

恶意代码分析与防治技术实验报告

Lab13

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
- Wireshark
- Resource Hacker
- WinHex
- KANAL

三、实验内容

Lab 13-1

Analyze the malware found in the file *Lab13-01.exe*.

Questions

1. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?
2. Use IDA Pro to look for potential encoding by searching for the string `xor`. What type of encoding do you find?
3. What is the key used for encoding and what content does it encode?
4. Use the static tools FindCrypt2, Krypto ANALyzer (KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?
5. What type of encoding is used for a portion of the network traffic sent by the malware?
6. Where is the Base64 function in the disassembly?
7. What is the maximum length of the Base64-encoded data that is sent? What is encoded?
8. In this malware, would you ever see the padding characters (= or ==) in the Base64-encoded data?
9. What does this malware do?

- 1、比较恶意代码中的字符串与动态分析得到的信息，基于此，哪些元素可能被加密？

首先动态分析，运行程序，利用 wireshark 抓包

```
[Expert Info (Chat/Sequence): GET /bmt1LWNjNmZjZmM4/ HTTP/1.1\r\n]
Request Method: GET
Request URI: /bmt1LWNjNmZjZmM4/
Request Version: HTTP/1.1
User-Agent: Mozilla/4.0\r\n
Host: www.practicalmalwareanalysis.com\r\n
\r\n
[Full request URI: http://www.practicalmalwareanalysis.com/bmt1LWNjNmZjZmM4/]
[HTTP request 1/2]
[Response in frame: 107]
0020  98 93 04 1a 00 50 62 65 db c9 5f 10 27 e3 50 18 .....Pbe ...'.P.
0030  fa f0 68 22 00 00 47 45 54 20 2f 62 6d 74 31 4c ...h"..GE T /bmt1L
0040  57 4e 6a 4e 6d 5a 6a 5a 6d 4d 34 2f 20 48 54 54 WNjNmZjZ mM4/ HTT
0050  50 2f 31 2e 31 0d 0a 55 73 65 72 2d 41 67 65 6e P/1.1..U ser-Agen
0060  74 3a 20 4d 6f 7a 69 6c 6c 61 2f 34 2e 30 0d 0a t: Mozill a/4.0..
0070  48 6f 73 74 3a 20 77 77 77 2e 70 72 61 63 74 69 Host: ww w.practi
0080  63 61 6c 6d 61 6c 77 61 72 65 61 6e 61 6c 79 73 calmalwa reanalys
0090  69 73 2e 63 6f 6d 0d 0a 0d 0a is.com... ..
HTTP User-Agent header (http.user-agent) Packets: 48 · Displayed: 48 (100.0%)
```

发现了一个 GET 请求，指定浏览器为 Mozilla/4.0，还看到一个网址以及一堆乱码

进入 IDA 查看字符串

```
.rdata:0000002C C R6009\r\n- not enough space for environment\r\n
.rdata:0000002A C R6008\r\n- not enough space for arguments\r\n
.rdata:00000025 C R6002\r\n- floating point not loaded\r\n
.rdata:00000025 C Microsoft Visual C++ Runtime Library
.rdata:0000001A C Runtime Error!\n\nProgram:
.rdata:00000017 C <program name unknown>
.rdata:00000013 C GetLastErrorPopup
.rdata:00000010 C GetActiveWindow
.rdata:0000000C C MessageBoxA
.rdata:00000008 C user32.dll
.rdata:0000000D C KERNEL32.dll
.rdata:00000008 C WS2_32.dll
.rdata:0000000C C WININET.dll
.data:00400000C C Mozilla/4.0
.data:00400000E C http://%s/%s/
.data:0040000014 C Could not load exe.
.data:004000001D C Could not locate dialog box.
.data:004000001B C Could not load dialog box.
.data:004000001B C Could not lock dialog box.
.data:0040000006 C %02x
.data:0040000006 C y !
```

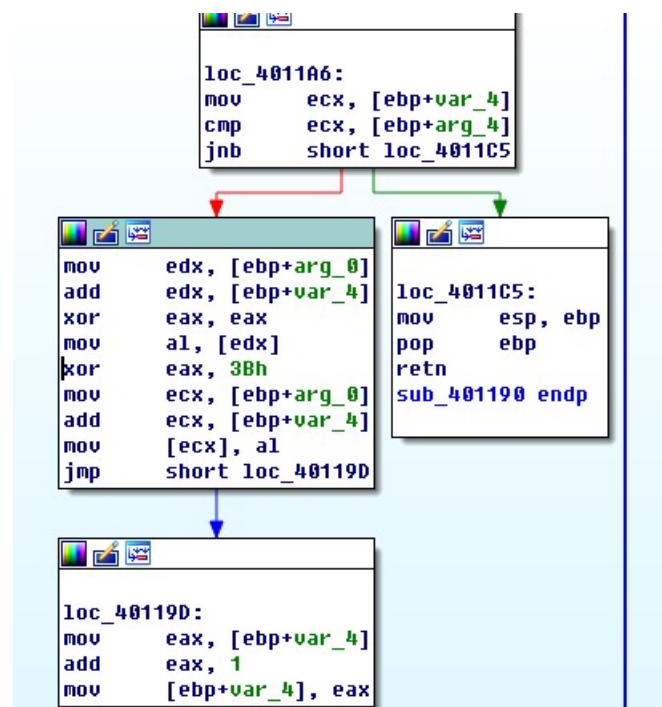
发现了浏览器版本字符串 Mozilla/4.0，没发现那个网址和乱码，所以这两个可能被加密了。

2、IDA Pro 搜索 xor，以此来查找潜在的加密，发现哪些加密类型？

IDA Pro 搜索 xor

Address	Function	Instruction
text:00401007	sub_401000	xor ecx, ecx
.text:0040101C	sub_401000	xor edx, edx
.text:00401029	sub_401000	xor ecx, ecx
.text:0040104E	sub_401000	xor eax, eax
.text:0040105C	sub_401000	xor edx, edx
.text:0040108D	sub_401000	xor ecx, ecx
.text:004011B4	sub_401190	xor eax, eax
.text:004011B8	sub_401190	xor eax, 3Bh
.text:004011D6	sub_4011C9	xor eax, eax
.text:004012A2	sub_4011C9	xor al, al
.text:004012E6	sub_4011C9	xor al, al
.text:004012FA	sub_4011C9	xor al, al
.text:00401332	sub_401300	xor eax, eax
.text:00401350	sub_401300	xor eax, eax
.text:0040138E	sub_401300	xor eax, eax
.text:00401463	_main	xor eax, eax
.text:004021E5		xor ecx, ecx
.text:00402202		xor edx, edx

经过分析，只有 004011B8 地址处的 xor 是用于加密，进入：



是一个循环，不断对 var_4 加一，eax 与 3Bh 异或，即对缓冲区的数据进行异或加密。

3、恶意代码使用什么密钥加密，加密的内容是？

有上一问可知异或加密使用的是 3Bh 加密，密钥是 3Bh

看看此处加密的是啥，看 401190 的交叉引用：

```

loc_401392:
mov     eax, [ebp+hResData]
push    eax                ; hResData
call    ds:LockResource
mov     [ebp+var_10], eax
mov     ecx, [ebp+dwBytes]
push    ecx
mov     edx, [ebp+var_10]
push    edx
call    sub_401190
add     esp, 8
mov     eax, [ebp+var_10]
jmp     short loc_4013E9

```

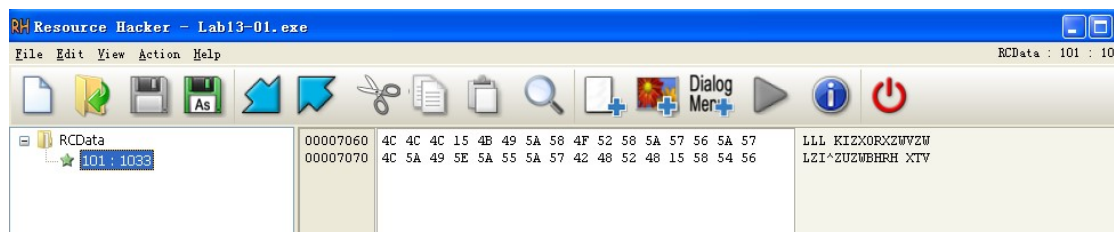
前面进行了一堆关于资源节的操作，重点关注 FindResourceA，看看加密的资源是什么东西

```

loc_401339:                ; lpType
push    0Ah                ; lpType
push    65h                ; lpName
mov     eax, [ebp+hModule]
push    eax                ; hModule
call    ds:FindResourceA
mov     [ebp+hResInfo], eax
cmp     [ebp+hResInfo], 0
jnz     short loc_401357

```

参数 lpType 的值是 0Ah，它表示资源数据是应用程序预定义的还是原始数据，参数 lpName，在这里它表示一个索引号，用 Resource Hacker 来看一下



在偏移 7060 处有 32 字节，用 WinHex 打开到此位置，用 3Bh 异或，得到解密数据

00007030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00	
00007040	09 04 00 00 48 00 00 00 60 80 00 00 20 00 00 00	H
00007050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00007060	77 77 77 2E 70 72 61 63 74 69 63 61 6C 6D 61 6C	www.practicalmal
00007070	77 61 72 65 61 6E 61 6C 79 73 69 73 2E 63 6F 6D	wareanalysis.com
00007080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00007090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

发现是那个网址。

4、使用静态工具 KANAL 等以及 IDA 熵插件识别一些其他类型的加密机制，你发现了什么？

使用 KANAL 发现在 4050E8 位置处有一个 Base64 编码表，IDA 跳转：

```

.rdata:004050E4          align 8
.rdata:004050E8 byte_4050E8 db 41h ; DATA XREF: sub_401000+1
.rdata:004050E8                                     ; sub_401000+3C↑r ...
.rdata:004050E9 db 42h ; B
.rdata:004050EA db 43h ; C
.rdata:004050EB db 44h ; D
.rdata:004050EC db 45h ; E
.rdata:004050ED db 46h ; F
.rdata:004050EE db 47h ; G
.rdata:004050EF db 48h ; H
.rdata:004050F0 db 49h ; I
.rdata:004050F1 db 4Ah ; J
.rdata:004050F2 db 4Bh ; K
.rdata:004050F3 db 4Ch ; L
.rdata:004050F4 db 4Dh ; M
.rdata:004050F5 db 4Eh ; N
.rdata:004050F6 db 4Fh ; O
.rdata:004050F7 db 50h ; P
.rdata:004050F8 db 51h ; Q
.rdata:004050F9 db 52h ; R
.rdata:004050FA db 53h ; S
.rdata:004050FB db 54h ; T
.rdata:004050FC db 55h ; U
.rdata:004050FD db 56h ; V
.rdata:004050FE db 57h ; W

```

看到这是一个标准的 Base64 编码表，

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-

可能存在 Base64 加密。

5、什么类型的加密被用来发送部分网络流量？

由上述分析，网址是经过异或加密的，所以异或加密被用于发送网络流量

下面来看看 Base64 有没有用到，查看 4050E8 的交叉引用：

都在函数 401000 里，进入：

```

or     edx, ecx
mov     eax, [ebp+arg_4]
mov     cl, ds:byte_4050E8[edx]
mov     [eax+1], cl
cmp     [ebp+arg_8], 1
jle     short loc_401077

```

```

mov     edx, [ebp+arg_0]
xor     eax, eax
mov     al, [edx+1]
and     eax, 0Fh
shl     eax, 2
mov     ecx, [ebp+arg_0]
xor     edx, ecx
mov     dl, [ecx+2]

```

```

loc_401077:
mov     [ebp+var_1], 3Dh

```

发现有等号，即 Base64 的填充字符，所以这里有 Base64 加密，再看看 401000 的交叉引用：

这里用 Base64 编码来创建字符串，可能是加密的内容，用于发送网络流量。

xrefs to sub_401000

Displacement	Type	Address	Text
D... p		sub_4010B1+9C	call sub_401000

OK Cancel Search Help

Line 1 of 1

```

.text:0040114C      push    eax
.text:0040114D      call    sub_401000
.text:00401152      add     esp, 0Ch

```

分析 4010B1 处的代码

```

result = strlen(a1);
v9 = result;
v10 = 0;
v4 = 0;
while ( (signed int)v10 < (signed int)v9 )
{
    v3 = 0;
    for ( i = 0; i < 3; ++i )
    {
        v7[i] = a1[v10];
        result = v10;
        if ( (signed int)v10 >= (signed int)v9 )
        {
            result = i;
            v7[i] = 0;
        }
        else
        {
            ++v3;
            ++v10;
        }
    }
    if ( v3 )
    {
        result = sub_401000(v7, v8, v3);
        for ( j = 0; j < 4; ++j )
        {
            result = j;
            *(_BYTE *)(v4++ + a2) = v8[j];
        }
    }
}
return result;

```

这里有一个 while 循环，v9 是源字符串长度，v10 循环自增，每轮循环中获取源字符串，三个为一组保存，然后 Base64 加密，查看 4010B1 的交叉引用：

.text:004011F2	add	esp, 0	
.text:004011F5	push	100h	; namelen
.text:004011FA	lea	edx, [ebp+name]	
.text:00401200	push	edx	; name
.text:00401201	call	gethostname	
.text:00401206	mov	[ebp+var_4], eax	
.text:00401209	push	0Ch	; size_t
.text:0040120B	lea	eax, [ebp+name]	
.text:00401211	push	eax	; char *
.text:00401212	lea	ecx, [ebp+var_18]	
.text:00401215	push	ecx	; char *
.text:00401216	call	_strncpy	
.text:0040121B	add	esp, 0Ch	
.text:0040121E	mov	[ebp+var_C], 0	
.text:00401222	lea	edx, [ebp+var_30]	
.text:00401225	push	edx	; int
.text:00401226	lea	eax, [ebp+var_18]	
.text:00401229	push	eax	; char *
.text:0040122A	call	sub_4010B1	
.text:0040122F	add	esp, 8	
.text:00401232	mov	byte ptr [ebp+var_23+3], 0	
.text:00401236	lea	ecx, [ebp+var_30]	
.text:00401239	push	ecx	
.text:0040123A	mov	edx, [ebp+arg_0]	
.text:0040123D	push	edx	
.text:0040123E	push	offset aHttpSS ; "http://%s/%s/"	
.text:00401243	lea	eax, [ebp+szUrl]	

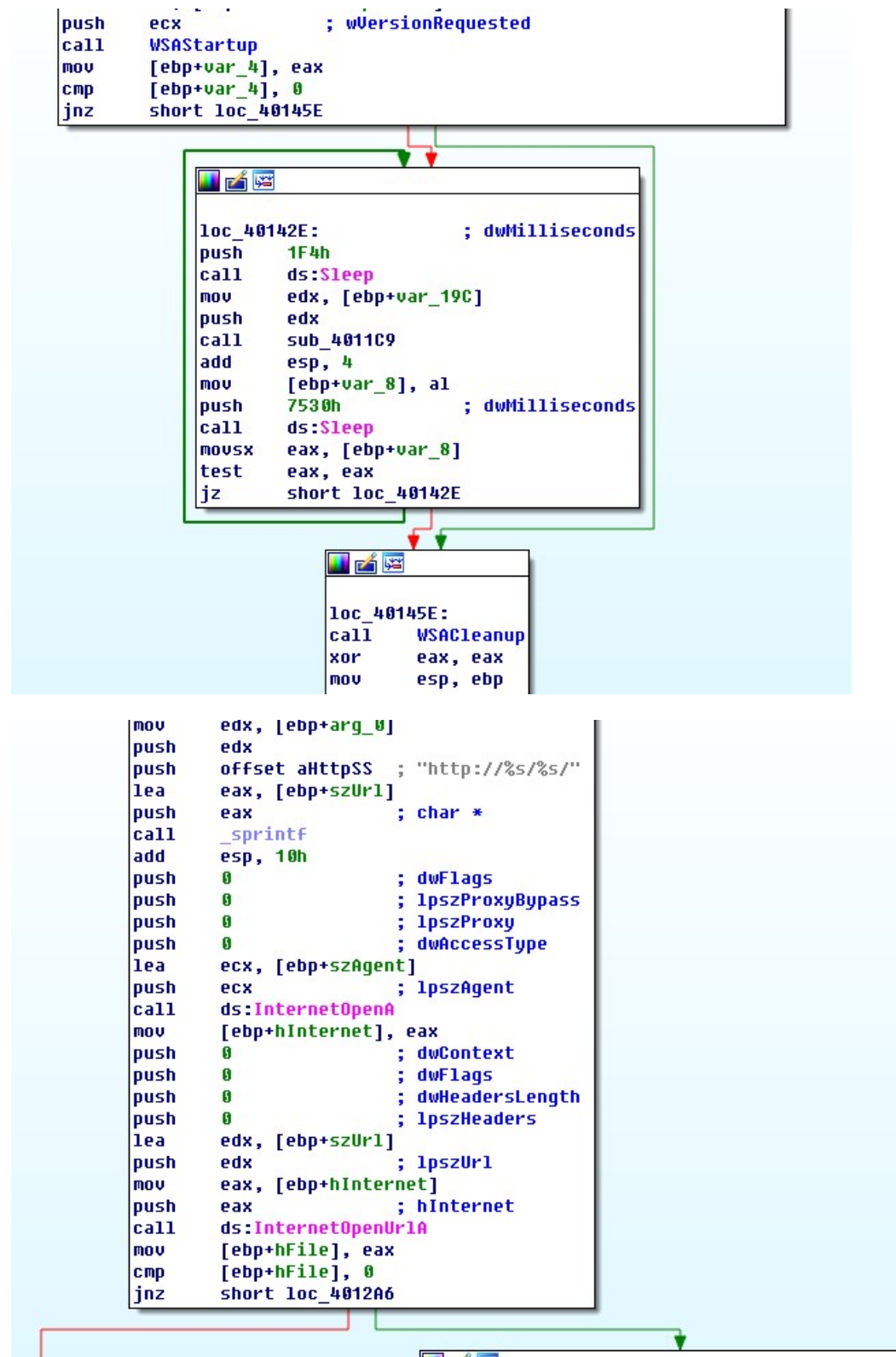
4010B1 的参数为 var_18，这个是 strncpy 的输出，_strncpy 的输入就是函数 gethostname 的输入，所以就是复制了主机名称的前 12 个字符。

所以输入的最长字符长度为 12。

8、恶意代码中，你是否在 Base64 加密数据中看到了填充字符（=或者=）？

当主机名长度不足 12 或者不能被 3 整除时，会出现填充字符。

9、恶意代码做了什么？



分析主程序，这个恶意程序就是将主机名称加密，然后向特定网址发送 GET 请求，如果收到以 o 开头的 web 响应，就退出程序。

Lab 13-2

Analyze the malware found in the file *Lab13-02.exe*.

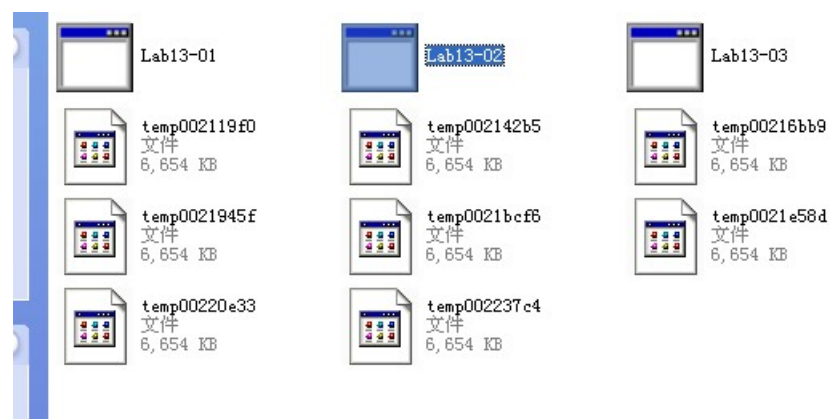
Questions

1. Using dynamic analysis, determine what this malware creates.
 2. Use static techniques such as an xor search, FindCrypt2, KANAL, and the IDA Entropy Plugin to look for potential encoding. What do you find?
 3. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?
 4. Where is the encoding function in the disassembly?
 5. Trace from the encoding function to the source of the encoded content. What is the content?
6. Can you find the algorithm used for encoding? If not, how can you decode the content?
 7. Using instrumentation, can you recover the original source of one of the encoded files?

1、动态分析，确定恶意代码创建了什么？

12:5...	services.exe	bb8	RegOpenKey	HKLM\System\CurrentControlSet\Control\DeviceClasses\{07760000-0000-1100-9402-000000000000}	SUCCESS	
12:5...	svchost.exe	852	CloseFile	C:\WINDOWS	SUCCESS	
12:5...	svchost.exe	852	CreateFile	C:\WINDOWS\system32	SUCCESS	Desired Access: Read Data/List Directory, Synch
12:5...	svchost.exe	852	QueryDirectory	C:\WINDOWS	SUCCESS	Filter: system32, 1: system32
12:5...	svchost.exe	852	CloseFile	C:\WINDOWS	SUCCESS	
12:5...	svchost.exe	852	CreateFile	C:\WINDOWS\system32	SUCCESS	Desired Access: Read Data/List Directory, Synch
12:5...	svchost.exe	852	QueryDirectory	C:\WINDOWS\system32\wbem	SUCCESS	Filter: wbem, 1: wbem

用 process monitor 监控，发现在程序的目录中创建了好多文件，还有好多文件写入的指令



每隔一段时间创建一个文件，这些文件都是以 temp 开头的，且文件大小一样，关掉 cmd 窗口程序就不再创建文件了。

2、静态分析，查找潜在的加密，发现了什么？

用 IDA 查找 xor

Address	Function	Instruction
.text:00401040	sub_401000	xor eax, eax
.text:004012D6	sub_40128D	xor eax, [ebp+var_10]
.text:0040171F		xor eax, [esi+edx*4]
.text:0040176F	sub_401739	xor edx, [ecx]
.text:0040177A	sub_401739	xor edx, ecx
.text:00401785	sub_401739	xor edx, ecx
.text:00401795	sub_401739	xor eax, [edx+8]
.text:004017A1	sub_401739	xor eax, edx
.text:004017AC	sub_401739	xor eax, edx
.text:004017BD	sub_401739	xor ecx, [eax+10h]
.text:004017C9	sub_401739	xor ecx, eax
.text:004017D4	sub_401739	xor ecx, eax
.text:004017E5	sub_401739	xor edx, [ecx+18h]
.text:004017F1	sub_401739	xor edx, ecx
.text:004017FC	sub_401739	xor edx, ecx
.text:0040191E	_main	xor eax, eax
.text:0040311A		xor dh, [eax]
.text:0040311E		xor [eax], dh

看到了好多异或指令，可能是异或加密，重点关注带有 var 操作数的异或指令，这些可能是对缓冲区的加密。

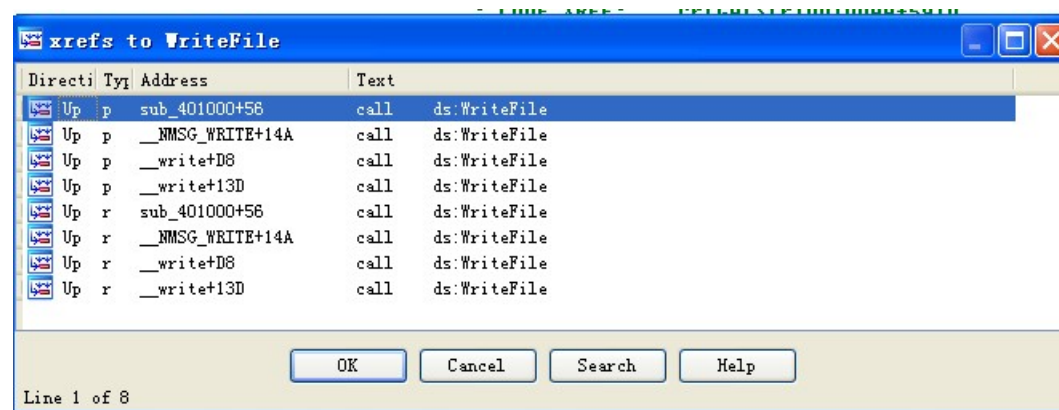
用 IDA 中的相关插件没有找到其他可能的加密。

3、基于问题 1 的回答，哪些导入函数将是寻找加密函数比较好的一个证据？

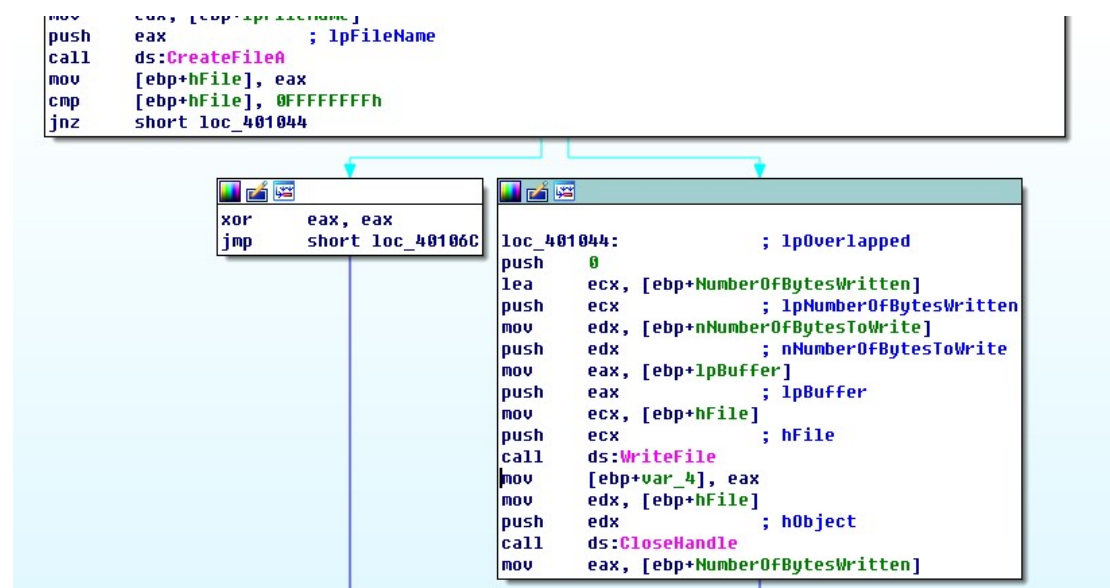
由于程序会间隔时间创建文件并向文件中写入数据，所以猜测加密函数可能会将数据加密人后写入创建的文件，所以重点关注 WriteFile 函数。

4、加密函数在反汇编的何处？

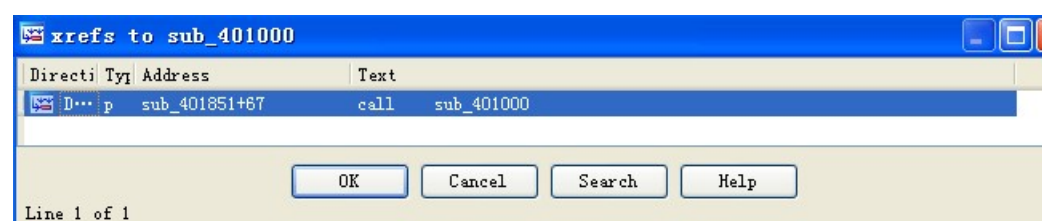
查看 WriteFile 函数的交叉引用



发现函数 sub_401000 对其进行了调用，进入查看：



这个函数就是创建或打开一个文件，然后将缓冲区的数据写入文件，下面看看缓冲区是啥，找到 401000 的交叉引用



进入



该函数首先调用 401070，查看：

```

push    ebp |
mov     ebp, esp
sub     esp, 78h
mov     [ebp+hdc], 0
push    0 ; nIndex
call    ds:GetSystemMetrics
mov     [ebp+var_1C], eax
push    1 ; nIndex
call    ds:GetSystemMetrics
mov     [ebp+cy], eax
call    ds:GetDesktopWindow
mov     hWnd, eax
mov     eax, hWnd
push    eax ; hWnd
call    ds:GetDC
mov     hDC, eax
mov     ecx, hDC
push    ecx ; hdc
call    ds>CreateCompatibleDC
mov     [ebp+hdc], eax
mov     edx, [ebp+cy]
push    edx ; cy
mov     eax, [ebp+var_1C]
push    eax ; cx
mov     ecx, hDC
push    ecx ; hdc
call    ds>CreateCompatibleBitmap
mov     [ebp+h] eax

```

经过分析就是获取当前的桌面信息，然后保存在缓冲区中，然后调用 40181F 函数，进入：

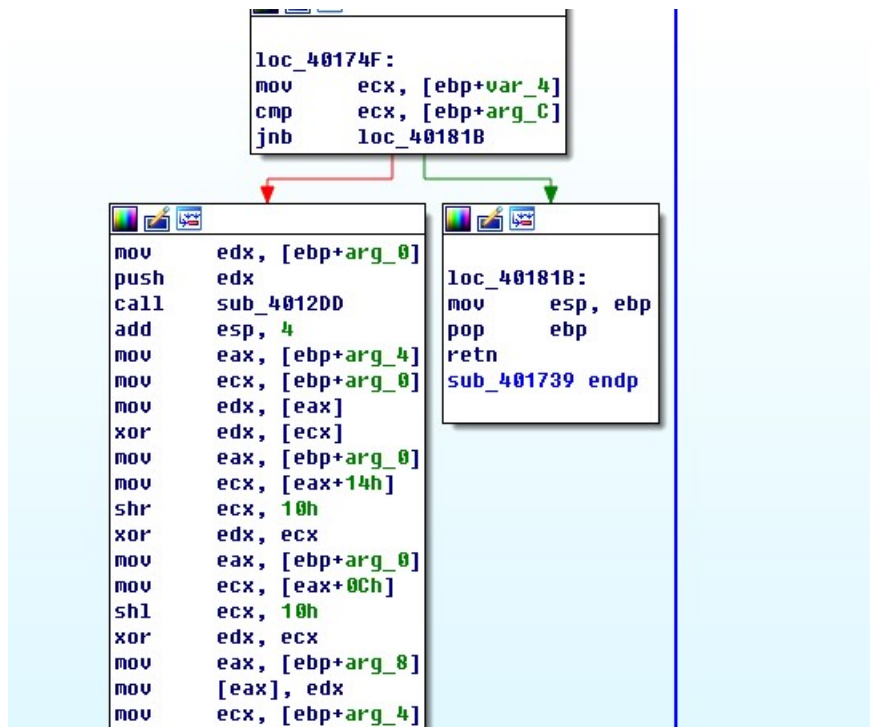
```

var_44= byte ptr -44h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch

push    ebp |
mov     ebp, esp
sub     esp, 44h
push    44h ; size_t
push    0 ; int
lea     eax, [ebp+var_44]
push    eax ; void *
call    _memset
add     esp, 0Ch
mov     ecx, [ebp+arg_4]
push    ecx
mov     edx, [ebp+arg_0]
push    edx
mov     eax, [ebp+arg_0]
push    eax
lea     ecx, [ebp+var_44]
push    ecx
call    sub_401739
add     esp, 10h
mov     esp, ebp
pop     ebp
retn
sub_40181F endp

```

调用了 4017389，进入



整体是一个大循环，对 401070 获取的桌面信息进行异或加密，所以这个就是加密函数的位置。

然后 401000 函数将加密后的内容写入文件，结合函数 GetTickCount 与 temp%08x 可以推测文件的名称就是系统启动到现在的时间。

5、从加密函数溯源原始加密内容，原始加密内容是什么？

由上述分析可知 401070 函数获取加密内容，内容为不断获取到的用户的桌面信息。

6、你是否能够找到加密算法？如果没有，如何解密？

可以找到加密算法，但是很复杂，不好分析

直接用 ollydbg 动态跟踪把原始数据拷贝下来，在调用 sub_40181F 之前去除截屏信息，在数据窗口选择复制全选，然后选择二进制复制，最后复制到 winhex，重命名为.bmp 文件，打开即可看到屏幕截图。

7、使用解密工具，能否恢复一个加密文件到原始文件？

如上一问

[S]	.rdata:00... 0000000B	C	USER32.dll
[S]	.rdata:00... 0000000B	C	WS2_32.dll
[S]	.data:004... 00000040	C	DEFGHIJKLMNOPQRSTUVWXYZAbcdefghijklmnopqrstuvwxyzab0123456789+/
[S]	.data:004... 0000003D	C	ERROR: API = %s.\n error code = %d.\n message = %s.\n
[S]	.data:004... 00000009	C	ReadFile
[S]	.data:004... 0000000D	C	WriteConsole
[S]	.data:004... 0000000C	C	ReadConsole
[S]	.data:004... 0000000A	C	WriteFile
[S]	.data:004... 00000010	C	DuplicateHandle
[S]	.data:004... 00000010	C	DuplicateHandle
[S]	.data:004... 00000010	C	DuplicateHandle
[S]	.data:004... 0000000C	C	CloseHandle
[S]	.data:004... 0000000C	C	CloseHandle
[S]	.data:004... 0000000D	C	GetStdHandle
[S]	.data:004... 00000008	C	cmd.exe
[S]	.data:004... 0000000C	C	CloseHandle
[S]	.data:004... 0000000C	C	CloseHandle
[S]	.data:004... 0000000C	C	CloseHandle
[S]	.data:004... 0000000D	C	CreateThread
[S]	.data:004... 0000000D	C	CreateThread
[S]	.data:004... 00000011	C	ijklmnopqrstuvwxyz
[S]	.data:004... 00000021	C	www.practicalmalwareanalysis.com
[S]	.data:004... 00000017	C	Object not Initialized
[S]	.data:004... 00000020	C	Data not multiple of Block Size

也看到了这个网址，可知没有加密网址信息，还发现一各大小写字母以及数字的字符串，为标准的 Base64 加密表，这里应该是自定义一个 Base64 加密。

此外还发现了一堆乱码的字符串，应该是被加密了，这里还不能看出本来是啥。

2、静态分析搜索 xor 查找潜在的加密，通过这，发现了什么类型的加密？

Address	Function	Instruction
.text:00401135	sub_401082	xor eax, eax
.text:0040123C	sub_401082	xor eax, eax
.text:00401310	sub_4012E9	xor ecx, ecx
.text:00401341	sub_40132B	xor eax, eax
.text:00401357	sub_40132B	xor eax, eax
.text:0040136D	sub_40132B	xor eax, eax
.text:004014A5	StartAddress	xor eax, eax
.text:004014BB	StartAddress	xor eax, eax
.text:00401873	sub_4015B7	xor eax, eax
.text:004019A5	_main	xor eax, eax
.text:00401A53	sub_401A50	xor eax, eax
.text:00401D51	sub_401AC2	xor edx, edx
.text:00401D69	sub_401AC2	xor eax, eax
.text:00401D88	sub_401AC2	xor eax, eax
.text:00401DA7	sub_401AC2	xor eax, eax
.text:00401EBD	sub_401AC2	xor eax, edx
.text:00401ED8	sub_401AC2	xor eax, edx
.text:00401EF3	sub_401AC2	xor eax, edx
.text:00401F08	sub_401AC2	xor eax, edx
.text:00401F13	sub_401AC2	xor edx, eax
.text:00401F56	sub_401AC2	xor eax, [esi+edx*4+414h]
.text:00401FB1	sub_401AC2	xor edx, [esi+ecx*4+414h]
.text:00402028	sub_401AC2	xor edx, ecx
.text:00402046	sub_401AC2	xor edx, ecx
.text:00402064	sub_401AC2	xor edx, ecx
.text:00402070	sub_401AC2	xor ecx, edx
.text:004020B3	sub_401AC2	xor edx, [esi+ecx*4+414h]

发现一堆 xor，经过分析，只有以下六个函数可能是加密函数：

sub_401AC2、sub_40223A、sub_4027ED、sub_402DA8、sub_403166 以及 sub_403990

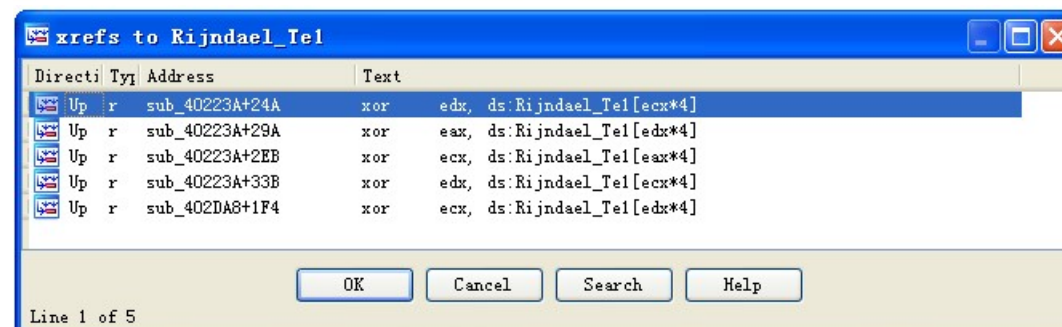
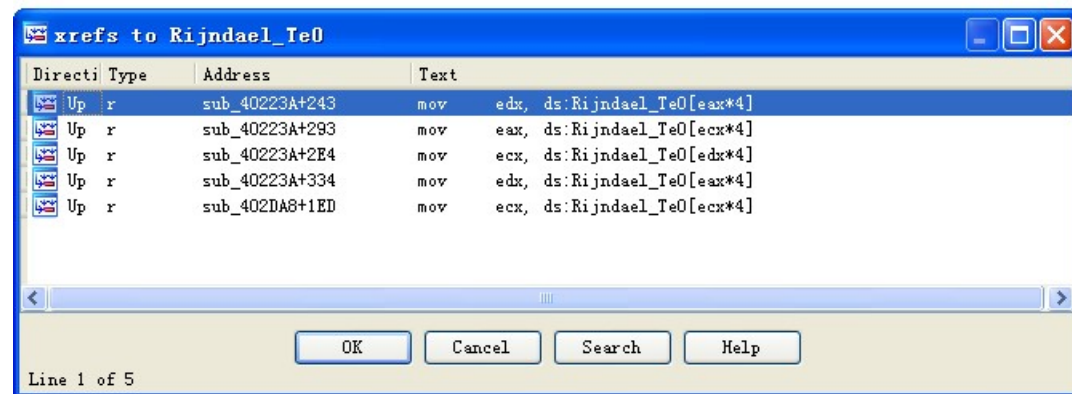
可能会存在异或加密。

3、使用静态工具，如 KANAL，识别一些其他类型的加密机制，发现的结果与搜索 xor 的结果比较如何？

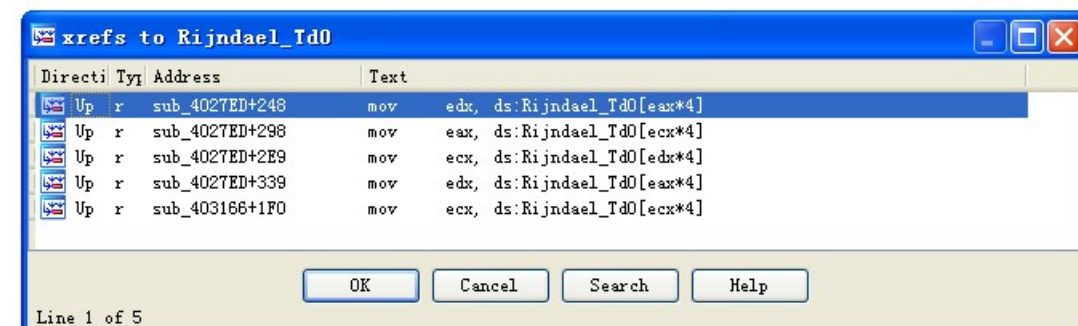
使用 IDA 的 FindCrypt 插件找到了 Rijndael 算法，也就是 AES 加密算法。

```
Caching 'Functions window'... OK
40CB08: found const array Rijndael_Te0 (used in Rijndael)
40CF08: found const array Rijndael_Te1 (used in Rijndael)
40D308: found const array Rijndael_Te2 (used in Rijndael)
40D708: found const array Rijndael_Te3 (used in Rijndael)
40DB08: found const array Rijndael_Td0 (used in Rijndael)
40DF08: found const array Rijndael_Td1 (used in Rijndael)
40E308: found const array Rijndael_Td2 (used in Rijndael)
40E708: found const array Rijndael_Td3 (used in Rijndael)
Found 8 known constant arrays in total.
```

搜索 xor 时得到六个可疑的加密函数，分析：



sub_40223A 和 sub_402DA8 使用了加密常量_TeX (X: 0-3)



xrefs to Rijndael_Td1			
Directi	Ty	Address	Text
Up	r	sub_4027ED+24F	xor edx, ds:Rijndael_Td1[ecx*4]
Up	r	sub_4027ED+29F	xor eax, ds:Rijndael_Td1[edx*4]
Up	r	sub_4027ED+2F0	xor ecx, ds:Rijndael_Td1[eax*4]
Up	r	sub_4027ED+340	xor edx, ds:Rijndael_Td1[ecx*4]
Up	r	sub_403166+1F7	xor ecx, ds:Rijndael_Td1[edx*4]

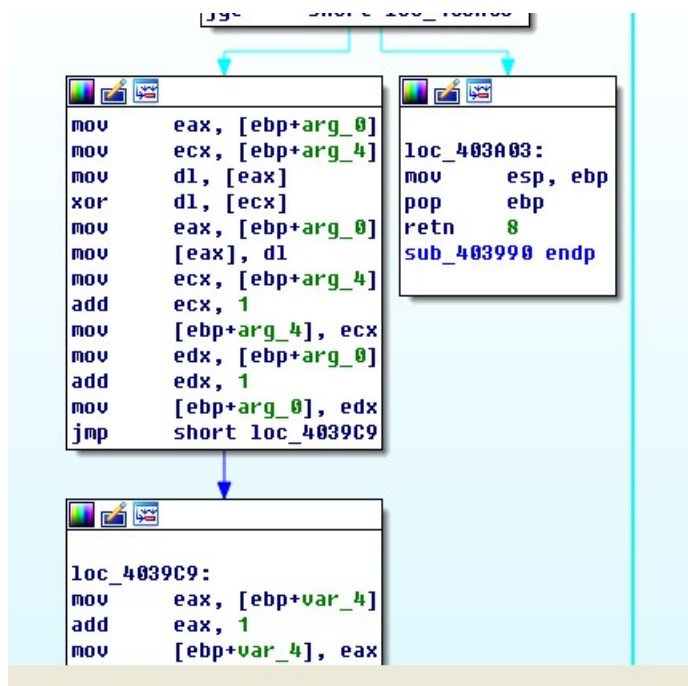
Line 1 of 5

OK Cancel Search Help

sub_4027ED 和 sub_403166 使用了加密常量_TdX (X: 0-3)

那么剩下的两个 sub_401AC2 和 sub_403990, 继续分析:

sub_403990



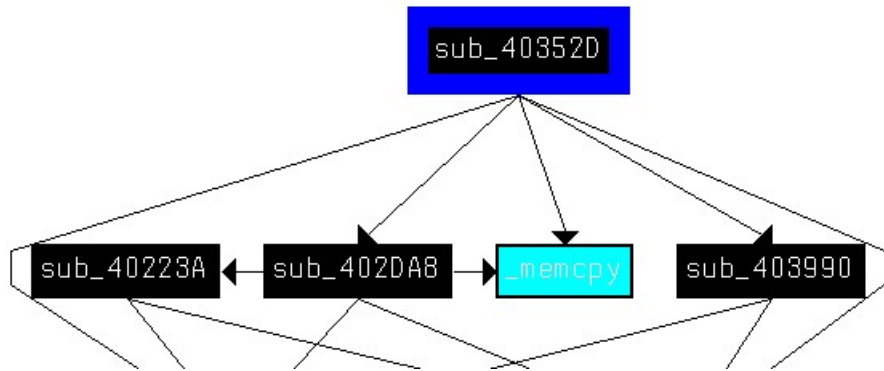
```

push    offset off_412240
lea     ecx, [ebp+var_10]
call    ??0exception@@QAE@ABQBD@Z ; exception::exception(char const * const &)
push    offset unk_410858
lea     edx, [ebp+var_10]
push    edx
call    __CxxThrowException@8 ; _CxxThrowException(x,x)

```

先循环异或加密, 再调用 AES 函数进行加密。

查看 403990 的交叉引用, 发现 40352D 函数调用了它, 查看这个函数:



发现这个函数同时调用了 sub_40223A 和 sub_402DA8，这是一个综合的加密函数，先看做整体，继续分析 sub_401AC2:

```

mov     [ebp+var_3C], offset aEmptyKey ; "Empty key"
lea     eax, [ebp+var_3C]
push    eax
lea     ecx, [ebp+var_38]
call    ??0exception@@QAE@ABQBD@Z ; exception::exception(char const * const &)
push    offset unk_410858
lea     ecx, [ebp+var_38]
push    ecx
call    __CxxThrowException@8 ; _CxxThrowException(x,x)

```

```

mov     [ebp+var_60], ecx
cmp     [ebp+arg_0], 0
jnz     short loc_401AF3

```

```

mov     [ebp+var_4C], offset aIncorrectKeyLe ; "Incorrect key length"
lea     edx, [ebp+var_4C]
push    edx
lea     ecx, [ebp+var_48]
call    ??0exception@@QAE@ABQBD@Z ; exception::exception(char const * const &)
push    offset unk_410858
lea     eax, [ebp+var_48]
push    eax
call    __CxxThrowException@8 ; _CxxThrowException(x,x)

```

```

mov     [ebp+var_5C], offset aIncorrectBlock ; "Incorrect block length"
lea     ecx, [ebp+var_5C]
push    ecx
lea     ecx, [ebp+var_58]
call    ??0exception@@QAE@ABQBD@Z ; exception::exception(char const * const &)
push    offset unk_410858
lea     edx, [ebp+var_58]
push    edx
call    __CxxThrowException@8 ; _CxxThrowException(x,x)

```

可以发现，这个函数一步一步地进行对传入的字符串进行比较，若不符合要求，就会返回 "Empty key", "Incorrect key length"和"Incorrect block length", 因此，这个函数可能是密钥的初始化代码。

查看其交叉引用，可知 main 函数调用，进入 main 函数:


```

enup= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 1ACh
push    10h           ; int
push    10h           ; int
push    offset unk_413374 ; void *
push    offset aijklmnopqrstuv ; "ijklmnopqrstuvwx"
mov     ecx, offset unk_412EF8
call    sub_401AC2
lea     eax, [ebp+WSAData]
push    eax           ; lpWSAData
push    202h          ; wVersionRequested
call    ds:WSAStartup
mov     [ebp+var_194], eax
cmp     [ebp+var_194], 0
jz      short loc_4018C5

```

看到 401AC2 函数的参数为 iijklmnopqrstuvwx，这个就是密钥。

4、恶意代码使用哪两种加密技术？

由上述分析，使用了 Base64 和 AES 加密技术。

5、对于每一种加密技术，密钥是什么？

Base64: CDEFGHIJKLMNOPQRSTUVWXYZABcdefghijklmnopqrstuvwxyz0123456789+/-

AES: iijklmnopqrstuvwx

6、对于加密算法，密钥足够可靠吗？除此之外还需要知道什么？

由前边的分析可知，40352D 函数为 AES 加密函数，查看交叉引用：

是在 sub_40132B 函数中被调用的，而 sub_40132B 函数又是在 sub_4015B7 函数中被调用的，

查看：

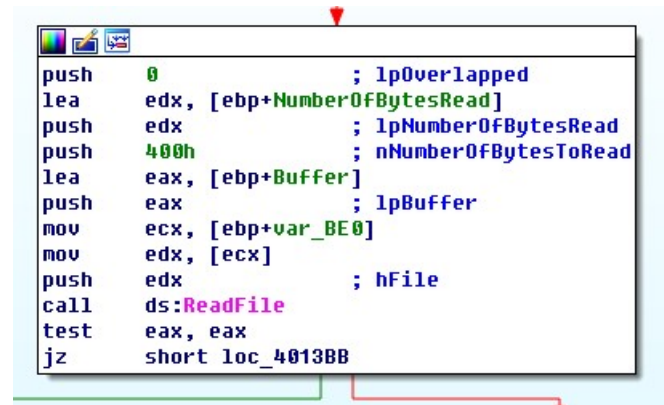
```

mov     eax, [ebp+var_18]
mov     [ebp+var_58], eax
mov     ecx, [ebp+arg_10]
mov     [ebp+var_54], ecx
mov     edx, dword_41336C
mov     [ebp+var_50], edx
lea     eax, [ebp+var_3C]
push    eax           ; lpThreadId
push    0             ; dwCreationFlags
lea     ecx, [ebp+var_58]
push    ecx           ; lpParameter
push    offset sub_40132B ; lpStartAddress
push    0             ; dwStackSize
push    0             ; lpThreadAttributes
call    ds:CreateThread
mov     [ebp+var_20], eax
cmp     [ebp+var_20], 0
jnz     short loc_401867

```

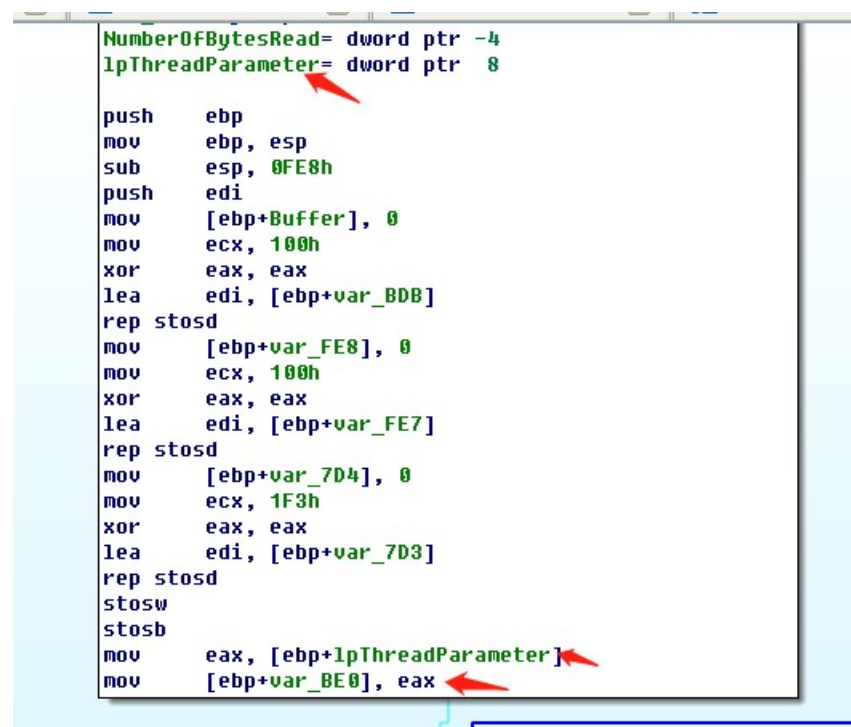
这里创建了一个线程，sub_40132B 是线程的开始。

传递给线程的参数保存在 lpParameter，也就是 var_58 中，00401826 处 var18 移入 var_58，0040182c 处 arg_10 移入 var_54，00401835 处 dwrdord_41336c 移入 var_50，然后进入 sub_40132B 看看具体流程：



```
push    0                ; lpOverlapped
lea     edx, [ebp+NumberOfBytesRead]
push    edx              ; lpNumberOfBytesRead
push    400h             ; nNumberOfBytesToRead
lea     eax, [ebp+Buffer]
push    eax              ; lpBuffer
mov     ecx, [ebp+var_BE0]
mov     edx, [ecx]
push    edx              ; hFile
call    ds:ReadFile
test    eax, eax
jz      short loc_4013BB
```

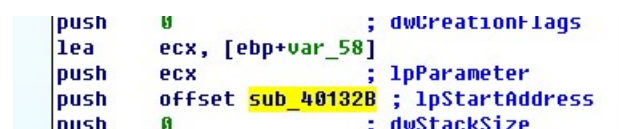
ReadFile 函数参数为 var_BE0，而 var_BE0 就是该函数的参数：



```
NumberOfBytesRead= dword ptr -4
lpThreadParameter= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 0FE8h
push    edi
mov     [ebp+Buffer], 0
mov     ecx, 100h
xor     eax, eax
lea     edi, [ebp+var_BDB]
rep stsd
mov     [ebp+var_FE8], 0
mov     ecx, 100h
xor     eax, eax
lea     edi, [ebp+var_FE7]
rep stsd
mov     [ebp+var_7D4], 0
mov     ecx, 1F3h
xor     eax, eax
lea     edi, [ebp+var_7D3]
rep stsd
stosw
stosb
mov     eax, [ebp+lpThreadParameter]
mov     [ebp+var_BE0], eax
```

再回到上层函数：



```
push    0                ; dwCreationFlags
lea     ecx, [ebp+var_58]
push    ecx              ; lpParameter
push    offset sub_40132B ; lpStartAddress
push    0                ; dwStackSize
```

该参数来自 var_58

再看看 sub_40132B 的 WriteFile 函数：

```

lea     edx, [ebp+var_FE8]
push    edx                ; void *
call    _memset
add     esp, 0Ch
push    1
mov     eax, [ebp+nNumberOfBytesToWrite]
push    eax
lea     ecx, [ebp+var_FE8]
push    ecx
lea     edx, [ebp+Buffer]
push    edx
mov     ecx, offset unk_412EF8
call    sub_40352D
push    0                  ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax                ; lpNumberOfBytesWritten
mov     ecx, [ebp+nNumberOfBytesToWrite]
push    ecx                ; nNumberOfBytesToWrite
lea     edx, [ebp+var_FE8]
push    edx                ; lpBuffer
mov     eax, [ebp+var_BE0]
mov     ecx, [eax+4]
push    ecx                ; hFile
call    ds:WriteFile
test    eax, eax
jnz     short loc_40146B

```

参数是 var_BE0+4, 也就是 var_54

而 var_58 和 var_18 持有一个管道的句柄, 并且这个管道的与一个 shell 命令的输出相连接

```

loc_401656:                ; dwOptions
push    2
push    0                  ; bInheritHandle
push    0                  ; dwDesiredAccess
lea     ecx, [ebp+var_18]
push    ecx                ; lpTargetHandle
call    ds:GetCurrentProcess
push    eax                ; hTargetProcessHandle
mov     edx, [ebp+hReadPipe]
push    edx                ; hSourceHandle
call    ds:GetCurrentProcess
push    eax                ; hSourceProcessHandle
call    ds:DuplicateHandle
test    eax, eax
jnz     short loc_401689

```

```

lea     ecx, [ebp+ProcessInformation]
push    edx                ; void *
call    _memset
add     esp, 0Ch
mov     eax, [ebp+hObject]
mov     [ebp+StartupInfo.hStdInput], eax
mov     ecx, [ebp+hWritePipe]
mov     [ebp+StartupInfo.hStdOutput], ecx
mov     edx, [ebp+hWritePipe]
mov     [ebp+StartupInfo.hStdError], edx
mov     eax, [ebp+StartupInfo.dwFlags]
or      ah, 1
mov     [ebp+StartupInfo.dwFlags], eax
lea     ecx, [ebp+ProcessInformation]
push    ecx                ; lpProcessInformation
lea     edx, [ebp+StartupInfo]
push    edx                ; lpStartupInfo
push    0                  ; lpCurrentDirectory
push    0                  ; lpEnvironment
push    0                  ; dwCreationFlags
push    1                  ; bInheritHandles
push    0                  ; lpThreadAttributes
push    0                  ; lpProcessAttributes
push    offset CommandLine ; "cmd.exe"
push    0                  ; lpApplicationName
call    ds:CreateProcessA
mov     [ebp+var_34], eax

```

命令 hSourceHandle 通过 DuplicateHandle 复制到 shell 命令的标准输出和标准错误, 这条 shell 命令由 CreateProcess 启动。

对于 var_54, 其是 sub_4015B7 的唯一参数, 交叉引用, 来到 main, 看看参数是啥

```

019A7: loc_40196E:
mov     ecx, [ebp+s]
push    ecx
sub     esp, 10h
mov     edx, esp
mov     eax, dword ptr [ebp+name.sa_family]
mov     [edx], eax
mov     ecx, dword ptr [ebp+name.sa_data+2]
mov     [edx+4], ecx
mov     eax, dword ptr [ebp+name.sa_data+6]
mov     [edx+8], eax
mov     ecx, dword ptr [ebp+name.sa_data+0Ah]
mov     [edx+0Ch], ecx
call    sub_4015B7
add     esp, 14h
xor     eax, eax

```

可知其参数来自 ebp+s, 而它是用 connect 调用创建的一个网络套接字

```

push    eax                ; name
mov     eax, [ebp+s]
push    eax                ; s
call    ds:connect
mov     [ebp+var_194], eax
cmp     [ebp+var_194], 0FFFFFFFh
jnz     short loc_40196E

```



至此, 也就知道了, sub_4015B7 用于读取 shell 命令的输出结果, 在写入网络套接字之前加密它。

综上：

程序通过创建线程的方式调用了 sub_40132B，还注意到还以线程方式调用 strataddress，我们发现(通过看 base64 的字符串引用,一直找上来)这个函数会调用 base64 的加密函数,并且也会在 readfile 和 writefile 之间调用。

函数 sub_40132B 读取 shell 命令的输出结果，将其加密之后写入网络套接字。Base64 加密函数 sub_401082 在一个由它们宿主线程启动的函数 sub_40147C 中使用。

查看主程序



可知，程序通过网络套接字与远程建立连接，Base64 线程读取远程套接字内容作为输入，经过函数解密后，再将结果发送，作为命令 shell 的输入。

对于自定义 Base64 加密的实现，索引字符串已经足够了。但是对于 AES，实现解密可能需要密钥之外的变量。如果使用密钥生成算法，则包括密钥生成算法、密钥大小、操作模式，如果需要还包括向量的初始化等。

7、恶意代码做了什么？

使用 Base64 算法来加密从远程套接字处获取的命令，然后使用 AES 加密作为传出 shell 命令，来建立反向 shell 连接。

8、构造代码来解密动态分析过程中生成的一些内容，解密后的内容是什么？

解密 base64 加密的数据：

```
import string
import base64

s = ""
tab = 'CDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
b64 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

ciphertext = 'BIaEi=='

for ch in ciphertext:
    if (ch in tab):
        s += b64[string.find(tab,str(ch))]
    elif (ch == '='):
        s += '='
```

结果为 dir，攻击者尝试发送的是一个 shell 命令 dir，用于列目录。

解密 AES 加密的数据：

用 wireshark 捕获的加密数据包为

```
37 f3 1f 04 51 20 e0 b5 86 ac b6 0f 65 20 89 92
4f af 98 a4 c8 76 98 a6 4d d5 51 8f a5 cb 51 c5
cf 86 11 0d c5 35 38 5c 9c c5 ab 66 78 40 1d df
4a 53 f0 11 0f 57 6d 4f b7 c9 c8 bf 29 79 2f c1
ec 60 b2 23 00 7b 28 fa 4d c1 7b 81 93 bb ca 9e
bb 27 dd 47 b6 be 0b 0f 66 10 95 17 9e d7 c4 8d
ee 11 09 99 20 49 3b df de be 6e ef 6a 12 db bd
a6 76 b0 22 13 ee a9 38 2d 2f 56 06 78 cb 2f 91
```

编写脚本：

```
from Crypto.Cipher import AES
import binascii
c = binascii.unhexlify(raw.replace(' ',''))
res = AES.new('ijklmnopqrstuvwxyz',AES.MODE_CBC)
print(obj.decrypt(c))
```

解密结果为：

```
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.
```

四、 实验心得

本次实验主要对恶意代码中的加密操作进行了分析，分别实践了异或加密、AES 加密、Base64 加密，体会了恶意程序加密的目的和方法，搜索加密的位置，加密的类型，分析加密的目的，得到恶意代码的功能，并编写基础的解密脚本，对加密信息进行解密来辅助分析。