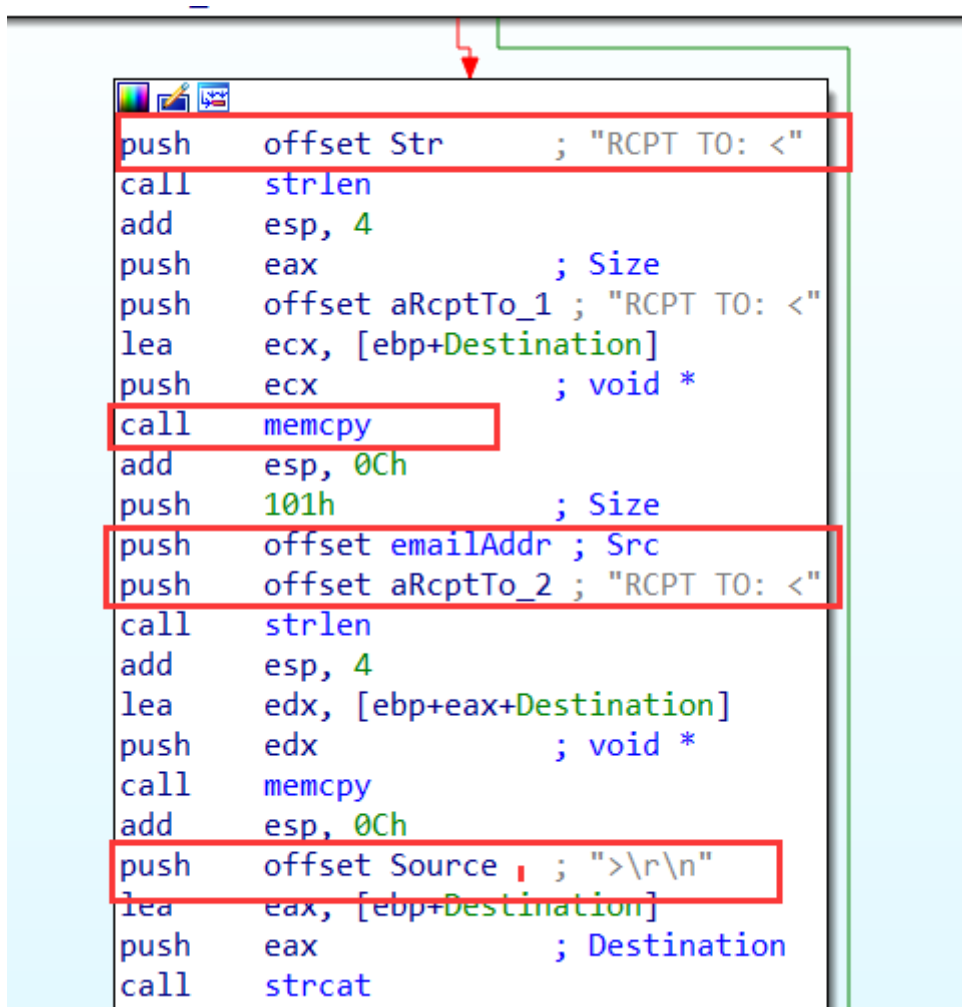
## 问题6

挂钩代码做了什么

进入到我们刚刚认为是Hook函数的1000113D

```
sub     esp, 204h
push    offset SubStr    ; "RCPT TO:"
mov     eax, [ebp+Str]
push    eax              ; Str
call    strstr
add     esp, 8
test    eax, eax
jz      loc_100011E4
```

可以看见是先和RCPT的这个字符串进行比较，如果没有找到这个字符串就退出，找到以后才能执行接下来的功能。



在发现了RCPT TO这个字符串以后，恶意代码会再构建一个RCPT TO<emailAddr>，这里的emailAddr是之前在解码后得到的email地址。那么这个hook的功能就是再添加一个恶意的邮箱地址，并进行发送。

## 问题7

哪个或者哪些进程执行这个恶意攻击，为什么？

在刚刚使用IDA进行分析时，这个程序层进行过一些比较

```

; "THEBAT.EXE"

Size
ce_0 ; "THEBAT.EXE"

ce ; "OUTLOOK.EXE"

; Size
ce_0 ; "OUTLOOK.EXE"

xe ; "MSIMN.EXE"

; Size
xe_0 ; "MSIMN.EXE"
1

```

在这里其实就是查看当前进程的名字是不是这几个，如果是的话才会执行攻击，否则退出。而这几个程序都是属于电子邮件客户端的程序，这样能够将自己的功能隐藏在这些进程中，不会容易被发现。

## 问题8

.ini文件的意义是什么

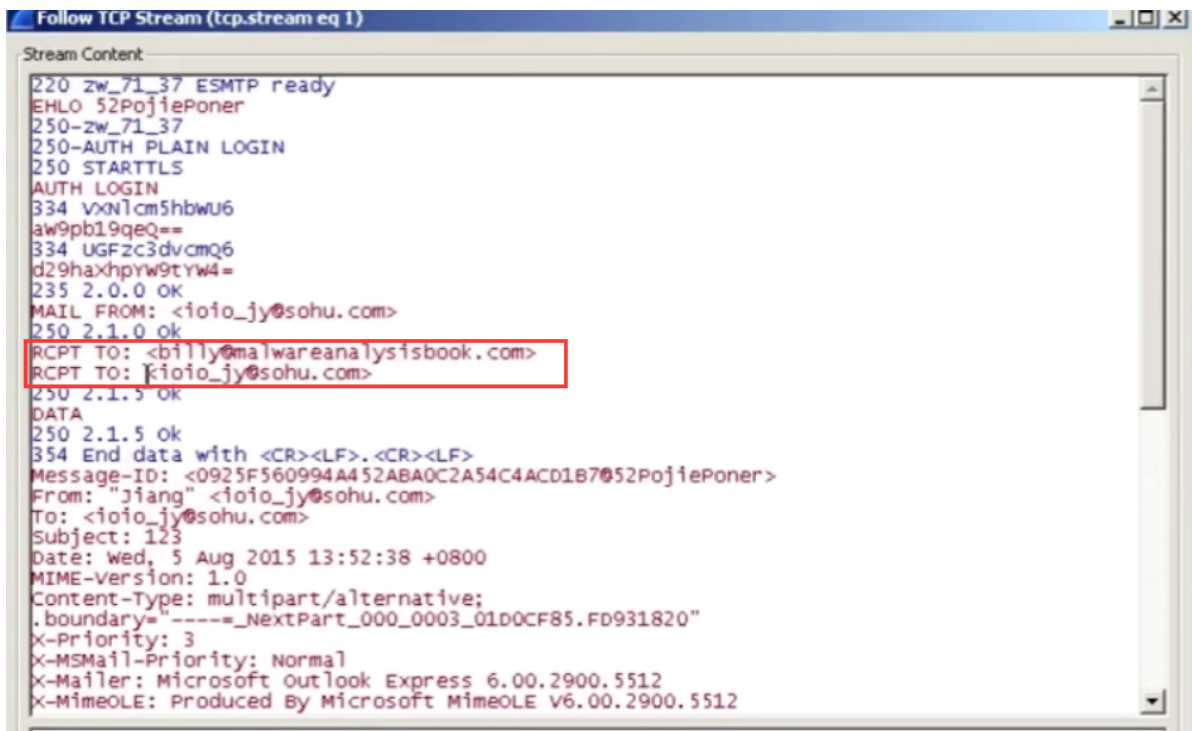
之前在分析的时候，我们发现这个恶意代码打开了这个ini文件，然后进行了解码，得到一个恶意邮箱的地址。那么这个文件其实就是用来保存编码过后的恶意邮箱的地址。

## 问题9

你怎样用Wireshark动态捕获这个恶意代码的行为

首先我们知道了这个恶意代码是附带在邮件程序中进行的，如果没有这几个程序是没有办法捕获到恶意代码的行为的，所以我们安装了一个outlook程序，用来收发邮件。

之后我们使用这个程序随便发送一封邮件，同时使用wireshark捕捉一下网络数据包



我们捕获到了一个SMTP的数据包，在打开以后发现，这里不仅发送了我们输入的邮箱地址，还发送到了之前解析出来的恶意邮箱

## Lab 11-3

### 问题1

使用基础的静态分析过程，你可以发现什么有趣的线索？

先简单使用srrgs工具查看一下有没有什么有趣的字符串

lab11-03.exe

```
C:\WINDOWS\System32\inet_epar32.dll
zzz69806582
.text
net start cisvc
C:\WINDOWS\System32\vs
cisvc.exe
Lab11-03.dll
C:\WINDOWS\System32\inet_epar32.dll
```

在exe的检查中可以看见有一个字符串是 `net start cisvc`，这里就是启动一个名为cisvc的服务，经过查询可以知道，这个服务是用来检测系统内存的。之后可以看见有一个系统路径的dll文件，说明这个恶意代码可能会在这个位置创建一个dll文件。

lab11-03.dll

```

<program name unknown>
GetLastActivePopup
GetActiveWindow
MessageBoxA
user32.dll
H:mm:ss
dddd, MMMM dd, yyyy
M/d/yy
December
November
October
September
August
July
June
April
March
February
January
Dec
Nov
Oct
Sep
Aug
Jul
Jun
May
Apr
Mar
Feb
Jan
Saturday
Friday
Thursday
Wednesday
Tuesday
Monday
Sunday
Sat
Fri
Thu
Wed
Tue
Mon
Sun
SunMonTueWedThuFriSat
JanFebMarAprMayJunJulAugSepOctNovDec

```

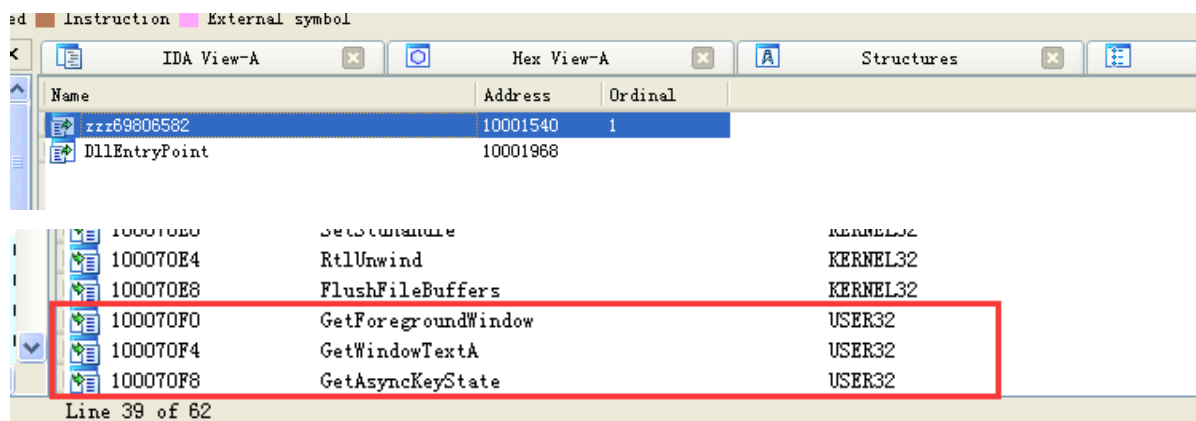
```

ActiveWindow
FlushFileBuffers
Lab1103.dll.dll
zzz69806582
0x0x
<SHIFT>
%s: %s
C:\WINDOWS\System32\kernel64.dll
y?
@~C

```

dll中出现的字符串就比较杂乱，其中有一些是关于星期、月份的，还有一个比较值得注意的就是图中框出来的系统路径下的dll文件。

使用IDA查看一下DLL文件的导入导出函数



在导入表中发现了一些比较有趣的函数，这几个函数结合使用，可以获取用户的键盘的输入。

## 问题2

当运行这个恶意代码时，发生了什么？

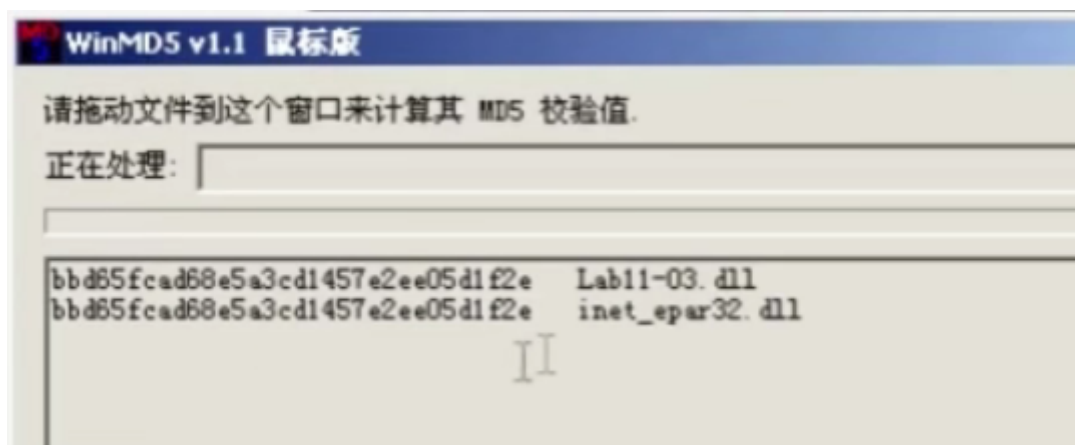
同样还是使用procmon进行监视，可以观察到弹出了一个什么内容都没有显示的命令行窗口，然后这个窗口很快就关闭了。

```

re 2484 QueryStream... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 QueryBasicI... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 QueryEaInfo... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 CreateFile C:\WINDOWS\system32\inet_epar32.dll
re 2484 CreateFile C:\WINDOWS\system32\inet_epar32.dll
re 2484 CloseFile C:\WINDOWS\system32\inet_epar32.dll
re 2484 QueryAttrib... C:\WINDOWS\system32\inet_epar32.dll
re 2484 QueryBasicI... C:\WINDOWS\system32\inet_epar32.dll
re 2484 QueryAttrib... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 SetEndOfFile... C:\WINDOWS\system32\inet_epar32.dll
re 2484 CreateFileM... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 QueryStanda... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 CreateFileM... C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 WriteFile C:\WINDOWS\system32\inet_epar32.dll
re 2484 SetBasicInf... C:\WINDOWS\system32\inet_epar32.dll
re 2484 CloseFile C:\Documents and Settings\Administrator\桌面\上机实验样本\Chapter_11\Lab11-03.dll
re 2484 CloseFile C:\WINDOWS\system32\inet_epar32.dll
re 2484 CreateFile C:\WINDOWS\system32\cisvc.exe
re 2484 QueryStanda... C:\WINDOWS\system32\cisvc.exe
re 2484 CreateFileM... C:\WINDOWS\system32\cisvc.exe

```

可以看见恶意代码在系统目录下创建了一个inet\_epar32.dll的文件，猜测这个dll文件和lab11-03.dll是同一个文件，所以还是计算一下MD5的值



果然是一样的，也就是说这个恶意代码把自己的dll文件复制到了系统目录下，并重命名为了 inet\_epar32.dll

并且发现这个代码试图打开之前说的那个exe文件



2484	CloseFile	C:\WINDOWS\system32\inet_eap32.dll	SUCCESS
2484	CreateFile	C:\WINDOWS\system32\cisvc.exe	SUCCESS
2484	QueryStand...	C:\WINDOWS\system32\cisvc.exe	SUCCESS
2484	CreateFileM...	C:\WINDOWS\system32\cisvc.exe	SUCCESS
2484	QueryStand...	C:\WINDOWS\system32\cisvc.exe	SUCCESS
2484	CloseFile	C:\WINDOWS\system32\cisvc.exe	SUCCESS
2484	QueryOpen	C:\WINDOWS\system32\cmd.exe	SUCCESS
2484	CreateFile	C:\WINDOWS\system32\cmd.exe	SUCCESS
2484	CreateFileM...	C:\WINDOWS\system32\cmd.exe	SUCCESS
2484	CreateFileM...	C:\WINDOWS\system32\cmd.exe	SUCCESS
2484	QueryOpen	C:\WINDOWS\system32\apphelp.dll	SUCCESS
2484	CreateFile	C:\WINDOWS\system32\apphelp.dll	SUCCESS
2484	CreateFileM...	C:\WINDOWS\system32\apphelp.dll	SUCCESS

## 问题3

Lab11-03.exe如何安装Lab11-03.dll使其长期驻留?

通过刚刚的行为可以知道这个恶意代码将lab11-03.dll文件重命名并复制到了Windows的系统目录下。

使用IDA进行分析

```

push    0 ; bFailIfExists
push    offset NewFileName ; "C:\\WINDOWS\\System32\\inet_eap32.dll"
push    offset ExistingFileName ; "Lab11-03.dll"
call    ds:CopyFileA
push    offset aCisvc_exe ; "cisvc.exe"
push    offset aCWindowsSyst_0 ; "C:\\WINDOWS\\System32\\%s"
lea     eax, [ebp+FileName]
push    eax ; char *
call    _sprintf

```

可以看见这个恶意代码先执行了之前说的复制行为，然后创建了一个

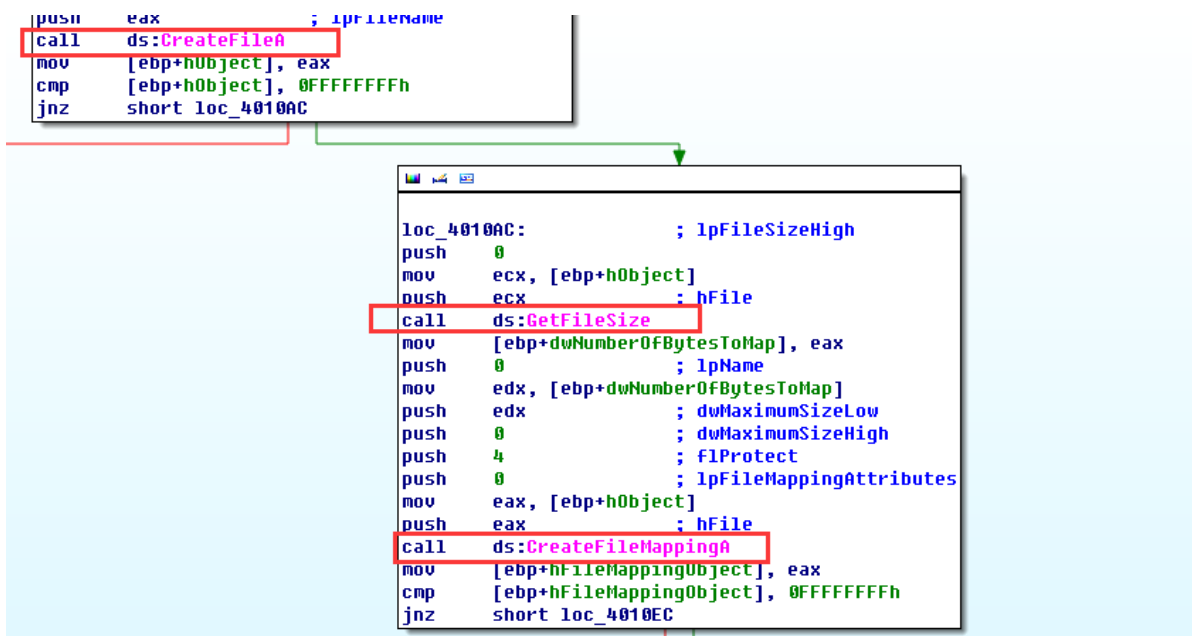
C:\\WINDOWS\\System32\\cisvc.exe 的字符串，并作为参数传递给下面的sub\_401070函数，最后执行了之前分析的net start cisvc 指令。

```

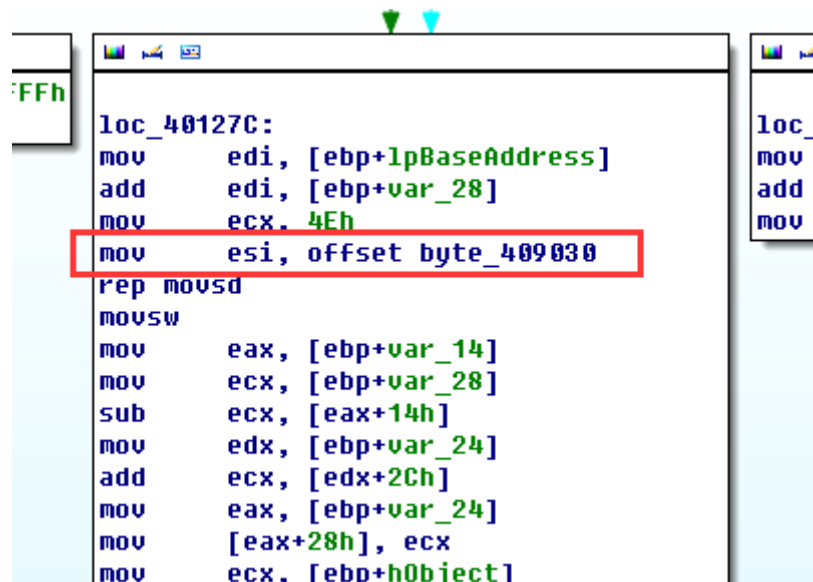
lea     eax, [ebp+FileName]
push    eax ; char *
call    _sprintf
add     esp, 0Ch
lea     ecx, [ebp+FileName]
push    ecx ; lpFileName
call    sub_401070
add     esp, 4
push    offset aNetStartCisvc ; "net start cisvc"
call    system
add     esp, 4
xor     eax, eax
mov     esp, ebp
pop     ebp

```

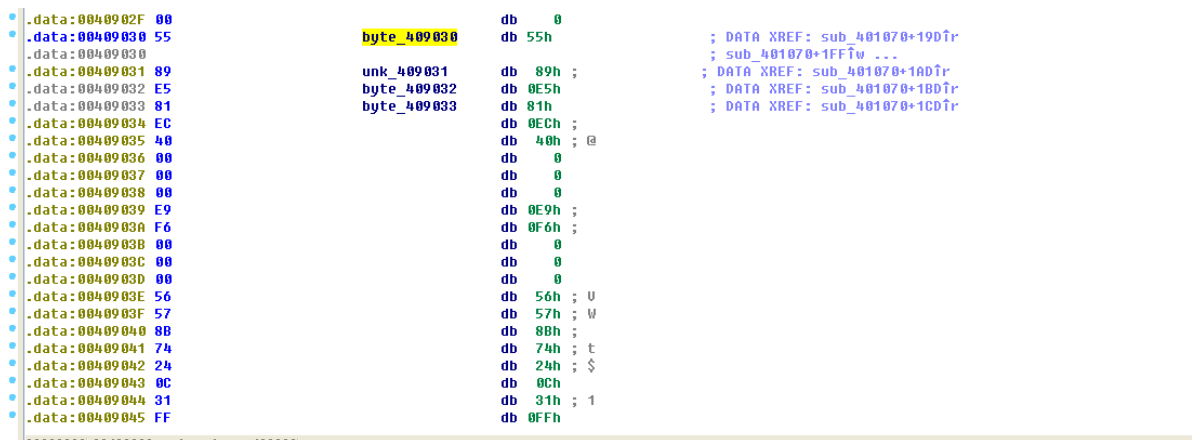
接下来查看一下401070这个函数的内容



发现这个函数创建了文件之后，获取文件的大小，后续的操作就是将这个文件映射到内存中，但是我们注意到一个地方：



查看这里的内容以后发现



这里是一串连续的但是看不懂的内容，尝试转换成字符串的时候还是看不懂，尝试转换成代码以后发现

```

030 ; -----
030
030 loc_409030: ; DATA XREF: sub_401070+19D↑r
030 ; sub_401070+1FF↑w ...
030 55 push ebp
031
031 loc_409031: ; DATA XREF: sub_401070+1AD↑r
031 ; sub_401070+18D↑r
031 89 E5 mov ebp, esp
033
033 loc_409033: ; DATA XREF: sub_401070+1CD↑r
033 81 EC 40 00 00 00 sub esp, 40h
039 E9 F6 00 00 00 jmp loc_409134
03E
03E ; ===== S U B R O U T I N E =====
03E
03E sub_40903E proc near ; CODE XREF: sub_40905F+2E↓p
03E
03E arg_0 = dword ptr 4
03E
03E 56 push esi
03F 57 push edi
040 8B 74 24 0C mov esi, [esp+8+arg_0]
044 31 FF xor edi, edi
046 FC ctd

```

原来这里就是一些汇编的代码，也就是shellcode，看见下面有一个跳转指令，切换过去可以看见：

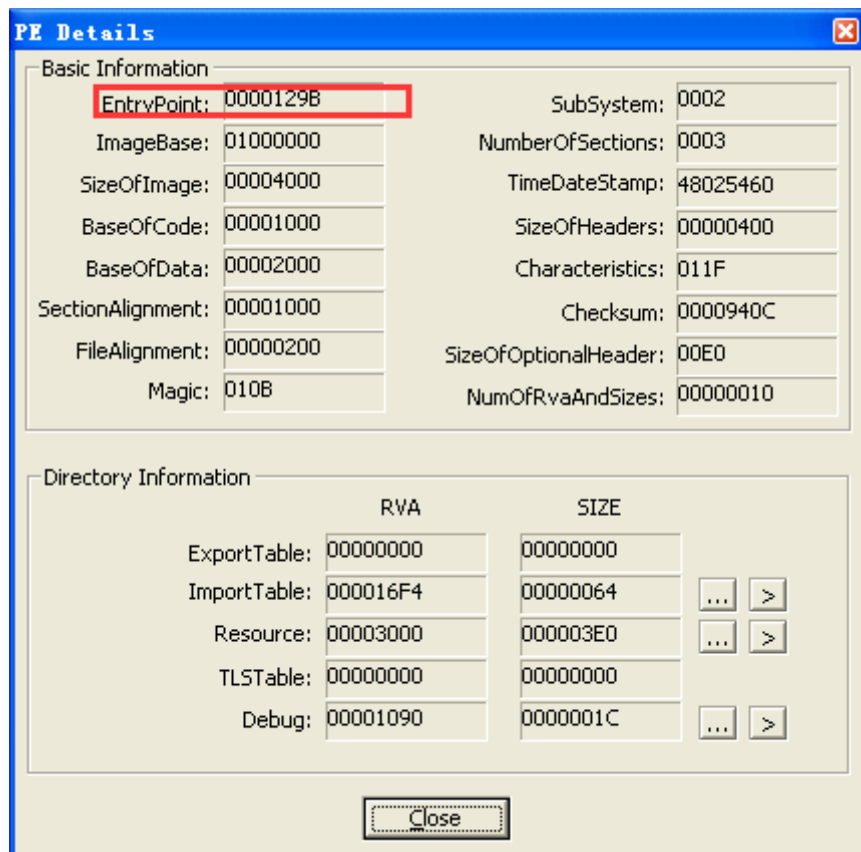
```

FF loc_409134: ; CODE XREF: .data:0040
; -----
db 43h ; C
db 3Ah ; :
db 5Ch ; \
db 57h ; W
db 49h ; I
db 4Eh ; N
db 44h ; D
db 4Fh ; O
db 57h ; W
db 53h ; S
db 5Ch ; \
db 53h ; S
db 79h ; y
db 73h ; s
db 74h ; t
db 65h ; e
db 6Dh ; m
db 33h ; 3
db 32h ; 2
db 5Ch ; \
db 69h ; i

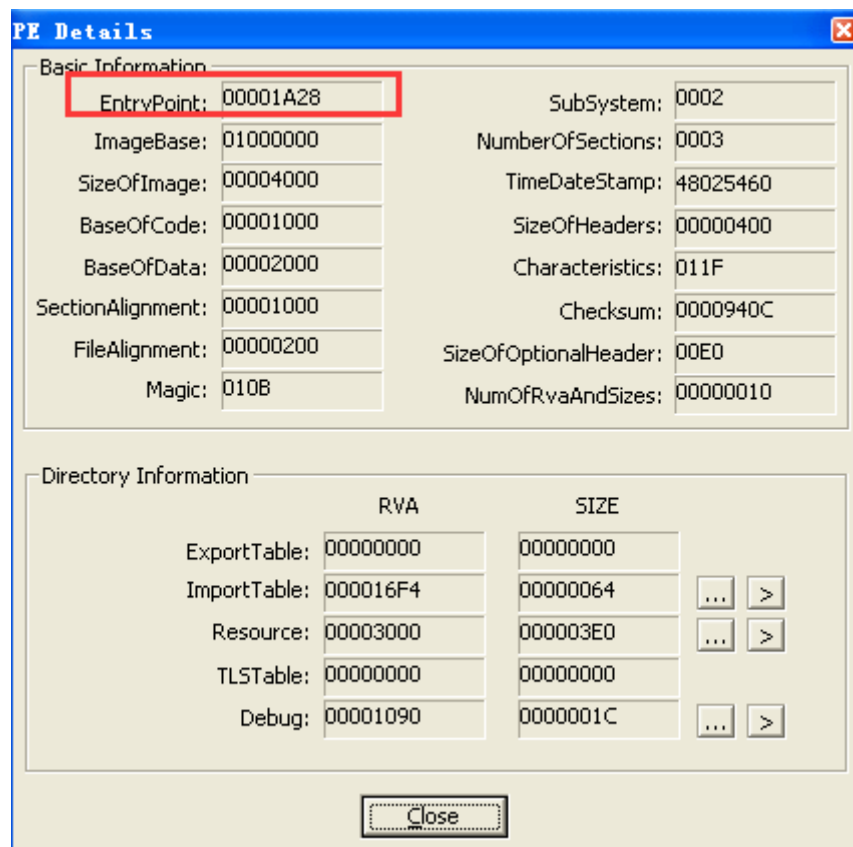
```

这里构建了两个字符串，一个是C盘目录下的那个dll文件，还有一个是dll文件的导出函数，那么就不难联想到这个恶意代码会调用这个dll文件的导出函数。

同时我们可以观察到，系统原本的cisvc.exe的入口点是这样的



在我们运行了恶意代码之后，程序的入口点变成了



为了能更加直观的看见这个变化，我们直接分析受到感染之后的文件。

使用IDA查看以后，可以发现这是一个调用链

```

start proc near

; FUNCTION CHUNK AT 01001B2C SIZE 00000036 BYTES

push    ebp
mov     ebp, esp
sub     esp, 40h
jmp     loc_1001B2C
start endp ; sp-analysis failed

```

```

; START OF FUNCTION CHUNK FOR start

loc_1001B2C:
call    sub_1001AD5
inc     ebx
cmp     bl, [edi+edx*2+49h]
dec     esi
inc     esp
dec     edi
push    edi

```

```

; FUNCTION CHUNK AT 01001407 SIZE 00000000 BYTES

pop     ebx
call    sub_1001AB4
mov     edx, eax
push    753A4FCh
push    edx
call    sub_1001A57
mov     [ebp-4], eax
push    7C0DFCAAh
push    edx
call    sub_1001A57
mov     [ebp-0Ch], eax
lea     eax, [ebx+0]
push    0
push    0
push    eax
call    dword ptr [ebp-4]
mov     [ebp-10h], eax
lea     eax, [ebx+24h]
push    eax
mov     eax, [ebp-10h]
push    eax
call    dword ptr [ebp-0Ch]
mov     [ebp-8], eax
call    dword ptr [ebp-8]
mov     esp, ebp

```

其中在 `call dword ptr [ebp-4]` 这里的时候，IDA就不好看出来究竟是什么了，所以接下来我们使用OD查看一下这一段到底干了什么

01001B09	. 50	PUSH EAX	
01001B0A	. FF55 FC	CALL DWORD PTR SS:[EBP-4]	kernel32.LoadLibraryExA
01001B0D	. 8945 F0	MOV DWORD PTR SS:[EBP-10], EAX	
01001B1A	. 50	PUSH EAX	
01001B1B	. FF55 F4	CALL DWORD PTR SS:[EBP-C]	kernel32.GetProcAddress
01001B1E	. 8945 F8	MOV DWORD PTR SS:[EBP-8], EAX	

此时栈上的状态为

0007FF78	10000000	
0007FF7C	01001B55	ASCII "zzz69806582"
0007FF80	00000000	
-----	-----	

之后调用栈上的函数

01001B1A	. 50	PUSH EAX	
01001B1B	. FF55 F4	CALL DWORD PTR SS:[EBP-C]	kernel32.GetProcAddress
01001B1E	. 8945 F8	MOV DWORD PTR SS:[EBP-8], EAX	

通过上面的依次查看，可以知道这里的代码就是执行了调用注入的恶意dll中的导出函数

```
D40 . 50 FOR EBP
B27 . ^ E9 6FF7FFFF JMP cisvc.0100129B
B2C > E8 A4FFFFFF CALL cisvc.01001AD5
```

之后执行正常的受感染文件的功能.

也就是说，这里是靠感染一个系统会使用的程序，让每次在执行这个程序之前调用注入的shellcode，从而达到驻留的目的。

## 问题4

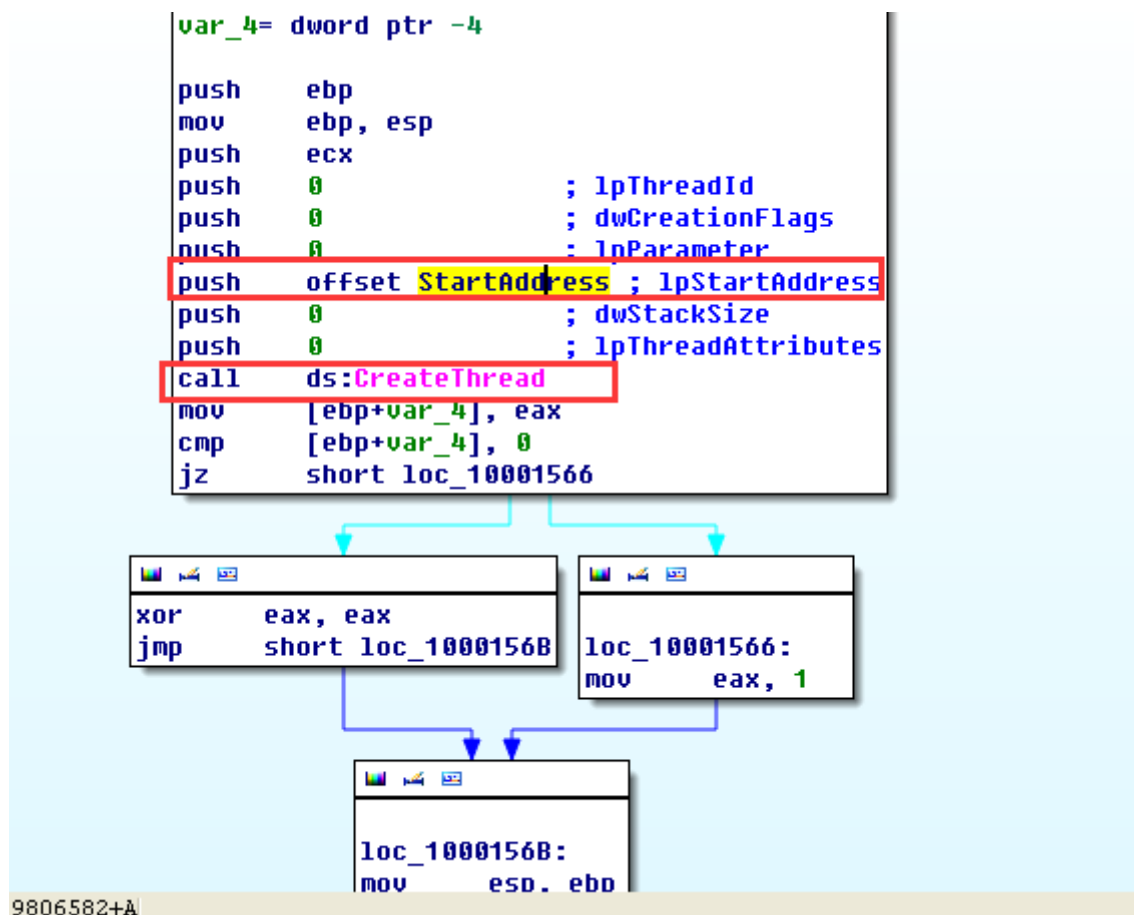
这个恶意代码感染Windows系统的哪个文件？

通过之前的分析可以看见是cisvc.exe，修改了这个文件的入口地址，使得程序每次执行的时候都会先执行shellcode

## 问题5

Lab11-03.dll做了什么？

我们使用IDA分析这个dll文件，同时通过之前的分析可以知道这个dll文件主要的不是dllmainentry，而是他的另一个导出函数，所以这里就只分析这个导出函数。



可以发现这个导出函数先创建了一个线程，然后这个函数就结束了，但是在创建线程的时候，其中设置了一个启动函数的地址。

```

push    ecx                ; size_t
push    offset byte_1000AD28 ; unsigned __int8 *
lea     edx, [ebp+var_80C]
push    edx                ; unsigned __int8 *
call    __mbsncpy
add     esp, 0Ch
push    offset Name        ; "MZ"
push    0                  ; bInheritHandle
push    1F0001h            ; dwDesiredAccess
call    ds:OpenMutexA
mov     [ebp+var_818], eax
cmp     [ebp+var_818], 0
jz      short loc_1000149D

```

进入这个函数可以发现，这个函数首先查看系统中有没有一个名为MZ的互斥量，如果没有就会创建。

```

loc_1000149D:                ; "MZ"
push    offset Name
push    1                   ; bInitialOwner
push    0                   ; lpMutexAttributes
call    ds:CreateMutexA
mov     [ebp+var_818], eax
cmp     [ebp+var_818], 0
jnz     short loc_100014BD

```

这个互斥量的作用自然就是防止多个程序的运行，确保每个时刻最多只有规定数量的程序在运行。

```

loc_100014BD:                ; hTemplateFile
push    0
push    80h                ; dwFlagsAndAttributes
push    4                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    1                   ; dwShareMode
push    0C0000000h         ; dwDesiredAccess
push    offset FileName    ; "C:\\WINDOWS\\System32\\kernel64x.dll"
call    ds:CreateFileA
mov     [ebp+hObject], eax
cmp     [ebp+hObject], 0
jnz     short loc_100014EB

```

然后可以看见这个代码创建了系统目录下的一个dll文件。

```

loc_100014EB:                ; dwMoveMethod
push    2
push    0                   ; lpDistanceToMoveHigh
push    0                   ; lDistanceToMove
mov     eax, [ebp+hObject]
push    eax                 ; hFile
call    ds:SetFilePointer
mov     ecx, [ebp+hObject]
mov     [ebp+var_4], ecx
lea     edx, [ebp+var_810]
push    edx
call    sub_10001380

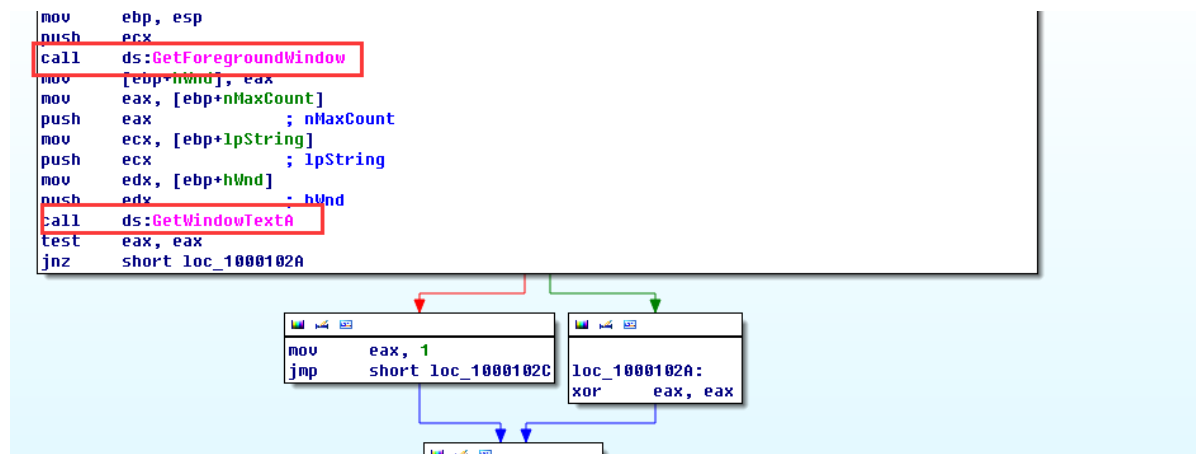
```

之后通过 `SetFilePointer` 函数，将指针移动到文件的末尾（`dwMoveMethod`的参数为2）

进入到后面调用的sub\_10001380函数，可以看见

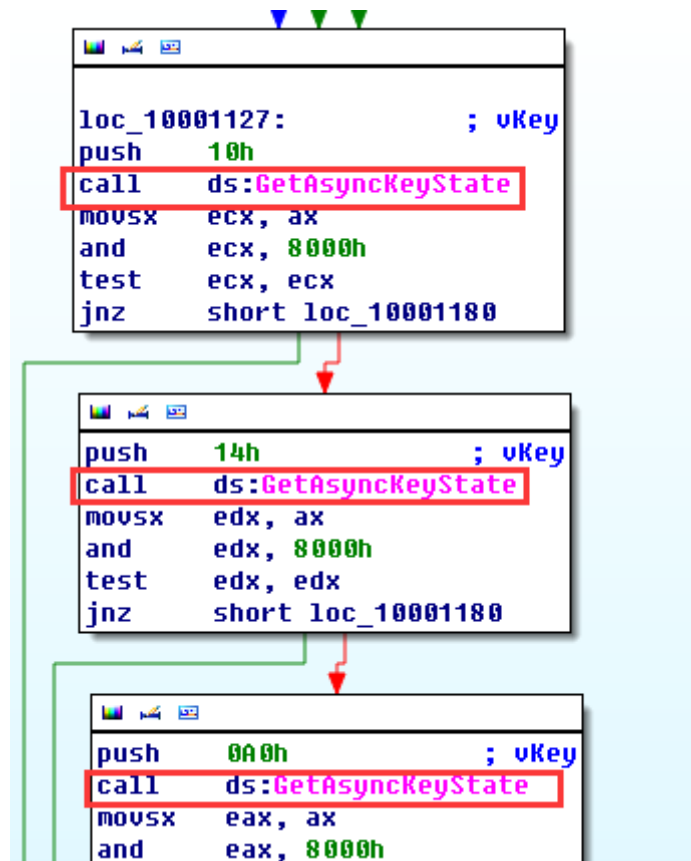
```
mov     edx, [ebp+arg_0]
add     edx, 408h
push    edx
mov     eax, [ebp+arg_0]
add     eax, 4
push    eax
push    offset aSS      ; "%S: %S\n"
lea     ecx, [ebp+Buffer]
push    ecx             ; char *
call    _sprintf
add     esp, 10h
push    0               ; lpOverlapped
lea     edx, [ebp+NumberOfBytesWritten]
push    edx             ; lpNumberOfBytesWritten
lea     edi, [ebp+Buffer]
or      ecx, 0FFFFFFFh
xor     eax, eax
repne scasb
not     ecx
add     ecx, 0FFFFFFFh
push    ecx             ; nNumberOfBytesToWrite
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
mov     ecx, [ebp+arg_0]
mov     edx, [ecx+80Ch]
push    edx             ; hFile
call    ds:WriteFile
```

这里调用了写文件的函数，以及上面出现了一些格式化字符串，那么整体来说这个函数的功能就是像刚刚打开的文件的末尾写入一些内容。



进入到嵌套调用的函数中可以发现这个恶意代码获取了当前哪个程序正在执行输入，并获取当前窗口的标题。





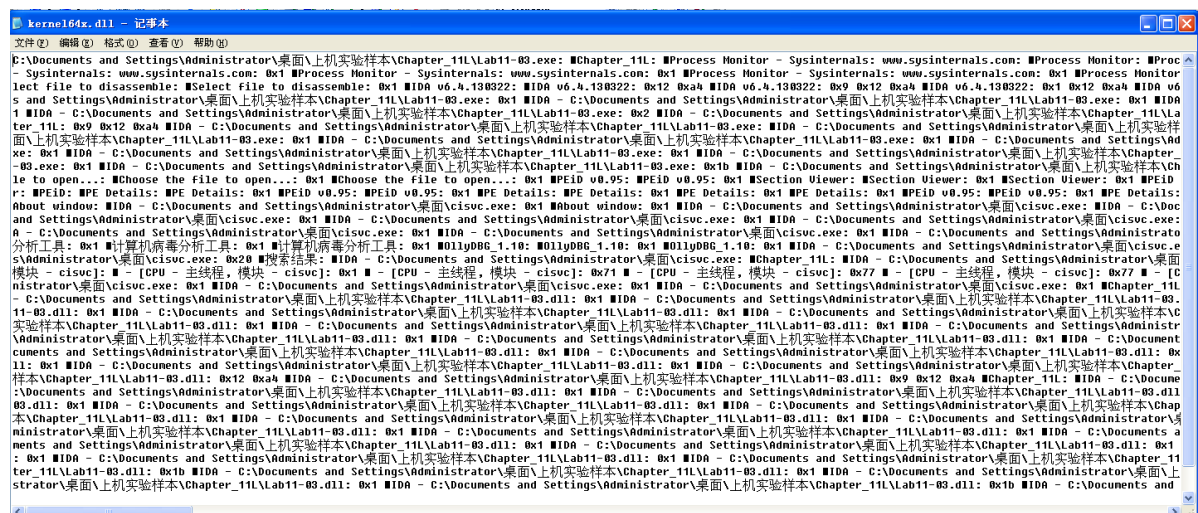
然后函数多次调用了 `GetAsyncKeyState` 这个函数，来识别一个按键是否被按下或者弹起（对比按键的状态），也就是轮询来获取键盘状态的变化，也就获得了键盘的输入。

综上，这个dll文件会创建一个线程，并通过创建互斥量来保证同时只运行一个线程，然后打开系统目录下的一个dll文件，通过对比键盘状态来记录当前窗口的输入。

## 问题6

这个恶意代码将收集的数据存放在何处？

通过问题5中的分析，可以知道保存在了 `C:\\WINDOWS\\System32\\kernel64x.dll` 中，找到这个文件我们可以看见确实已经记录了很多内容



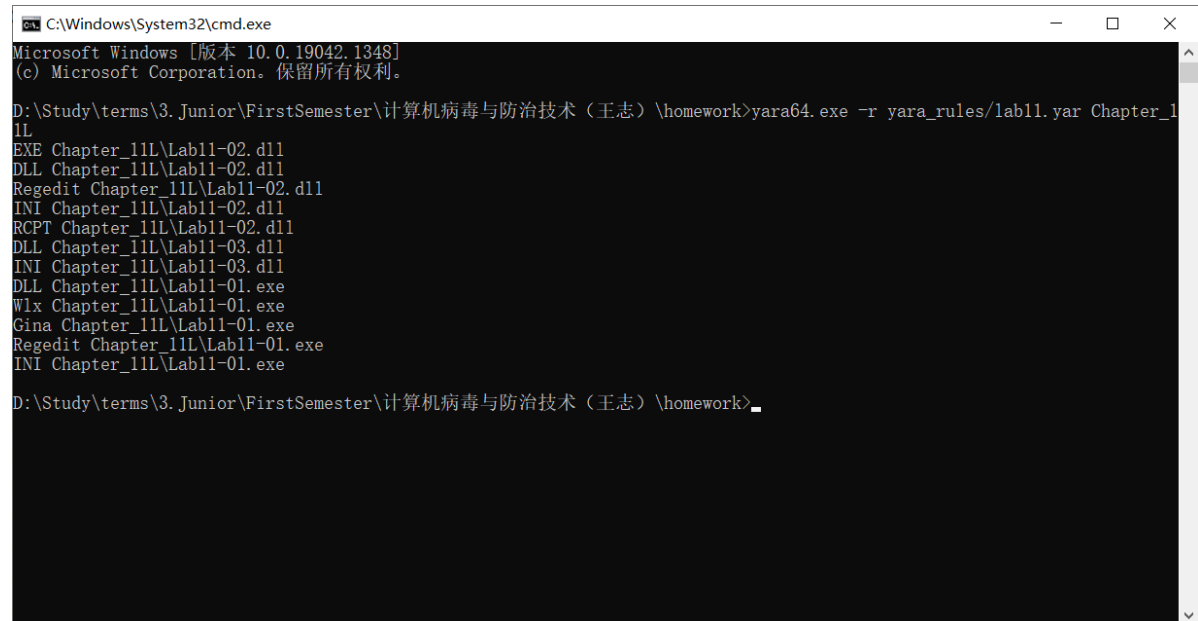
# Yara

根据内容编写yara规则如下:

```
1  import "pe"
2
3  rule EXE {
4      strings:
5          $exe = ".exe" nocase
6      condition:
7          $exe
8  }
9
10 rule DLL {
11     strings:
12         $dll = /[a-zA-Z0-9_]*.dll/
13     condition:
14         $dll
15 }
16
17 rule Wlx {
18     strings:
19         $WlxFuncs = /Wlx[a-zA-Z]*/
20     condition:
21         $WlxFuncs
22 }
23
24 rule Gina {
25     strings:
26         $name = "Gina"
27     condition:
28         $name
29 }
30
31 rule Regedit {
32     strings:
33         $system = "NT"
34         $software = "SOFTWARE"
35         $winlogon = "winlogon"
36     condition:
37         $system or $software or $winlogon
38 }
39
40 rule INI {
41     strings:
42         $name = /[a-zA-Z0-9_]*.ini/
43     condition:
44         $name
45 }
46
47 rule Service {
48     strings:
49         $start = "net start"
50     condition:
51         $start
52 }
```

```
53  
54 rule RCPT {  
55     strings:  
56         $name = "RCPT TO"  
57     condition:  
58         $name  
59 }
```

实验结果如下



```
C:\Windows\System32\cmd.exe  
Microsoft Windows [版本 10.0.19042.1348]  
(c) Microsoft Corporation。保留所有权利。  
  
D:\Study\terms\3. Junior\FirstSemester\计算机病毒与防治技术（王志）\homework>yara64.exe -r yara_rules\lab11.yar Chapter_11L  
EXE Chapter_11L\Lab11-02.dll  
DLL Chapter_11L\Lab11-02.dll  
Regedit Chapter_11L\Lab11-02.dll  
INI Chapter_11L\Lab11-02.dll  
RCPT Chapter_11L\Lab11-02.dll  
DLL Chapter_11L\Lab11-03.dll  
INI Chapter_11L\Lab11-03.dll  
DLL Chapter_11L\Lab11-01.exe  
Wlx Chapter_11L\Lab11-01.exe  
Gina Chapter_11L\Lab11-01.exe  
Regedit Chapter_11L\Lab11-01.exe  
INI Chapter_11L\Lab11-01.exe  
  
D:\Study\terms\3. Junior\FirstSemester\计算机病毒与防治技术（王志）\homework>
```