## 课本实验13实验报告

姓名：魏伯繁 学号：2011395 专业：信息安全

### **一、实验工具：**

IDAPro7.0 IDAPro7.7 win10虚拟机 winxp虚拟机、Wireshark、Ollydbg、strings、ProcessMonitor、Process Explorer、RegShot等

### 二、实验环境：

静态分析主要在win10虚拟机完成。

动态分析在winxp虚拟机上完成

内存4GB

主频2.9GHZ

### 三、实验内容

#### 第一部分：Lab13-01

1.比较恶意代码中的字符串(字符串命令的输出)与动态分析提供的有用信息，基于这些比较，哪些元素可能被加密?

2.使用IDA Pro 搜索恶意代码中字符串xor’，以此来查找潜在的加密，你发现了哪些加密类型?

3.恶意代码使用什么密钥加密，加密了什么内容?

4.使用静态工具FindCrypt2 Krypto ANALyzer (KANAL)以及IDA插件识别一些其他类型的加密机制，你发现了什么?

5.什么类型的加密被恶意代码用来发送部分网络流量?

6.Base64编码函数在反汇编的何处?

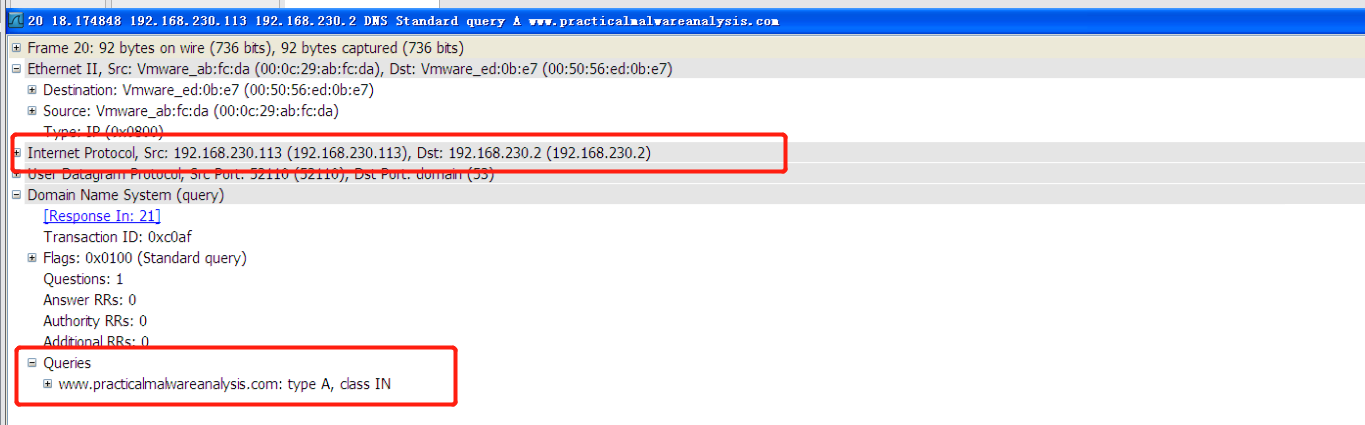
7.恶意代码发送的 Base64 加密数据的最大长度是什么?加密了什么内容?

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==)?

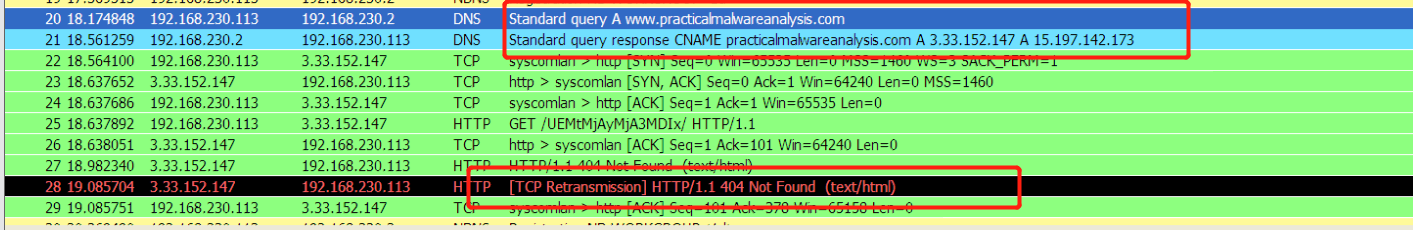
9.这个恶意代码做了什么?

分析过程：

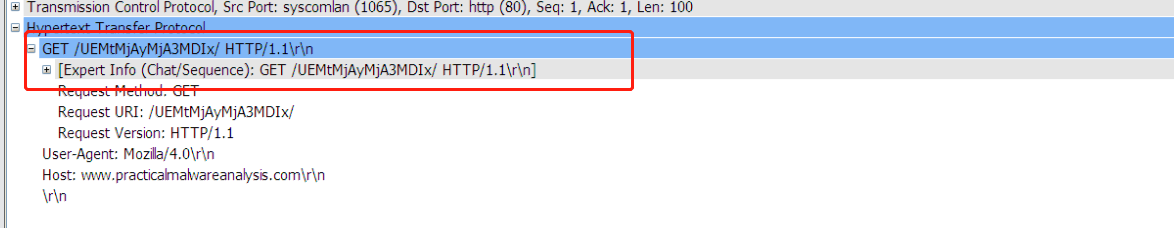
首先我们可以运行Lab13-01.exe来分析其具体的行为，并通过wireshark进行抓包分析。我们可以通过结果简单发现他请求了一个网站<www.practicalmalwareanalysis.com>:



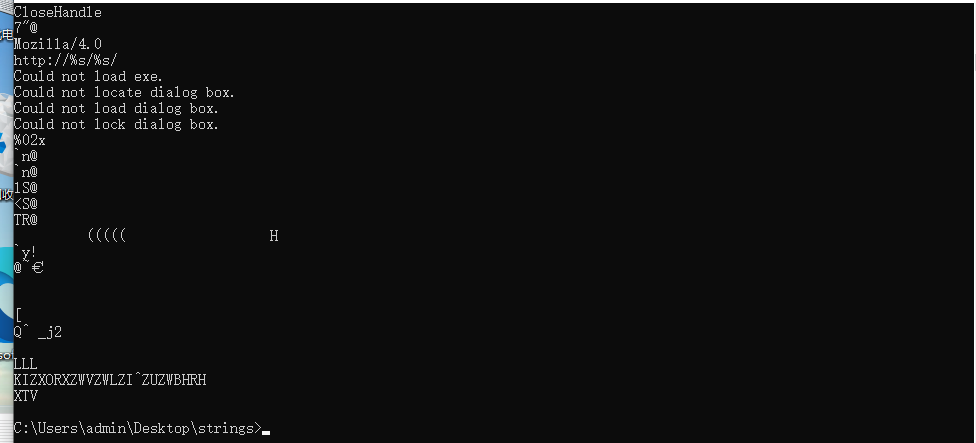
并且观察抓包结果我们还可以得出他并没有访问成功，只是得到了404的结果：

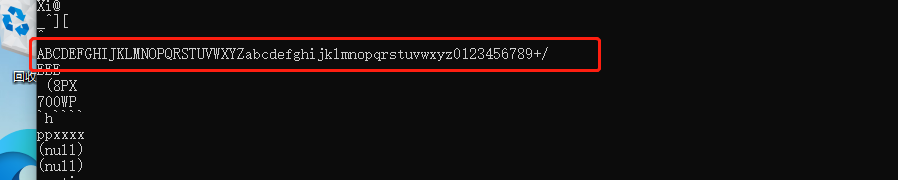


当然，重点在于，当我们对他的GET请求进行分析时分析出了一些乱码，现在我们还不知道这个乱码的含义，但可以肯定的是这很有可能和我现在使用的计算机系统是相关的



更奇怪的是，如果我们对Lab13-01使用strings工具，可以发现我们并不能观察到其访问的网址以及那个奇怪的乱码字符串，但是我们却可以观察到a-zA-Z0-9，所以我们可以猜测有一些重要字符串被加密了

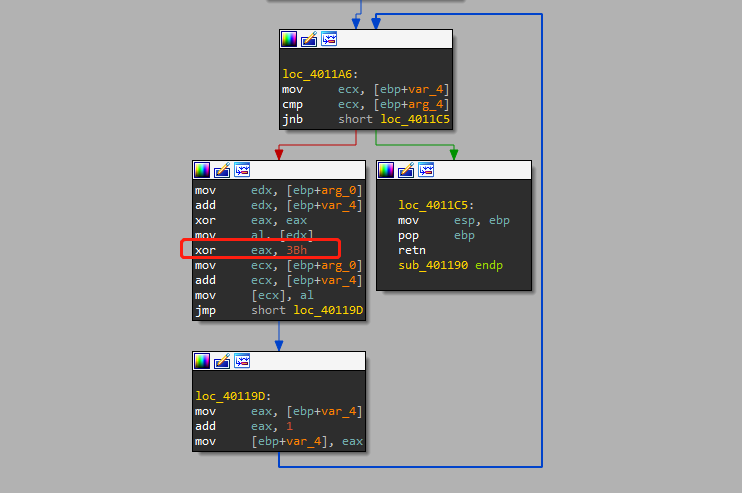




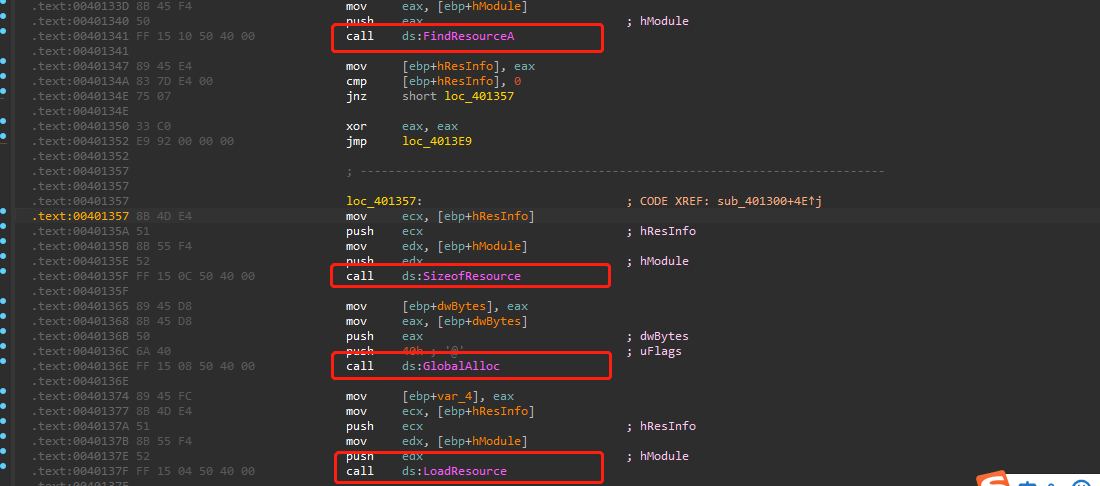
接下来，我们可以使用静态分析来分析程序是如何运作的：

在IDAPro中搜索XOR指令，并且将一些对库函数的引用去除后我们决定将分析的重点聚焦在sub401190函数中，函数的执行流程图如下图：

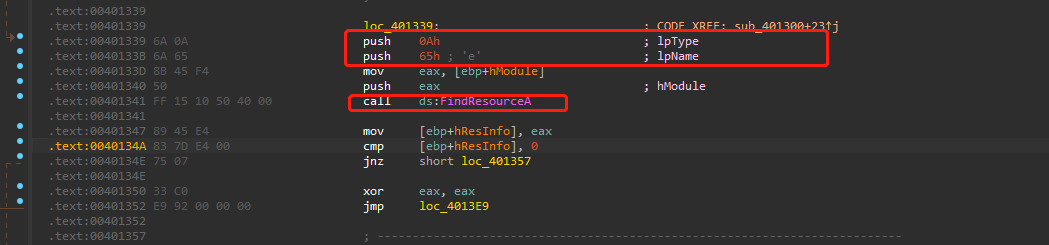
很明显它使用了声明的变量var4用做递增计数器，并且用0x3B与缓冲区第一个参数中原始内容的异或结果来修改缓冲区。其他被异或的参数 ，也就是第二个参数是缓冲区的长度。它用静态字节0x3B执行单字节XOR加密，并将缓冲区及其长度作为它的参数。



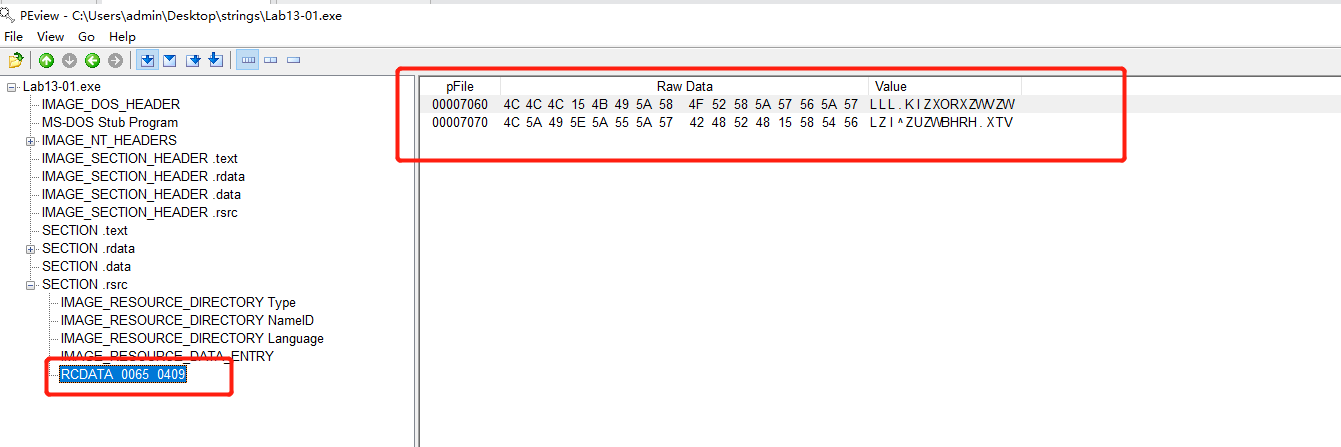
然后我们需要知道究竟是谁调用了这个函数，并且参数是什么字符串：函数sub401300是调用上面函数的唯一函数。跟踪它调用解密函数之前的代码块，我们依次看到调用GetModuleHandleA、FindResourceA、SizeofResource、GlobalAlloc、LoadResource以及LockResource。调用解密函数之前，恶意代码会对资源做一些处理。在这些资源相关的函数中，我们应该研究让我们找到资源数据的函数FindResourceA。



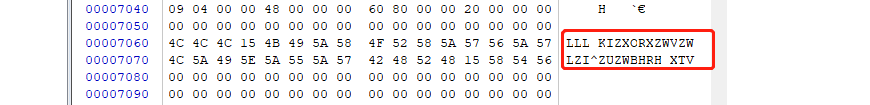
其中的重点毫无疑问是加载资源的LoadResourceA函数，我们将针对该函数进行具体的分析：其中lpType是0xA，它表示资源数据是应用程序预定义的还是原始数据。lpName既可以是一个名字，也可以是一个索引号。本例中，它是一个索引号。由于函数引用了ID为101的资源



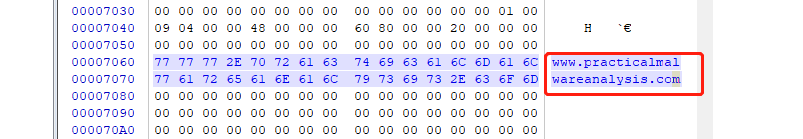
于是我们去PEVIEW中寻找该资源，也就是索引号为0x65的资源，查找也比较简单，打开资源节找到非常醒目的RCDATA就可以找到目标字符串，在资源节偏移0x7060的位置



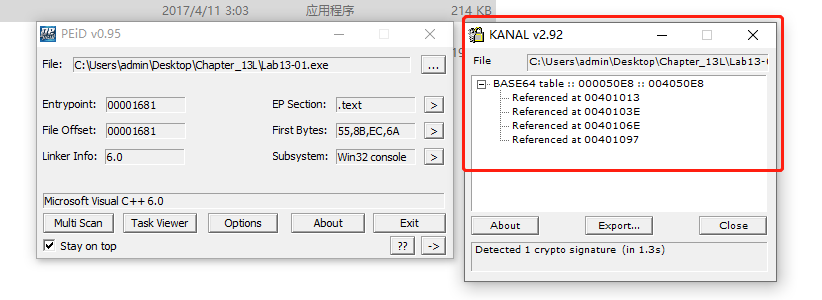
然后我们使用WinHEX对这部分资源进行转换：



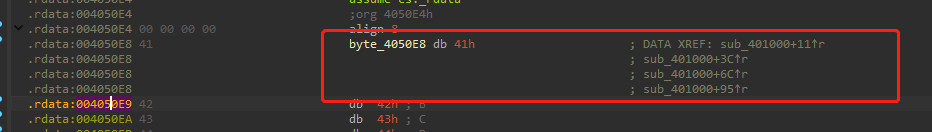
选择转转方式为XOR然后输入要异或的数字3B即可，稍等片刻WINHEX就会帮我们更改目标文件：



有两个字符串也可能被加密，我们发现了域名，但是没有发现GET请求字符串。为了找到GET字符串，我使用PEiD的KANAL插件，它在0x004050F8处找到一个Base64编码表。

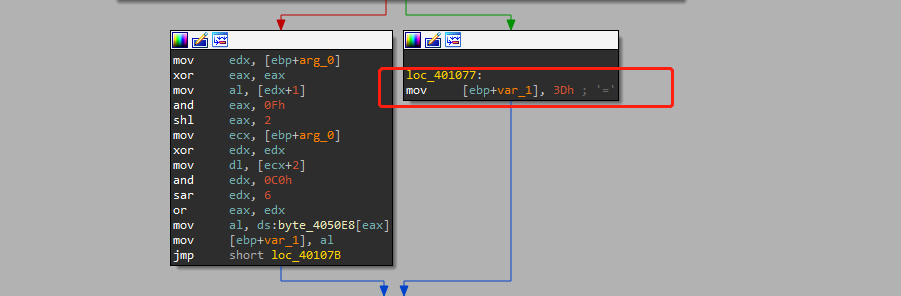


随后，我们找到在基本静态分析时找到的典型的Base64编码表的位置，发现他有四个交叉引用，并且全部位于sub401000函数



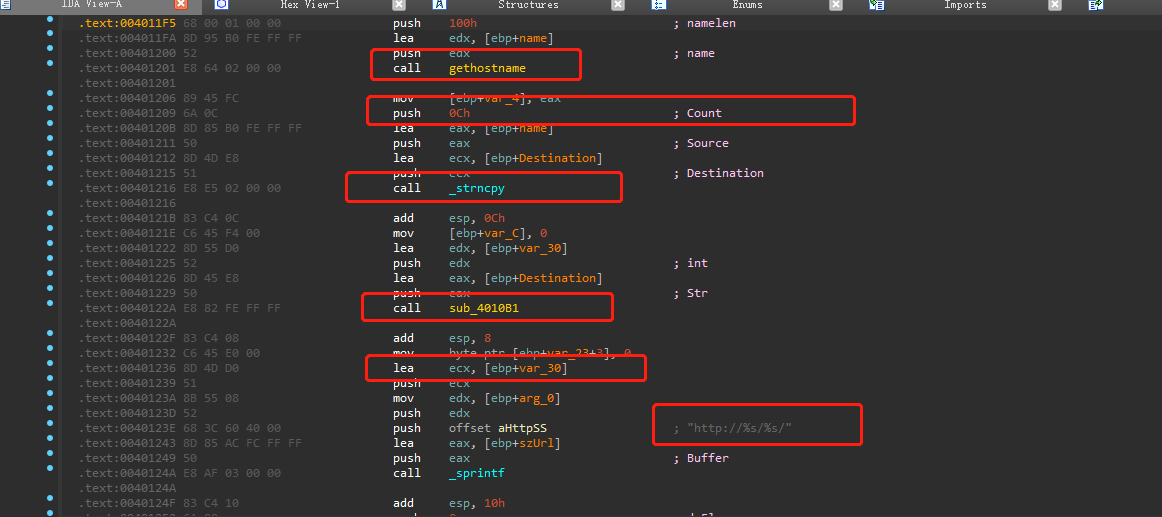
我们马上来到该函数：

我们可以看到，图中右侧的框中有一个字符=的交叉引用。这证实了我们的判断:sub401000与Base64编码相关，因为在Base64加密中以“=’作为填充字符。调用本函数的函数(4010B1) 是真正的Base64加密函数。它的目的是将源字符串划分成3个字节的块，并且将3个字节块传递给本函数，从而将传入的3个字节加密成4个字一些线索使得这更加明显:函数开头使用strlen查找源字符串的长度，在外部循环(代码块loc 401100)的开头与3比较 ，当本函数返回结果后，在内部写循环的开头与4比较。因此我们得出结论: 前面分析的解密函数就是Base64编码的主函数，他用源字符和Base64加密转换的目的缓冲区作为参数



我们还可以发现一个对前面分析的解密函数逇交叉引用，在4011C9的位置：

看一下传递到解密函数目的字符串，我们看到，它作为sprintf的第4个参数入栈。具体说，以<http://%s/%s/(>处)格式的第2个字符串是URI的路径。这与我们前面识别的字符串乱码是一致。接下来，我们跟踪传递给解密函数的源字符串，可以看到，它是strncpy函数的输出。并且strncpy的输入是处gethostname的输出。因此，我们可以明白加密的URI路径源字符串是主机名。strncpy函数仅复制了主机名的前12个字节，如所示。



后面的代码它使用了WinINet (InternetopenA、InternetOpenUrIA和InternetReadFile)打开并且读取代码清单解密出的URL。返回数据的第一个字符与字母o比较。如果第一个字母是o，则返回1，否则返回0。主函数由一个循环sleep调用以及beacon调用组成。当beacon (0x004011C9)返回真时(获得以o开始的Web响应)则循环退出程序终止。  
概括来说，这个恶意代码发送通信信号beacon，让攻击者知道它在正常运行。恶意代码用加密的(有可能截断)主机名作为标识符发出一个特定的通信信号，当接收到一个特定的回应后，则终止程序的运行



#### 第二部分：Lab13-02：

1.使用动态分析，确定恶意代码创建了什么?

2.使用静态分析技术，例如 xor 指令搜索、FindCrypt2、KANAL 以及IDA 插件，查找潜在的加密，你发现了什么?

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?

4.加密函数在反汇编的何处?

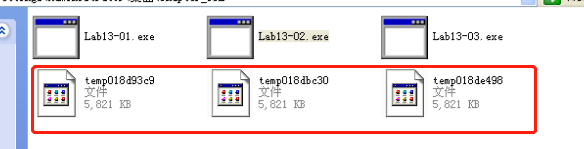
5.从加密函数追溯原始的加密内容，原始加密内容是什么?

6.你是否能够找到加密算法?如果没有，你如何解密这些内容?

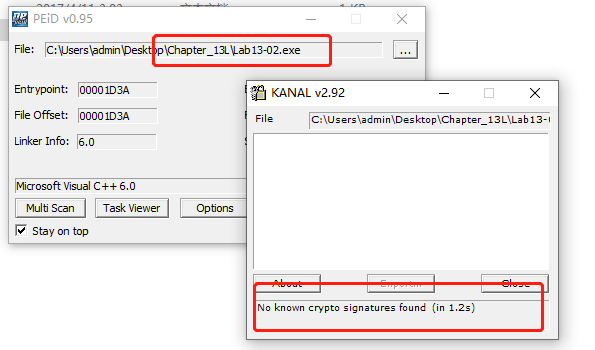
7.使用解密工具，你是否能够恢复加密文件中的一个文件到原始文件?

分析过程：

首先，双击运行该恶意代码，我们可以观察到，他会在原始目录下创建出很多temp文件，而且文件普遍较大：

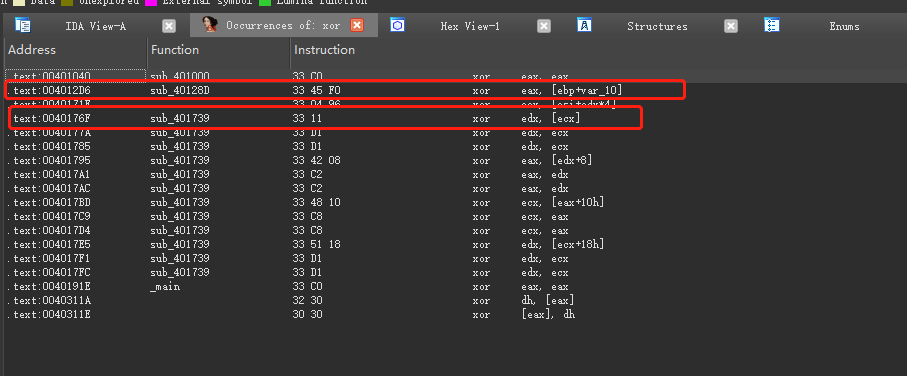


使用peid扫描该可执行文件未发现可以被轻易检测的加密证据：

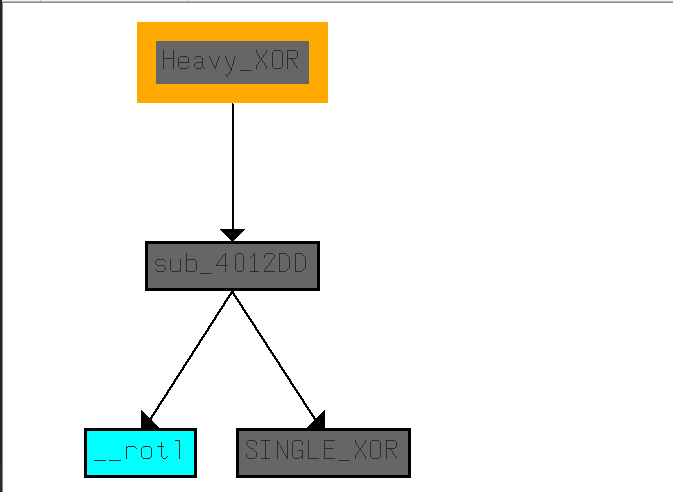


接下来，我们使用静态分析，进入IDAPro搜索XOR指令，去除掉一些库函数的引用、寄存器清零等无特殊含义的异或指令调用，我们可以找到两个很有价值的异或指令：分别位于004012D6和0040176F

尤其要注意位于0x0040171F的xor指令，我们看到它位于一个函数中，但是由于很少使用，这个函数并没有被自动识别。在0x00401570定义函数导致创建包含了前面孤立的xor指令函数。这个未使用的函数也与同组内可能的加密函数有关联。



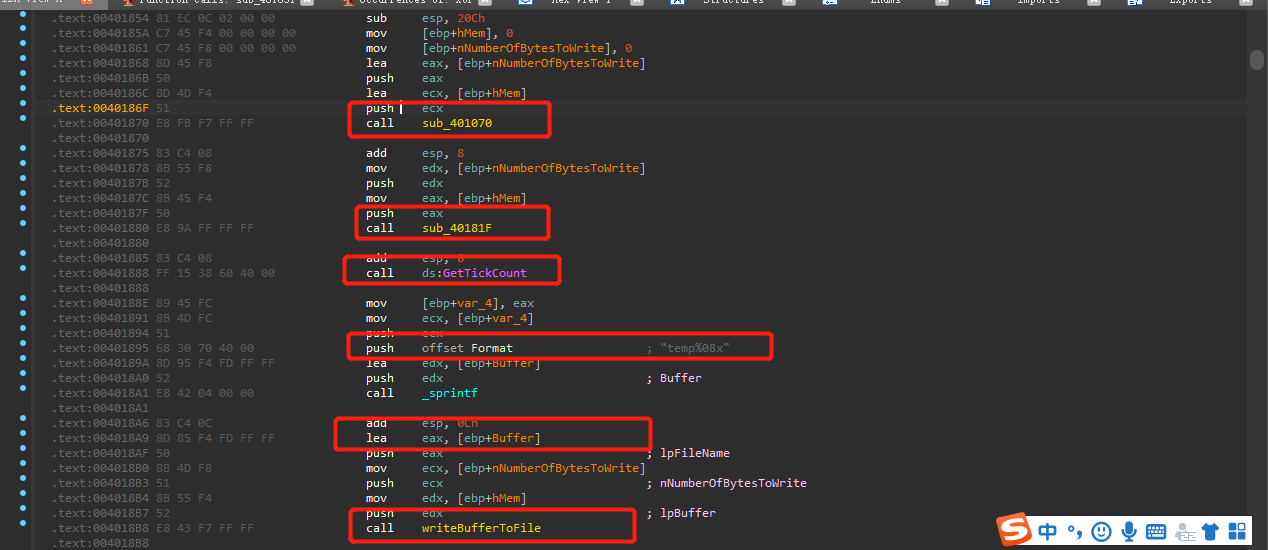
因为sub401739拥有非常多的xor指令，我们将sub 491739称为Heavy\_xor。而sub40128D拥有一条xor指令，我们将sub40128D称为SINGLE\_xor。heavyxor带有4个参数，它是一个单循环，除了xor指令以外，拥有包含多个包含SHL和SHR指令的代码块。看一下heavy\_xor调用的函数，我们发现singlexor与heavy\_xor相关，因为singlexor的调用者也被heavy\_xor调用。



为了确定heavyxor的确是一个加密函数，让我们来看一下它与写入到磁盘上temp文件有何种关系我们找到了数据写入到磁盘的位置，然后向前回溯确定怎么使用的加密函数。查看一下导入函数，我们看到了存在函数调用WriteFile。检查一下writeFile的交叉引用，我们找到了sub 401000,它用一个缓冲区，一个缓冲区长度一个文件名作为参数，它打开一个文件并且将缓冲区数据写入到这个文件。我们将sub 401000重命名为writeBufferToFile。sub 401851是调用wirteBufferToFile的唯一函数。下图展示了sub 401851的内容(我们将它命名为doStuffAndwriteFile)，作为0处调用writeBufferToFile函数前的准备

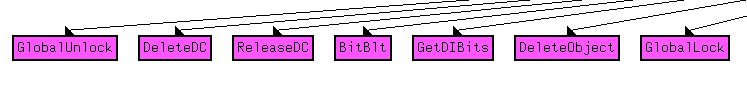
在下图开始的位置，可以发现两处函数调用sub401070和sub 40181F它们都是用缓冲区以及缓冲区的长度作为参数。结合GetTickCount 结果的格式化字符串"temp%08x”发现了原文件名，它是当前时间的十六进制打印。IDA Pro已经标注了文件名，我们可以猜测处的sub 40170用来获取内容(我们称它为getContent)，sub\_40181F用来加密内容(我们重命名为encodingwrapper)

首先看一下我们假设的加密函数encodingwrapper(在0x0040181F)，可以看到它仅仅是一个heavy\_xor函数的封装器。这证实了我们前面的猜想所说的该函数是加密函数。函数encodingwrapper首先设置了4个加密的参数:一个使用之前清空的本地变量，从dostuffAndwriteFile传递过来指向同-个缓冲区的两个指针以及一个从dostuffAndwriteFile传递过来的缓冲区大小。两个指针指向同一个内存缓冲区暗示编码函数携带源和目的缓冲区以及一个长度，这种情况下，加密将在适当的地方执行。



接下来，我们确定加密并且写入磁盘的原始内容。如前面提到的一样，函数getContent(在0x00401070)似乎获取一些内容。看一下getContent，我们看到一个拥有多个系统函数的代码块。

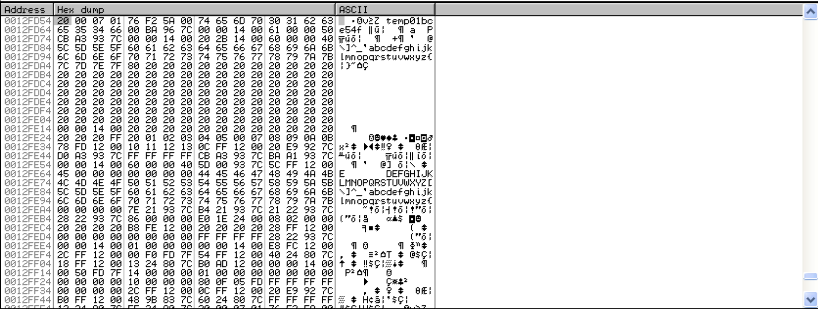
我们可以根据他的系统函数调用列表进行猜测，一种较好的猜测是:这个函数试图抓屏。值得注意的是，GetDesktopwindow获取覆盖整个屏幕桌面窗口的一个句柄，函数BitBlt和GetDIBits(粗体所示)获取位图信息并将它们复制到缓冲区。因此，我们得出结论，恶意代码反复抓取用户的桌面并且将加密版本的抓屏信息写入到一个文件



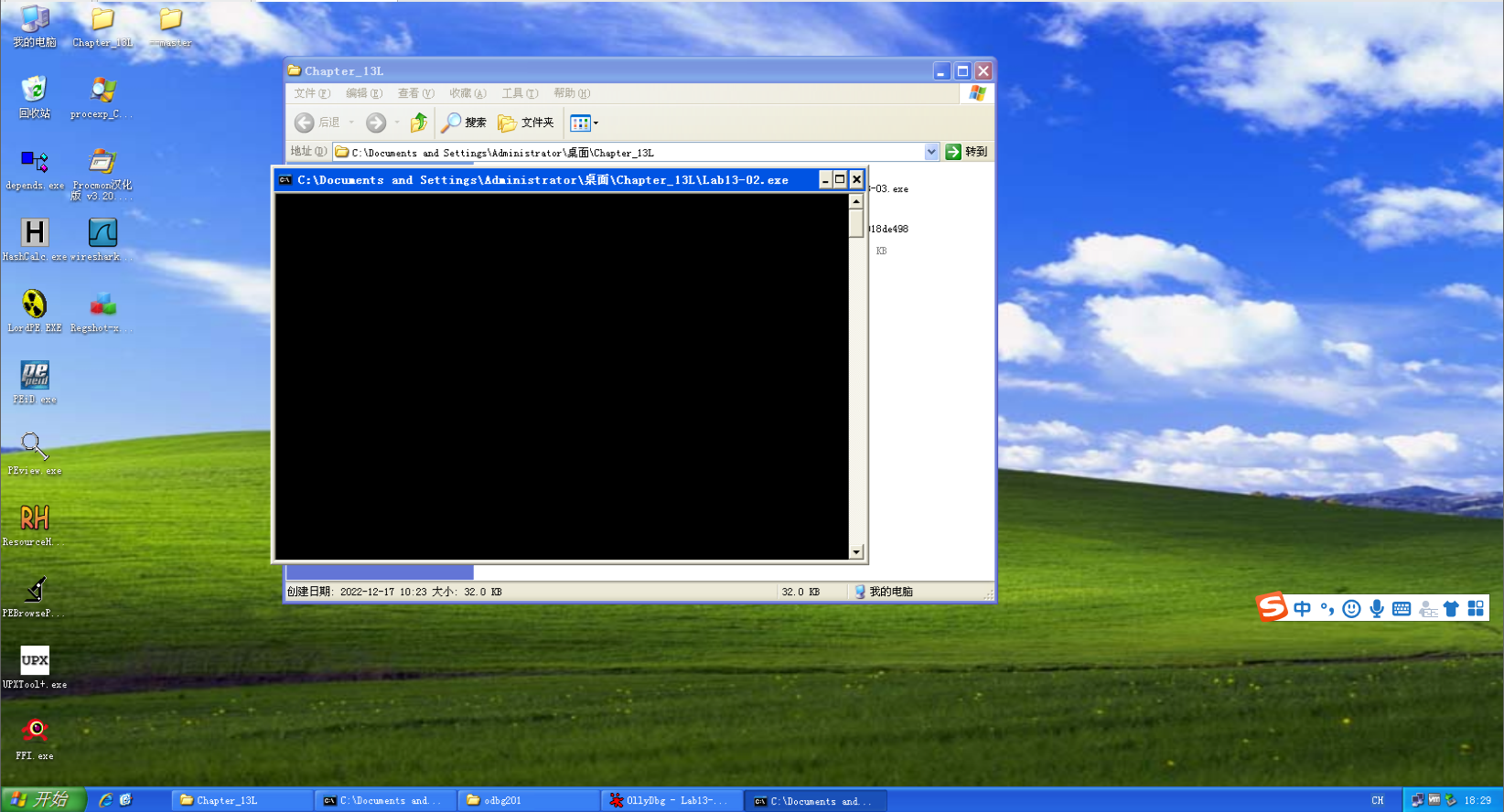
比较正常的一种思路就是使用ollydbg进行解密，这时我们需要设置两个断点，第一个事在加密开始前，也就是在0x00401880作为断点，第二个断电我们可以设置在文件写入到内存后，也就是位于0x0040190A，此时我们可以根据观察寄存器和堆栈的值来进行解密。



此时一个比较重要的值是esp，我们需要在真正的内存中取出对应位置的值，当我们找到对应位置时果然发现了一些数据，但这些数据我们是无法直接读取的，我们需要做的就是先把他们复制出来：



然后我们可以点击十六机制的转存，随后点击ollydbg的运行键让程序运行到最后䘝断点的位置。这个时候可以检查恶意代码目录中与之前产生的文件有相同名字的文件，并给他添加扩展名bmp再次打开他就可以观察到一张关于屏幕的截屏了



#### 第三部分：Lab13-03：

1.比较恶意代码的输出字符串和动态分析提供的信息，通过这些比较，你发现哪些元素可能被加案?

2.使用静态分析搜索字符 xor 来查找潜在的加密。通过这种方法，你发现什么类型的加密?

3使用静态工具，如 FindCvpt2、KANAL 以及DA 插件识别一些其他类型的加案机制。发现的结果与搜索字符XOR结果比较如何?

4.恶意代码使用哪两种加密技术?

5.对于每一种加密技术，它们的密钥是什么?

6.对于加密算法，它的密钥足够可靠吗?另外你必须知道什么?

7.恶意代码做了什么?

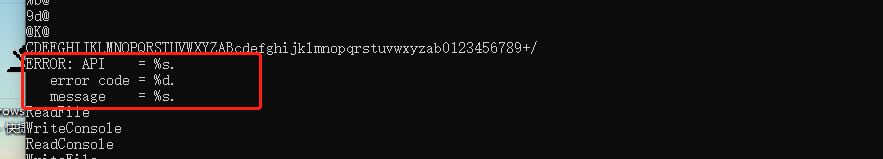
8.构造代码来解密动态分析过程中生成的一些内容，解密后的内容是什么?

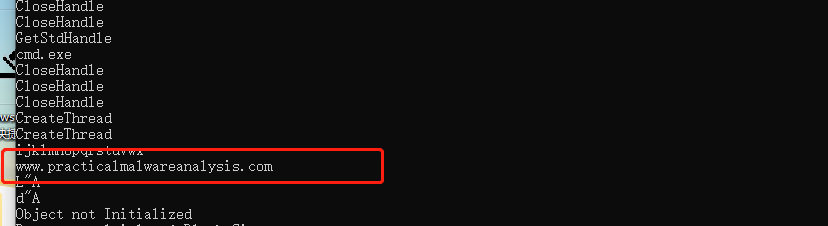
分析过程：

我们首先点击运行可执行文件并配置wireshark进行监听，我们发现应用程序对www.pracitcalmalwareanalysis进行了访问，也就是说，本恶意代码是具有网络特征的

接下来我们再使用基本静态分析进行检测：

我们可以发现很对格式字符串以及和Base64相关的字符串，这说明有一些内容很有可能被加密了，但是网站的域名却没有被加密，同时我们观察这个很像Base64加密编码的字符序列，发现他并不是最常见的编码方式，因为AB和ab都被放在了最后，而他们却是以C和c开始的





并且，这里有一些比较新颖的win系统函数，我们可以通过查阅官方文档来了解这些函数的定义

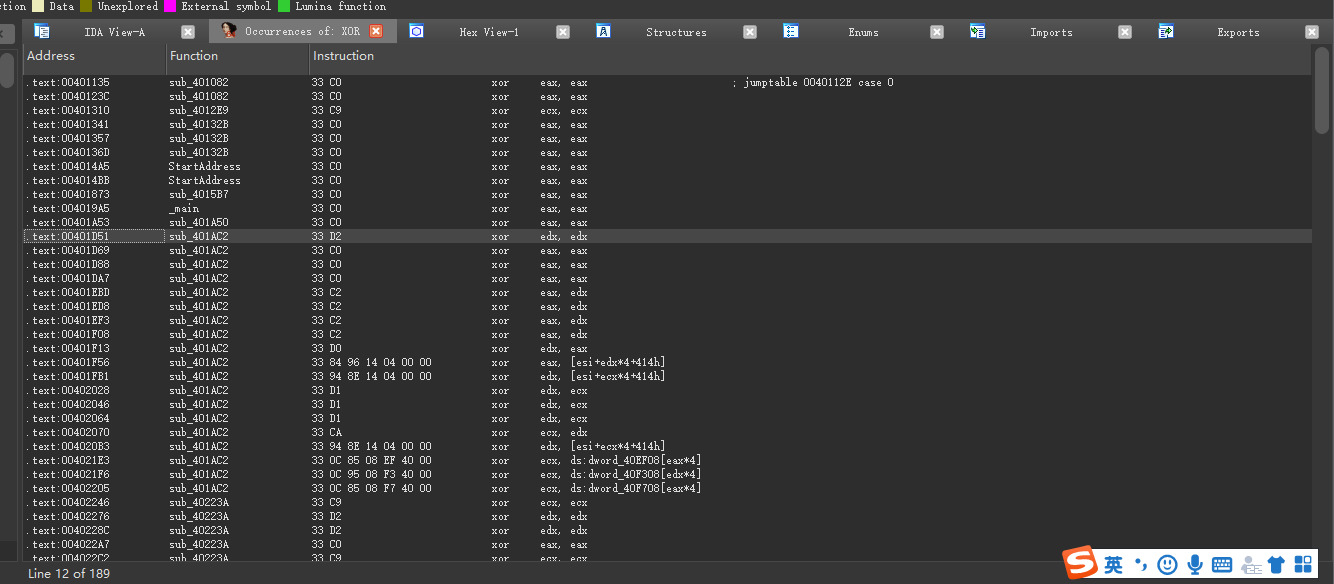
WriteConsole：从当前光标位置开始，将字符串写入控制台屏幕缓冲区

ReadConsole：从控制台输入缓冲区读取字符输入，并将其从缓冲区中删除。

DuplicateHandle：复制对象句柄



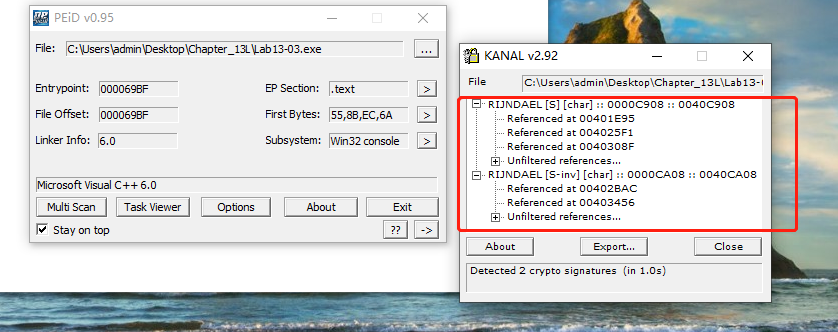
然后，我们安装管理对exe中的xor指令进行搜索，但这次搜索出的结果数量却异常庞大，所以我们必须对他们进行仔细的筛选，筛选出对理解程序可能有帮助的指令所在的函数进行重点分析：



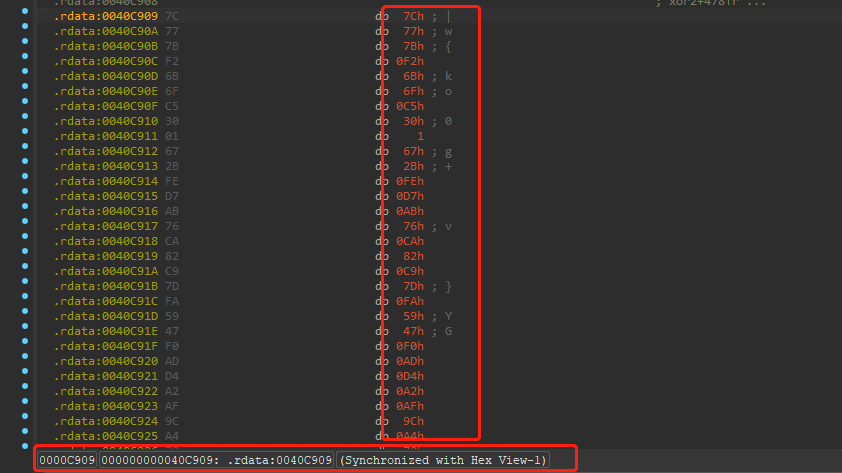
最后我们根据简单的挑选挑选出了六个比较有分析价值的函数并对他们进行重新命名以便于后续分析：



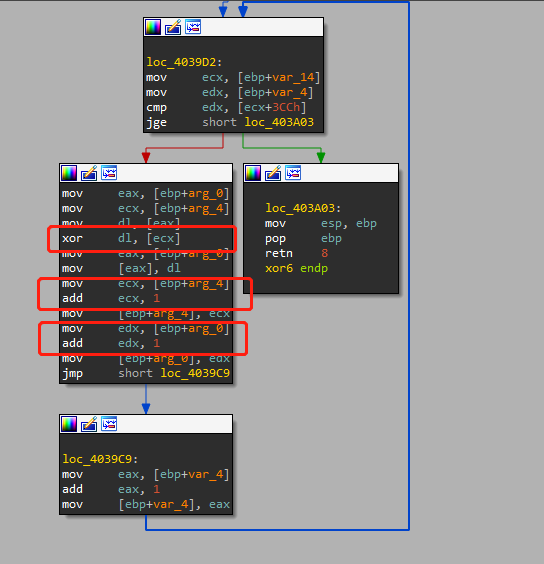
其次，我们很简单的可以观察到lab13-03.exe的函数非常对，程序结构非常复杂，于是我们使用peid的插件来查看程序是否使用了现代的加密算法，简单检查后发现果然使用了AES算法



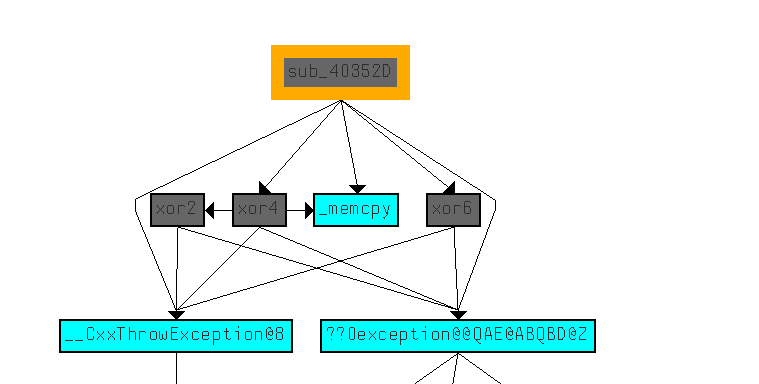
然后我们使用IDA熵插件来显示熵比较高的位置，我们可以观察到在数据段0x0040C900开始的位置和AES中使用的S-box区域是相同的，也就是说通过不同的插件确认，我们可以确定该恶意代码使用了AES进行加密



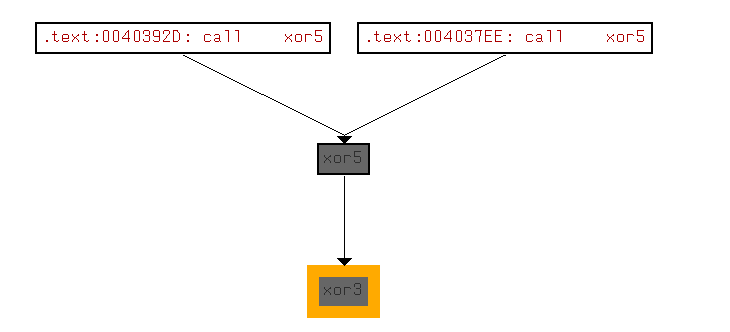
经过对源代码的分析，我们可以得出一个结论就是关于xor相关的函数使用：其中我们标注的xor2和xor4函数是AES的加密相关函数，而xor3和xor6函数是AES的解密函数。下图就是xor6的xor指令相关的循环处理函数，其中第一个参数是一个指针，他指向了进行转换的原缓冲区，第二个参数也是一个指针，他指向了异或原数据的缓冲区，也就是该和谁异或。



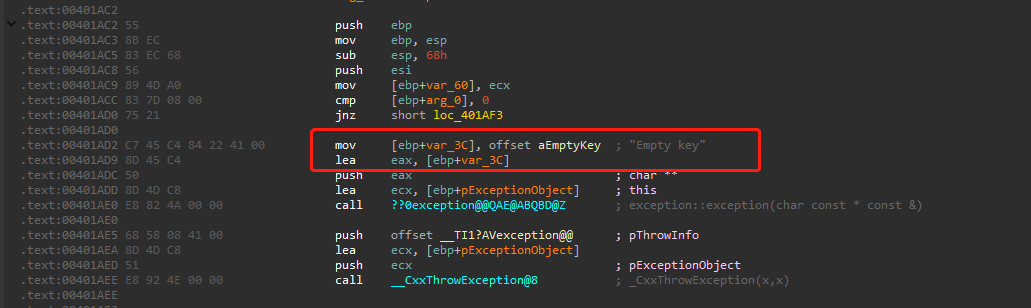
为了判断这几个函数的像话调用关系，我们查看这些函数的交叉引用图，虽然我们此时已经判定了xor3和xor5与解密AES有关，但是和这三个函数的关系似乎并不明朗，而当我们着眼于xor5的时候发现他被两个函数所调用，但是这两个位置似乎没有被识别为函数，所以说当AES代码链接到恶意代码时没有使用解密。

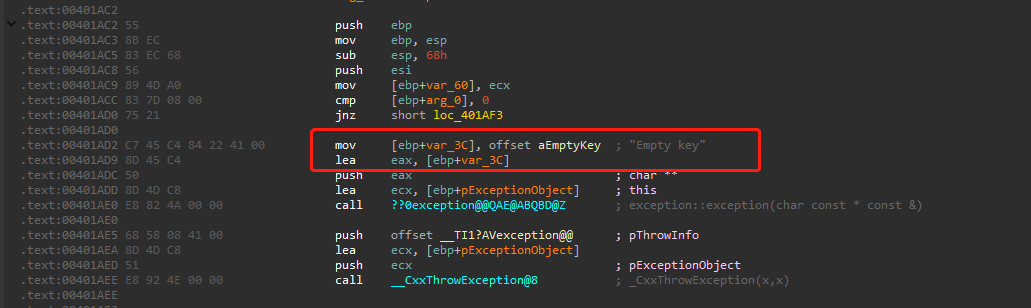


同样，我们对xor5生成指定的交叉引用图就可以看到其和xor3的关系：

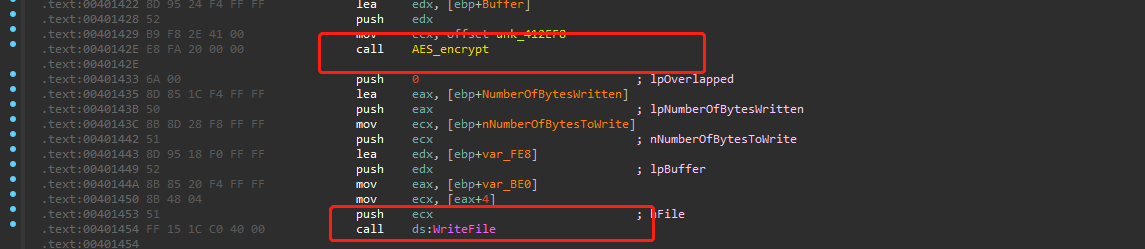


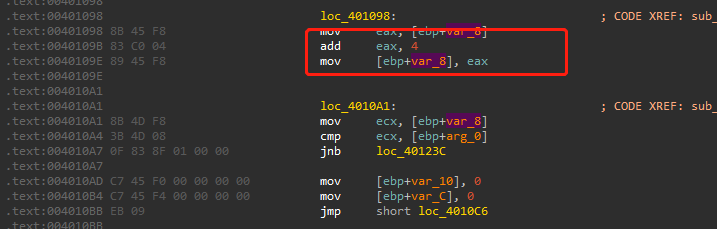
此时我们将关注点聚焦到xor1上来，他并不直接服务于AES算法，但是他会处理好和加密相关信息的初始化并做出判断，包括如果密钥为空或者密钥的长度不正确都会被识别。而xor1和其他函数的关系可以查看xor1的调用函数，在xor1被调用之前412EF8首先被调用，他将一个偏移量给予了xor1，而412EF8在加密之前被载入到ECX，所以我们不难判断他其实是一个C++对象，或者说是一个AES的加密器，而xor1接受了这个参数arg0，如果完成判断则空密钥的提升被发出，所以说arg0一定是一个密钥，而在main中xor1的参数在0x401895被设置，这个字符串将被用于加密。



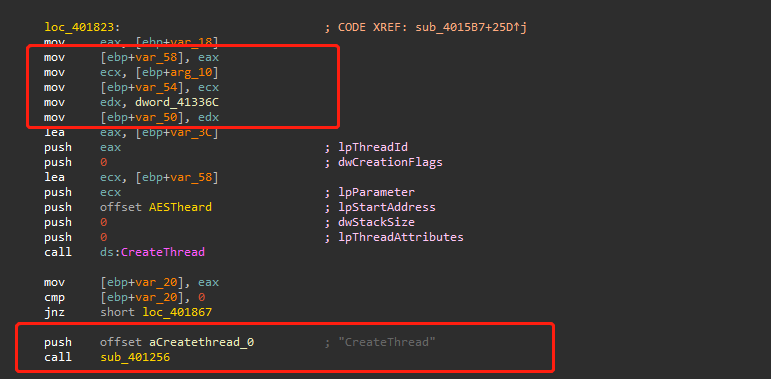


然后，接下来我们就需要明确这个AES代码在程序中做了一件什么事：0040132B的位置调用了加密函数，而这个加密函数发生在读文件前，在加密后完成了写函数。xor1只在启动的时候被调用一次，他设置了密钥。并且Base64页参与了加密，检查对编码表的引用，这个字符串在0x0040103F函数中，函数索引编码表并且将解密后的字符串分成32bits的块，并且自定义了一个解码函数，同时也在读文件和写文件之间调用了它

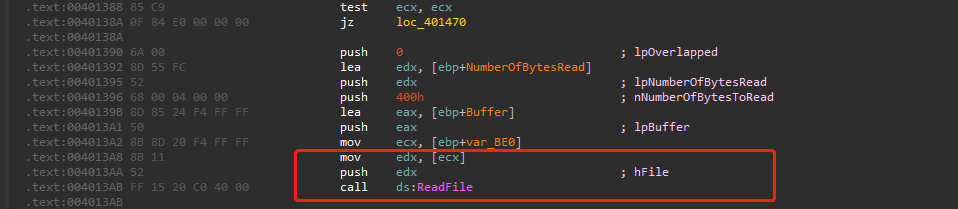


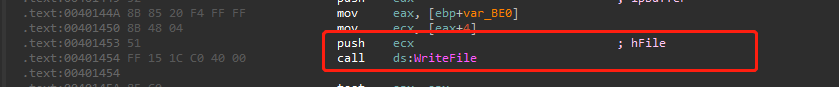


接下来，我们需要了解加密与解密之间的联系，在这时我们注意到了一个新的线程，他的位置在0x0040132B，我们可以将他命名为AESTheard我们先后可以看到三个参数的压栈，分别是var58 arg10和var50，而他们值的来源分别是var18、arg10和dword41336C

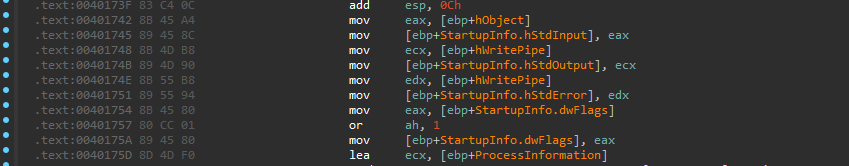


接下来我们会看到这些参数的使用：在ReadFile之前压入栈中的值hfile其实就是var58压入的值，而在WriteFile之前压入栈中的值就是arg10和var54





跟踪句柄值，找到生成它的地方，首先我们发现var 58和var 18持有一个管道的句柄，早些时候，这个管道在函数0x0040132B中创建,并且它与一个shell命令的输出相连接。命令hSourceHandle被复制到shell命令的标准输出和标准错误，这条shell命令由0x0040177B处的CreateProcess命令启动



解密算法：具体的解密算法可以参考书后给出的代码即可对其加密的内容进行解密，

具体的Base64算法为：

import string
  
import base64
  
S=""
  
tab="CDEFGHIJKLMNOPORSTUVWXYZABcdefghiklmnopqrstuvwxyzab0123456789+/"
  
b64='ABCDEFGHIJKLMNOPORSTUVWXYZabcdefghiklmnopqrstuvwxyz0123456789+/
  
ciphertext= 'BInaEi=='
  
for ch in ciphertext:if (ch in tab):s+= b64[string.find(tab,str(ch))]elif(ch==='):S+='-l
  
print base64.decodestring(s)

具体的AES的解密算法为：

