# 第六章实验报告

学号：2011395 姓名：魏伯繁 专业：信息安全

### 一、实验工具：

IDAPro7.0 IDAPro7.7 win10虚拟机 winxp虚拟机

### 二、实验环境：

静态分析主要在win10虚拟机完成。

内存4GB

主频2.9GHZ

### 三、实验内容

第一部分：教材第五章实验：

1、DLLMain的地址是什么

**.text段的0x1000D02E**

2、使用Imports窗口并浏览到gethostbyname，导入函数定位到什么地址

**.idata段0x100163CC**

3、有多少函数调用了gethostbyname

**5个函数**

4、将精力集中于位于0x10001757处的对gethostbyname的调用，你能找出哪个DNS请求被触发吗？

**pics.practicalmalwareanalysis.com**

5、IDAPro识别了在0x10001656处的子过程有多少个局部变量？

**23个临时变量**

6、IDAPro识别了在0x10001656处的子过程有多少个参数？

**1个参数**

7、使用strings窗口，来在反汇编中定位字符串\cmd.exe /c 。它位于哪？

**0x10095B34**

8、在引用\cmd.exe /c的代码所在区域发生了什么？

**连接了一个远程会话**

9、在同样的区域，在0x100101c8处，看起来好像dword\_1008E5C4是一个全局变量，它帮助决定走哪条路径，那恶意代码是如何设置dword\_1008E5C4的呢？

**将版本号保存在了全局变量中**

10、在位于0X1000FF58处的子过程的几百行指令中，一系列使用memcmp来比较字符串的比较。如果对robotwork的字符串的比较是成功的（memcmp返回0），会发生什么？

**会有一个访问注册表的句柄，访问其具体workstime的值并通过一个套接字S发送给远程会话**

11、PSLIST导出函数做了什么？

**pslist可以获取进程名并将它发送到远端会话**

12、使用图模式来绘制出对于sub\_10004E79的交叉引用图。当进入这个函数的时候，哪个API函数可能被调用？仅仅基于这些API函数，你会如何重命名这个函数？

**调用了send、GetSystemDefaultLangID等函数，可以将这个函数重命名为toSendLangID**

13、Dllmain直接调用了多少个WindowsAPI？多少个在深度为2时被调用？

**直接调用了4个API，间接调用了非常非常多的WindowsAPI**

14、在0x10001358处有一个对sleep的调用，顺着代码往后看，如果这段代码执行，这个程序会睡眠多久？

**会休眠30毫秒**

15、在0x10001701处有一个对socket的调用，它的三个参数是什么？

**6、1、2**

16、使用MSDN页面的socket和IDApro中的命名符号变量，你能使参数更加有意义吗？在你应用了修改后，参数是什么？

**当然可以，这其实就就是指定sokcet的类型，可以将其一次替换为IPPROTO\_TCP、SOCK\_STREAM、AF\_INET**

17、搜索in指令（opcode 0xED）的使用，这个指令和一个魔术字符串VMXh用来进行VMware检测。这在这个恶意代码中被使用了吗？使用对in指令函数的交叉引用，能发现进一步的VMware的证据吗？

**可以找到，对16进制字符串的转换以及对函数的交叉引用都可以找到结果。包括字符串VMXh和found virtual machine，install cancel，代表着可能使用了反虚拟机技术。**

18、讲你的光标移动到0X1001D988处，你发现了什么？

**发现了一些没有说明实际含义的字符串**

19、如果你安装了IDA python插件，运行Lab05-01.py，在运行这个脚本后发生了什么？

**乱码被成功翻译成为了有意义的语句**

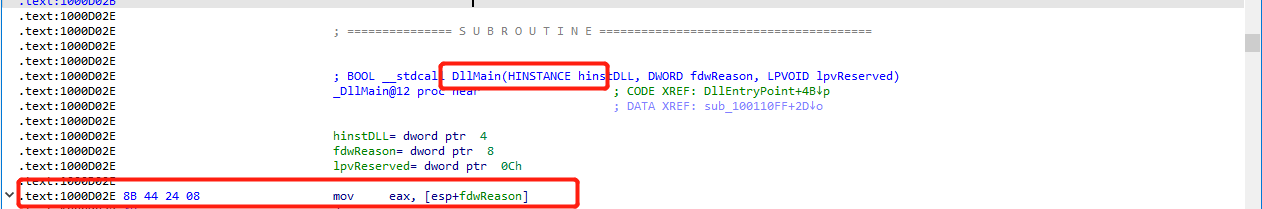
20、将光标放在同一位置，你如何将这个数据转换成一个单一的字符串？

**按下a键就可以获得脚本运行效果**

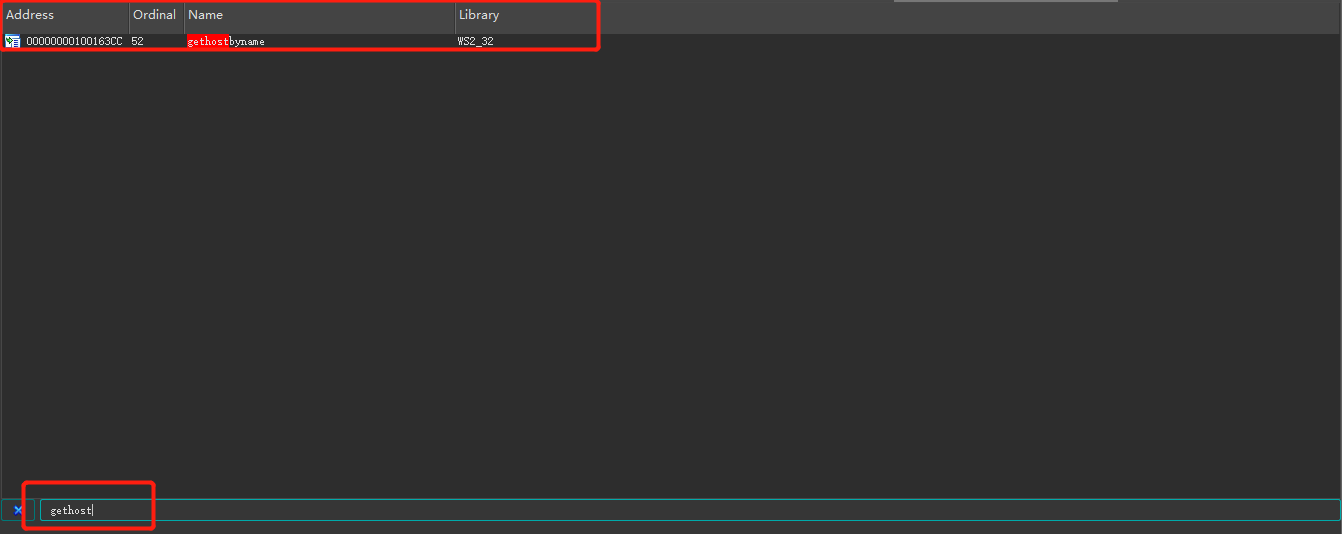
21、使用文本编辑器打开.py文件，它是如何工作的？

**使用了简单的异或运算，与0x55进行异或**

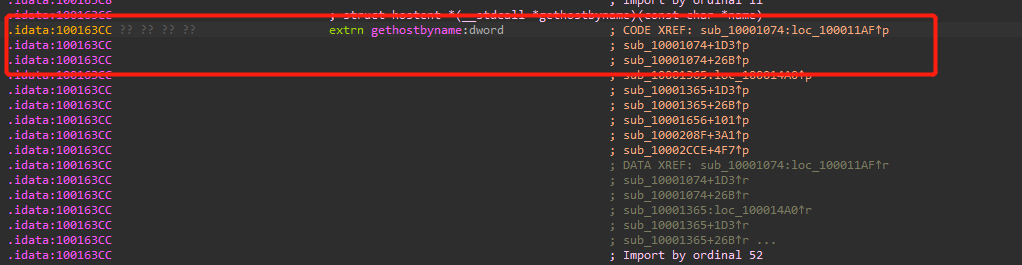
当我们使用IDAPro加载dll文件时，IDA将自动跳转到DLLmain的地址处，因为从DLLEntryPoint到DLLMain之间的代码都是编译器自动生成的，在恶意代码分析时我们应该主要分析被人编写的代码，而不是着重关注那些编译器自动生成的代码。从下面这张图片可以看到，DLLMain的地址位于.text段的1000D02E的位置



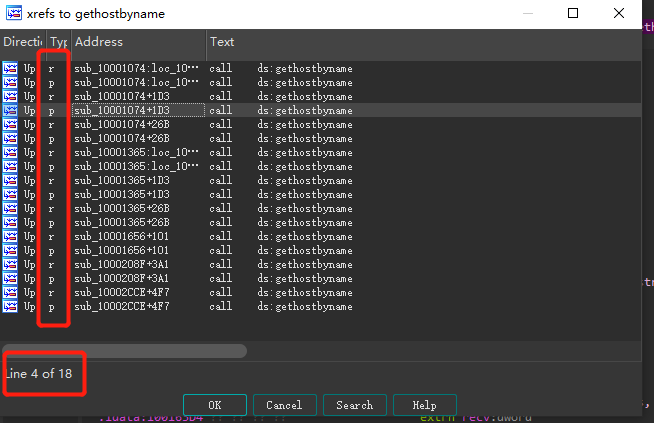
然后查看导入函数，直接点击View---Open Subviews之后找到Imports，在导入表中搜索到我们需要的dll--gethostbyname，可以查看到，他的位置是0x100163CC



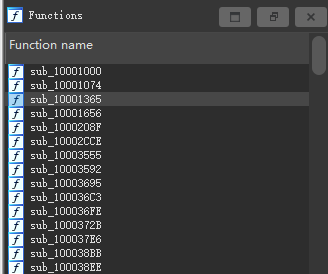
双击后可以查看函数详情



为了计算出到底有多少个函数调用过gethostbyname，我们找到这个函数并按下ctrl+x进行查看交叉引用表如下：总共显示了18条信息，但是是成对出现的类型p表示是被调用的引用，类型r代表是被读取的引用，因为对每一个call指令，cpu都需要先读取导入项，然后再调用它。

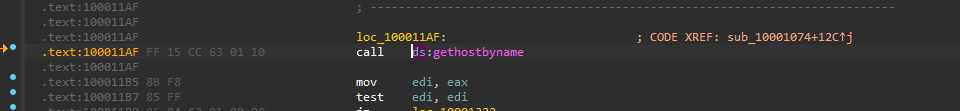


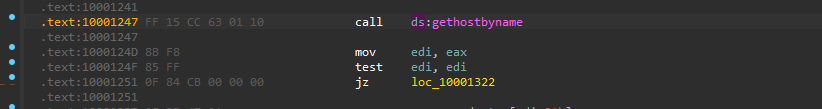
但是，有一些调用可能是发生在一个函数中的，这时候就需要我们手动去检查：下图是函数在.text段的地址：

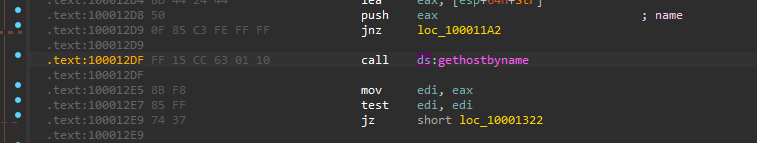


我们通过一个一个点击交叉引用表去查看发现：

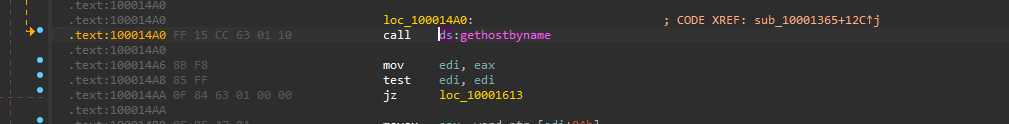
前三组交叉引用都在sub\_10001074函数中：

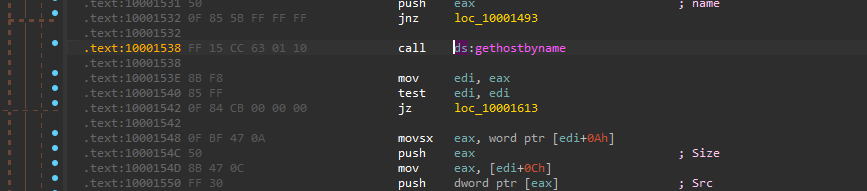


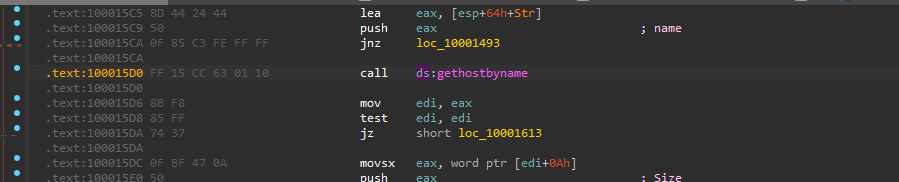




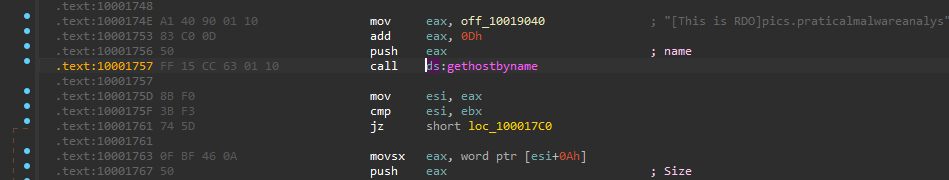
这三个引用发生在sub\_10001365函数中：



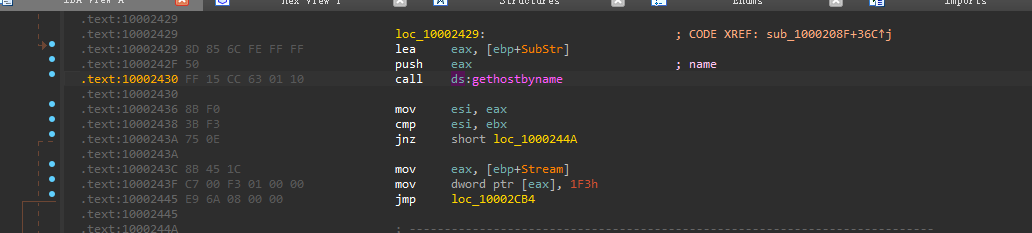




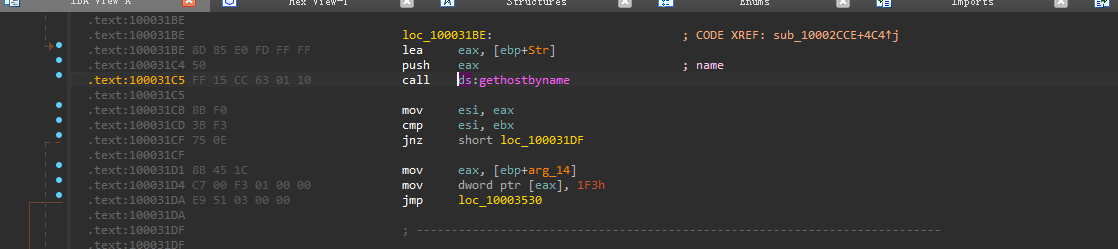
这次引用发生在sub\_10001656



这次引用发生在sub\_1000208f

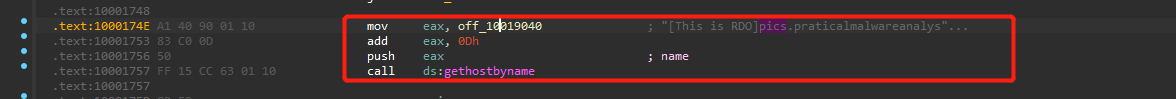


这次引用发生在sub\_10002CCE

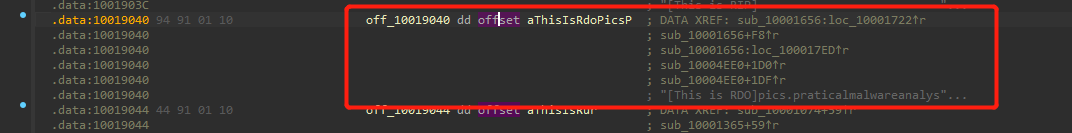


总共在五个函数发生了这个dll的调用

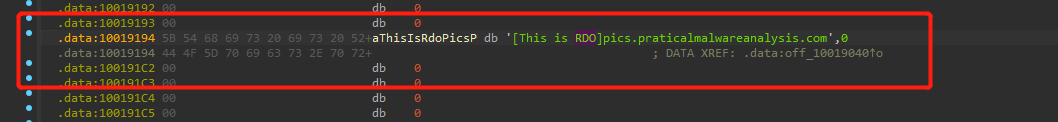
重点关注0x10001757处的函数调用，点击之后找到为gethostbyname的参数传递过程：



点击对应地址查看内容：

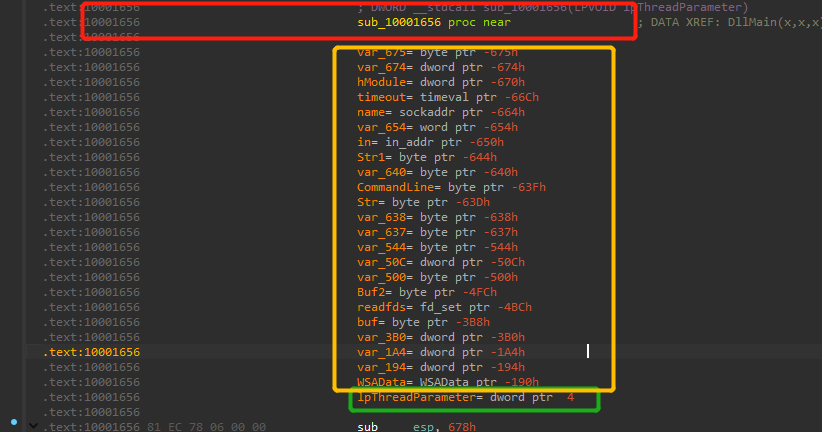


再次点击偏移信息：发现了这个熟悉的网址pics.practicalmalwareanalysis.com

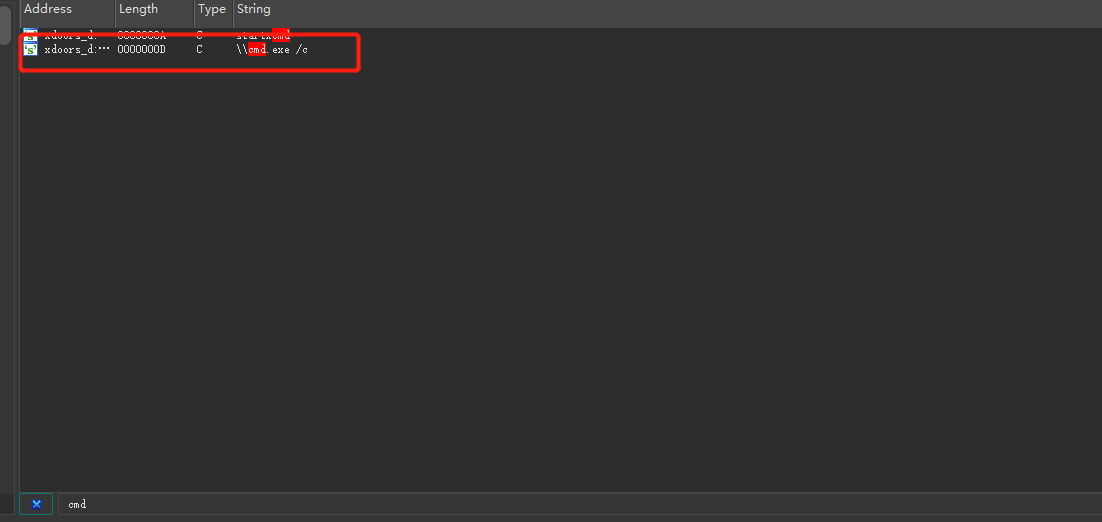


想要分析0x10001656处的代码，就点击g并输入想要前往的地址即可：

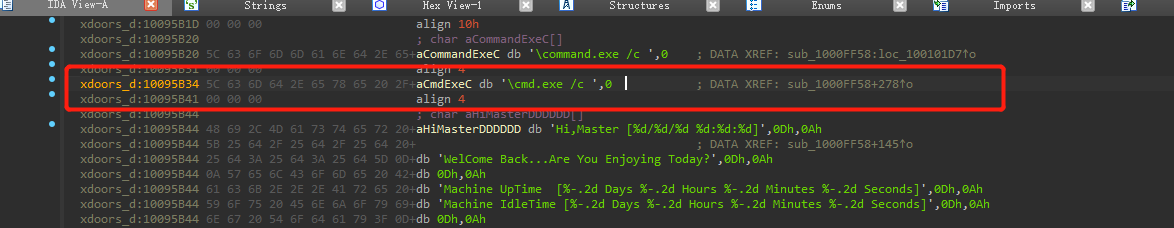
在0x10001656，IDAPro已经帮我们分析好了局部变量以及参数，简单的数一下可以发现总共有23个临时变量和1个参数



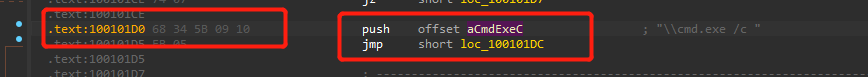
查找特定的string还是和之前的策略一致，利用ida提供的搜索机制进行搜索，可以轻松找到需要的cmd.exe字符串



点击进去就可以看到这个字符串的位置位于0x10095B34的位置

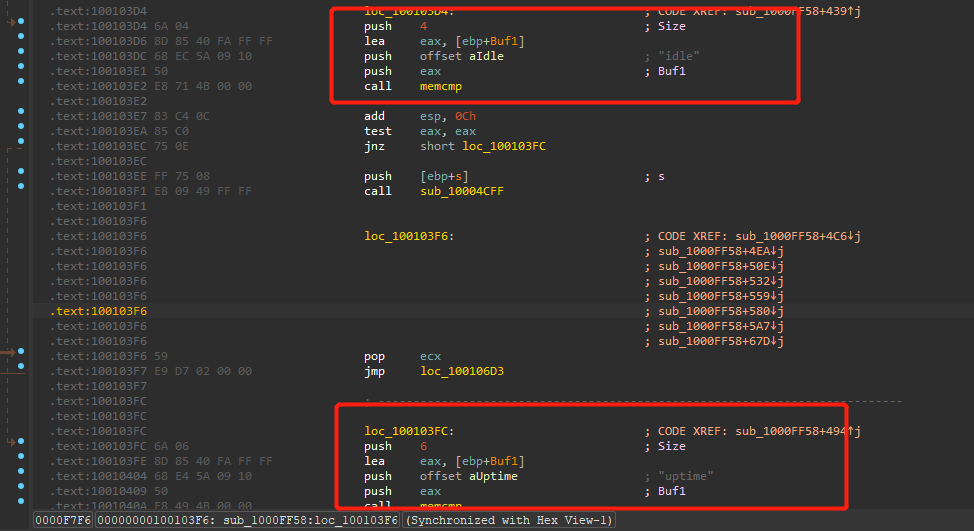


为了分析对这个字符串的引用还是按住ctrl+x随后来到了如下位置：字符串被压栈后跳转到别的地方进行继续执行



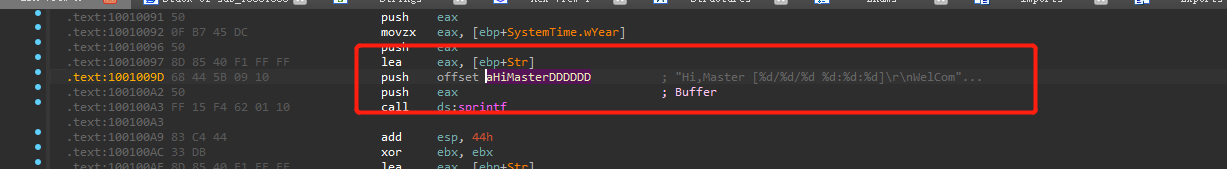
按照题目要求观察上下文：首先查看下文，发现代码进行了一系列的比较操作：包括与quit、exit、cd、idle、uptime、minstall、inject等进行比较，非常像在操控命令行程序。



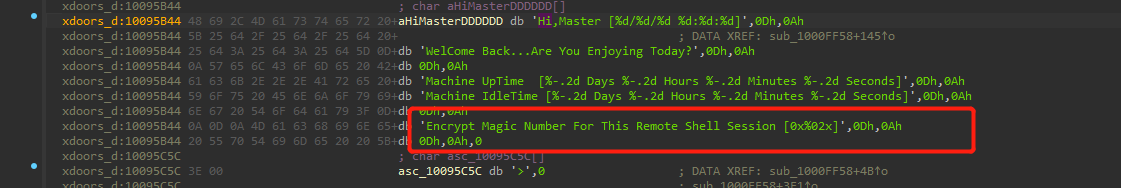




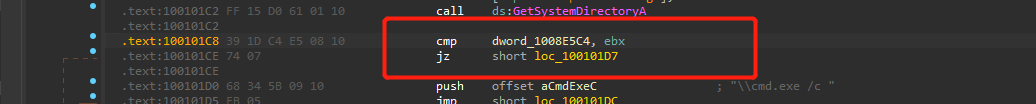
查看上文：发现引用了一大段字符串，我们还是点进去查看一下完整的字符串内容



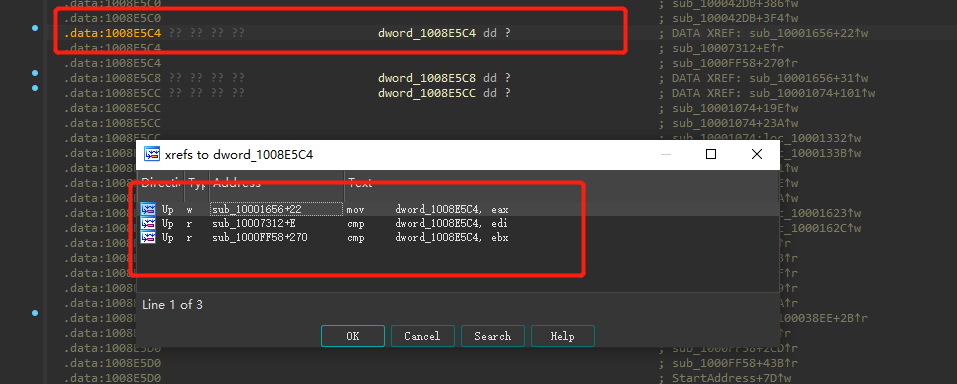
虽然程序我看不太懂，但是英文还是认识一些，可以判断这个阶段应该是想远程连接一个会话（remote shell session）



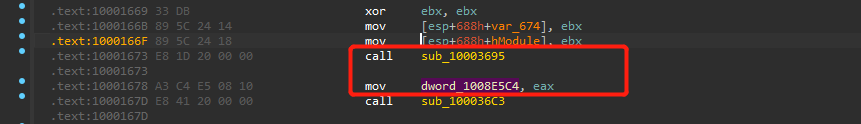
找到对应的全局变量的位置，随后双击进入查看详情。



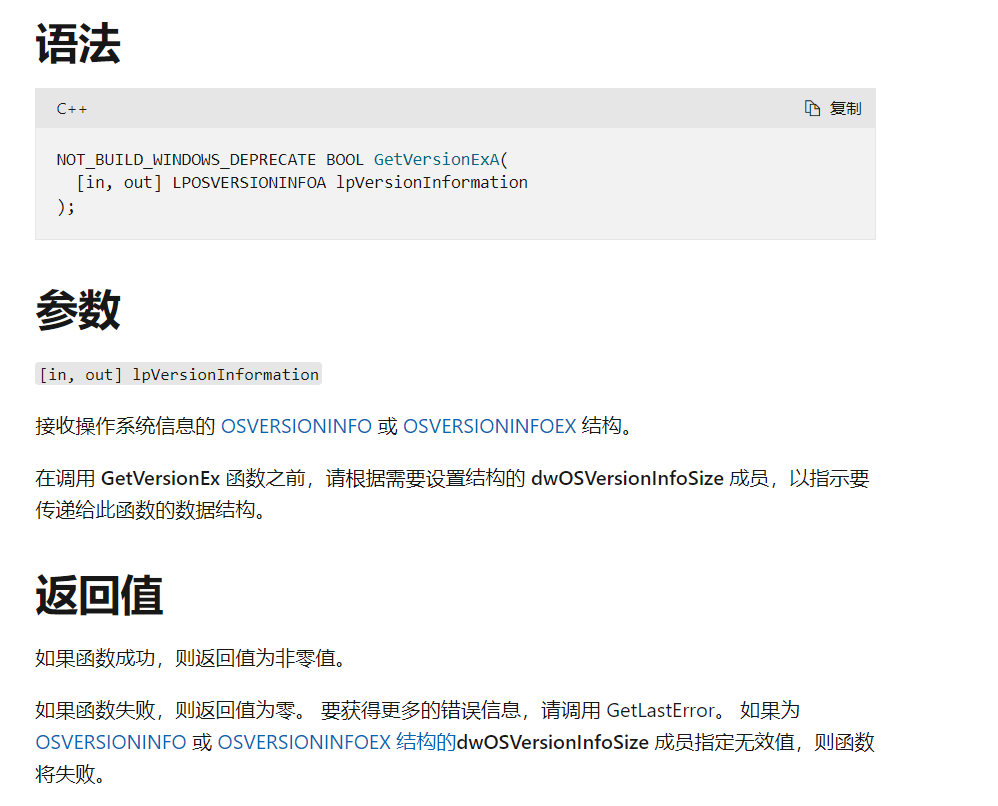
点击ctrl+x进行交叉引用，发现有三个地方对这个变量进行了引用，但是只有一个地方对其进行了改值操作，我们点击进入

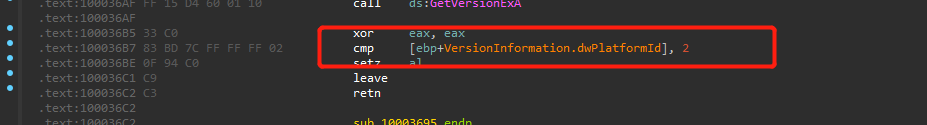


可以看到，程序将上一个调用的函数的值赋值给了这个全局变量，于是我们点击进入该函数：



发现它实际上调用了GetVersionExA函数，我们通过查看微软官方文档来确定他的含义。并且将platformId和2进行比较来设置AL的值，也就是说会简单的判断系统是否是win2000或更高的版本。



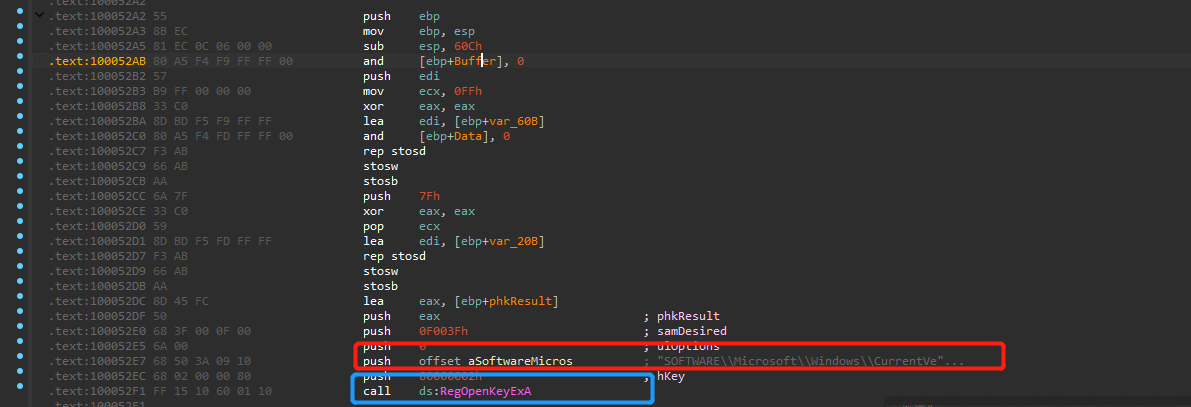


然后，我们来分析这个函数一系列比较中的一个：通过ida给我们的注释找到robotwork我们发现，如果字符串等于robotwork，就会跳转到绿色的位置并执行相应函数，我们点击进入该函数：

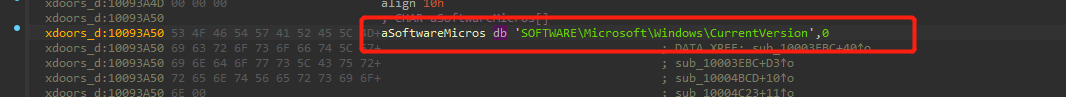


发现它将一个字符串压入了栈并调用了函数，随后就又调用了RegOpenKeyExA，同样通过查看微软官方文档来获取其定义：这是为了打开一个注册表的句柄。

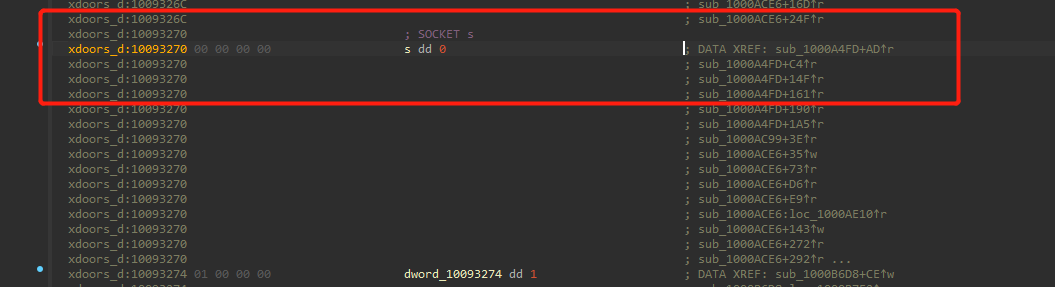




查看其字符串完整内容：是查找了注册表某一个键的值，然后将这个值返回给了S

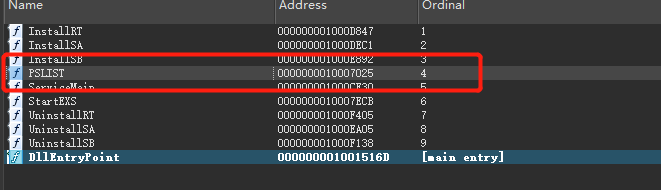


点进去看看s是什么，发现是一个套接字S，很可能就是返回给刚才远程会话连接的套接字。

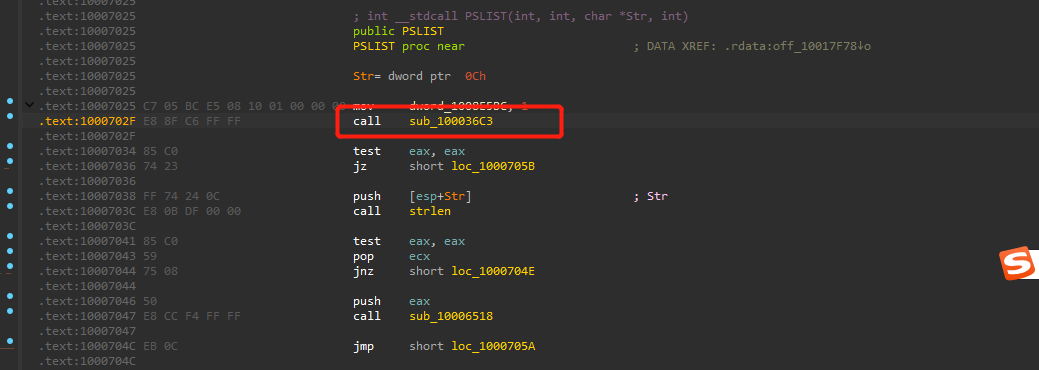


**（问题十一解决办法）**

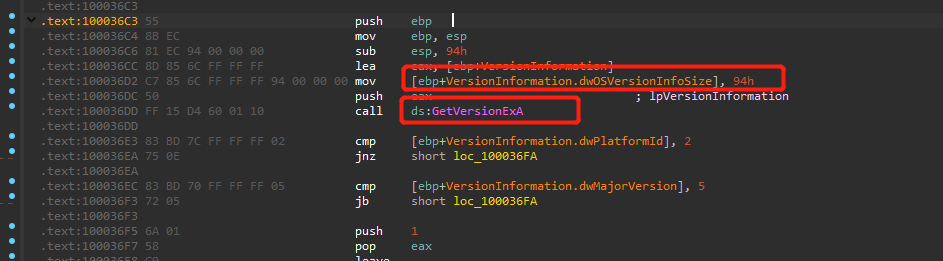
想要查看导出表，我们也是使用open subview并选择export，可以看到目标导出函数：



点击进入，查看该函数的具体动作：可以看到根据sub\_100036C3的返回值不同，函数走的路径也不同

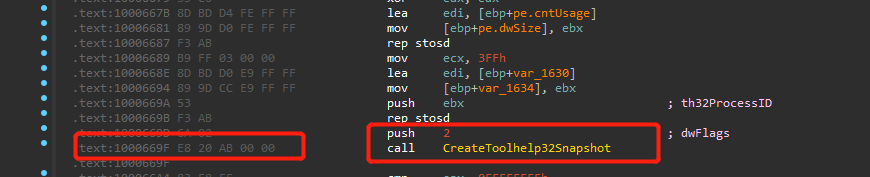


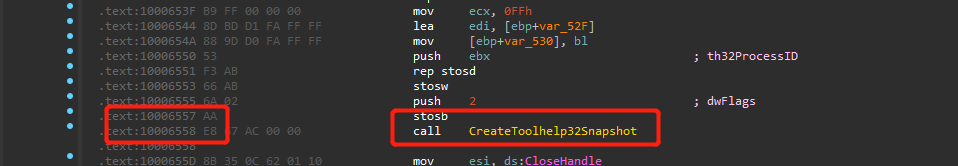
查看该函数的具体含义：可以判断该函数是用来判断操作系统的版本并返回的



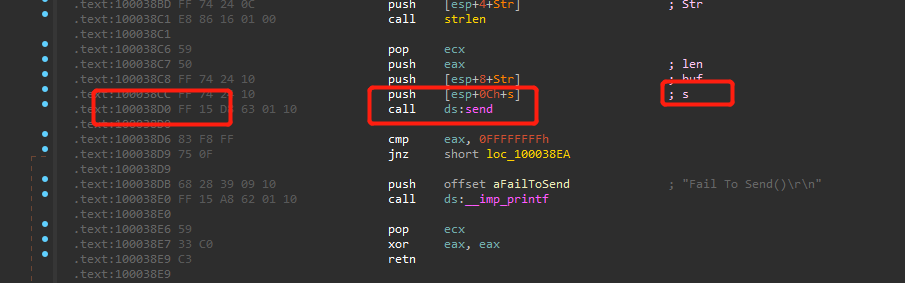
查看其可能选择的路径：

两条路径其实要做的事情是一样的，都调用CreateToolhelp32Snapshot来获取进程列表



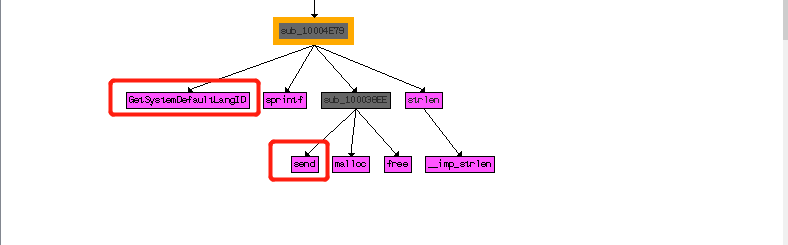


然后通过send发送给s，从而完成与远程会话的通信，s也就是绑定了远程会话的套接字（前面分析过）



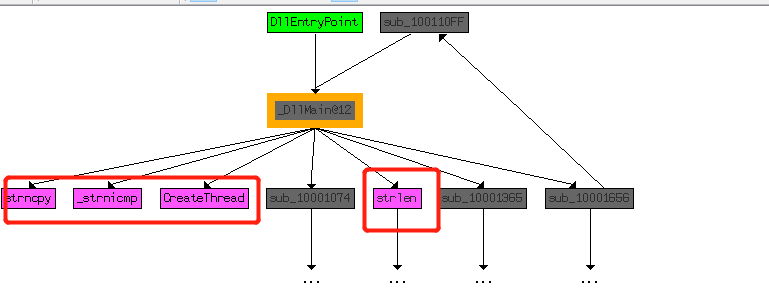
**（问题十二解决办法）**

通过view目录下显示user Xref Chart来显示交叉引用，发现它调用了GetSystemDefaultLangID和send，说明他把得到的信息通过套接字发送给了远程会话，所以我们不妨把他改名为toSendLangID

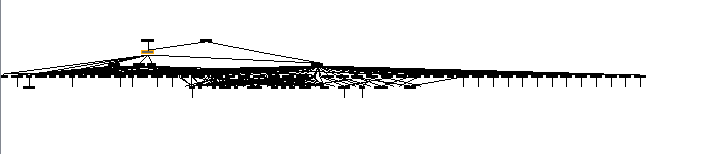


**（问题十三解决办法）**

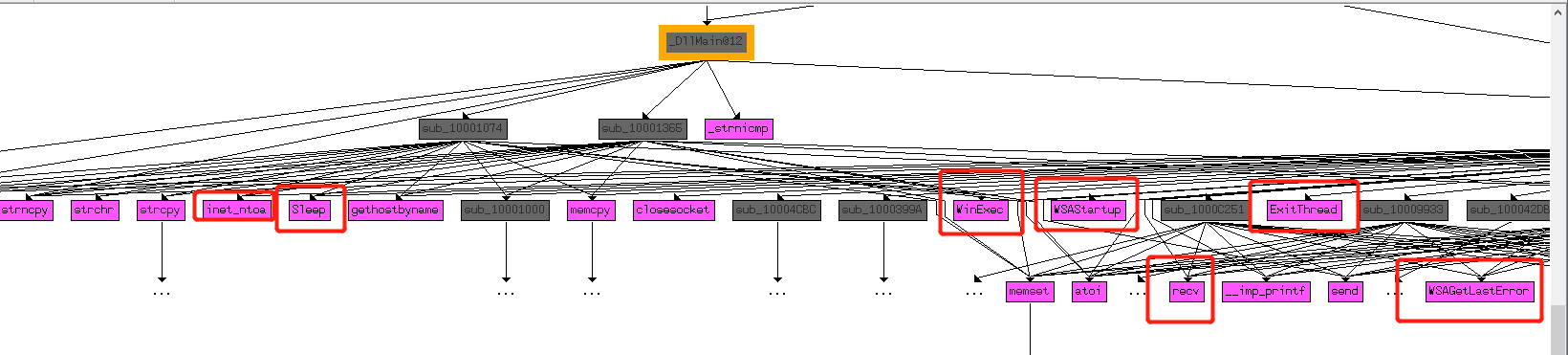
使用同样的方法，只不过要回到问题一中已经找到的DLLMain的位置，是1000D02E，然后查看一层的交叉引用图，发现他直接引用了4个API函数



更改查看的调用深度，发现引用树一下膨胀了很多

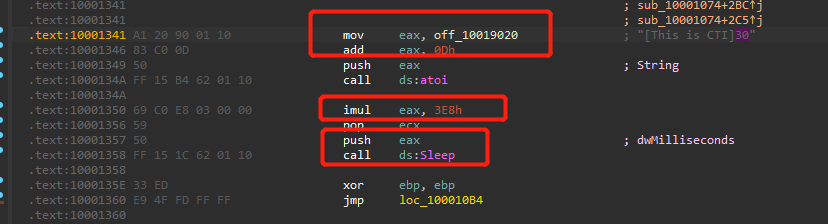


慢慢放大仔细去看，发现了很多和之前分析有关的函数，包括初始化socket.dll、网络连接函数、进程创建函数、执行一个二进制文件函数等等，说明DLL的在本机将展开很多恶意行为。

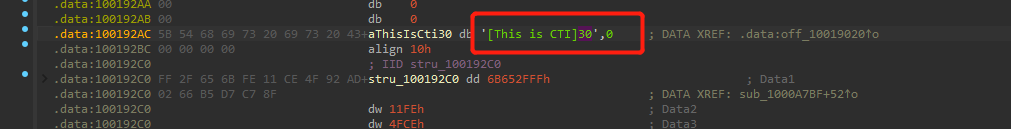


**（问题十四解决办法）**

使用g+地址的方法进入相应地址并查看源代码：发现eax被作为参数传入sleep的调用，eax先被赋予了0ff\_10019020位置的值，随后使用atoi将字符串变成了int，最后又被乘了1000（转换为16进制就是3E8h）

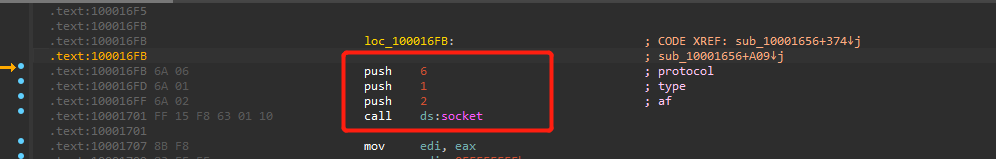


下面查看0ff\_10019020的值，发现是30，所以真正休眠的时间就是30秒，因为30\*1000的单位是毫秒

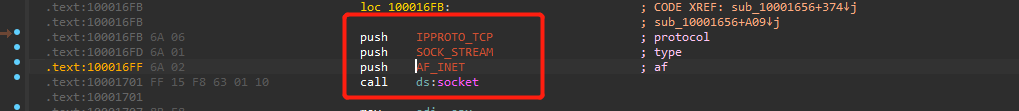


**（问题十五、十六解决办法）**

可以查看到在这个位置为了构造socket传入了三个参数分别是6、1、2，根据后面的注释很简单的可以猜出来是代表这使用IPV4还是IPV6，选择什么样的传输层协议以及使用的数据类型（流类型还是其他）

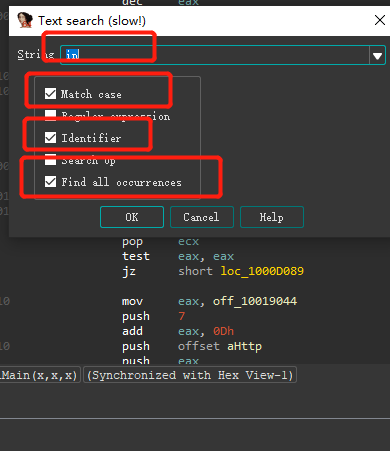


于是右键使用symbolic constant进行替换，可以帮助我们更好的理解程序：

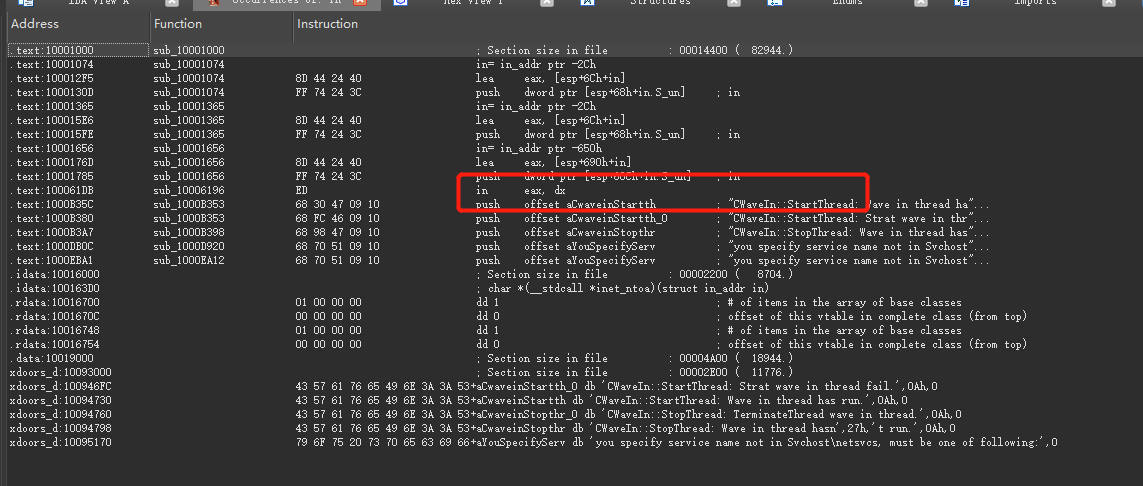


**（问题十七解决办法）**

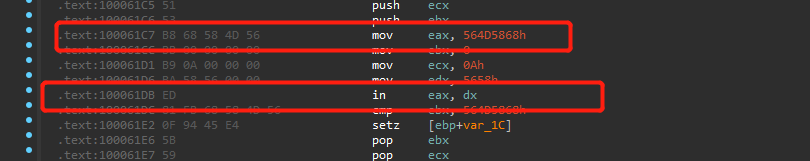
首先查找字符串，但由于包括in的单词很多，所以我们要对字符串做筛选，首先是大小写敏感，其次是选择identifier，也就是必须是独立的in，像main这种在单词中包含in的不能被选择。



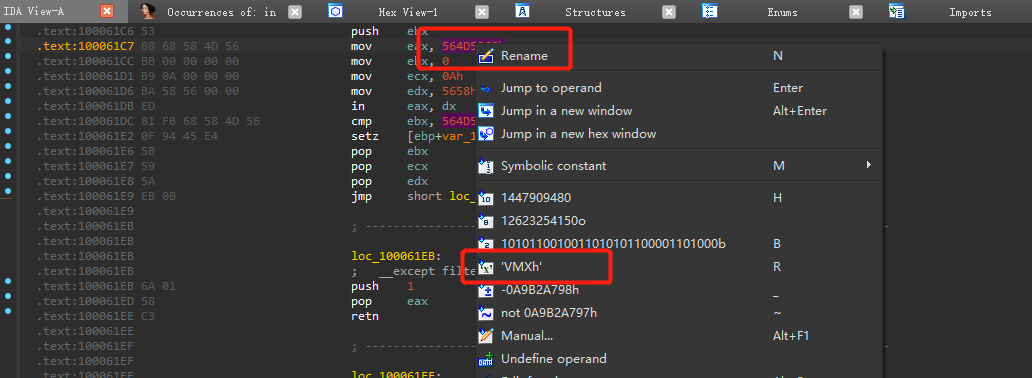
我们仔细检查，发现只有一处出现了in作为指令，我们点击进行查看



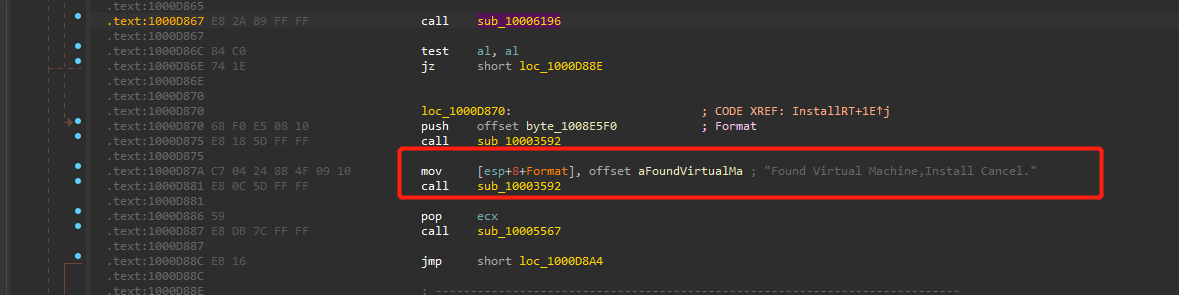
检查该部分代码，其实比较重要的是搞懂那个16进制字符串是什么意思



右键字符串，ida会自动将有选项其替换为成asc码，可以看到代表着VMXh

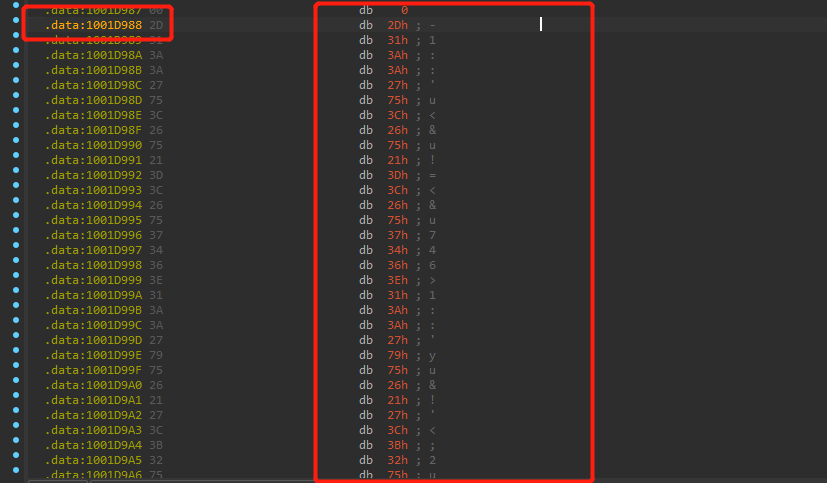


对这个位置所在函数应用交叉引用检查：发现其确实与虚拟机有关，找到了相关字符串found virtual machine，install cancel，很有可能应用了反虚拟机技术。

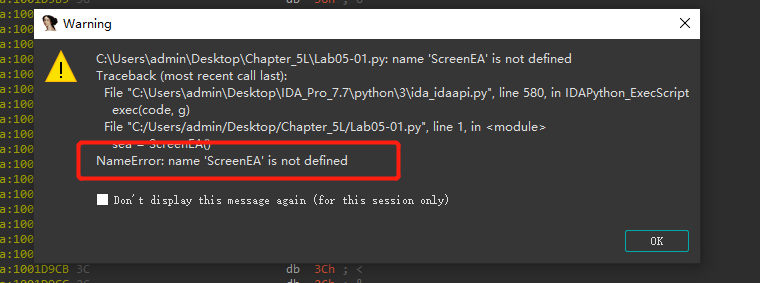


**（问题十八到二十一解决办法）**

来到指定位置，我们看到了一些没有什么实际意义的字符串：



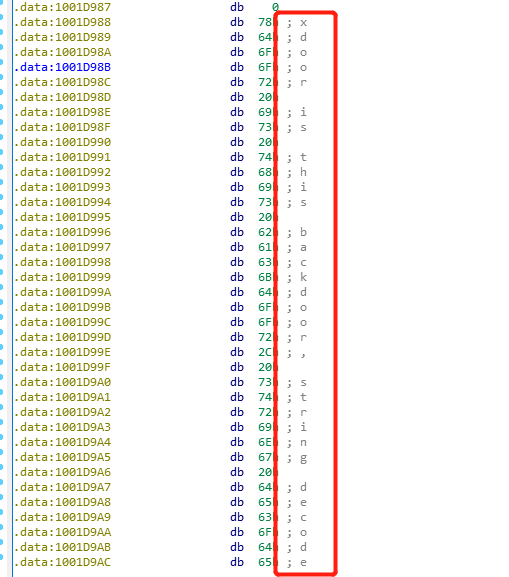
然后我们使用file一加载，发现了一个神奇的报错：系统说这个识别光标的函数我没办法检测，有点奇怪。

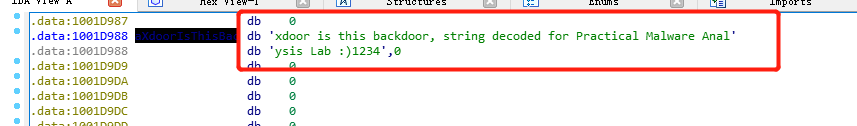


然后我抱着侥幸的心理来试了一下运行：发现没有什么用，识别出来的东西还是乱码。

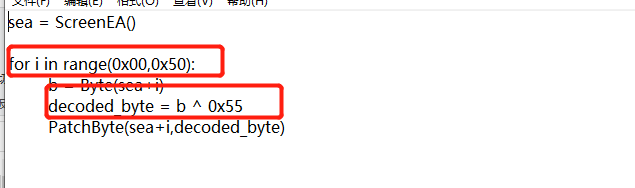


因为我用的是idapro7.7，他适配的idapython是3.8版本的python，我想是不是可能因为版本太高了呢，于是我又查看了ida7.0，他使用的是python2.7，于是我又在我的idapro7.0上进行了这个实验，发现就可以成功跑通了：





最后，我们点开这个脚本看一看他是如何实现的转换：发现他是对光标所在的位置向后50个字节进行异或解密，与55进行异或



yara规则编写：

import "pe"  
rule IsPE{  
 meta:  
 description = "检查文件是否为PE文件"  
 author="wbf"  
 date="2022-10-29"  
 condition:  
 uint16(0) == 0x5A4D and //“MZ”头  
 uint32(uint32(0x3C)) == 0x00004550 // “PE”头  
  
}  
  
rule smallFileSize{  
 meta:  
 description = "检查文件小是否大概率为exe文件"  
 author="wbf"  
 date="2022-10-29"  
 condition:  
 filesize<500KB  
}  
  
rule mayBeMaleware{  
 meta:  
 description = "检查文件小是否大概率为maleware文件"  
 author="wbf"  
 date="2022-10-29"  
 strings:  
 $a = { E8 00 00 00 00 }  
 condition:  
 $a at pe.entry\_point  
}  
  
rule WebConnect{  
 meta:  
 description = "检查文件是否可能有网络行为"  
 author="wbf"  
 date="2022-10-29"  
 strings:  
 $a = "WS2\_32.dll" nocase  
 $b = "http"  
 $c = "https"  
 $d = "www"  
 $e = "URL"  
 $f = "ftp://" nocase  
 $g = "socket" nocase  
 $h = "GET"  
 condition:  
 (5 of them) and IsPE and smallFileSize  
}  
  
rule getMessageOfComputer{  
 meta:  
 description = "该软件可能获取了大量本机信息"  
 author="wbf"  
 date="2022-10-29"  
 strings:  
 $a = "thread" nocase  
 $b = "windows"  
 $c = "CreateToolhelp32Snapshot"  
 $d = "GetDiskFreeSpaceExA"  
 $e = "HardwareInformation.MemorySize"  
 $f = "HostInfo"   
 $g = "GroupInfo"  
 $h = "ProcessID"  
 $i = "regedit.exe"  
 condition:  
 (7 of them) and IsPE and smallFileSize  
}  
  
rule changeMessageOfComputer{  
 meta:  
 description = "该软件可能更改本机配置"  
 author="wbf"  
 date="2022-10-29"  
 strings:  
 $a = "SeShutdownPrivilege"   
 $b = "OpenProcessToken"  
 $c = "RegCreateKey(Parameters)"  
 $d = "OpenThread"  
 $e = "xinstall.dll"  
 $f = "AdjustTokenPrivileges"  
 condition:  
 (3 of them) and IsPE and smallFileSize  
}

ida python辅助工具编写：

查找指定地址的参数个数：(第五问)

ea=0x10001656  
idaapi.get\_arg\_addrs(ea)

查找指定函数的汇编语句：（第十问）

dism\_addr=list(idautils.funcItems(here()))  
for line in dism\_addr:  
 print("0x%x%s"%(line,idc.generate\_disasm\_line(line,0)))

查看交叉引用：（第九问、第17问）

wf\_addr = idc.get\_name\_ea\_simple("in")  
print("0x%x %s" % (wf\_addr,idc.generate\_disasm\_line(wf\_addr,0)))  
for addr in idautils.CodeRefsTo(wf\_addr,0):\  
 print("0x%x %s" % (wf\_addr,idc.generate\_disasm\_line(wf\_addr,0)))

搜索字符串（第七问、第14问）

cur\_addr= idc.get\_inf\_attr(INF\_MIN\_EA)  
for x in range(0,5):  
 cur\_addr = ida.search.find\_text(cur\_addr,0,0,"sleep",ida\_search.SEARCH\_DOWN)  
 if addr == idc.BADADDR:  
 break  
 print("0x%x %s" % (wf\_addr,idc.generate\_disasm\_line(wf\_addr,0)))  
 cur\_addr = idc.next\_head(cur\_addr)

### 四、实验心得

通过本次实验，我复习了在大二学年在汇编课程上学习的IDAPro的使用，通过完成例题加深了我对IDA各种功能的使用理解。同样，通过本次课学习的IDAPyhton脚本编写加速了我分析各种恶意代码的速度，提升了完成作业的效率。