

**恶意代码课程实验报告**

**实验五**

**实验目的：**

**（1）完成教材上lab5的题目；**

**（2）在样本分析结果的基础上，编写样本的Yara检测规则。**

**（3）尝试编写IDA Python脚本来辅助样本分析。**

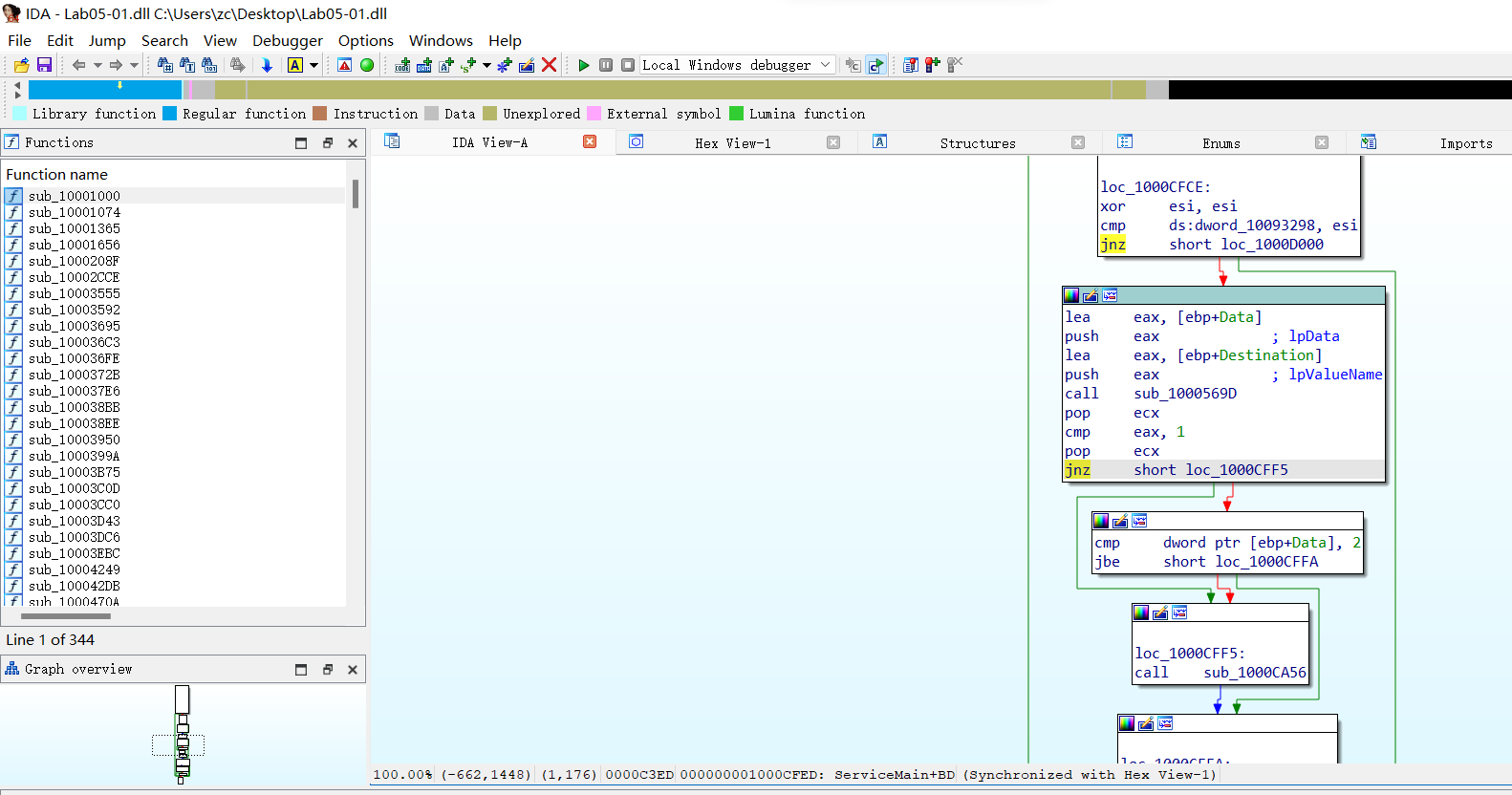
**实验过程：**

### Lab5-1

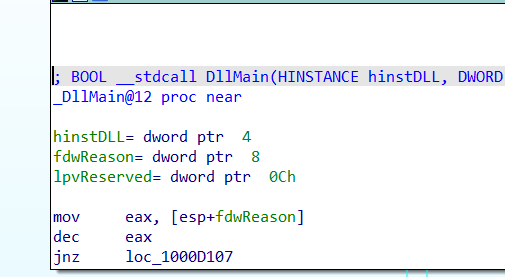
##### DLLMain的地址是什么？

在win系统进行ida分析即可。

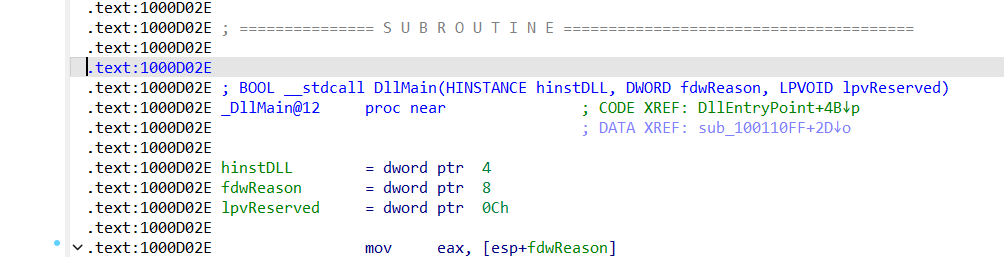
实验ida打开Lab05-01.dll:



事实上打开时鼠标所在位置就是DLLMain位置：



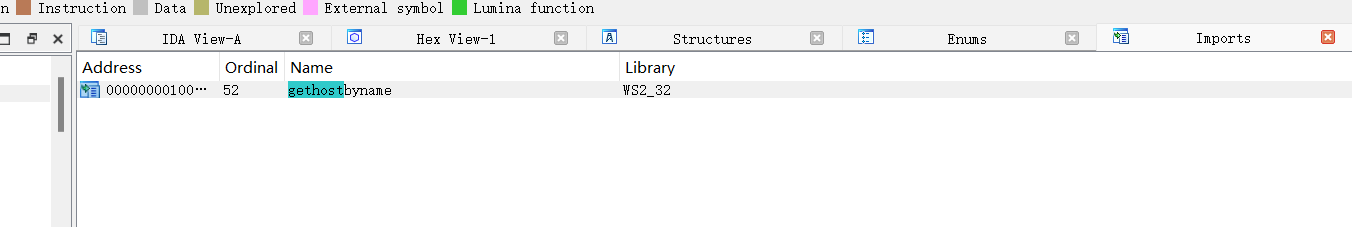
按空格键进入反汇编：



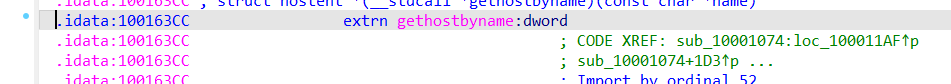
可知，DLLMain的地址是 **.text:1000D02E**

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

在import窗口，可通过Ctrl+f查找gethostbyname:



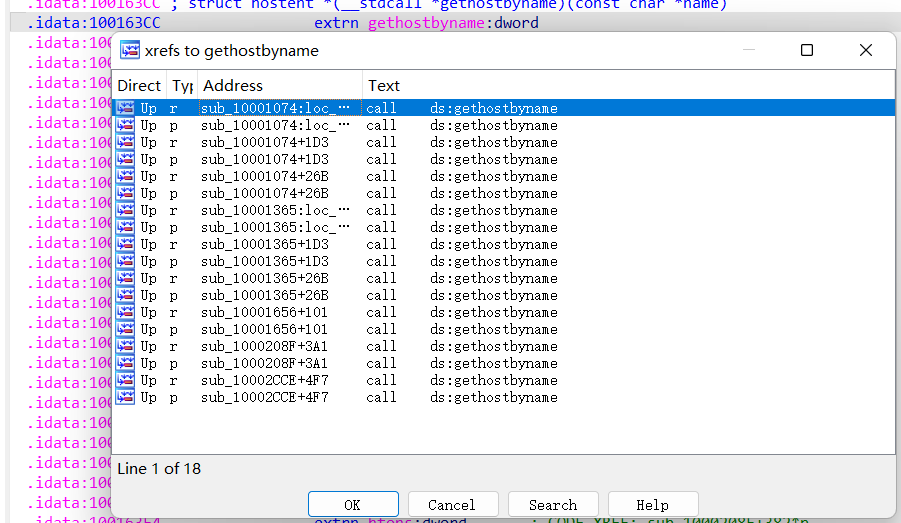
双击查看：



可知定位地址在 **.data:100163CC**

### 有多少函数调用了gethostbyname？

在上一题的步骤后，按Ctrl+X，出现交叉引用窗口：

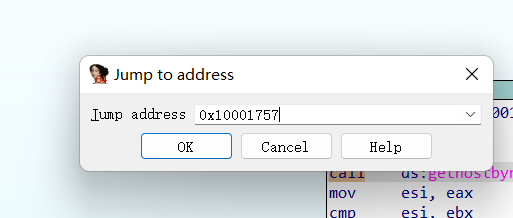


需要在这18个结果里查找，Type为p（调用）且Address为不同函数。

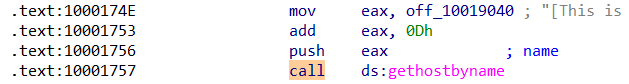
最终知道有5个函数调用了gethostbyname，总计调用9次。

### 将精力集中在位于0x10001757处的对gethostbyname的调用，你能找出哪个DNS请求将被触发吗？

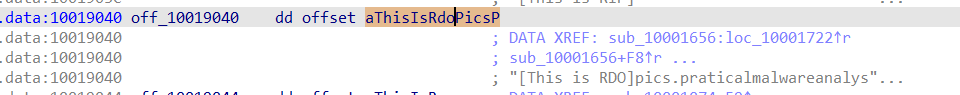
按下G键快速跳转：



在反汇编页面可以看到参数为eax，值为off\_10019040+0Dh:



点击off\_10019040:



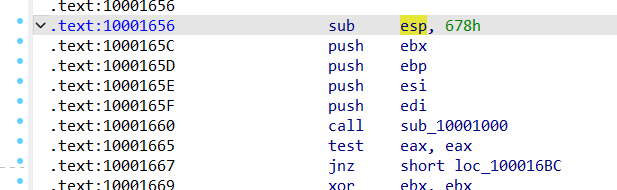
可以进一步点击：



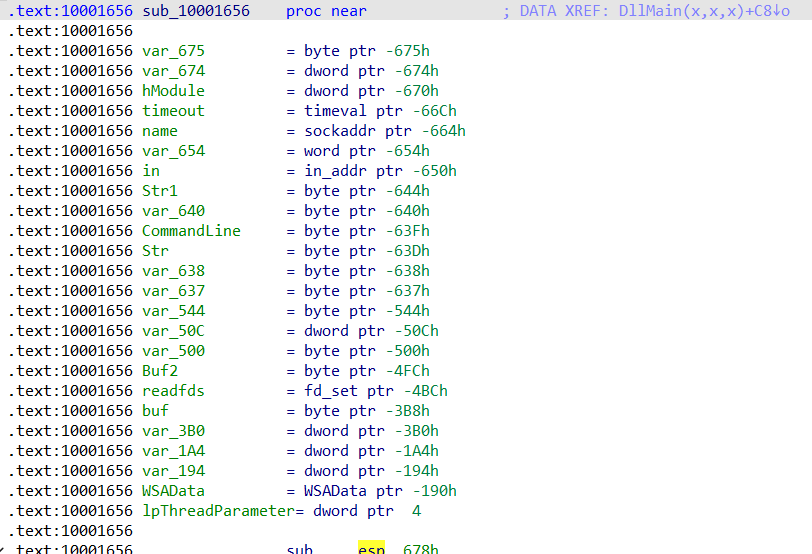
故解析的就是：**pics.praticalmalwareanalys.com**

### IDA Pro 识别了在0x10001656处的子过程中的多少个局部变量？

G键跳转:



后面为带负号的是局部变量：



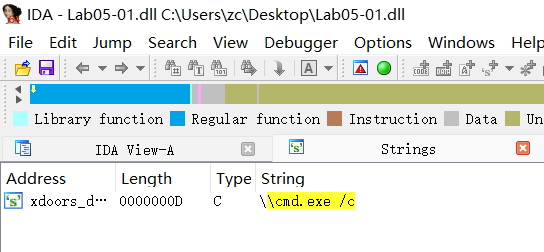
共有23个局部变量。

### IDA Pro识别了在0x10001656处的子过程中的多少个参数？

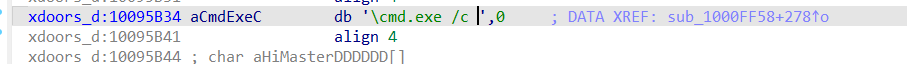
同上一题，后面是正值的是参数，故有一个参数。

### 使用Strings窗口，来在反汇编中定位字符串 \cmd.exe /c, 它位于哪儿？

view->string窗口，Ctrl+f查找：



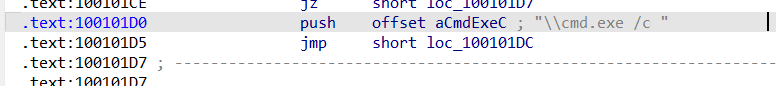
进入：



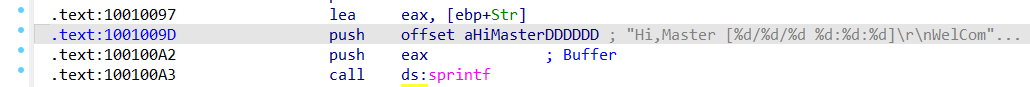
可知位置在： **xdoors\_d:10095B34**

### 在引用 \cmd.exe /c的代码所在区域发生了什么？

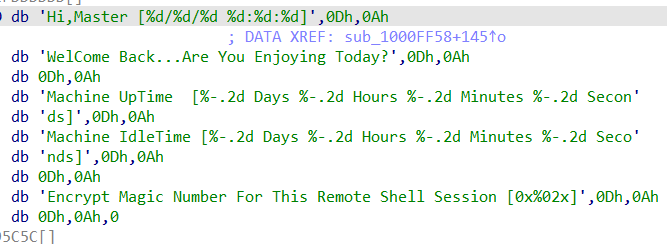
双击后面向上的箭头：



检查区域，在上方有字符串：

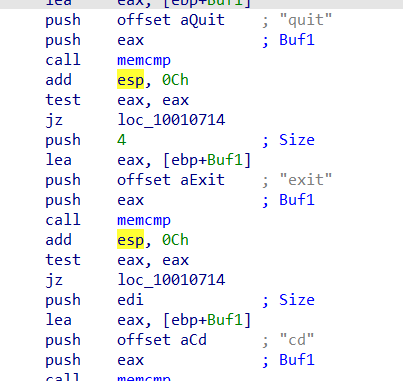


双击查看：



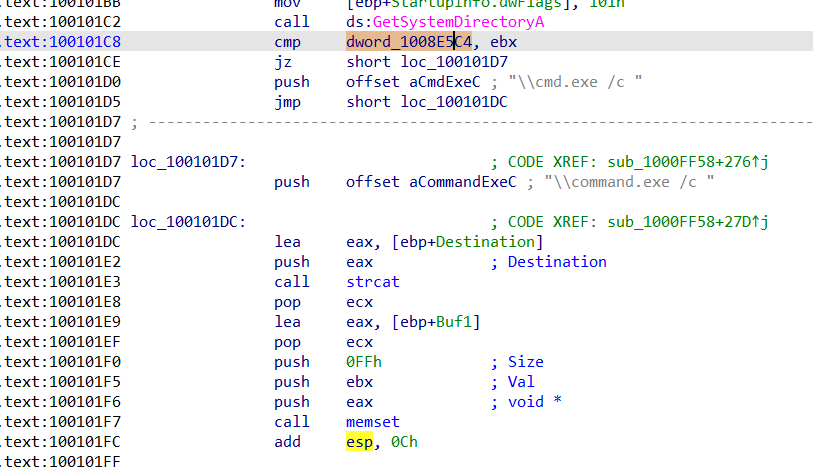
从Remote Shell Session 可以怀疑程序会开启远程的shell会话。

同时区域下方的api函数和字符串比对也更有理由来怀疑：

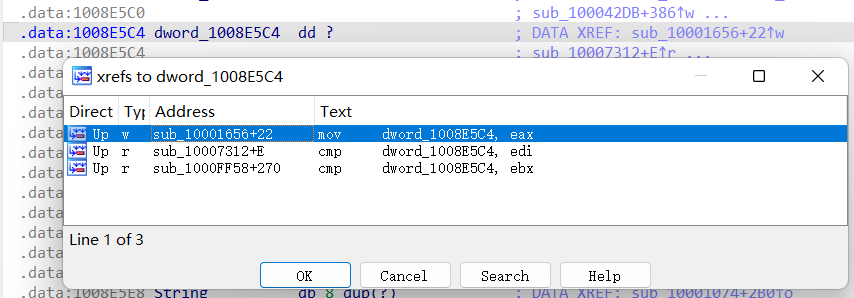


### 在同样的区域，在0x100101C8处，看起来dword\_1008E5C4好像是一个全局变量，它帮助决定走哪条路径。那恶意代码是如何设置dword\_1008E5C4的呢？

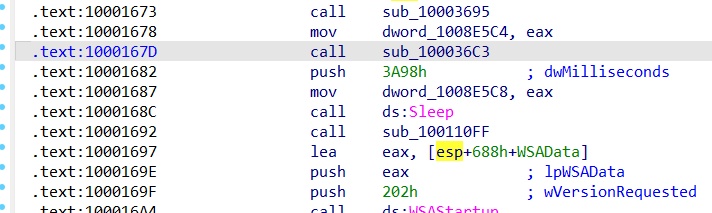
跳转到0x100101C8：



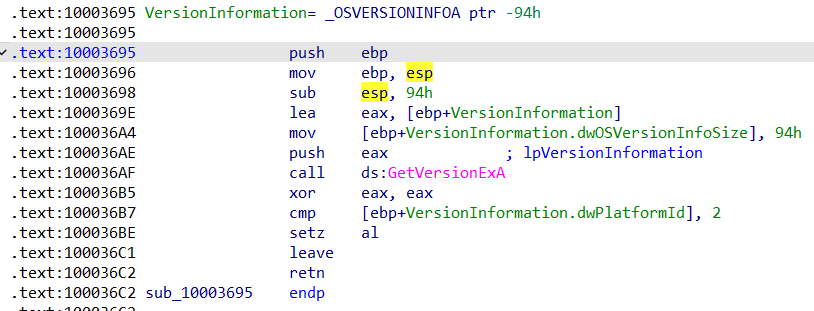
双击进入，Ctrl+X交叉引用：



显然Type为w（写入eax）是我们分析的点：



eax是call sub\_10003695的返回值：

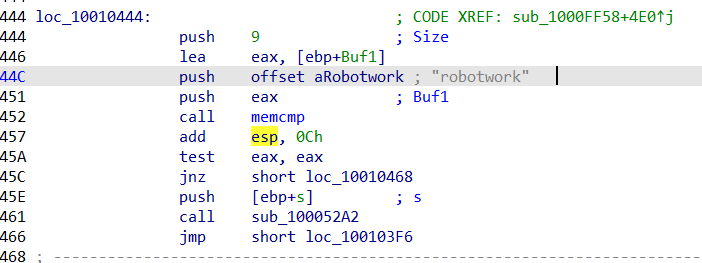


这个函数的作用是验证系统的版本。

所以dword\_1008E5C4承载的是操作系统的版本信息。

### 在位于0x1000FF58处的子过程中的几百行指令中，一系列使用memcmp来比较字符串的比较。如果对robotwork的字符串比较是成功的（当memcmp返回0），会发生什么？

跳转并找到robotwork:

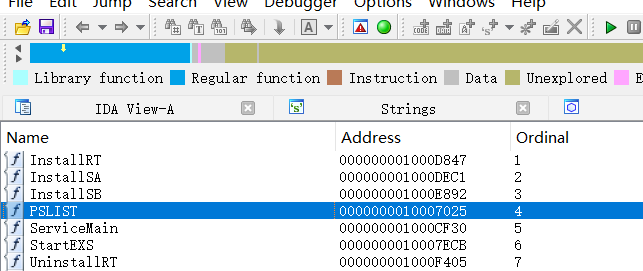


若对robotwork的字符串比较是成功的，jnz不执行，会call sub\_100052A2:

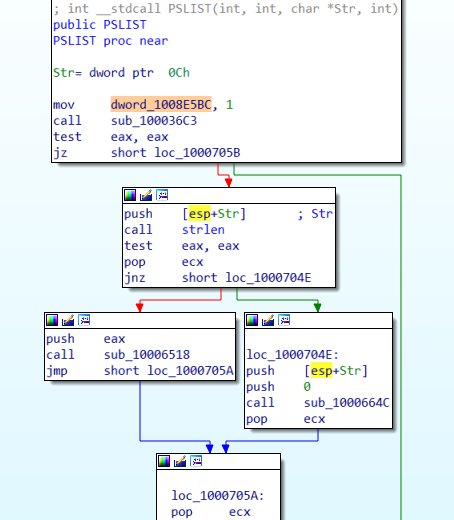


此函数会访问注册表，并调用send发送。

### PSLIST导出函数做了什么？



进入：



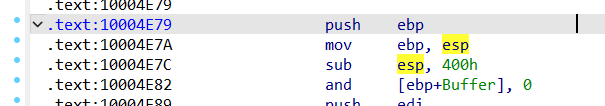
上方的 call sub\_100036C3 函数仍然是查看目前的操作系统版本信息。

对于左下的分支call sub\_10006518,以及右下的分支call sub\_1000664C,都有获取系统的进程信息的功能。

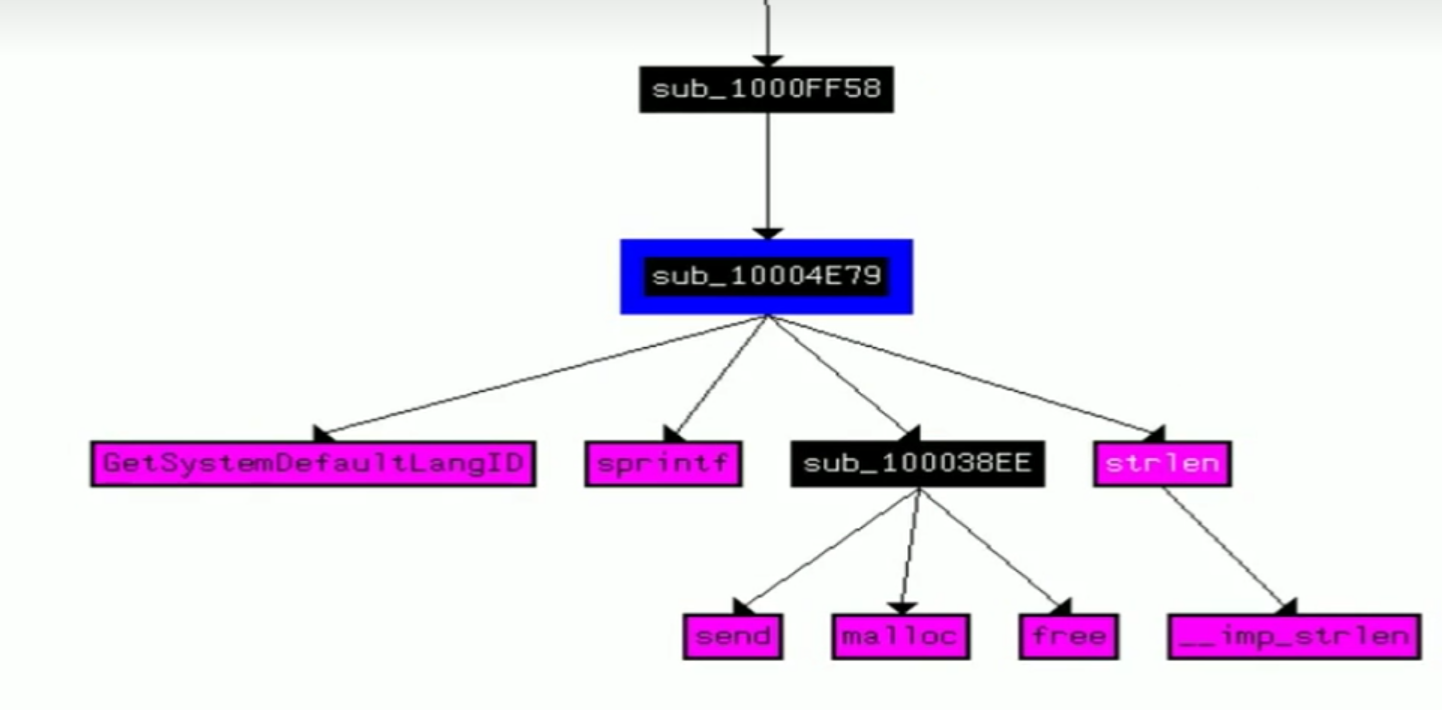
故而，我们怀疑PSLIST可能发送我们系统的进程信息，或者寻找进程的某一特定进程并进行下一步操作。

### 使用图模式来绘制出对sub\_10004E79的交叉引用图。当进入这个函数时，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？

找到函数4E79：



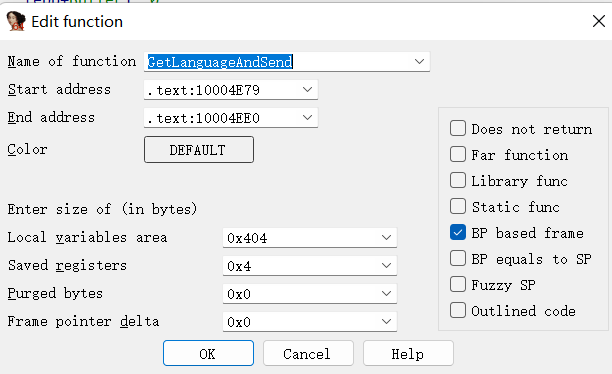
vier->graphs->User xrefs chart获取图模式：



可能被调用的函数：

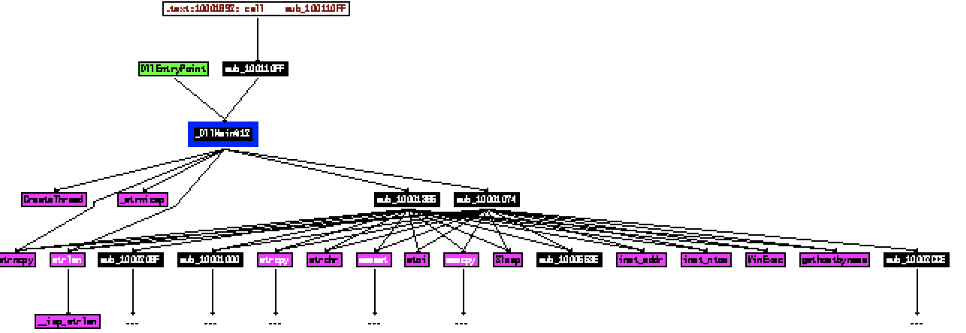
send: socket发送  
 malloc: 分配栈堆   
 free：释放堆栈   
 GetSystemDefaultLangID: 获取系统的默认语言

重命名，在左窗口edit函数：



### DllMain直接调用了多少个Windows API？多少个在深度为2时被调用？

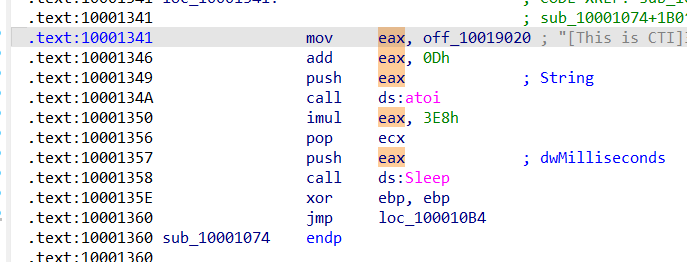
找到DLLMain进入图模式，绘制时选择深度为2：



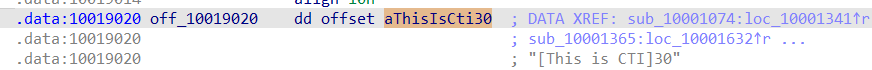
可以看到，DLLMain直接调用了strncpy，strnicmp、CreateThread 和 strlen这些API。进一步地，调用了非常多的API，包括Sleep、WinExec、getthostbyname，等等。

### 在0x10001358处，有一个对Sleep（一个使用一个包含要睡眠的毫秒数的参数的API函数）的调用。顺着代码向后看，如果这段代码执行，这个程序会睡眠多久？

找到sleep,可以看到睡眠时间取决于eax:



进入off\_10019020:



可以看到字符串有30

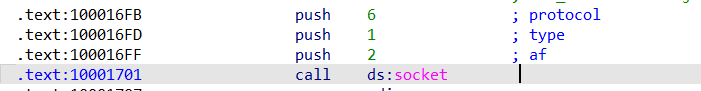
后面add eax,0Dh 的操作是为了偏移，来指向30

对于imul eax,3E8h ：3E8h是十进制1000，atoi调用结果乘以1000,即程序休眠时间为

30X1000=30000毫秒 = 30秒

### 在0x10001701处是一个对socket的调用，它的3个参数是什么？

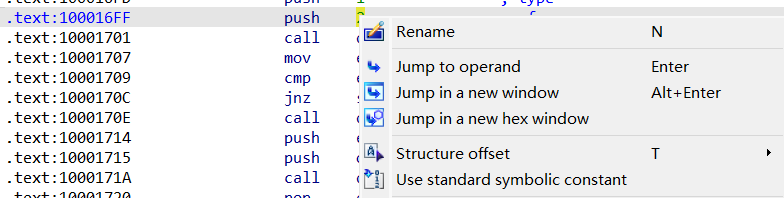
跳转：



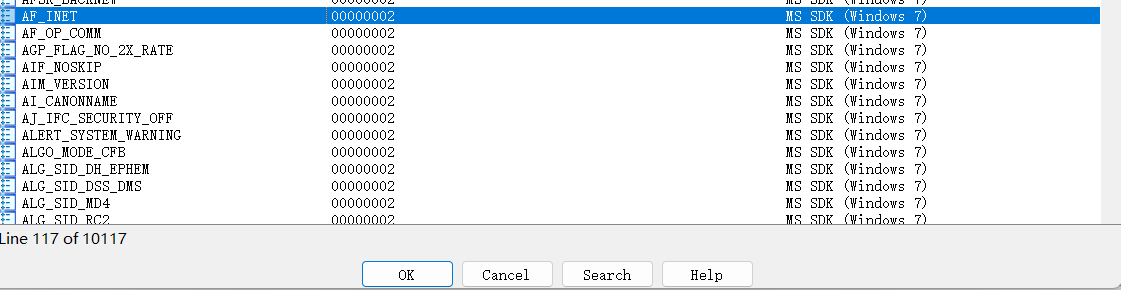
可知三个参数是：2 ，1 ，6

### 使用MSDN页面的socket和IDA Pro中的命名符号常量，你能使参数更有意义吗？修改后，参数是什么？

右键，选择Use standard symbolic constant:



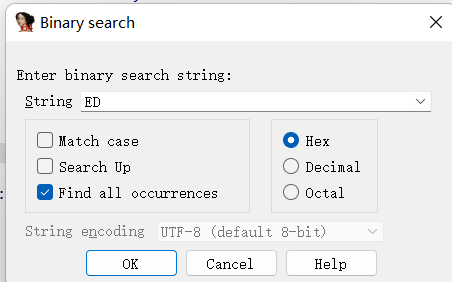
查找msdn，将1替换为SOCK\_STREAM，2替换为AF\_INET，6替换为IPPORTO\_TCP:



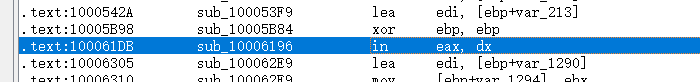
转换后更容易理解。

### 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。在这个恶意代码中被使用了吗？使用对执行in指令函数的交叉引用，能发现进一步检测Vmware的证据吗？

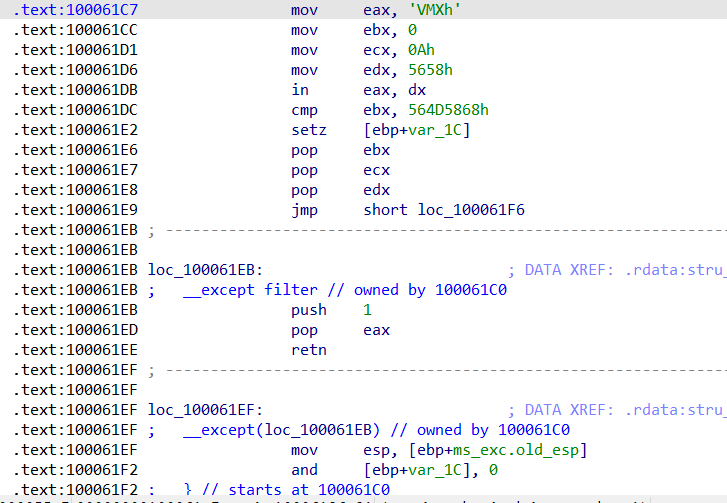
查找：



结果：

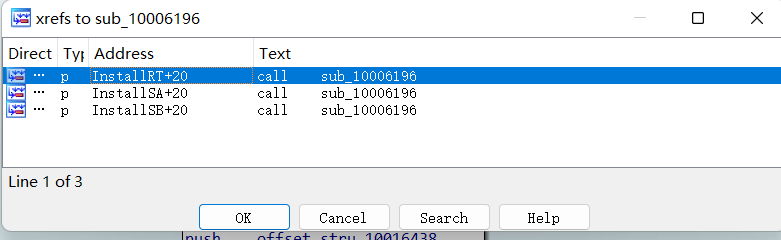


进入：



可以看到VMXh已经出现。

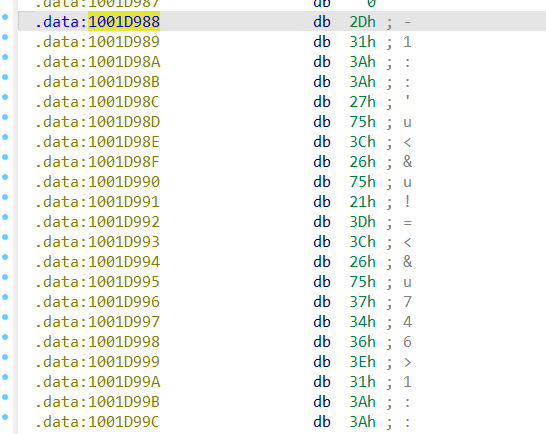
在函数头部，交叉引用：



函数判断程序是否在VMWare虚拟机中。

### 将你的光标跳转到0x1001D988处，你发现了什么？

G跳转：

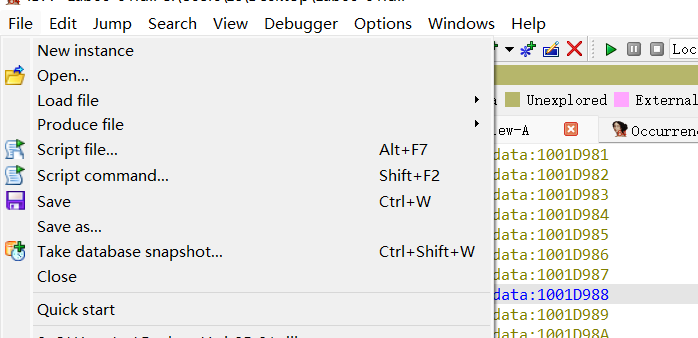


看似是乱码。

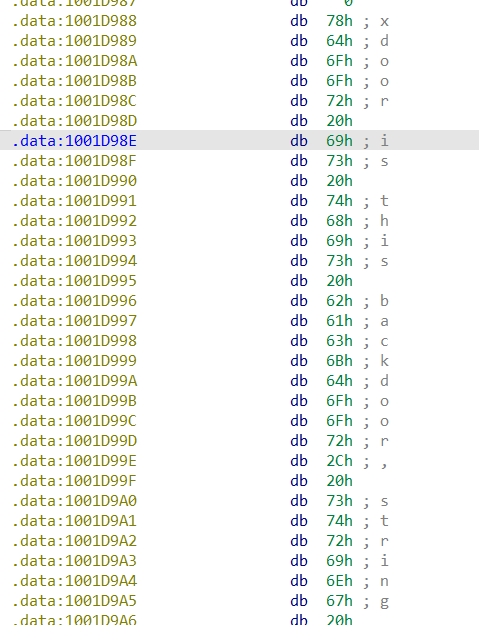
### 如果你安装了IDA Python插件（包括IDA Pro的商业版本插件），运行Lab05-01.py，一个IDA Pro Python脚本（确定光标是在0x1001D988处），在运行该脚本后发生了什么？

取出每一行乱码的字节，与0x55进行异或运算。

file->Scrip file,选择编写的idc文件：



运行后：



解密的结果为：

**xdoor is this backdoor, string decoded for Pratical Malware Analysis Lab:)1234**

### 将光标放于同一位置，你如何将这个数据转成一个单一的ASCII字符串？

代码首先获取当前屏幕的起始地址 ScreenEA()，然后使用一个循环从起始地址开始，依次读取每个字节的值。

### 使用一个文本编辑器打开这个脚本。它是如何工作的？

一旦脚本文件被加载和执行，它将自动对当前打开的二进制文件进行解密操作。

具体来说，它会对二进制文件的前 0x50 个字节进行异或解密，解密密钥是 0x55。解密后的结果将被写回到原始地址中。

### Yara编写

private global rule FileSizeAndIsPE {

condition:

filesize < 150KB and

uint16(0)== 0x5A4D and // MZ 头

uint32(uint32(0x3C))== 0x00004550 // PE 头

}

rule get {

strings:

$get = "gethostbyname"

$url = "pics.practicalmalwareanalysis.com"

$ver = "VersionInformation.dwPlatformId"

$version = "VersionInformation.dwMajorVersion"

$a = "malloc"

$b = "send"

$c = "free"

condition:

($get or $url or $ver or $version or $a or $b or $c) and FileSizeAndIsPE

}

rule remote\_shell\_session {

strings:

$remote = "Remote Shell Session"

condition:

$remote and FileSizeAndIsPE

}

rule get\_systemlang\_process {

strings:

$lang = "GetSystemDefaultLangID"

$process = "CreateToolhelp32Snapshot"

condition:

($lang or $process) and FileSizeAndIsPE

}

rule sleep {

strings:

$sleep = "sleep" nocase

condition:

$sleep and FileSizeAndIsPE

}

rule Internet {

strings:

$a = "IPPROTO\_TCP"

$b = "SOCK\_STREAM"

$c = "AF\_INET"

condition:

($a or $b or $c) and FileSizeAndIsPE

}

rule VM {

strings:

$VM = "VMXh"

condition:

$VM and FileSizeAndIsPE

}

rule xdoor {

strings:

$a = "xdoor is this backdoor, string decoded for practical Malware Analysis Lab"

condition:

$a and FileSizeAndIsPE

}

**实验结论及心得体会**

之前学习的IDA操作其实已经忘得差不多了，这次实验也算是又熟练了一遍，还结合了恶意代码分析。

IDA Python的知识在课上能听懂，但在实际操作的时候又茫然了。