**恶意代码分析与防治技术实验报告**

**Lab13**

**学号：6016252 姓名：马世骐 专业：**

**一、实验环境**

Windows7(VMWARE虚拟机)，Windows11本机

**二、实验工具**

STRINGS, IDAPro，PEVIEW，EXEInfo，YARA，IDA Python，Netcat，Wireshark，Procmon,WinDbg，resource\_hacker，winhex， FindCrypt2、Krypto ANALyzer（KANAL）

**三、实验内容**

**LAB13-1**

**问题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？ |

**答： 动态分析出的网络行为中出现了两个strings工具未检测出的字符串，分别是eHBsaQ==和 www.practicalmalwareanalysis.com，可能被加密了。**

分析过程：

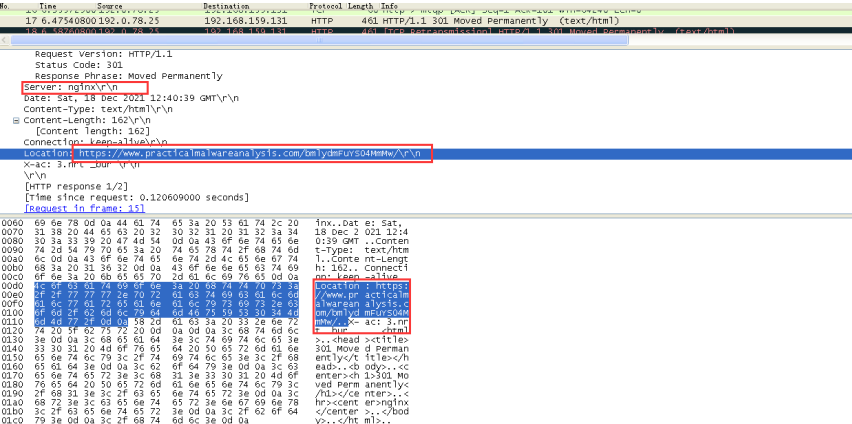
首先使用strings简单查看一下这个程序





可以其中有一部分的字符串枚举了所有的大小写字母和数字，以及+/，很明显这是一个base64编码的内 容那么猜测这个恶意代码会存在有base64加密的内容。之后可以看见有一部分看似是乱码的内容： KIZXORXZWVZWLZI^ZUZWBHRH ，结合刚刚猜测的会存在有base64编码，想来这里应该就是加密过后的 内容。 同时我们还可以发现里面有 http:// 的字样，而且这里是一个格式化字符串，上面还有一个 Mizilla/4.0 ，暂时还不知道是什么作用。

之后我们实际运行一下这个程序，由于之前分析出来可能会有网络行为，所以这里使用wireshark抓包分析一下



可以发现有一个HTTP请求，并且这个请求是一个GET请求，请求的网址我们之前在string中并没有看 见，想来可能和加密的内容有关。与找到的字符串进行对比，相同的有Mozilla/4.0，却没有看到eHBsaQ==和 www.practicalmalwareanalysis.com，推测这些字符串被加密了。

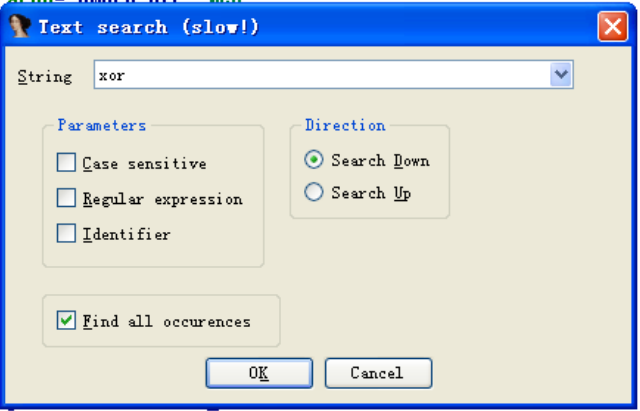
**问题2**

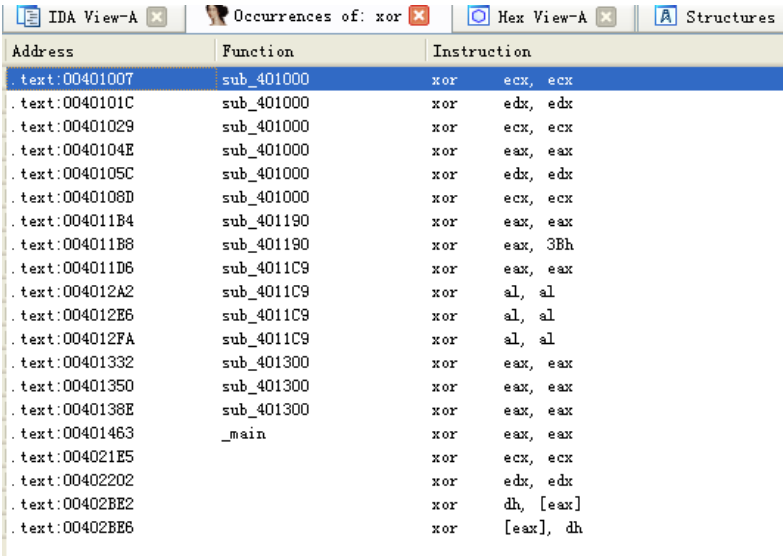
|  |
| --- |
| 使用IDA Pro 搜索恶意代码中字符串'xor'，以此来查找潜在的加密，你发现了哪些加密类型？ |

**答：只找到了一处，004011B8处的xor eax,38h是一个 XOR 循环加密的指令。**

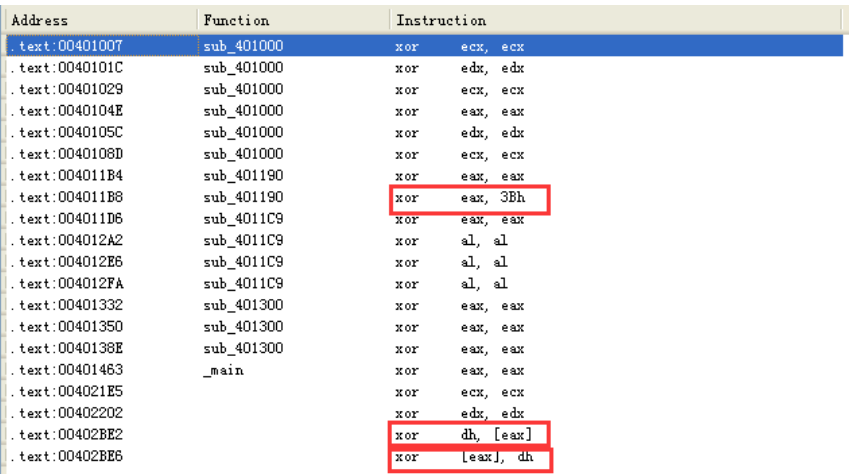
分析过程：

使用IDA打开程序，并进行搜索



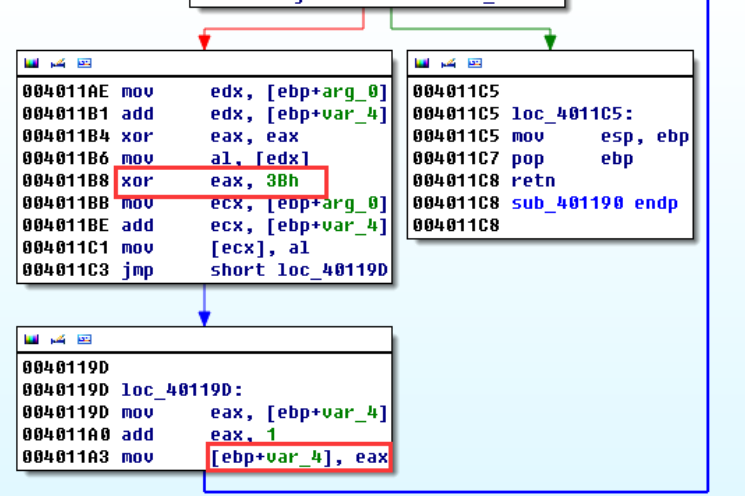


可以发现有非常多的地方都出现了xor，但是大部分都是自身进行异或，也就是清空，那么这些对于加密是没有任何作用的，所以这里忽略这些，之后我们发现还剩下3个地方需要注意



发现在第一处他异或的是3Bh，猜测这里可能是使用了异或进行加密，可能密钥就是3Bh

双击点过去可以看见：



这里就是一个循环进行异或的操作，它递增var\_4，并且用0x3B与edx 中的内容进行异或，edx中存的是缓冲区[ebp+arg\_0]偏移[ebp+var\_4]的值，每次递增的是var\_4中的内容（在这里就是长度），异或的内容是 arg\_0，那么这里很明显就是一个异或加密的操作了，密钥是3Bh，所以这个sub\_401190函数就是进行异或加密的函数

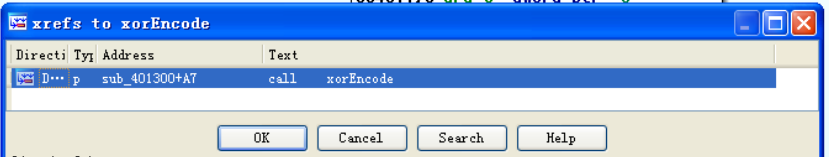
**问题3**

|  |
| --- |
| 恶意代码使用什么密钥加密，加密了什么内容？ |

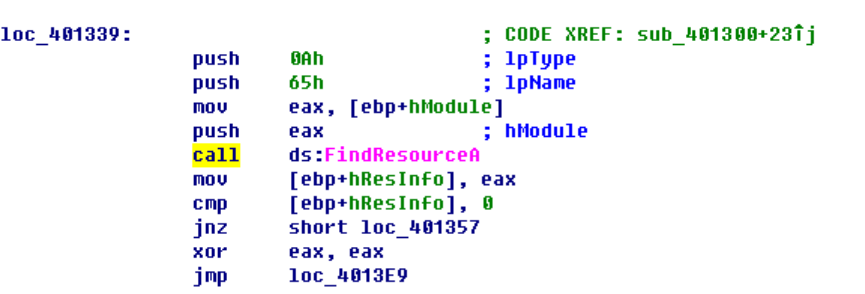
**答：密钥是0x3B，加密了域名www.practicalmalwareanalysis.com。**

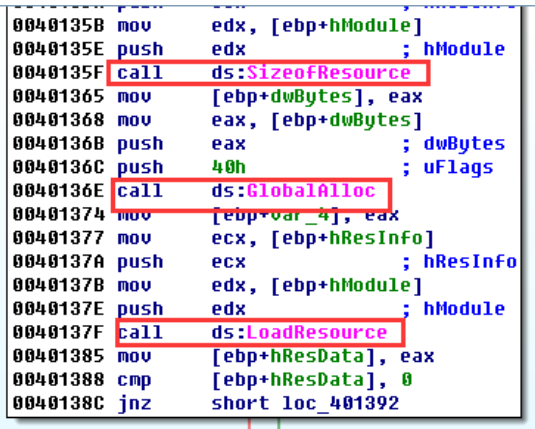
分析过程：

根据刚刚的分析可以知道这个异或加密的密钥是3Bh

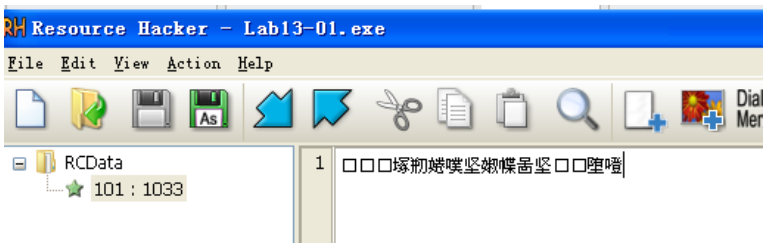


通过查找交叉引用，发现该加密函数仅被函数sub\_401300调用。于是查看函数sub\_401300。可以看到在调用 加密函数之前有一系列对资源节的操作：依次调用了GetModuleHandleA、FindResourceA、 SizeofResource、GlobalAlloc、LoadResource和LockResource，通过FindResource的参数来寻找操作的 对象：其中lpType=0xA，表示资源数据原始数据，lpName=65h，在这里它是一个索引号，表示引用ID为0x65 的资源。





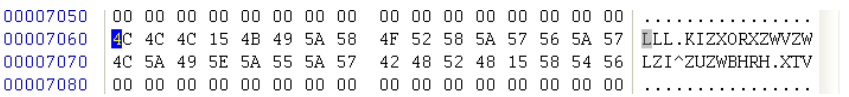
可以看见在异或操作之前，这里释放了资源节的内容，那么很明显这个异或操作就是对资源节中的内容进行解密。



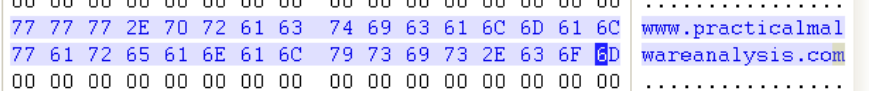
于是我们使用PEview查看恶意代码的资源节，查找索引号为65h的资源，偏移为0x7060，然后用WinHex打开， 定位到该资源节处，选择Edit->Modify Data->XOR，输入3B，得到异或结果，为域名 www.practicalmalwareanalysis.com。



同时通过resource\_hacker可以看见资源节里是一串乱码，更加印证了刚刚我们猜测他是对资源节中的内容 进行解密了



使用winhex工具定位到刚刚的位置，可以看见刚刚的乱码，对其进行异或操作得到：



可以看见这里就是刚刚wireshark中抓到的那个url

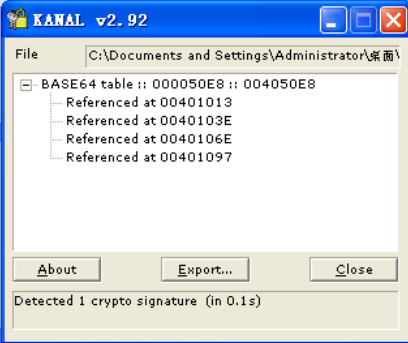
**问题4**

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

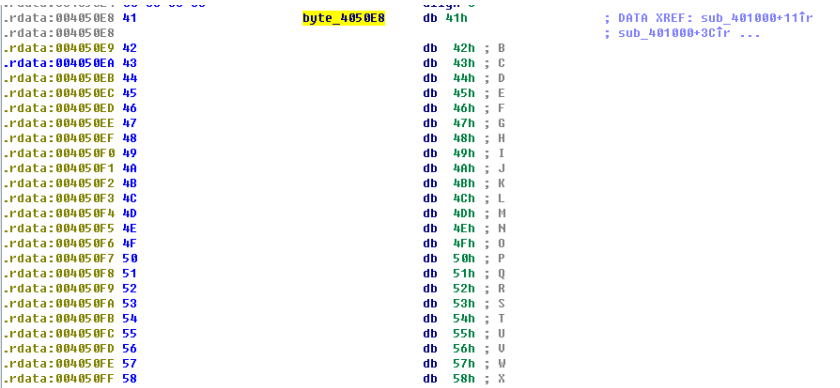
**答： 用PEiD的KANAL插件和IDA的熵插件，发现了恶意代码使用的Base64编码字符串: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/**

分析过程：

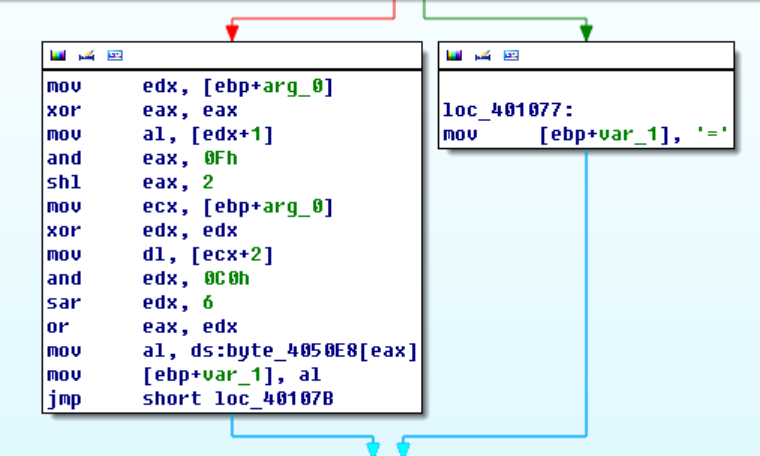
使用PEiD的Krypo ANALyzer插件，发现该恶意代码有一个BASE64 table，地址是0x004050E8，于是我们在 IDA中跳转到该位置，看到这里存着的是字符串 ABCDEFGHIJKLMNOPQRSTUVXYZabcdefghijklmnopqrstuvwxyz0123456789+/。



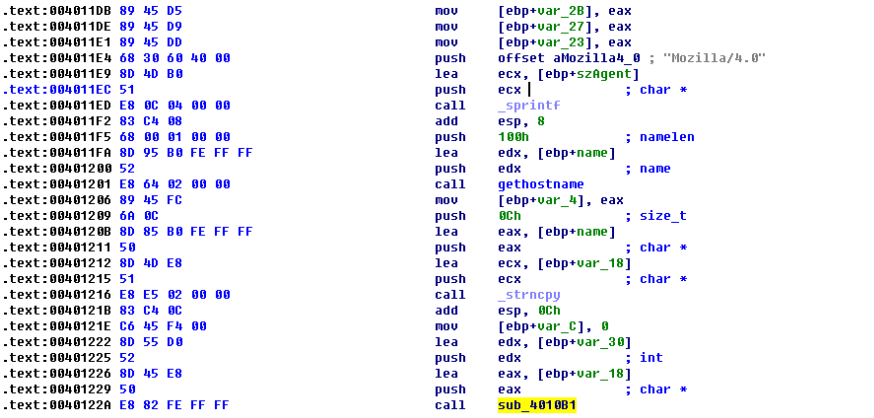
在如图中所示的几个位置还存在有Base64加密



这个字符串在函数sub\_401000中被引用了四次，注意到该函数中有一处对=的引用，这是Base64加密中的填充 字符，因此该函数执行加密操作，调用该函数的函数是Base64加密函数。可以看到，sub\_4010B1调用了该函 数，于是进入这个函数查看。调用了strlen函数得到参数中字符串的长度，将该字符串以3字节为单位分组， 分别传给sub\_401000进行加密操作，得到该分组的4字节密文。



函数sub\_4011C9调用了这个加密函数，查看其参数，源字符串是strncpy的返回值，而strncpy的输入是函数 gethosename的输出，即要加密的字符串是主机名的前12个字节。而其目的字符串作为sprintf的参数，即加 密后的字符串被输出，也就是我们在动态分析阶段看到的那样。



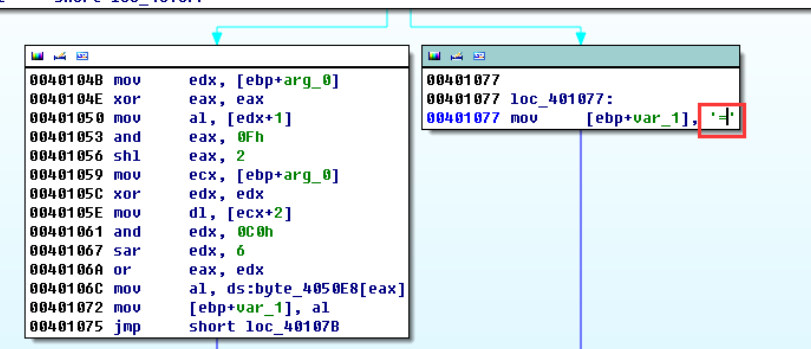
**问题5**

|  |
| --- |
| 什么类型的加密被恶意代码用来发送部分网络流量？ |

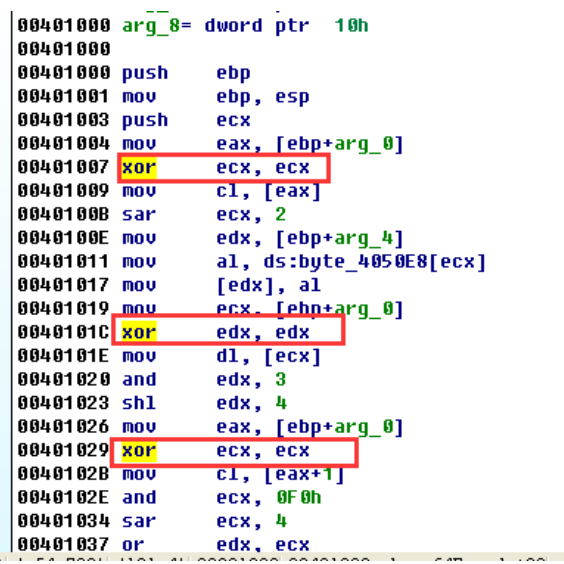
**答： 使用Base64加密来构造GET请求字符串。**

分析过程：

通过刚刚我们分析的可以知道，url是使用的xor进行加密，还有一个就是GET请求我们没有找到，那么 GET请求想来就是使用的这个base64进行加密



进入到其交叉引用的地方可以看见这里出现了=，也就是标准base64编码使用的填充符号



同时在上面我们可以注意到这里一共进行了三次的清空寄存器和存放的操作，很明显这里就是 对"GET"字样进行一个解密了

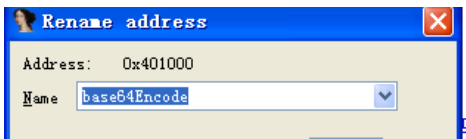
**问题6**

|  |
| --- |
| Base64 编码函数在反汇编的何处？ |

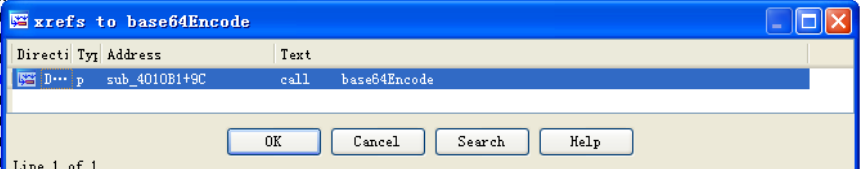
**答： 从0x004010B1处开始。**

分析过程：

通过交叉引用可以发现刚刚标准的base64编码在401000位置被引用



我们将其改名为base64Encode，再次通过交叉应用可以发现



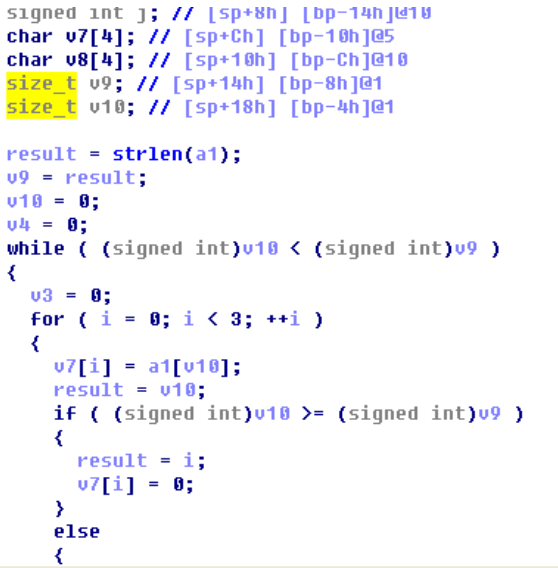
**问题7**

|  |
| --- |
| 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？ |

**答： 12字节，加密了主机名，这使得用于构造GET请求的字符串不超过16字节。**

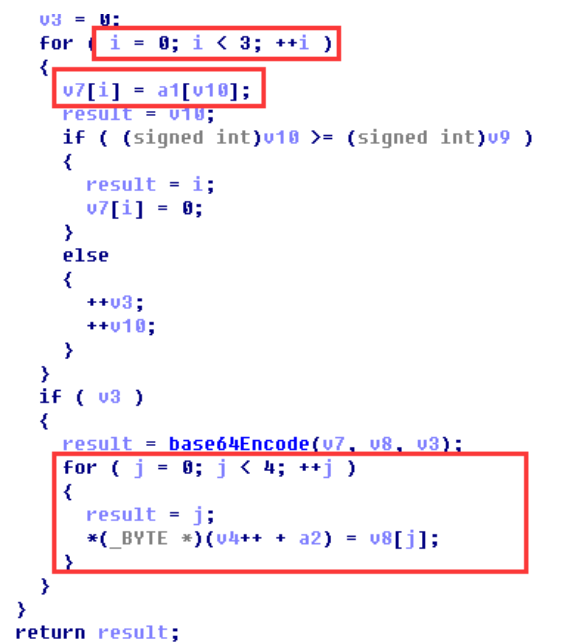
分析过程：

使用F5键进入到C语言代码的形式



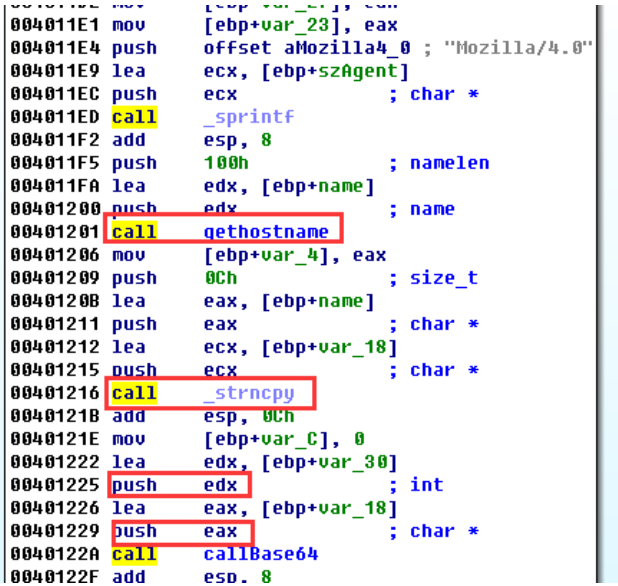
可以看见这个循环的判定条件是v10和v9的大小，并且v10的初始值是0，在之后的循环体内部v10会执行一个自增的操作，那么这个v10其实也就是相当于一个下角标的作用，来控制循环次数。并且v9字符串的初始值设置的是 strlen(a1) ，也就是a1字符串的长度，那么也就是说这里循环就是遍历了整个字符串。

查看循环体内部

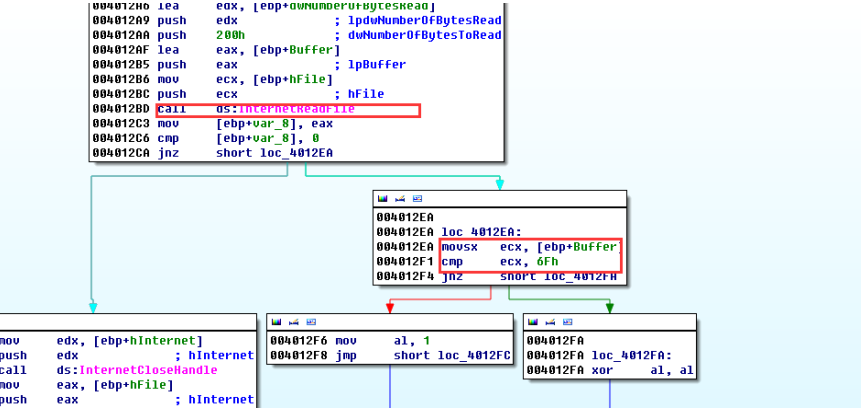


可以看见循环体内部就是每次取出来三个字符，然后利用base64进行解密的操作

回到刚刚的函数体，查看一下交叉引用



可以看见传递给这个函数的参数有两个，其中有一个来自上面的strncpy，这个函数的内容来自上面的 gethostname的前12个字节。那么也就是说这里给到的是前12个字符。



之后可以看见他获取了网络上的资源之后并对第一个字符进行比较，如果第一个字符是o则返回1，否则返回0

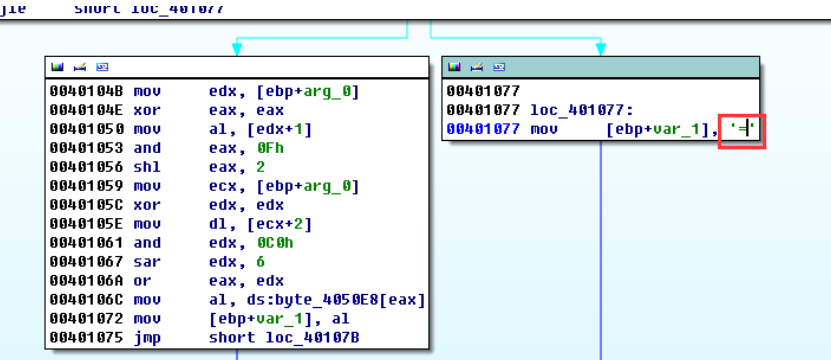
**问题8**

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

**答： 本次得到的加密数据是eHBsaQ==，有填充字符，这是因为本机的主机名小于12字节并且不能被3整除。**

分析过程：

通过问题5中的分析我们可以看见



这里确实是使用了=进行填充，但是这里其实是当长度不是3的倍数或者是小于12个字符的时候才会填 充，而这里因为之前取的就是12，所以虽然有填充在这里的，但是在本段恶意代码中不会发生。

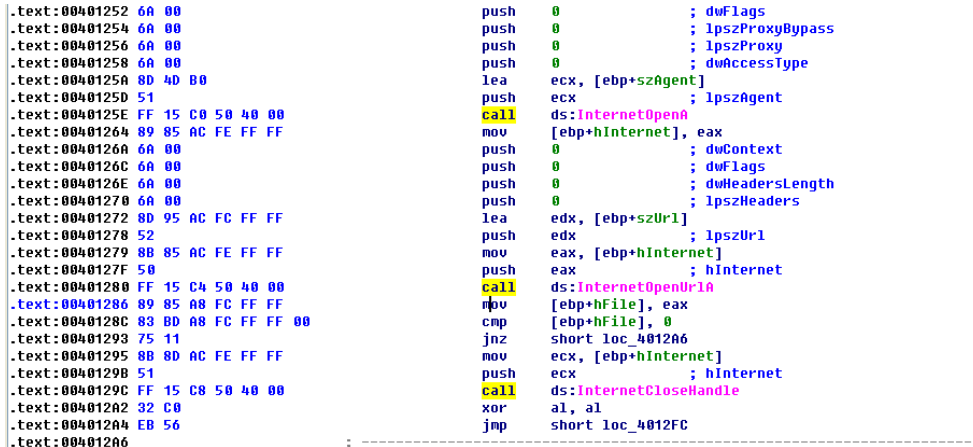
**问题9**

|  |
| --- |
| 这个恶意代码做了什么？ |

**答： 这个恶意代码用经Base64加密的主机名构造出一个GET发出请求，当接收到以o开始的Web响应后才会终止。**

分析过程：

依次调用了InternetOpenA、InternetOpenUrlA和InternetReadFile，用于打开并且读取之前构成的 URL。并将读到的数据中的第一个字符与字母o比较，如果相同则返回1，否则返回0。



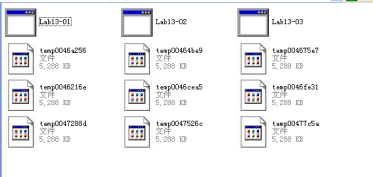
**LAB13-2**

**问题1**

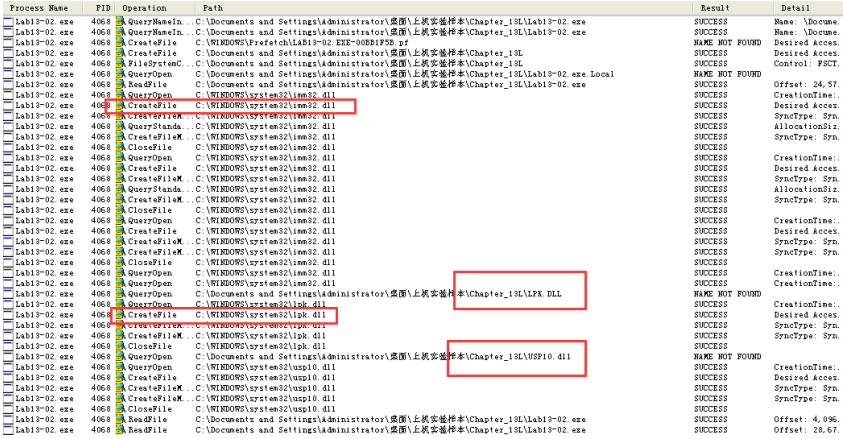
|  |
| --- |
| 用动态分析，确定恶意代码创建了什么？ |

**答：运行恶意代码，可以看到其打开一个命令行窗口，每隔几秒就会在当前目录下创建一个新文件，这些文件的大 小相同，但名字不同（看起来像随机数）。用promon可以监测到它确实有大量创建文件、写文件的操作。**

分析过程：



使用procmon工具进行监控



可以发现确实多出来了一些文件，可以发现这些文件的特点就是以temp开头，而且他们的大小都是 6654KB

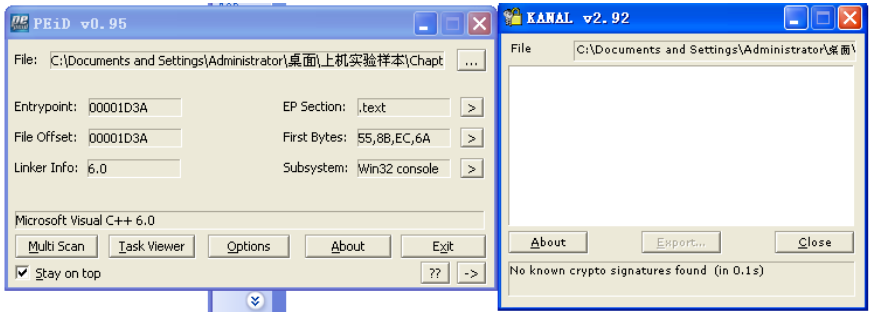
**问题2**

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

**答：多个xor指令的函数会调用一个4012DD函数，然后这个函数会调用40128D**

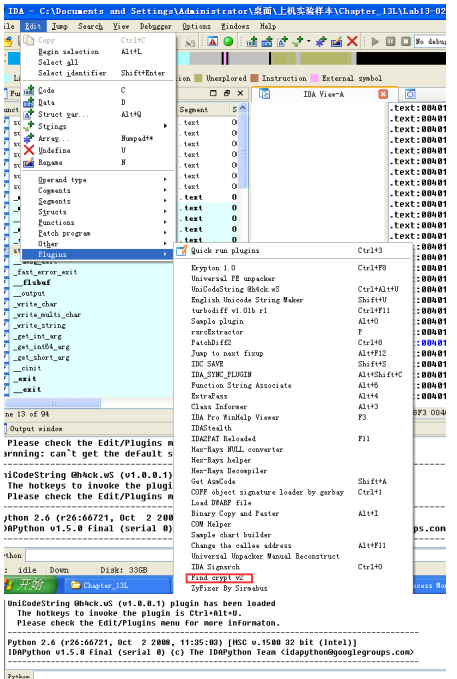
分析过程：

首先使用KANAL查看



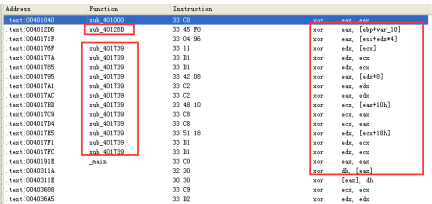
发现PEID中的插件并没有发现什么内容

之后使用IDA中的FindCrype插件查看



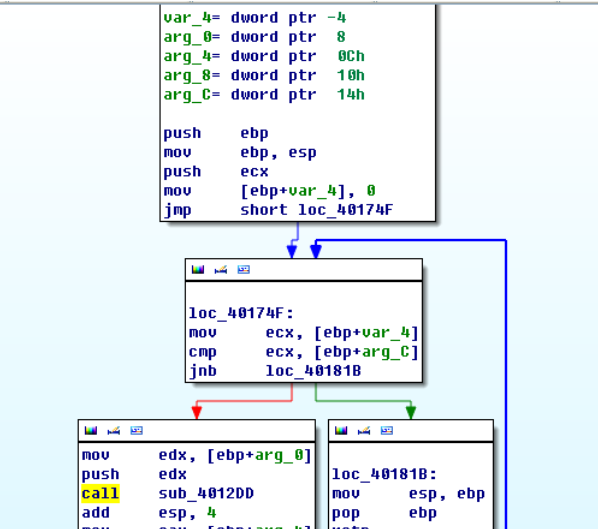
输出框依旧没有什么内容，也就是说这里没有找到东西

接下来搜索xor指令

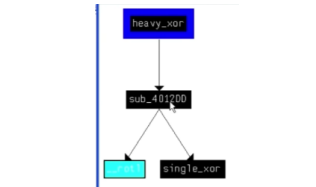


可以看见有很多条的xor指令，并且其中大部分来着sub\_401739，还有一个条来自sub\_40128D

忽略其中用于将寄存器清零的和库函数中的操作，其余的xor指令集中在函数sub\_40128D和sub\_401739中，并 且搜索结果中的第三条指令并没有对应的函数。 我们先查看拥有最多xor指令的函数sub\_401739，它包含一个循环，循环体中大多是xor、SHL和SHR指令，只有一次函数调用，调用的是sub\_4012DD，而这个函数又会调用sub\_40128D，即拥有xor指令的另一个函数， 推测这两处xor指令相关。



通过查看交叉引用图可以发现



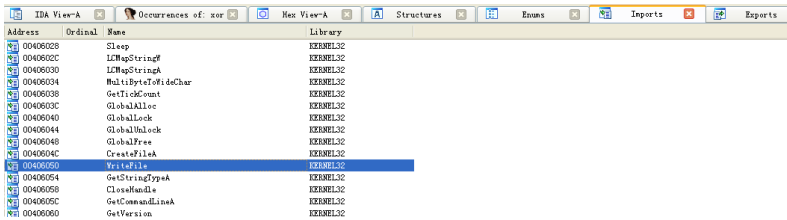
刚刚我们发现的使用多个xor指令的函数会调用一个4012DD函数，然后这个函数会调用40128D

**问题3**

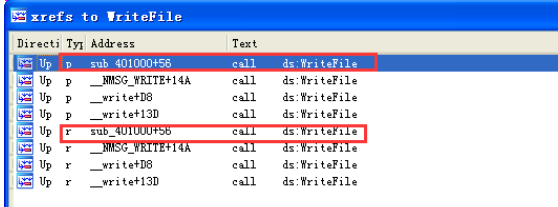
|  |
| --- |
| 基于问题1的回答，哪些导入函数将是寻找加密函数比较好的一个证据？ |

**答： 查看导入函数表，有WriteFile函数，由于该恶意代码的行为是不断创建文件并写文件，因此可能会在 WriteFile函数附近寻找到加密函数。**

分析过程：



问题1中的特征我们发现是创建了很多具有一定特征的文件，而创建文件要使用的就是WriteFile函数， 接下来我们针对这个函数的调用情况进行分析



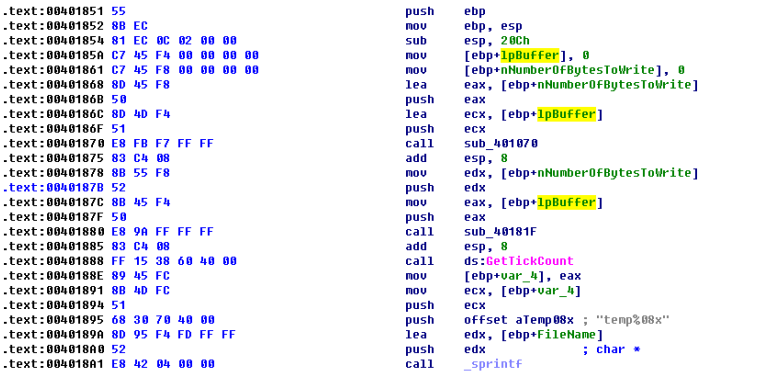
可以看见有两个地方被调用。那么由此也就可以认为WriteFile函数就是我们需要关注的函数

**问题4**

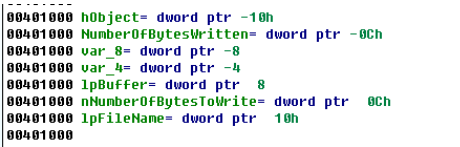
|  |
| --- |
| 加密函数在反汇编的何处？ |

**答：0x0040181F处。**

分析过程：

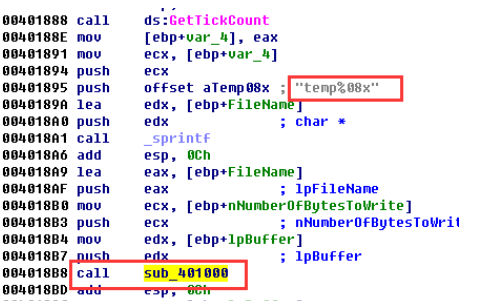


根据刚刚对WriteFile函数的分析我们可以知道主要是sub\_401000函数调用了这个函数，查看一下这个 函数

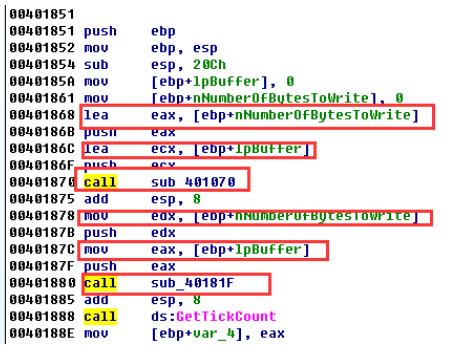


可以看见这个函数的参数有如上几个，并且通过参数的名称就可以看出有一个参数是要写入的byte数 量，一个是文件名，还有一个是缓冲区的指针。

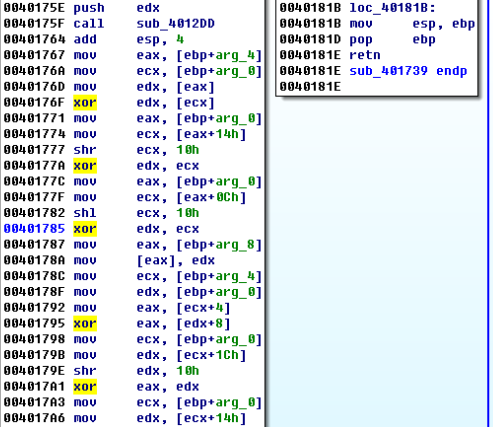
找到调用这个函数的位置



看见出现了刚刚创建的文件的文件名的一部分，同时我们注意到，在上面有一个函数是 GetTickCount ，也就是获取系统启动了的时间，那么猜测这里的文件名就是创建这样文件名的一个文件。 再往上可以发现有：



在上面先调用了另外两个函数，并且两个函数的参数都是指针和缓冲区的大小，那么根据逻辑顺序猜测第一个函数应该是获取文件，第二个函数是对文件进行加密或者解密的操作。 进入到第二个函数发现他就是刚刚调用xor函数的函数，并且密钥为10h



那么这个加密函数就在反汇编中的sub\_401739

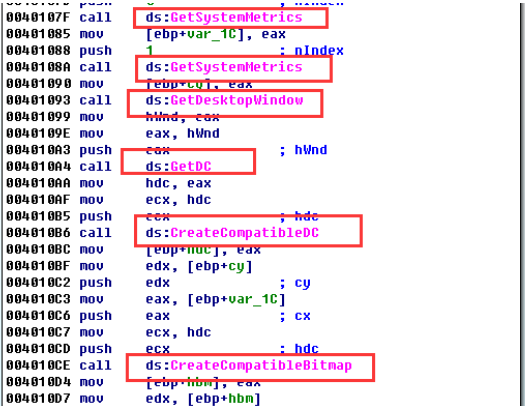
**问题5**

|  |
| --- |
| 从加密函数追溯原始的加密内容，原始加密内容是什么？ |

**答： 当前的屏幕截图。**

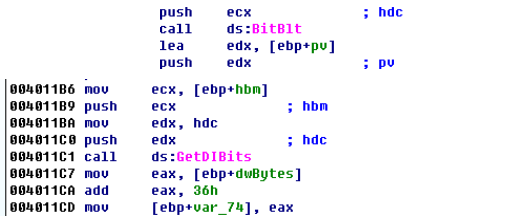
分析过程：

根据刚刚的分析可以知道，两个函数中第二个函数是用来加密的，那么第一个函数就是获取要加密的文件的内容，对这个函数进行分析



发现这个函数调用了一串的系统函数，其中有一个GetDesktopWindow是用来获取桌面窗口的距离

之后还有



这两个函数用来获取位图的信息，并放到缓冲区。

经过网上资料的查询可以知道，这几个函数连用的作用就是获取用户桌面的信息，那么综合以上内容就知道其实这里获取的是用户桌面的内容，之后进行加密。

**问题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？ |

**答： 查看用于加密操作的代码块，发现使用的是用户自定义的算法，而不是什么标准的加密算法。可以通过 Ollydbg来解密。**

分析过程：

根据刚刚的分析，我们认为这个加密算法使用的就是一个简单的异或加密，对于异或加密操作来说，解密和加密是使用的同一套流程，所以解密的时候同样是使用10h进行异或操作即可。 或者是在恶意代码获取了缓冲区的内容后，将缓冲区里的内容进行修改，改成以前得到的加密文件，然后让他再次执行xor的操作，就能够达到解密的效果

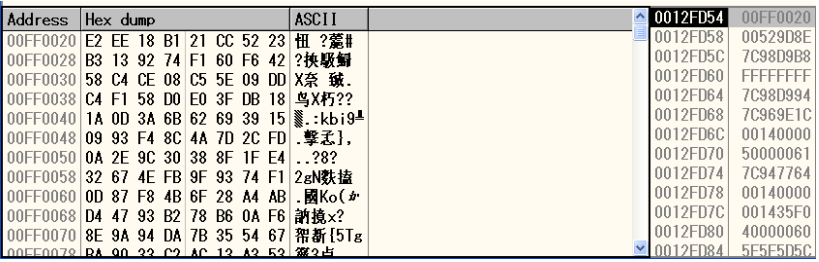
**问题7**

|  |
| --- |
| 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？ |

**答： 能**

分析过程：

使用OllyDbg加载该程序，在加密之前先设置一个断点，并且保证在这时缓冲区Buffer中已经保存了要加密的屏幕截图信息，因此可以选择0x00401880作为断点，运行，命中断点后，此时堆栈上的参数表示的就是加密缓冲区Buffer和长度，查看堆栈中的值，如下：

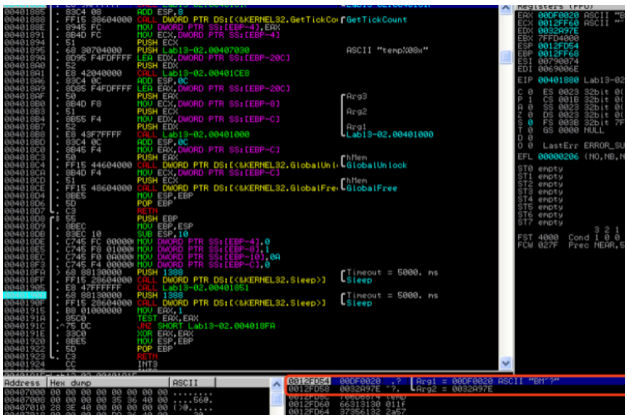


然后运行WinHex，打开一个恶意代码创建的加密文件，选择Edit→CopyAll→Hex Values，并在OllyDbg中将复制的内容 粘贴到缓冲区中。 设置第二个断点为0x0040190A处，即在第一个文件写入后，继续运行程序，命中断点，在当前目录中找与先前 创建文件具有相同命名约定的新文件，修改扩展名为.bmp，就恢复出了屏幕截图。（在这里我们假设加密和解密用的是同样的函数）

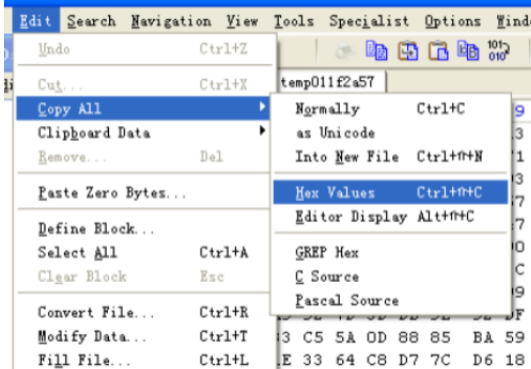
使用immunity工具进行中间内容的修改，首先载入程序，并在加密之前下一个断点（在ida中可以看见 函数的位置为401880），然后在写入文件之后的位置再下一个断点（位置为：401905）。

想法是在到达他加密的位置时，将里面的内容替换为之前已经加密好的一个文件，然后执行加密操作 （在这里就变成了解密），之后写入将解密的内容写入到新文件中去，查看新文件即可。

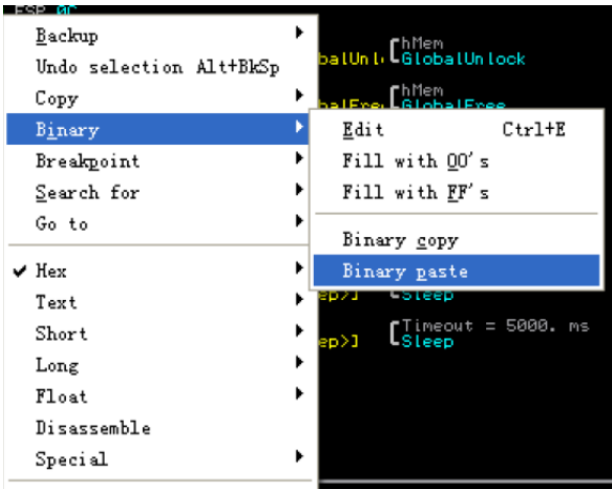
运行到第一个断点



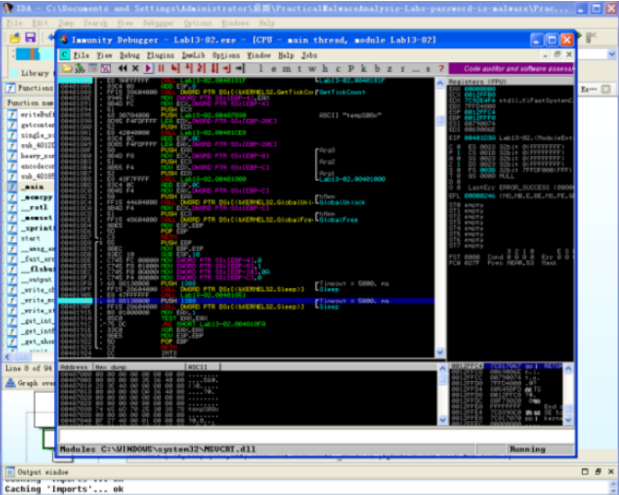
可以看见右下角这里就是即将加密的缓冲区和缓冲区的长度，在左边定位到缓冲区里的内容，然后使用 winhex将我们要解密的文件以16进制的方式进行复制



再将缓冲区的内容全选以后进行替换



之后继续运行，程序运行结束之后得到一个新的文件。将这个文件的后缀改为.bmp之后可以双击打开



发现就是刚刚我们电脑状态的截图，解密成功

**LAB13-3**

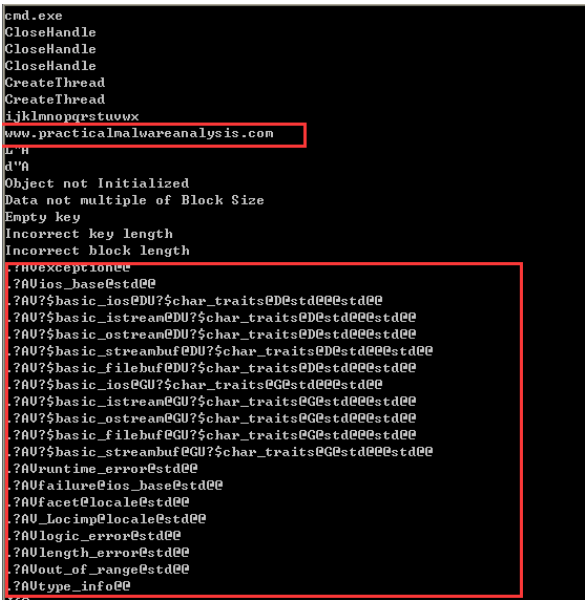
**问题1**

|  |
| --- |
| 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密？ |

**答：在执行动态分析时看到一些随机内容，推测其被加密了。而在输出字符串中找不到什么可能被加密的内容。**

分析过程：

首先使用strings工具简单查看一下有哪些字符串

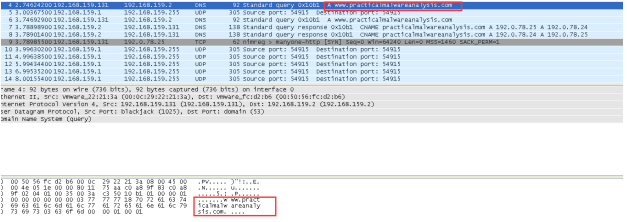


发现在上面出现了一个url，在下面出现了有点像是乱码的字符串，但是好像又不是乱码



同时还可以注意到这里有一个类似于base64加密的字符串，发现下面的ERROR后面有一个=，还有格式化字符串，猜测会对这里进行一个加密

根据刚刚的分析我们可以认为这个程序有访问网络的行为，所以我们使用wireshark进行分析



可以看见出现了刚刚分析得到的url

进行基础动态分析技术，可以看到恶意代码尝试解析域名www.practicalmalwareanalysis.com，并且向外 连接那个主机的TCP端口8910。如果使用Netcat向这个连接发送内容，可以看到恶意代码用一些随机内容作为 响应，但是看起来都是不可读的乱码（可能被加密了），如果我们从Netcat端终止这个连接，可以看到：

|  |
| --- |
| Go ERROR: API = ReadConsole.  error Code = 0.  message = The operation completed successfully. |

与输出字符串进行对比，看到了域名和看起来像前面提到的出错信息的字符串。

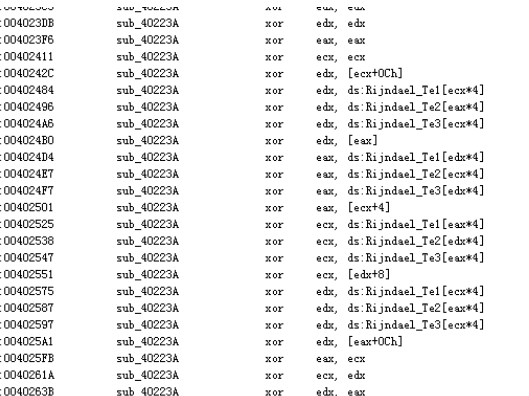
**问题2**

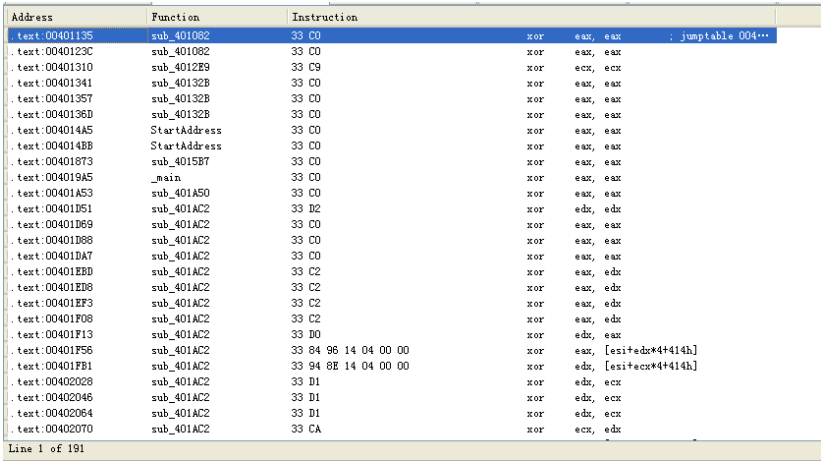
|  |
| --- |
| 使用静态分析搜索字符串xor来查找潜在的加密。通过这种方法，你发现什么类型的加密？ |

**答：可能是异或**

分析过程：

搜索字符串xor，共得到191条结果，忽略用于清空寄存器的和库函数中的操作，发现了6个包含xor指令的函数，为了便于后续识别，我们将其重命名。而进入这些函数查看xor指令的上下文，并不能识别出是哪一种类型的加密。





有非常多的地方都使用了xor，一共是有6个函数使用了xor，那么这些函数可能存在有加密的行为，对 这6个函数都进行重命名，分别为x1到x6。那么根据刚刚的分析，猜测有6处都有加密的可能，并且使用的加密方式可能是异或

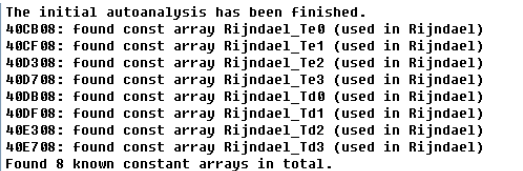
**问题3**

|  |
| --- |
| 使用静态工具，如 FindCrypt2、KANAL 以及IDA熵插件识别一些其他类型的加密机制。发现的结果与搜索字符 XOR结果比较如何？ |

**答：使用这三个插件都识别出了AES算法，它与搜索xor识别出的6个函数相关。而IDA的熵插件还识别出了一个自定 义的 Base64索引字符串。具体见分析过程**

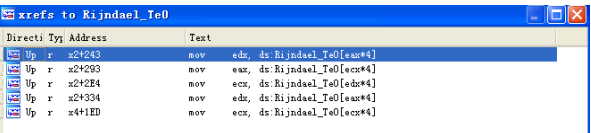
分析过程：

使用IDA的FindCrypt2插件进行查找

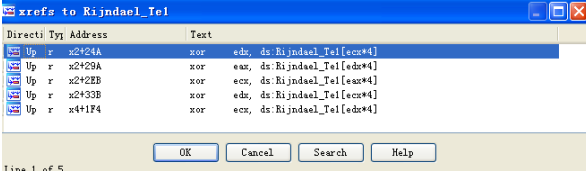


可以发现找到了8处位置使用了加密算法，并且这个标注的Rijndael就是指的AES中的算法 分别查看一下这8个位置的交叉引用

1



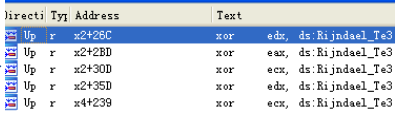
2



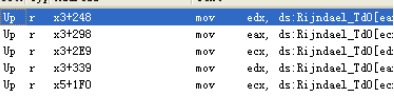
3



4



5



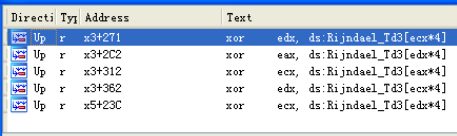
6



7



8



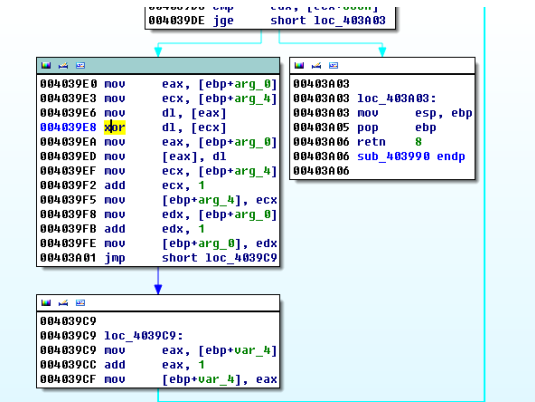
发现这8处一共出现了两种组合：3和5以及2和4，前4个地方使用2和4进行加密；后4个地方使用3和5进行解密

使用PEiD工具的插件可以发现



也是发现了两处位置进行了加解密的操作。那么根据刚刚的内容我们可以知道，3和5是为AES的解密；2 和4是AES的加密

我们注意到x6和x1还没有被引用过，所以接下来先分析x6的作用是什么

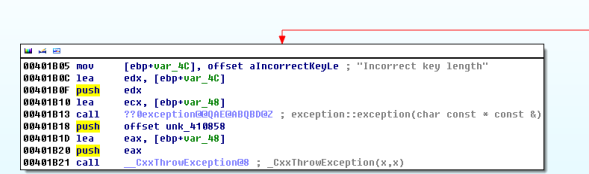


可以发现这里就是一个异或的加密操作，其中在异或的时候有两个参数，一个是arg0，也就是加密前的内容，还有一个是arg4，也就是加密后的内容。

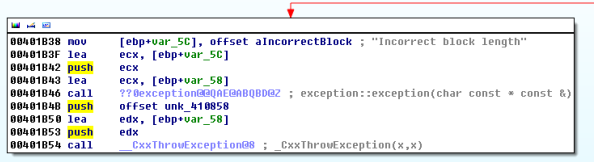
再来看一下x1



发现x1是首先判断一下密钥是否为空，如果是空的就返回一个提示信息

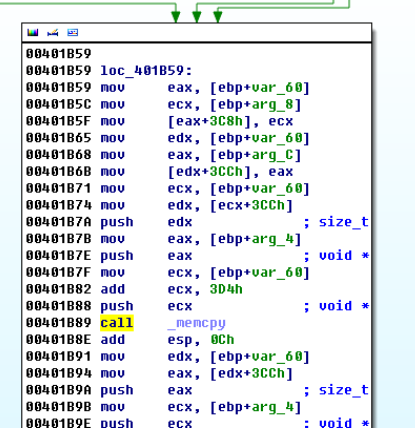


然后检查密钥的长度是否符合要求，不符合的话同样返回一个提示信息

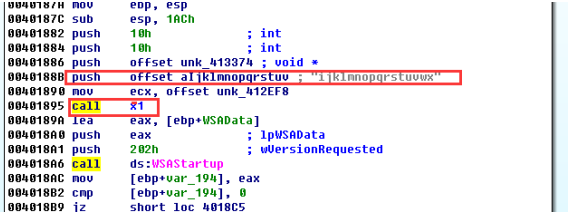


最后检查的是块的长度，不符合的话返回一个提示信息

当这些检查都通过以后会执行后面的操作



查看x1的交叉应用可以发现



这个函数是由main函数直接进行调用的，而且在这个函数被调用之前有一个unk\_412ef8的一个引用； 通过交叉引用可以发现这个还被其他位置进行了调用



可以看见这个偏移也被引入到了x6中，经过分析我们可以得知这个x6是AES的一个启动函数，所以这个其实就是AES要加密的一个对象。而刚刚那个x1对这个对象进行了一系列的检查，那么其实这个x1就是加密器的初始化函数。

**问题4**

|  |
| --- |
| 恶意代码使用哪两种加密技术？ |

**答： AES加密算法和自定义的Base64加密算法。**

分析过程：

根据问题3中的分析可以知道为AES的Rijndeal 算法和自定义的一个Base64加密技术

**问题5**

|  |
| --- |
| 对于每一种加密技术，它们的密钥是什么？ |

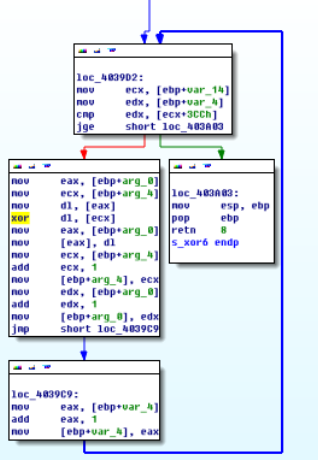
**答： AES的密钥是ijklmnopqrstuvwx**

**自定义的Base64加密的密钥是CDEFGHIJKLMNOPQRSTUVWXYZABcdefghijklmnopqrstuvwxyzab0123456789+/**

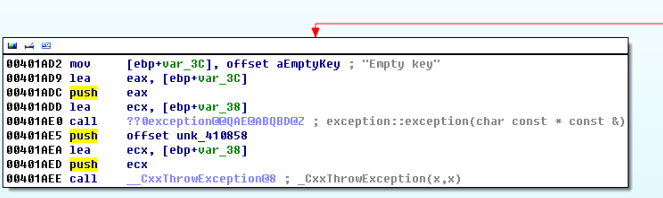
分析过程：

首先查看AES算法，我们已经知道，函数s\_xor2和s\_xor4与AES加密有关，s\_xor3和s\_xor5与AES解密函数有 关。然后查看剩下的两个函数s\_xor1和s\_xor6，寻找其中的关联。

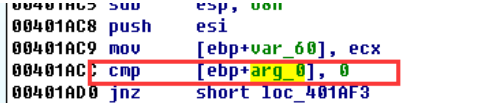
首先查看s\_xor6的代码，可以看到其xor操作在一个循环中，变量arg\_0指向被加密的原缓冲区，arg\_4指向用 来提供异或值的缓冲区。查看该函数的交叉引用，可以看到s\_xor6与AES加密函数s\_xor2和s\_xor4相关。



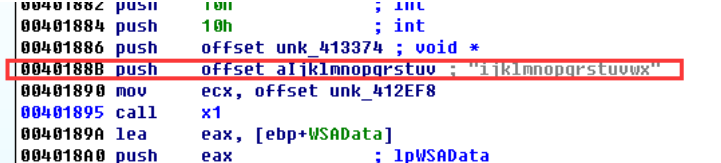
根据之前的分析我们知道了x1就是AES的初始化函数，那么密钥就是在x1中进行了检查



在x1中有一个地方会提示空密钥，而这个提示是基于arg0的

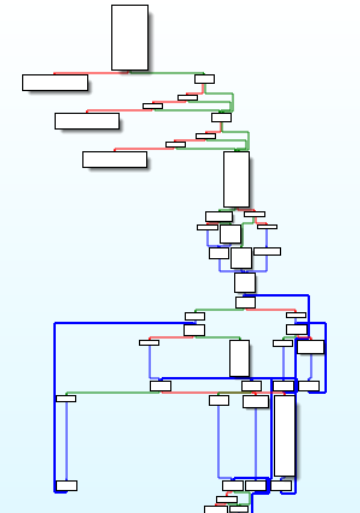


回到调用他的地方查看参数



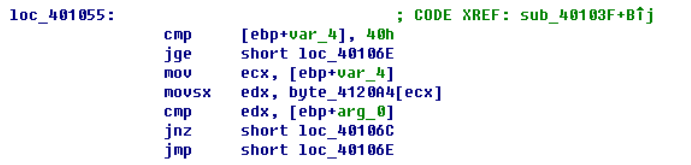
可以看到这里这个位置是 ijklmnopqrstuvwx ，那么其实AES的密钥就是 ijklmnopqrstuvwx

事实上我们查看s\_xor1的代码，发现其非常复杂，有很多的cmp操作引出的分支，当传给该函数的参数不正确时，开始 走向不同的分支，而不同的分支有不同的错误信息，包括：空密钥、不正确的密钥长度、不正确的块长度，由此推测该函数用来进行密钥的初始化。

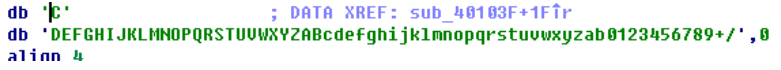


首先比较了arg\_0与0，如果该参数为空，对应的错误信息是Empty key，因此arg\_0就是密钥。到s\_xor1被调 用的地方，可以看到该参数是ijklmnoparstuvwx，这就是被用来AES加密的字符串。 查看该函数的交叉引用，看到在s\_xor1被调用之前，存在一个对unk\_412EF8的引用，将这个偏移量作为参数 传入s\_xor1函数。查看unk\_412EF8的其他引用，发现0x401429是该偏移量载入ECX的地方之一，而它在调用 函数sub\_403745之前被载入到ECX。由此unk\_412EF8是一个表示AES加密器的C++对象，并且s\_xor1是加密器的初始化函数。至此AES的分析结束

而base64的密钥就是strings中看见的那个字符串，也就是



再查看Base64加密算法。检查对字符串 CDEFGHIJKLMNOPQRSTUVWXYZABcdefghijklmnopqrstuvwxyzab0123456789+/的引用，看到这个字符串在 0x0040103F中被使用，该函数用索引var\_4查询这个字符串并且调用函数sub\_401082，将解密的字符串分成4 字节的块，可以看到调用sub\_401082的上下文，发现在其前后分别调用了ReadFile和WriteFile，推测函数 sub\_401082就是该恶意代码自定义的Base64解码函数。



**问题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？ |

**答： 对于AES算法，解密还需要密钥之外的变量，包括密钥生成算法、密钥大小、操作模式，以及一些常量的初始 化等；而对于这个自定义的Base64加密，当前已知的索引字符串已经足够了。**

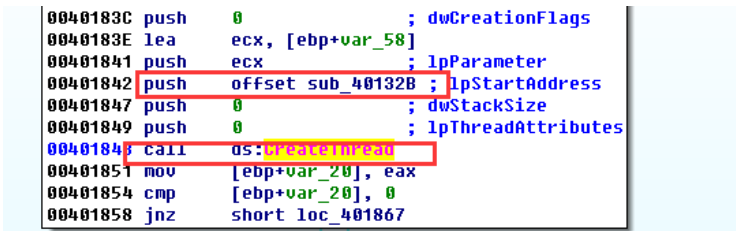
**问题7**

|  |
| --- |
| 恶意代码做了什么？ |

**答： 还建立了一个shell的后门，并且后门是使用base64对发过来的 指令进行解密，用AES对执行结果进行加密再发送回去。该恶意代码用AES用于在写入网络套接字前加密shell命令的输出结果，而这个自定义的Base64用来加密传入的命令。**

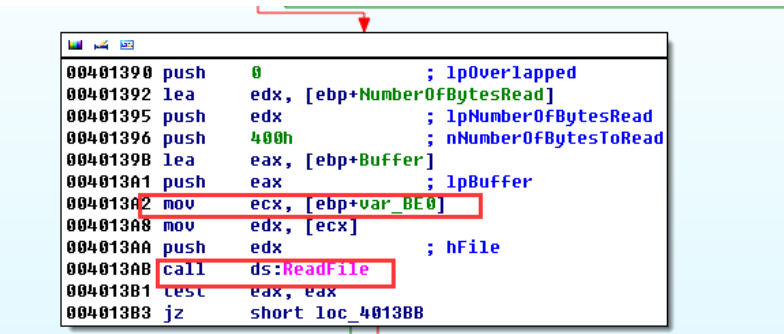
分析过程：

在main函数中我们可以看见



这里创建了一个线程，并且线程的起始地址就是加密函数的起始地址

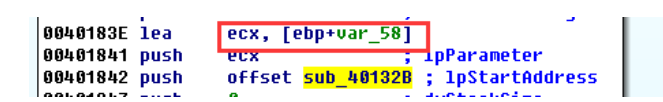
进入到这个函数里，查看一下参数都是在哪里进行了使用



首先这里有一个readfile的函数，这个函数的参数是var\_BE0，

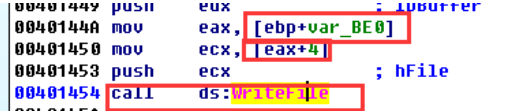


而这个BE0来自于arg\_0

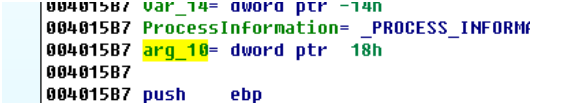


也就是创建线程的时候传入的var\_58

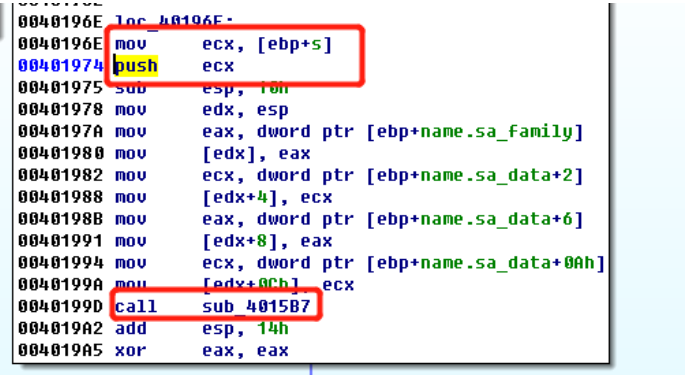
线程内还有一个writefile的函数，这函数的参数



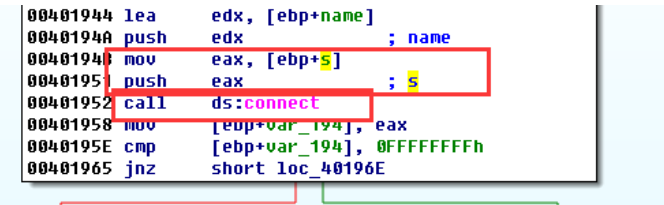
从边上的注释可以看出来这个参数是arg\_10



往上走可以发现这里其实就是这个函数的一个参数

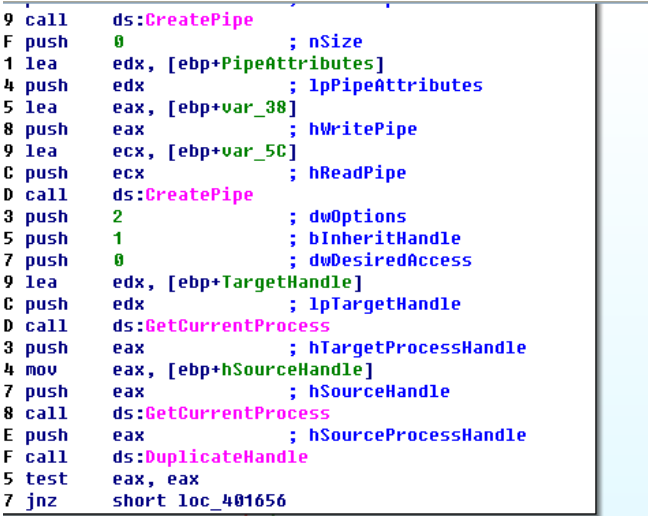


从交叉引用可以发现这里的arg\_10来自ebp+s

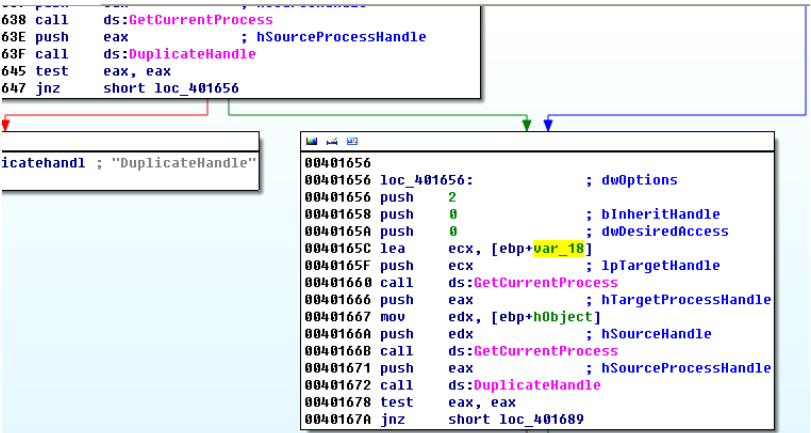


再往上看可以发现这个s就是connect函数创建的socket

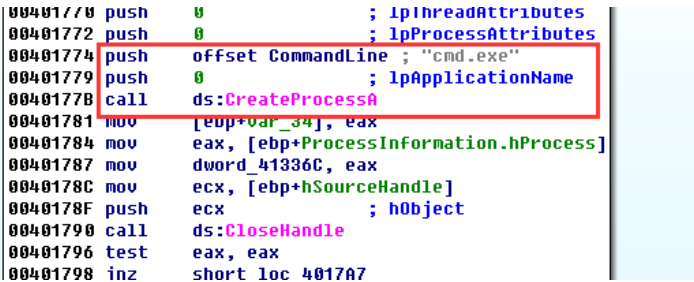
回到函数内，我们可以发现



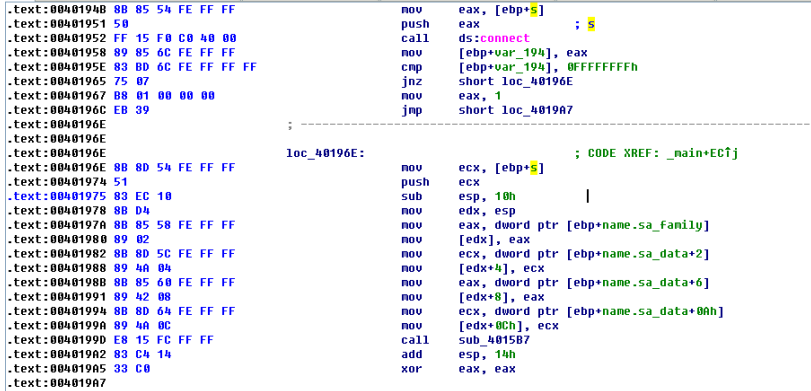
我们已经知道加密函数在sub\_40132b中被调用，而这个函数又被sub\_4015b7引用，跟进，在这附近调用了 CreateThread创建了一个新线程，sub\_40132b作为参数，就是这个新线程的开始位置，传递给这个新线程的 参数保存在lpParameter，也就是var\_58中。跟进该函数，查看这些参数的具体意义。 该函数首先调用了ReadFile，其参数hFile为var\_BE0，是传入该函数的唯一参数，我们已经知道该参数是 var\_58。之后又调用了WriteFile，其参数hFile为var\_BE0+4，也就是var\_54，我们看到var\_58和var\_18 持有一个管道的句柄，并且这个管道的与一个shell命令的输出相连接，令hSourceHandle通过函数 DuplicateHandle复制到shell命令的标准输出和标准错误，这条shell命令由通过调用CreateProcess来启动。



回溯这个频繁被用到的参数var\_54，可以看到它是sub\_4015b7的唯一参数，在main函数中我们可以知道这个 参数是[ebp+s]，它是调用函数connnect后创建的一个网络套接字。



这一系列的操作就是典型的创建了一个反向shell，建立后门，使用 CreatePorcessA 进行启动。而根据 之前的调用base64和AES的位置我们可以发现，这两个都是在readfile和writefile之间被调用的，然后 base64的调用是在AES之前，也就是说，这两个应该是有一个先后顺序：首先使用base64对传递来的指令进行一个解密操作。然后在本地执行完指令之后，使用AES对执行结果进行加密，并反馈给远端。



**问题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？ |

**答： 见分析过程**

分析过程：

两个加密方式中base64相对较为简单，这里先尝试解密Base64产生的一些内容

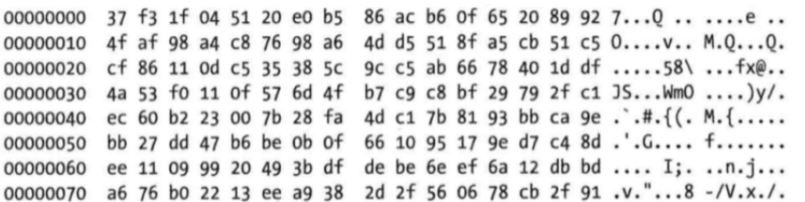
假设截取到的网络通信的部分信息为BInaEi==

python脚本如下：

|  |
| --- |
| Python import string  import base64  result = ""  ciphter\_content = "CDEFGHIJKLMNOPQRSTUVWXYZABabcdefghijklmnopqrstuvwxyz0123456789+/" standard\_b64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"  ciphter\_text = "BInaEi=="  for each\_ch in ciphter\_text:   if each\_ch in ciphter\_content:   result += starndard\_64[string.find(ciphter\_content, str(each\_ch))]   elif each\_ch == '=':   s += '='  result = base64.decodestring(result)  print(result) |

得到的解密结果为：dir ，也就是说此时攻击者执行的指令是dir，想要获得当前路径下的目录列表

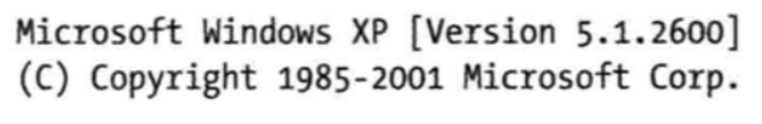
再尝试使用python解密AES的内容，在wireshark中抓到内容为：



python脚本为：

|  |
| --- |
| Python from Crypto.Cipher import AES  import binascii  ciphter\_text = "37 f3 1f 04 51 20 e0 b5 86 ac b6 b5 20 89 92 4f af 98 a4 c8 76 98 a6 4d d5 51 8f a5 cb 51 c5 cf 86 11 0d c5 35 38 5c 9c c5 ab 66 78 40 1d df 4a 53 f0 11 0f 57 6d 4f b7 c9 c8 bf 29 79 2f c1 ec 60 b2 23 00 7b 28 fa 4d c1 7b 81 93 bb ca 9e bb 27 dd 47 b6 be 0b 0f 66 10 95 17 9e d7 c4 8d ee 11 09 99 20 49 3b df de be 6e ef 6a 12 db bd a6 76 b0 22 13 ee a9 38 2d 2f 56 06 78 cb 2f 91 af 64 af a6 d1 43 f1 f5 47 f6 c2 c8 6f 00 49 39"  ciphter\_text = binascii.unhexlify(ciphter\_text.replace(' ', ''))  obj = AES.new('ijklmnopqrstuvwx', AES.MODE\_CBC)  print(obj.decrypt(ciphter\_text)) |

得到的解密结果



**IDA和YARA**

本次实验没有要求，故不再编写。其中LAB13-3的问题8已经可以反映出IDA PYTHON辅助分析的细节了。

**四、实验心得**

**本次实验，我熟悉了对恶意代码分析工具有了更深入的理解，加深了我对恶意代码的理解和相关知识的掌握。在实验过程中，通过亲手分析恶意代码，我收获了很多，过程是非常快乐的。最后，我对本门课程的实验开始得心应手，做实验的速度越来越快了。**