网络安全入门实验01：逆向分析三个文件

原创

Krumitz 于 2019-03-08 00:06:12 发布 3079 收藏 14
分类专栏： 网络安全实验报告

目录

0.前言

大学三年终于上到了最最最想学的课。

经历过买了很多相关书籍、看了很多相关视频，但是发现其中大部分讲的太过浅显，感觉并没有什么实际作用、不知道从哪里入手、讲的太难又看不懂的迷惘

曾经觉得“入门太简单，但要做的很好太难”

曾经怀疑过，为什么别人都可以自学，为什么自己这么差劲

在那一个个失落的夜晚之后，今天终于真的真的真的要入门了。

发现了原来在推开实现梦想的大门，踏上第一级阶梯之前，一切的失意都不值得一提。

甚至兴奋地颤抖。

所以在这里，将课堂的实验，以实验报告的形式发表博客，记录下自己学习的过程，也希望可以跟大家分享这个过程

如有不足、如果错误，还希望大家指正

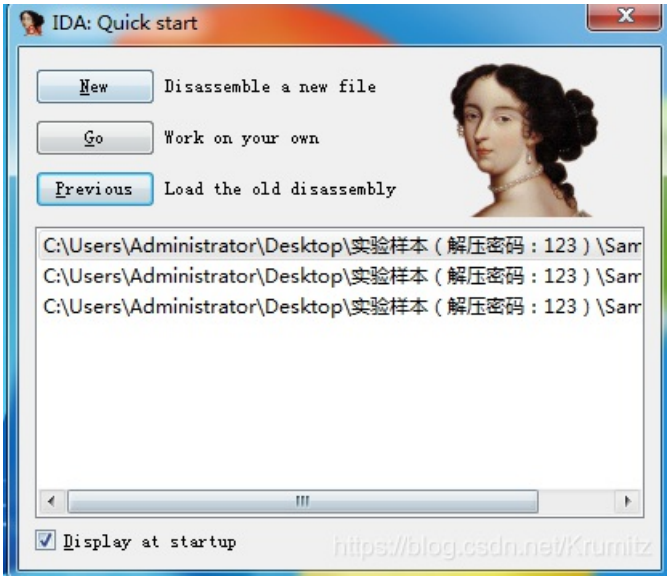
1.实验环境搭建

本周的第一个实验是逆向分析三个简单的文件，不需要搭配什么特别的环境

需要的软件：IDA PRO

p.s PRO专业版和普通的版本打开文件后的默认窗口是略有差别的，提供的功能也是不一样的

2.IDA PRO简单介绍

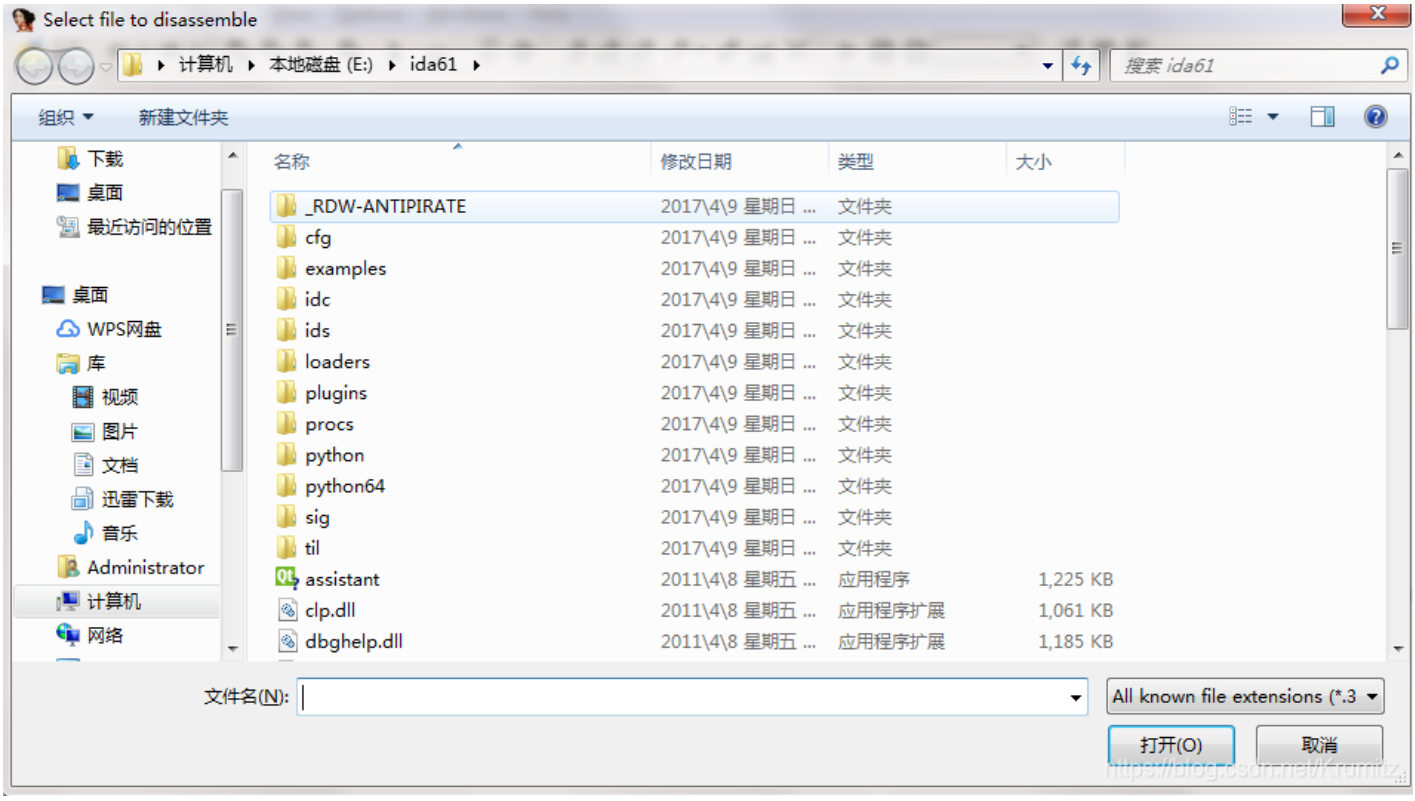


打开IDA PRO会显示快速启动的界面

其中有三个选项：“New”、“Go”、“Previous”

这里我们选择第一个，解析一个新的文件

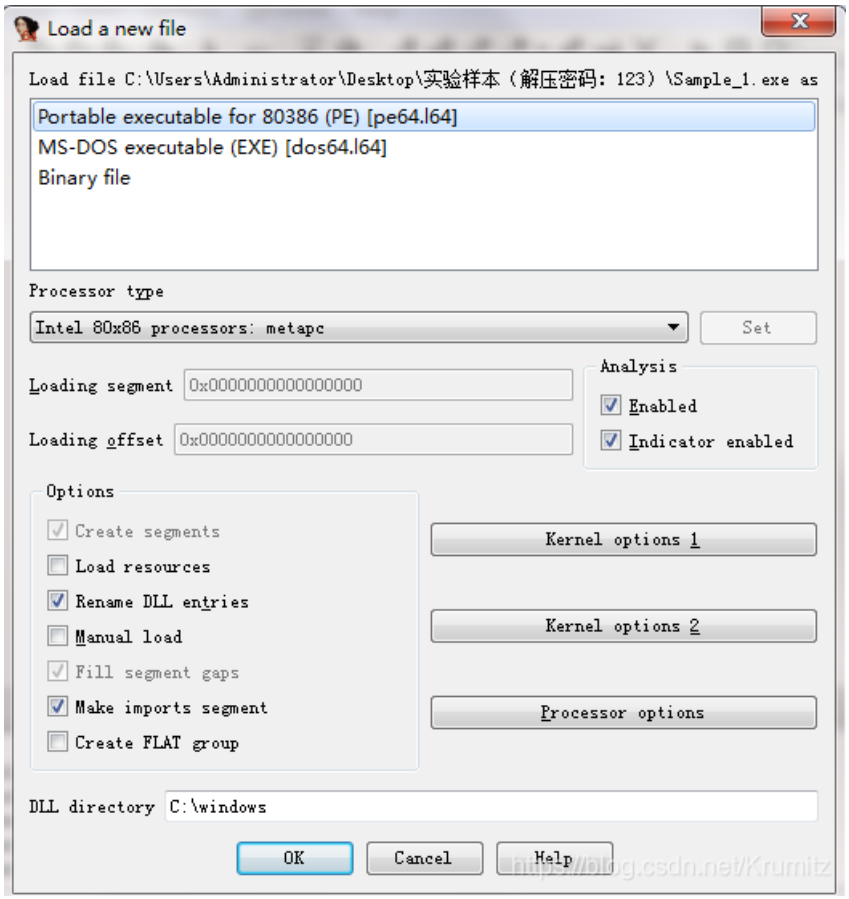
然后会默认打开一个“打开文件的窗口”



我们可以从这里选择要解析的文件，也可以直接拖动文件到IDA PRO里面进行解析

```
Drag a file here to disassemble it
```

在加载文件时有多个选项



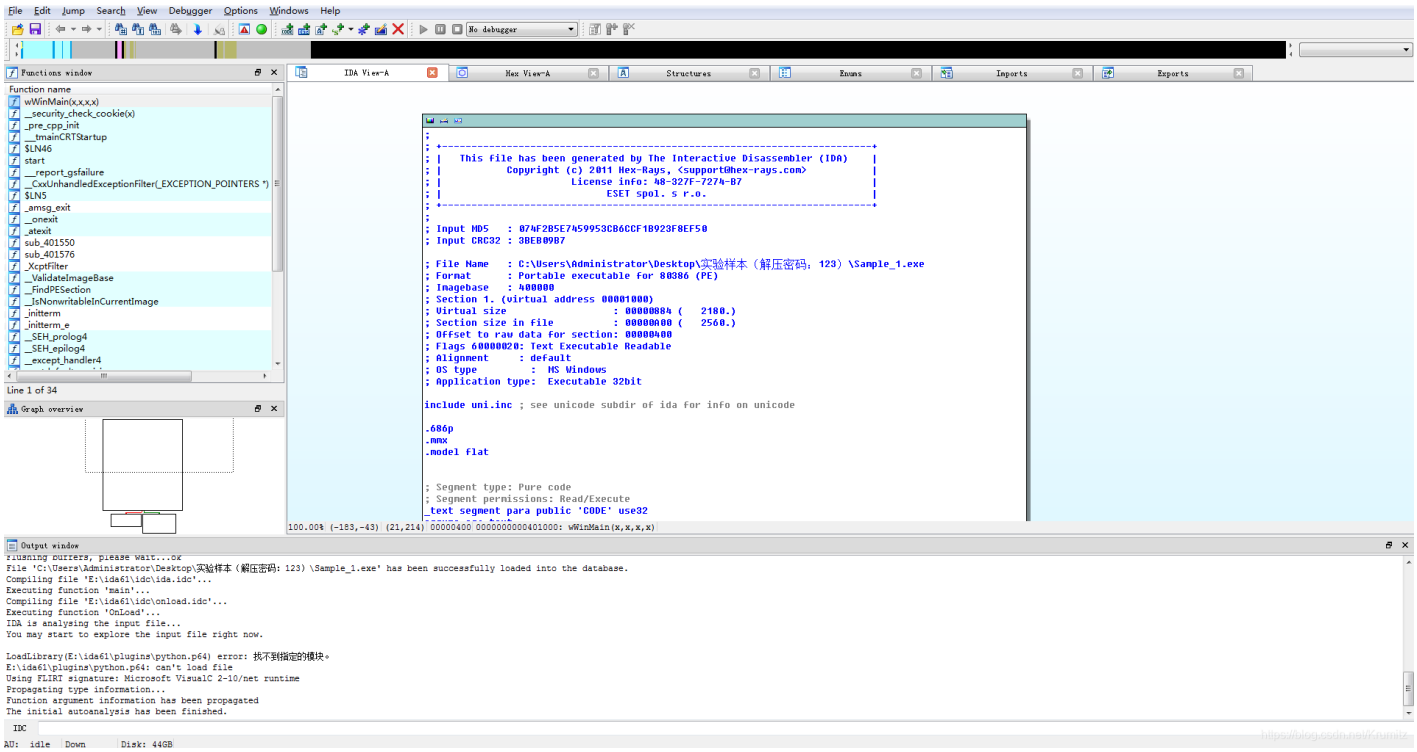
主要是在窗口最上方的选项

“Load file xxx as”（程序的类型）

这里我们选择默认的第一个，其他选项也按照默认的参数来，点击OK即可

我们选择的文件是Sample_1的可执行文件

OK后，开始Disassemble我们选择的文件，出现如下窗口



我们可以看到默认显示的有

“Function Name”窗口，即显示可执行文件所调用的函数

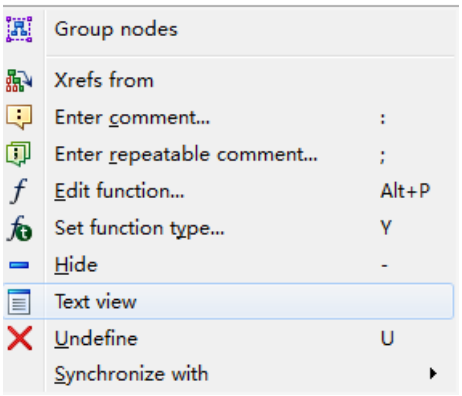
“IDA View-A”窗口，即可执行文件的反汇编窗口

“Output Window”，即可执行文件的输出

这里我们主要关注的是反汇编窗口，

其余的有关以十六进制显示的窗口还没有学到，在此先不做研究

在反汇编窗口点击鼠标右键，可以看到有多个选项，我们可以通过点击“Text View”来更改视图（默认以Graph View显示）



其余相关介绍可以参考其他的参考资料，本文仅粗略介绍本次实验需要用到的部分

3.Sample_01的逆向分析

```
; int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
_wWinMain@16 proc near

hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpString1= dword ptr  10h
nShowCmd= dword ptr  14h

push    ebp
mov     ebp, esp
mov     eax, [ebp+lpString1]
push    offset String2  ; "2012"
push    eax             ; lpString1
call    ds:lstrcmpW
push    0               ; uType
push    offset Caption  ; "MESSAGE"
test    eax, eax
jnz     short loc_401035
```

```
push    offset Text     ; "Hello! 2012"
call    ds:GetActiveWindow
push    eax             ; hWnd
call    ds:MessageBoxW
xor     eax, eax
pop     ebp
retn    10h
```

```
loc_401035:             ; "Hello! Windows"
push    offset aHelloWindows
call    ds:GetActiveWindow
push    eax             ; hWnd
call    ds:MessageBoxW
xor     eax, eax
pop     ebp
retn    10h
_wWinMain@16 endp
```



```
; int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
_wWinMain@16 proc near

hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpString1= dword ptr  10h
nShowCmd= dword ptr  14h

push    ebp
mov     ebp, esp
mov     eax, [ebp+lpString1]
push    offset String2  ; "2012"
push    eax             ; lpString1
call    ds:lstrcmpW
push    0               ; uType
push    offset Caption  ; "MESSAGE"
test    eax, eax
jnz     short loc_401035
```


打开Sample_01.exe后，跳过上面的文件信息的部分，我们就找到了这个可执行文件的Main函数的入口（黄色选中部分）

那么黑框里面的，就肯定是主函数的部分啦

前三行我们可以先不详细分析，因为本次实验的目的主要是通过逆向分析猜测可执行文件会执行一个怎么样的操作，并且解释相关函数的作用

那么我们从第四行开始看

```
push    offset String2  ; "2012"
push    eax             ; lpString1
```

我们可以看到，函数将“2012”，和lpString1 push入栈

接着就来到了

```
call    ds:lstrcmpW
```

call表示调用了一个函数，这个函数的名称为“lstrcmpW”

通过参考微软的函数手册，我们可以看到

Syntax

C++ 复制

```
int lstrcmpW(
  LPCWSTR lpString1,
  LPCWSTR lpString2
);
```

Parameters

`lpString1`

Type: LPCTSTR

The first null-terminated string to be compared.

`lpString2`

Type: LPCTSTR

The second null-terminated string to be compared.

Return Value

Type: int

If the string pointed to by *lpString1* is less than the string pointed to by *lpString2*, the return value is negative. If the string pointed to by *lpString1* is greater than the string pointed to by *lpString2*, the return value is positive. If the strings are equal, the return value is zero.

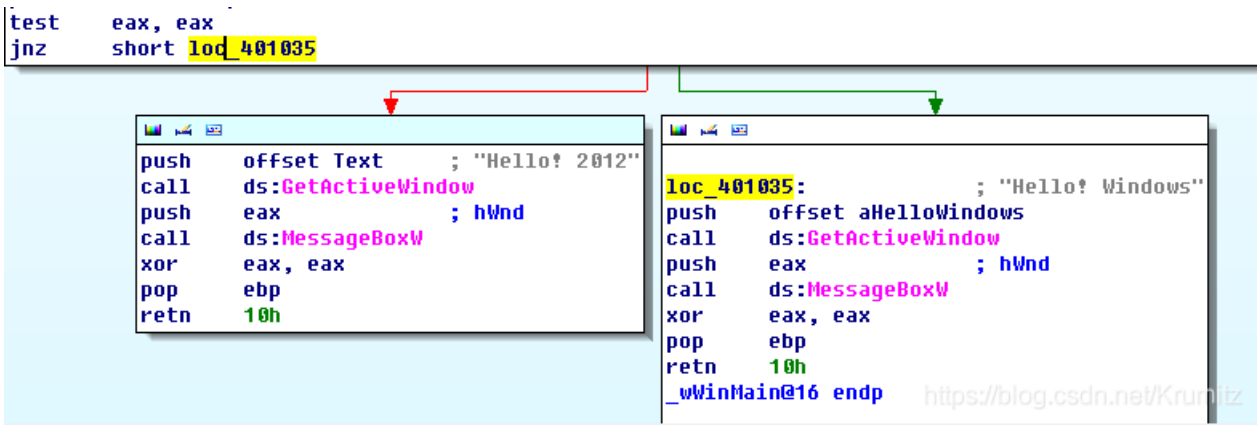
lstrcmpw函数比较了两个字符串lpString1和lpString2的值

如果lpString1小于lpString2，则函数返回一个负数

如果lpString大于lpString2，则函数返回一个正数

如果两个字符串相等，返回值为0

所以在这里我们就可以进行猜测了，这个程序首先将我们从命令行输入的值跟“2012”进行了比较，比较之后做了什么呢？让我们继续往下看



接下来test了两个值，通过查阅资料，我们知道了在反汇编中

test和jnz在这里是连用的

首先 test eax eax的含义是

```
if(EAX == 0)
ZF = 1
else
ZF = 0
```

而JNZ short loc_401035是

```
if (ZF == 0)
GOTO loc_401035
```

那么就是，首先test了一个值，看这个值是否为0

如果为0，则ZF = 1，如果不为0，ZF = 0

然后

如果ZF == 0

就跳转到loc_401035

那loc_401035是啥

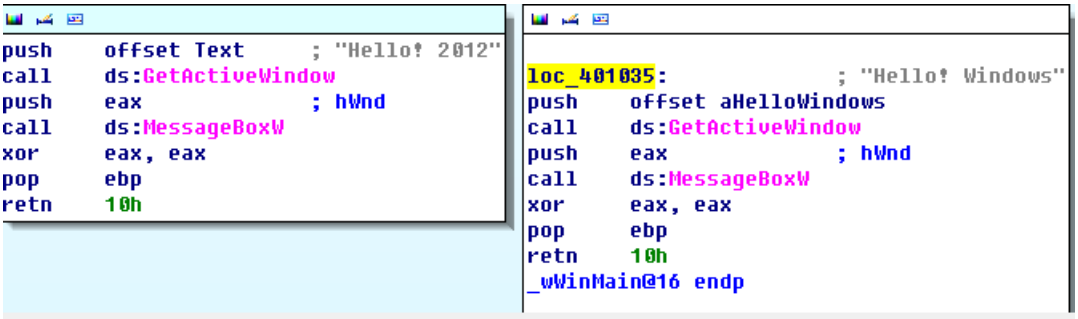
还好当你选中这个值的时候，软件会自动帮你选中这个窗口中所有相同的值

那我们就可以猜测

比较了lstrcmpW函数的值是否为0

如果函数返回0，则ZF为1，则不会跳转

如果函数返回不为0，则ZF为0，则会跳转到loc_401035，执行他接下来的代码



接下来就很简单了，我们看到两边的代码（跳转 or 没有跳转）

分别有“Hello ！ 2012”和“Hello！Windows”

然后都call了“GetActiveWindow”和“MessageBoxW”函数

那么可以猜测

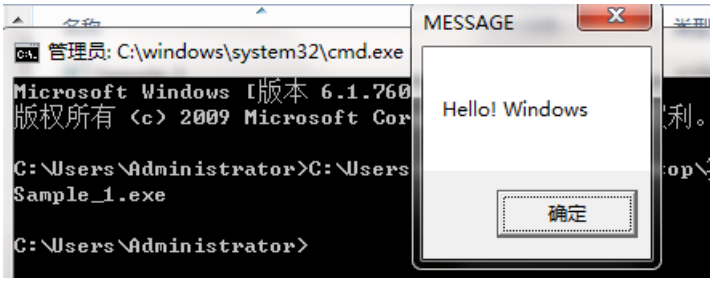
如果没有跳转，则程序获取一个窗口句柄，并show了一个MessageBox，显示Hello！2012

如果跳转了，则程序获取一个窗口句柄，并show了一个MessageBox，显示Hello！Windows

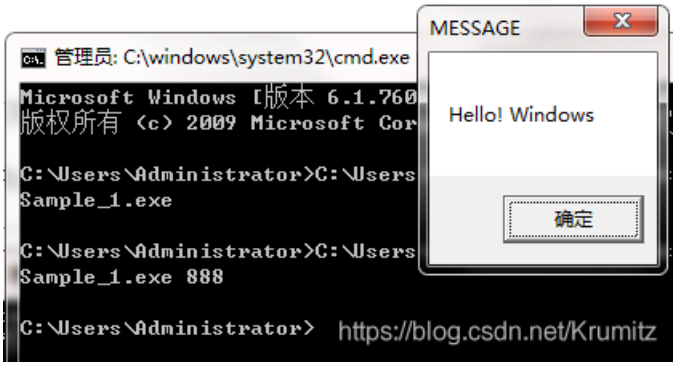
最后程序return结束

我们可以来测试一下

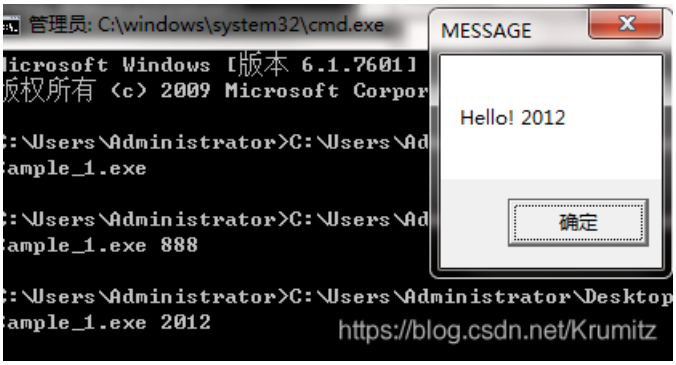
打开命令行



如果不输入任何值，很明显，lstrcmpW返回肯定不为0，显示Hello！Windows



如果输入不为2012，同理可得，也是现实Hello！Windows



输入为2012呢？那当然会显示Hello！2012了

Sample_01总结（来自图灵社区相关文章）

http://www.ituring.com.cn/book/tupubarticle/9632

“和无参数的情况相比，这次显示出来的消息变成了 Hello! 2012。

可能有人要问：“那又如何？”我们发现了通过传递 2012 这个参数，程序的显示结果会发生变化，这很重要。因为我们在“完全没有源代码的情况下，搞清楚了程序的行为”。

这就是逆向工程。

刚才这个参数是我们猜测出来的，其实只要阅读汇编语言代码，就可以发现其中使用了 lstrcmpW 对字符串 2012 和命令行参数进行了比较操作。”

“我们没必要看懂全部的汇编语言代码。和刚才使用二进制编辑器的时候一样，只要一眼望去能大概理解这段代码做了什么事就可以了。”

“刚一听到“逆向工程”“汇编”这些词的时候，大家总会以为它们很难，但实际上并非如此。使用 IDA，我们就可以将可执行文件转换成像 C 语言一样容易理解（实际上还是有差距的）的汇编代码。尤其是它的 Graph view 十分强大，可以让我们十分清晰地看出程序的分支逻辑。

只要一定程度上掌握这些工具的使用方法，大家就可以完成很多软件分析工作了。”

这三句话，也恰恰好的解释了上文所说的“本次实验的目的主要是通过逆向分析猜测可执行文件会执行一个怎么样的操作，并且解释相关函数的作用”

4.Sample_02的逆向分析

实验要求：Sample_2分析范围：0x00401000至0x0040105E

所以我们同样把Sample_02的可执行文件拖入IDA PRO，并右键切换到Text View视图

方便我们按范围分析

```
---------------
.text:00401000 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
.text:00401000 _WinMain@16     proc near               ; CODE XREF: ___tmainCRTStartup+14B↓p
.text:00401000
.text:00401000 Msg             = tagMSG ptr -820h
.text:00401000 String2         = byte ptr -804h
.text:00401000 String1         = byte ptr -404h
.text:00401000 var_4           = dword ptr -4
.text:00401000 hInstance       = dword ptr  8
.text:00401000 hPrevInstance   = dword ptr  0Ch
.text:00401000 lpCmdLine       = dword ptr  10h
.text:00401000 nCmdShow        = dword ptr  14h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 820h
.text:00401009                 mov     eax, ___security_cookie
.text:0040100E                 xor     eax, ebp
.text:00401010                 mov     [ebp+var_4], eax
.text:00401013                 push    esi
.text:00401014                 mov     esi, [ebp+hInstance]
.text:00401017                 push    400h            ; nSize
.text:0040101C                 lea     eax, [ebp+String2]
.text:00401022                 push    eax             ; lpFilename
.text:00401023                 push    0               ; hModule
.text:00401025                 call    ds:GetModuleFileNameA
.text:0040102B                 push    0               ; fCreate
.text:0040102D                 push    7               ; csidl
.text:0040102F                 lea     ecx, [ebp+String1]
.text:00401035                 push    ecx             ; pszPath
.text:00401036                 push    0               ; hwnd
.text:00401038                 call    ds:SHGetSpecialFolderPathA
.text:0040103E                 push    offset String2  ; "\\0.exe"
.text:00401043                 lea     edx, [ebp+String1]
```


首先我们还是找到文件的主函数入口

从0x00401000至0x0040105E的范围内，一共call了四个函数

```
.text:00401025                 call    ds:GetModuleFileNameA
.text:0040102B                 push    0               ; fCreate
.text:0040102D                 push    7               ; csidl
.text:0040102F                 lea     ecx, [ebp+String1]
.text:00401035                 push    ecx             ; pszPath
.text:00401036                 push    0               ; hwnd
.text:00401038                 call    ds:SHGetSpecialFolderPathA
.text:0040103E                 push    offset String2  ; "\\0.exe"
.text:00401043                 lea     edx, [ebp+String1]
.text:00401049                 push    edx             ; lpString1
.text:0040104A                 call    ds:lstrcatA
.text:00401050                 lea     eax, [ebp+String2]
.text:00401056                 push    eax             ; lpString2
.text:00401057                 lea     ecx, [ebp+String1]
.text:0040105D                 push    ecx             ; lpString1
.text:0040105E                 call    ds:lstrcmpA
```

分别查阅函数的作用

函数分析：

GetModuleFileNameA函数：

Syntax

C++ Copy

```
DWORD GetModuleFileNameA(
  HMODULE hModule,
  LPSTR   lpFilename,
  DWORD   nSize
);
```


Parameters

`hModule`

A handle to the loaded module whose path is being requested. If this parameter is **NULL**, **GetModuleFileName** retrieves the path of the executable file of the current process.

The **GetModuleFileName** function does not retrieve the path for modules that were loaded using the **LOAD_LIBRARY_AS_DATAFILE** flag. For more information, see LoadLibraryEx.

`lpFilename`

A pointer to a buffer that receives the fully qualified path of the module. If the length of the path is less than the size that the *nSize* parameter specifies, the function succeeds and the path is returned as a null-terminated string.

If the length of the path exceeds the size that the *nSize* parameter specifies, the function succeeds and the string is truncated to *nSize* characters including the terminating null character.

Windows XP: The string is truncated to *nSize* characters and is not null-terminated.

The string returned will use the same format that was specified when the module was loaded. Therefore, the path can be a long or short file name, and can use the prefix "\?". For more information, see Naming a File.

`nSize`

The size of the *lpFilename* buffer, in TCHARs.

Return Value

If the function succeeds, the return value is the length of the string that is copied to the buffer, in characters, not including the terminating null character. If the buffer is too small to hold the module name, the string is truncated to *nSize* characters including the terminating null character, the function returns *nSize*, and the function sets the last error to ERROR_INSUFFICIENT_BUFFER.

Windows XP: If the buffer is too small to hold the module name, the function returns *nSize*. The last error code remains ERROR_SUCCESS. If *nSize* is zero, the return value is zero and the last error code is ERROR_SUCCESS.

If the function fails, the return value is 0 (zero). To get extended error information, call GetLastError.

hModule参数为NULL时，函数获取当前进程的可执行文件的路径，复制到lpFilename指向的缓冲区中，如果lpFilename指向的缓冲区足够容纳该路径，则函数成功，path中存储了一个路径相关的字符串

SHGetSpecialFolderPathA函数：

Syntax

C++ Copy

```
BOOL SHGetSpecialFolderPathA(
  HWND  hwnd,
  LPSTR pszPath,
  int   csidl,
  BOOL  fCreate
);
```


Parameters

`hwnd`

Type: **HWND**

Reserved.

`pszPath`

Type: **LPTSTR**

A pointer to a null-terminated string that receives the drive and path of the specified folder. This buffer must be at least MAX_PATH characters in size.

`csidl`

Type: **int**

A CSIDL that identifies the folder of interest. If a virtual folder is specified, this function will fail.

`fCreate`

Type: **BOOL**

Indicates whether the folder should be created if it does not already exist. If this value is nonzero, the folder is created. If this value is zero, the folder is not created.

Return Value

Type: **BOOL**

TRUE if successful; otherwise, **FALSE**.

获取一个指定的系统路径

lstrcatA函数：

Syntax

C++ Copy

```
LPSTR lstrcatA(
  LPSTR  lpString1,
  LPCSTR lpString2
);
```

Parameters

`lpString1`

Type: **LPTSTR**

The first null-terminated string. This buffer must be large enough to contain both strings.

`lpString2`

Type: **LPTSTR**

The null-terminated string to be appended to the string specified in the *lpString1* parameter.

Return Value

Type: **LPTSTR**

If the function succeeds, the return value is a pointer to the buffer.

If the function fails, the return value is **NULL** and *lpString1* may not be null-terminated.

返回了一个指向长度足以存储lpString1和lpString2的缓冲区的指针

可以理解为返回了一个LPTSTR形的字符串

此字符串为lpString1和lpString2的连接字符串

lstrcmpA函数：

Syntax

C++ Copy

```cpp
int lstrcmpA(
  LPCSTR lpString1,
  LPCSTR lpString2
);
```

Parameters

`lpString1`

Type: LPCTSTR

The first null-terminated string to be compared.

`lpString2`

Type: LPCTSTR

The second null-terminated string to be compared.

Return Value

Type: int

If the string pointed to by *lpString1* is less than the string pointed to by *lpString2*, the return value is negative. If the string pointed to by *lpString1* is greater than the string pointed to by *lpString2*, the return value is positive. If the strings are equal, the return value is zero.

如果lpString1小于lpString2，返回负数

如果大于，返回正数

如果相等，返回0

分析与验证

看到这里一度觉得十分的尴尬，从0x00401000至0x0040105E的范围，无非就是做了

获取文件名，获取一个路径，将路径和

```
.text:0040103E                 push    offset String2  ; "\\0.exe"
```

"\\0.exe"拼接，然后做了一个比较

可以强行猜测一波，向这个路径复制了一个名为0的可执行文件

就没了啊，那我怎么继续逆向分析呢。。。

没办法，只好硬着头皮继续往下做

Sample_02拓展

发现做到这里再继续“正叙”写下去是根本一点也写不下去了

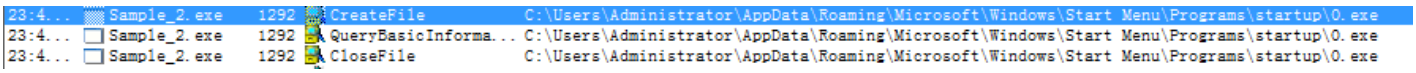
emmm可能这就是为什么只让我们分析到这里吧

因为后面的那些函数真是看不懂啊！

但是不能这么尴尬

所以我们偷偷懒，“倒叙”来看看程序是怎么走的

我们用Process Monitor来看看



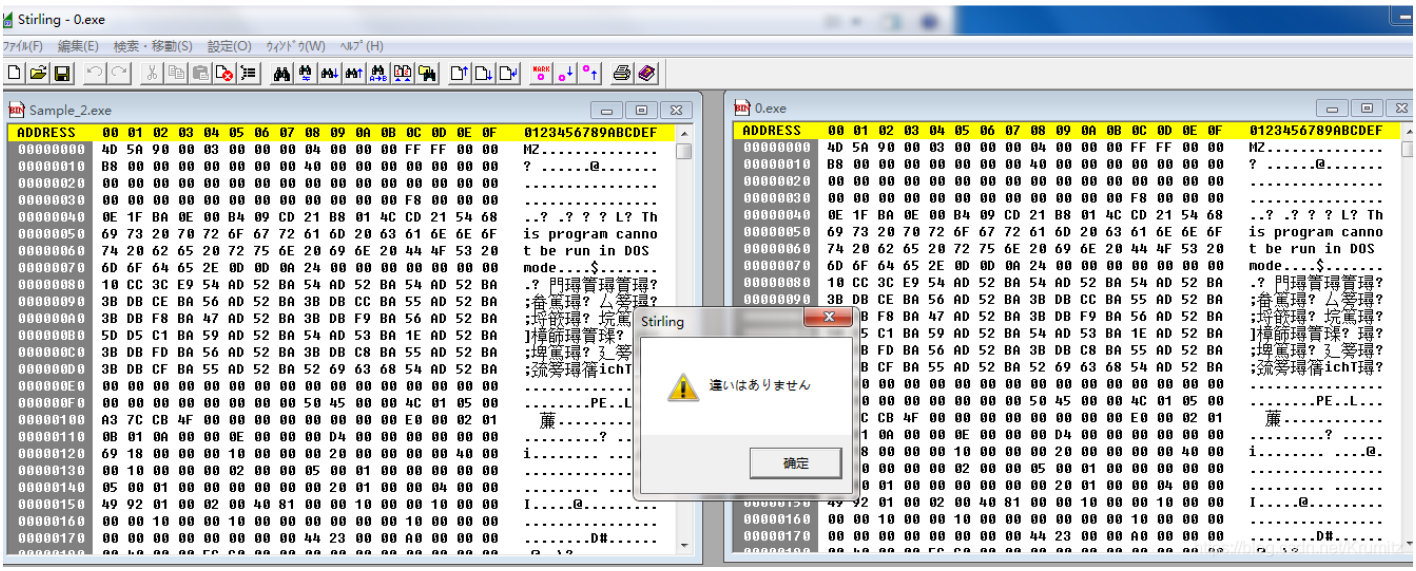
看到Sample_02向

C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\startup

这个路径下Create了一个0.exe的File

而根据我们之前的分析和函数解释，大概可以猜到，这个0.exe跟Sample_02.exe是一样的

我们用Stirling来验证一下



发现确实如此

嗯。。。虽然没搞懂IDA里面的东西，但是通过借助其他工具的帮助，我们还是大致知道了，这个Sample_02会执行一个怎么样的操作

5.Sample_03逆向分析

实验要求：Sample_3分析范围：0x00401000至0x00401062

```
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 mov     eax, 2004h
.text:00401008                 call    __alloca_probe
.text:0040100D                 mov     eax, ___security_cookie
.text:00401012                 xor     eax, ebp
.text:00401014                 mov     [ebp+var_4], eax
.text:00401017                 push    1000h           ; nSize
.text:0040101C                 lea     eax, [ebp+ExistingFileName]
.text:00401022                 push    eax             ; lpFilename
.text:00401023                 push    0               ; hModule
.text:00401025                 call    ds:GetModuleFileNameW
.text:0040102B                 lea     ecx, [ebp+NewFileName]
.text:00401031                 push    ecx             ; pszPath
.text:00401032                 push    0               ; dwFlags
.text:00401034                 push    0               ; hToken
.text:00401036                 push    7               ; csidl
.text:00401038                 push    0               ; hwnd
.text:0040103A                 call    ds:SHGetFolderPathW
.text:00401040                 push    offset String2  ; "\\wsample01b.exe"
.text:00401045                 lea     edx, [ebp+NewFileName]
.text:0040104B                 push    edx             ; lpString1
.text:0040104C                 call    ds:lstrcatW
.text:00401052                 push    0               ; bFailIfExists
.text:00401054                 lea     eax, [ebp+NewFileName]
.text:0040105A                 push    eax             ; lpNewFileName
.text:0040105B                 lea     ecx, [ebp+ExistingFileName]
.text:00401061                 push    ecx             ; lpExistingFileName
.text:00401062                 call    ds:CopyFileW
.text:00401068                 mov     ecx, [ebp+var_4]
.text:0040106B                 xor     ecx, ebp
.text:0040106D                 xor     eax, eax
.text:0040106F                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401074                 mov     esp, ebp
.text:00401076                 pop     ebp
.text:00401077                 retn
```


咦让我们分析的部分没有找到函数入口啊

没事，我们往下拉一点可以看到

```
.text:00401062                 call    ds:CopyFileW
.text:00401068                 mov     ecx, [ebp+var_4]
.text:0040106B                 xor     ecx, ebp
.text:0040106D                 xor     eax, eax
.text:0040106F                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401074                 mov     esp, ebp
.text:00401076                 pop     ebp
.text:00401077                 retn
.text:00401077 sub_401000      endp
.text:00401077
.text:00401077 ; ---------------------------------------------------------------------------
.text:00401078                 align 10h
.text:00401080
.text:00401080 ; =============== S U B R O U T I N E =======================================
.text:00401080
.text:00401080
.text:00401080 ; int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
.text:00401080 _wWinMain@16    proc near               ; CODE XREF: ___tmainCRTStartup+153↓p
.text:00401080
.text:00401080 hInstance       = dword ptr  4
.text:00401080 hPrevInstance   = dword ptr  8
.text:00401080 lpCmdLine       = dword ptr  0Ch
.text:00401080 nShowCmd        = dword ptr  10h
.text:00401080
.text:00401080                 call    sub_401000
.text:00401085                 push    0               ; uType
.text:00401087                 push    offset Caption  ; "MESSAGE"
.text:0040108C                 push    offset Text     ; "Copied!"
.text:00401091                 call    ds:GetActiveWindow
.text:00401097                 push    eax             ; hWnd
.text:00401098                 call    ds:MessageBoxW
.text:0040109E                 xor     eax, eax
.text:004010A0                 retn    10h
.text:004010A0 _wWinMain@16    endp
```


程序的主函数上来讲就先call了我们要分析的这一部分

所以我们分析出来那一部分的子函数会做什么，也就可以知道这个程序会做什么了

函数分析：

GetModuleNameW函数：

Syntax

C++ Copy

```
DWORD GetModuleFileNameW(
  HMODULE hModule,
  LPWSTR  lpFilename,
  DWORD   nSize
);
```

Parameters

`hModule`

A handle to the loaded module whose path is being requested. If this parameter is **NULL**, **GetModuleFileName** retrieves the path of the executable file of the current process.

The **GetModuleFileName** function does not retrieve the path for modules that were loaded using the **LOAD_LIBRARY_AS_DATAFILE** flag. For more information, see LoadLibraryEx.

`lpFilename`

A pointer to a buffer that receives the fully qualified path of the module. If the length of the path is less than the size that the *nSize* parameter specifies, the function succeeds and the path is returned as a null-terminated string.

If the length of the path exceeds the size that the *nSize* parameter specifies, the function succeeds and the string is truncated to *nSize* characters including the terminating null character.

Windows XP: The string is truncated to *nSize* characters and is not null-terminated.

The string returned will use the same format that was specified when the module was loaded. Therefore, the path can be a long or short file name, and can use the prefix "\?". For more information, see Naming a File.

`nSize`

The size of the *lpFilename* buffer, in **TCHARs**.

Return Value

If the function succeeds, the return value is the length of the string that is copied to the buffer, in characters, not including the terminating null character. If the buffer is too small to hold the module name, the string is truncated to *nSize* characters including the terminating null character, the function returns *nSize*, and the function sets the last error to **ERROR_INSUFFICIENT_BUFFER**.

Windows XP: If the buffer is too small to hold the module name, the function returns *nSize*. The last error code remains **ERROR_SUCCESS**. If *nSize* is zero, the return value is zero and the last error code is **ERROR_SUCCESS**.

If the function fails, the return value is 0 (zero). To get extended error information, call GetLastError.

与Sample_02中的GetModuleFileNameA相似

SHGetFolderPathW函数：

Syntax

C++ Copy

```
SHFOLDERAPI SHGetFolderPathW(
  HWND   hwnd,
  int    csidl,
  HANDLE hToken,
  DWORD  dwFlags,
  LPWSTR pszPath
);
```


Parameters

`hwnd`

Type: **HWND**

Reserved.

`csidl`

Type: **int**

A CSIDL value that identifies the folder whose path is to be retrieved. Only real folders are valid. If a virtual folder is specified, this function fails. You can force creation of a folder by combining the folder's **CSIDL** with **CSIDL_FLAG_CREATE**.

`hToken`

Type: **HANDLE**

An access token that can be used to represent a particular user.

Microsoft Windows 2000 and earlier: Always set this parameter to **NULL**.

Windows XP and later: This parameter is usually set to **NULL**, but you might need to assign a non-**NULL** value to *hToken* for those folders that can have multiple users but are treated as belonging to a single user. The most commonly used folder of this type is **Documents**.

The calling process is responsible for correct impersonation when *hToken* is non-**NULL**. The calling process must have appropriate security privileges for the particular user, including TOKEN_QUERY and TOKEN_IMPERSONATE, and the user's registry hive must be currently mounted. See Access Control for further discussion of access control issues.

Assigning the *hToken* parameter a value of -1 indicates the Default User. This enables clients of **SHGetFolderPath** to find folder locations (such as the Desktop folder) for the Default User. The Default User user profile is duplicated when any new user account is created, and includes special folders such as My Documents and Desktop. Any items added to the Default User folder also appear in any new user account.

`dwFlags`

Type: **DWORD**

Flags that specify the path to be returned. This value is used in cases where the folder associated with a KNOWNFOLDERID (or CSIDL) can be moved, renamed, redirected, or roamed across languages by a user or administrator.

The known folder system that underlies **SHGetFolderPath** allows users or administrators to redirect a known folder to a location that suits their needs. This is achieved by calling IKnownFolderManager::Redirect, which sets the "current" value of the folder associated with the SHGFP_TYPE_CURRENT flag.

The default value of the folder, which is the location of the folder if a user or administrator had not redirected it elsewhere, is retrieved by specifying the SHGFP_TYPE_DEFAULT flag. This value can be used to implement a "restore defaults" feature for a known folder.

For example, the default value (SHGFP_TYPE_DEFAULT) for FOLDERID_Music (CSIDL_MYMUSIC) is "C:\Users**user name\Music". If the folder was redirected, the current value (SHGFP_TYPE_CURRENT) might be "D:\Music". If the folder has not been redirected, then SHGFP_TYPE_DEFAULT and SHGFP_TYPE_CURRENT retrieve the same path.**

SHGFP_TYPE_CURRENT

Retrieve the folder's current path.

SHGFP_TYPE_DEFAULT

Retrieve the folder's default path.

`pszPath`

Type: **LPTSTR**

A pointer to a **null**-terminated string of length MAX_PATH which will receive the path. If an error occurs or S_FALSE is returned, this string will be empty. The returned path does not include a trailing backslash. For example, "C:\Users" is returned rather than "C:\Users".

与Sample_02中的SHGetSpecialFolderPathA功能类似

CopyFileW函数：

Syntax

C++ Copy

```
BOOL CopyFileW(
  LPCWSTR lpExistingFileName,
  LPCWSTR lpNewFileName,
  BOOL    bFailIfExists
);
```

Parameters

`lpExistingFileName`

The name of an existing file.

In the ANSI version of this function, the name is limited to **MAX_PATH** characters. To extend this limit to 32,767 wide characters, call the Unicode version of the function and prepend "\?" to the path. For more information, see Naming a File.

> **Tip** Starting with Windows 10, version 1607, for the unicode version of this function (**CopyFileW**), you can opt-in to remove the **MAX_PATH** limitation without prepending "\\?\". See the "Maximum Path Length Limitation" section of **Naming Files, Paths, and Namespaces** for details.

If *lpExistingFileName* does not exist, **CopyFile** fails, and GetLastError returns **ERROR_FILE_NOT_FOUND**.

`lpNewFileName`

The name of the new file.

In the ANSI version of this function, the name is limited to **MAX_PATH** characters. To extend this limit to 32,767 wide characters, call the Unicode version of the function and prepend "\?" to the path. For more information, see Naming a File.

> **Tip** Starting with Windows 10, version 1607, for the unicode version of this function (**CopyFileW**), you can opt-in to remove the **MAX_PATH** limitation without prepending "\\?\". See the "Maximum Path Length Limitation" section of **Naming Files, Paths, and Namespaces** for details.

`bFailIfExists`

If this parameter is **TRUE** and the new file specified by *lpNewFileName* already exists, the function fails. If this parameter is **FALSE** and the new file already exists, the function overwrites the existing file and succeeds.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

复制一个文件，参数分别为现有文件名，目标文件名，和一个bool值

如果bool值为TRUE，且目标文件名已存在的话，函数失败

如果为FALSE，目标文件名已存在的话，函数将覆盖已有文件，并成功返回

分析与验证

这就比Sample_02容易猜测这个程序会执行一个什么操作了，毕竟有一个**CopyFileW**的存在嘛

所以在这里我们就已经大致可以猜测，这个程序

获取了一个系统的路径

```
text:00401040                 push    offset String2  ; "\\wsample01b.exe"
```

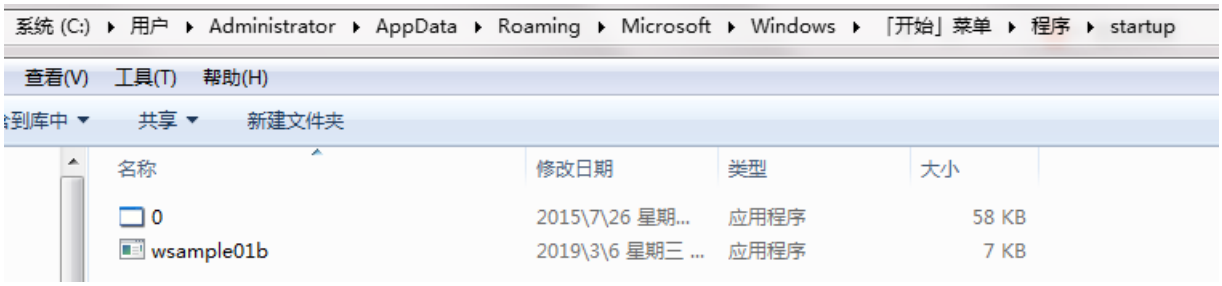
并朝这个路径下复制了一个名为wsample01b的可执行文件，而这个可执行文件跟Sample_03应该是一样的

我们打开Process Monitor来查看一下

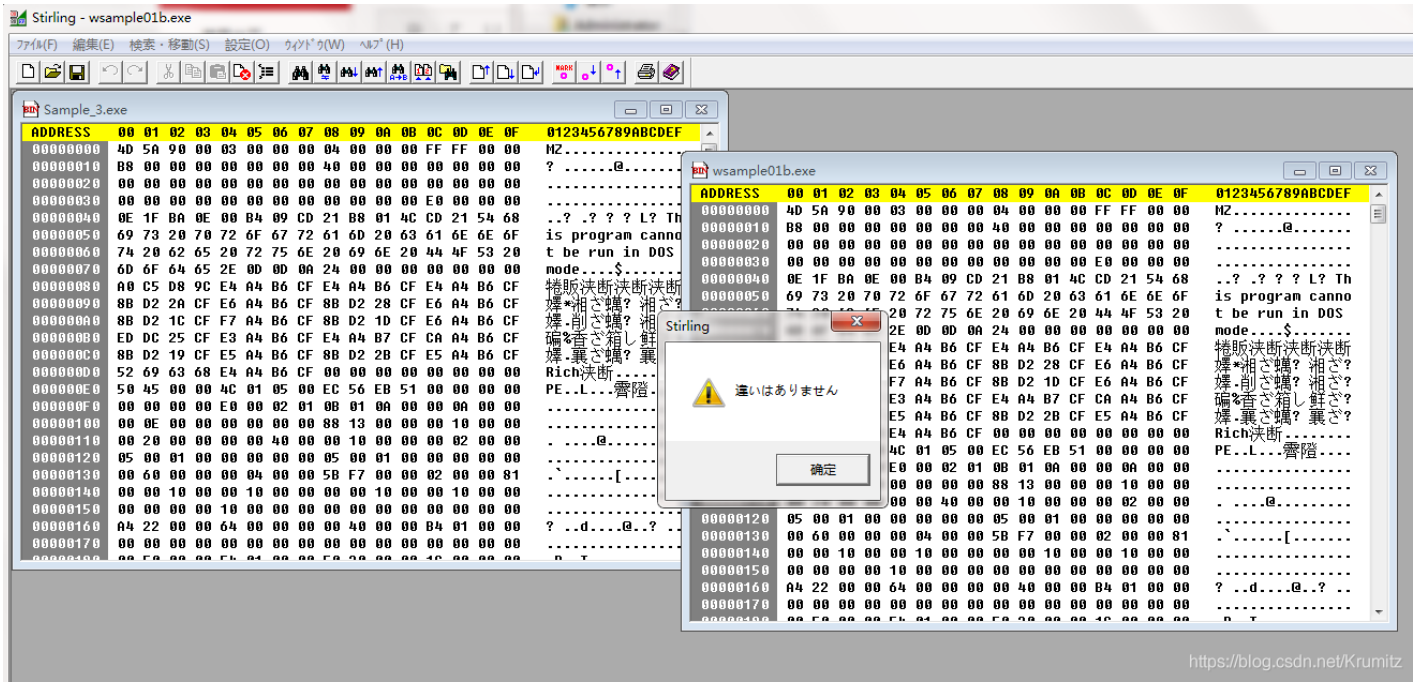
17:3...	Sample_3.exe	7336	CreateFile	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\startup	SUCCESS
17:3...	Sample_3.exe	7336	QueryBasicInforma...	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\startup	SUCCESS
17:3...	Sample_3.exe	7336	CloseFile	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\startup	SUCCESS
17:3...	Sample_3.exe	7336	RegQueryKey	HKLM	SUCCESS
17:3...	Sample_3.exe	7336	RegQueryKey	HKLM	SUCCESS
17:3...	Sample_3.exe	7336	RegOpenKey	HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Explorer\KnownFolderSett...	NAME NOT F

发现Sample_3在C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\startup这个路径下CreateFile

打开这个文件夹，发现



确实有wsample01b的可执行文件



用Stirling二进制编辑器，对比Sample3和wsample01b，发现他们是一样的

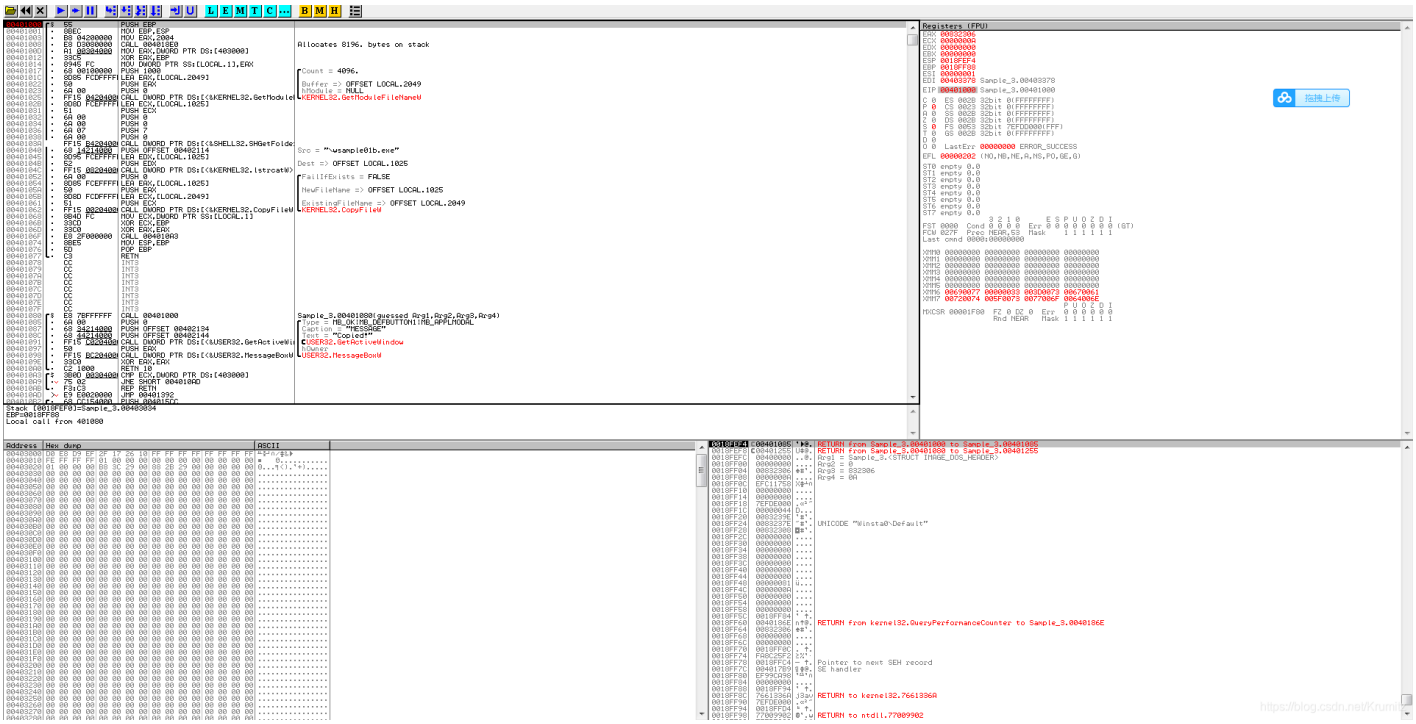
至此即可确定我们之前的分析是大致正确的

Sample_03拓展

在IDA中找了很久，都没有找到Sample_03.exe将获取到的系统路径存放在了哪里

提出疑问，如果不通过进程监视器来看，我们应该怎么知道程序向哪条路径CopyFile了呢

我们选择使用OllyDbg调试，将Sample_03拖入到OllyDbg中，如下显示



在00401000处设置断点，然后按下F8逐步执行指令

运行到0040103A处，显示如下

```
0040103A    FF15 B4204000  CALL DWORD PTR DS:[<&SHELL32.SHGetFolde
00401040  · 68 14214000    PUSH OFFSET 00402114                     Src = "\wsample01b.exe"
00401045  · 8D95 FCEFFFFF  LEA EDX,[LOCAL.1025]
0040104B  · 52             PUSH EDX                                 Dest => OFFSET LOCAL.1025
0040104C  · FF15 0820400   CALL DWORD PTR DS:[<&KERNEL32.lstrcatW>
00401052  · 6A 00          PUSH 0                                   FailIfExists = FALSE
00401054  · 8D85 FCEFFFFF  LEA EAX,[LOCAL.1025]                     NewFileName => OFFSET LOCAL.1025
0040105A  · 50             PUSH EAX
0040105B  · 8D8D FCDFFFFF  LEA ECX,[LOCAL.2049]                     ExistingFileName => OFFSET LOCAL.2049
00401061  · 51             PUSH ECX
00401062  · FF15 0020400   CALL DWORD PTR DS:[<&KERNEL32.CopyFileW  KERNEL32.CopyFileW
00401068  · 8B4D FC        MOV ECX,DWORD PTR SS:[LOCAL.1]
0040106B  · 33CD           XOR ECX,EBP
0040106D  · 33C0           XOR EAX,EAX
0040106F  · E8 2F000000    CALL 004010A3
00401074  · 8BE5           MOV ESP,EBP
00401076  · 5D             POP EBP
00401077  · C3             RETN
00401078    CC             INT3
00401079    CC             INT3
0040107A    CC             INT3
0040107B    CC             INT3
0040107C    CC             INT3
0040107D    CC             INT3
0040107E    CC             INT3
0040107F    CC             INT3
00401080  $ E8 7BFFFFFF    CALL 00401000                            Sample_3.00401080(guessed Arg1,Arg2,Arg3,Arg4)
00401085  · 6A 00          PUSH 0                                   Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
00401087  · 68 34214000    PUSH OFFSET 00402134                     Caption = "MESSAGE"
0040108C  · 68 44214000    PUSH OFFSET 00402144                     Text = "Copied!"
00401091  · FF15 C020400   CALL DWORD PTR DS:[<&USER32.GetActiveWi  USER32.GetActiveWindow
00401097  · 50             PUSH EAX                                 hOwner
00401098  · FF15 BC20400   CALL DWORD PTR DS:[<&USER32.MessageBoxW  USER32.MessageBoxW
0040109E  · 33C0           XOR EAX,EAX
004010A0  · C2 1000        RETN 10
004010A3  $ 3B0D 0030400   CMP ECX,DWORD PTR DS:[403000]
004010A9  · 75 02          JNE SHORT 004010AD
004010AB  · F3:C3          REP RETN
004010AD    E9 E0020000    JMP 00401392
004010B2  · 68 CC154000    PUSH 004015CC
[004020B4]=75125590 (SHELL32.SHGetFolderPathW)
```


此处即将执行SHGetFolderPathW函数

运行到下一行后，显示如下

```
00401040  · 68 14214000    PUSH OFFSET 00402114                     Src = "\wsample01b.exe"
00401045  · 8D95 FCEFFFFF  LEA EDX,[LOCAL.1025]
0040104B  · 52             PUSH EDX                                 Dest => OFFSET LOCAL.1025
0040104C  · FF15 0820400   CALL DWORD PTR DS:[<&KERNEL32.lstrcatW>
00401052  · 6A 00          PUSH 0                                   FailIfExists = FALSE
00401054  · 8D85 FCEFFFFF  LEA EAX,[LOCAL.1025]                     NewFileName => OFFSET LOCAL.1025
0040105A  · 50             PUSH EAX
0040105B  · 8D8D FCDFFFFF  LEA ECX,[LOCAL.2049]                     ExistingFileName => OFFSET LOCAL.2049
00401061  · 51             PUSH ECX
00401062  · FF15 0020400   CALL DWORD PTR DS:[<&KERNEL32.CopyFileW  KERNEL32.CopyFileW
00401068  · 8B4D FC        MOV ECX,DWORD PTR SS:[LOCAL.1]
0040106B  · 33CD           XOR ECX,EBP
0040106D  · 33C0           XOR EAX,EAX
0040106F  · E8 2F000000    CALL 004010A3
00401074  · 8BE5           MOV ESP,EBP
00401076  · 5D             POP EBP
00401077  · C3             RETN
00401078    CC             INT3
00401079    CC             INT3
0040107A    CC             INT3
0040107B    CC             INT3
0040107C    CC             INT3
0040107D    CC             INT3
0040107E    CC             INT3
0040107F    CC             INT3
00401080  $ E8 7BFFFFFF    CALL 00401000                            Sample_3.00401080(guessed Arg1,Arg2,Arg3,Arg4)
00401085  · 6A 00          PUSH 0                                   Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
00401087  · 68 34214000    PUSH OFFSET 00402134                     Caption = "MESSAGE"
0040108C  · 68 44214000    PUSH OFFSET 00402144                     Text = "Copied!"
00401091  · FF15 C020400   CALL DWORD PTR DS:[<&USER32.GetActiveWi  USER32.GetActiveWindow
00401097  · 50             PUSH EAX                                 hOwner
00401098  · FF15 BC20400   CALL DWORD PTR DS:[<&USER32.MessageBoxW  USER32.MessageBoxW
0040109E  · 33C0           XOR EAX,EAX
004010A0  · C2 1000        RETN 10
004010A3  $ 3B0D 0030400   CMP ECX,DWORD PTR DS:[403000]
004010A9  · 75 02          JNE SHORT 004010AD
004010AB  · F3:C3          REP RETN
004010AD    E9 E0020000    JMP 00401392
004010B2  · 68 CC154000    PUSH 004015CC
Stack [0018DEE8]=0018EEEC, UNICODE "C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"
Imm=Sample_3.00402114, UNICODE "\wsample01b.exe"
```


然后我们再确认一下Stack[0018DEE8]的内容

```
0018DEE8  0018EEEC  ∞€↑.   UNICODE "C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"
```

可以看到，虽然我们不知道SHGetFolderPathW函数设置了什么参数，但是我们知道了执行完这个函数之后，系统将获取到的路径push入栈，而我们的目的也达到了，寻找到了Sample_03.exe向哪一个路径复制了程序

最后我们用32位的IDA PRO查看我们分析范围0x00401000至0x00401062的这段子函数的代码，以确认之前逆向分析的结果（32位的可以按F5查看程序伪代码，64位的好像并没有这个功能）

```
int sub_401000()
{
  WCHAR Filename; // [sp+0h] [bp-2004h]@1
  WCHAR pszPath; // [sp+1000h] [bp-1004h]@1

  GetModuleFileNameW(0, &Filename, 0x1000u);
  SHGetFolderPathW(0, 7, 0, 0, &pszPath);
  lstrcatW(&pszPath, L"\\wsample01b.exe");
  CopyFileW(&Filename, &pszPath, 0);
  return 0;
}
```

可以发现，是大致正确的

6.实验01总结

三个实验做下来，其实并没有“写下这篇实验报告类型的博客”看起来这么简单与轻描淡写

还是花费了许多时间的

三个实验中Sample_01和Sample_03还是比较让人满意的

Sample_02就真的，太打击人了

希望在日后的学习中可以继续好好努力，不耻下问

至少总有一天得搞懂Sample_02的函数吧（小声bb）

虽然不是第一次在CSDN上发文了，（虽然以前也只发过一次）

但是这次有一点赶作业的嫌疑，并没有怎么认真排版，如果让大家阅读不便，敬请谅解！

附上三个Sample的IDA DATABASE文件和做作业时候的参考网址吧，原执行文件就不附上了

链接：https://pan.baidu.com/s/1v227QBlmELFtbgMqmi8u5A
提取码：syll

http://www.ituring.com.cn/book/tupubarticle/9632