第六章实验报告

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

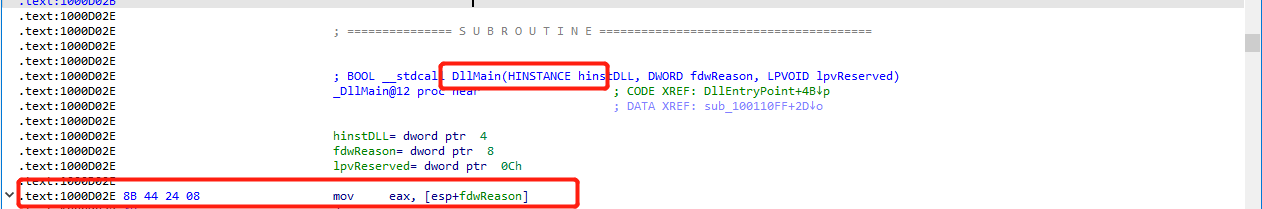
9、在同样的区域，在0x100101c8处，看起来好像dword\_1008E5C4是一个全局变量，它帮助决定走哪条路径，那恶意代码是如何设置dword\_1008E5C4的呢？

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？

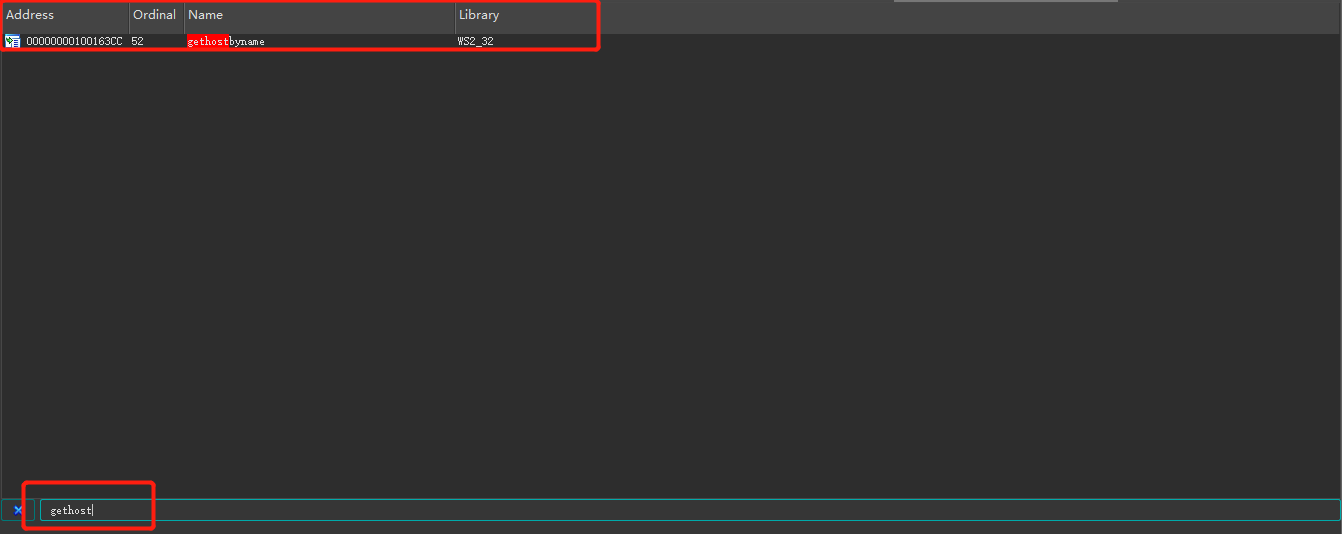
会有一个访问注册表的句柄，访问其具体workstime的值并通过一个套接字S发送给远程会话

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

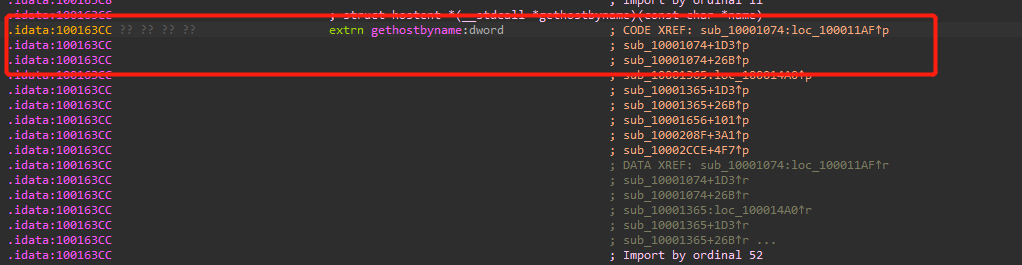


（小小的更新了一下背景颜色

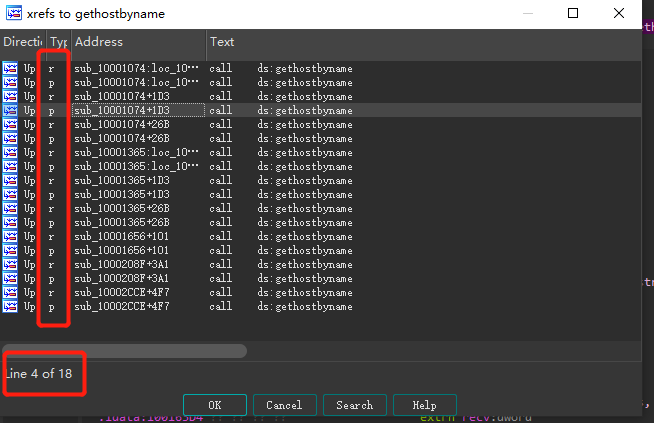
然后查看导入函数，直接点击View---Open Subviews之后找到Imports，在导入表中搜索到我们需要的dll--gethostbyname，可以查看到，他的位置是0x100163CC



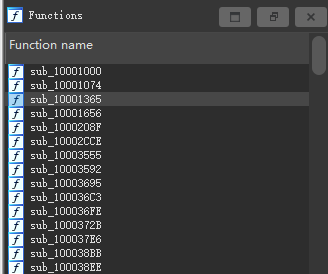
双击后可以查看函数详情



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。

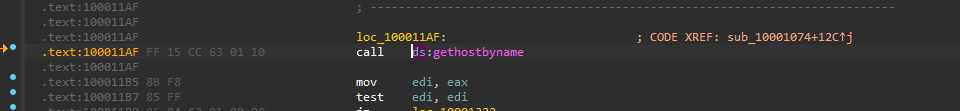


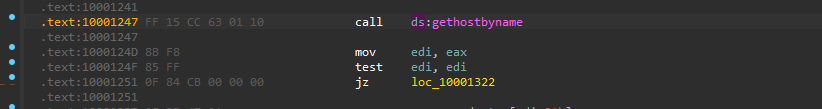
但是，有一些调用可能是发生在一个函数中的，这时候就需要我们手动去检查：下图是函数在.text段的地址：

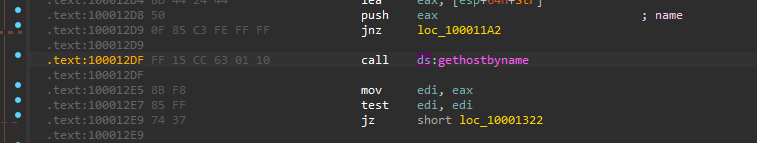


我们通过一个一个点击交叉引用表去查看发现：

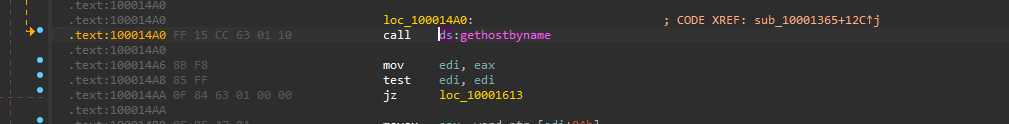
前三组交叉引用都在sub\_10001074函数中：

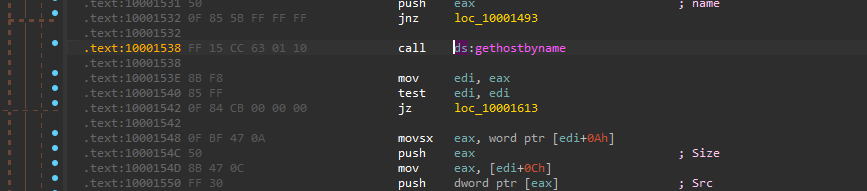


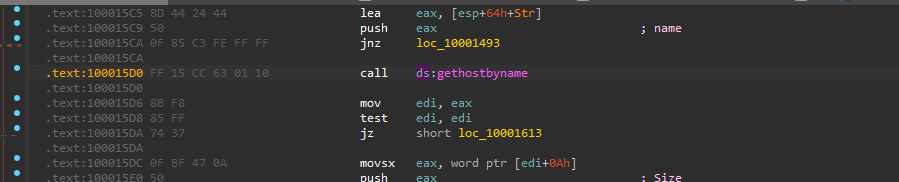




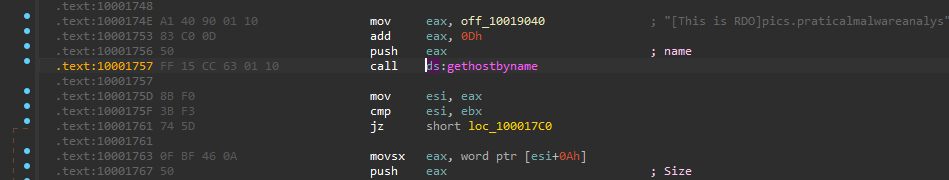
这三个引用发生在sub\_10001365函数中：



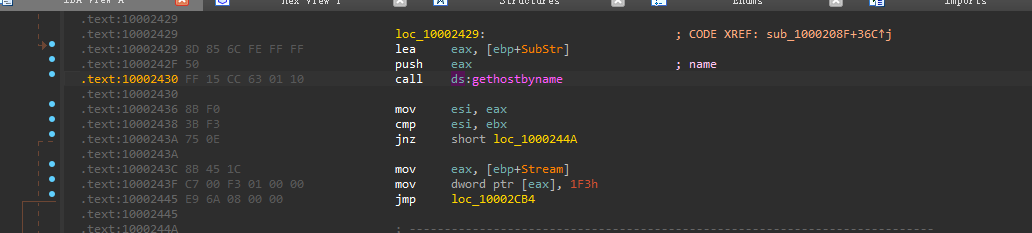




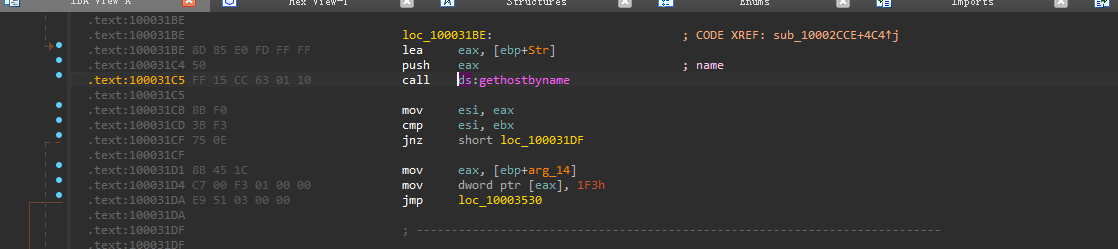
这次引用发生在sub\_10001656



这次引用发生在sub\_1000208f

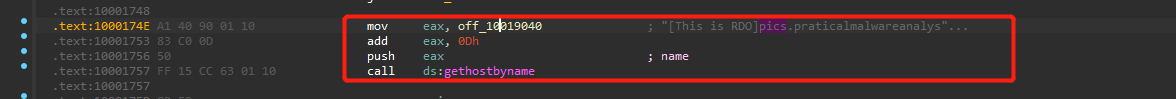


这次引用发生在sub\_10002CCE

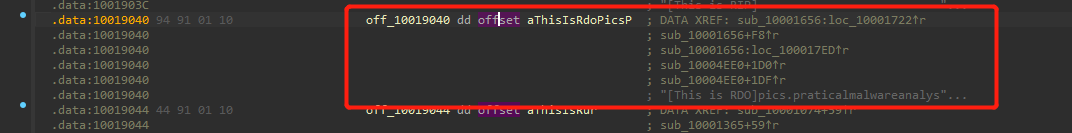


总共在五个函数发生了这个dll的调用

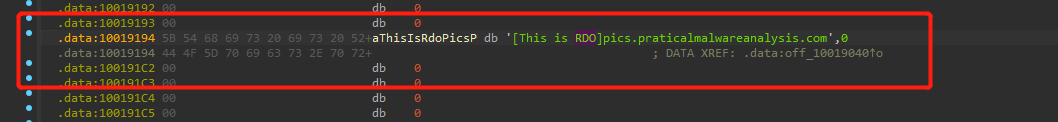
重点关注0x10001757处的函数调用，点击之后找到为gethostbyname的参数传递过程：



点击对应地址查看内容：

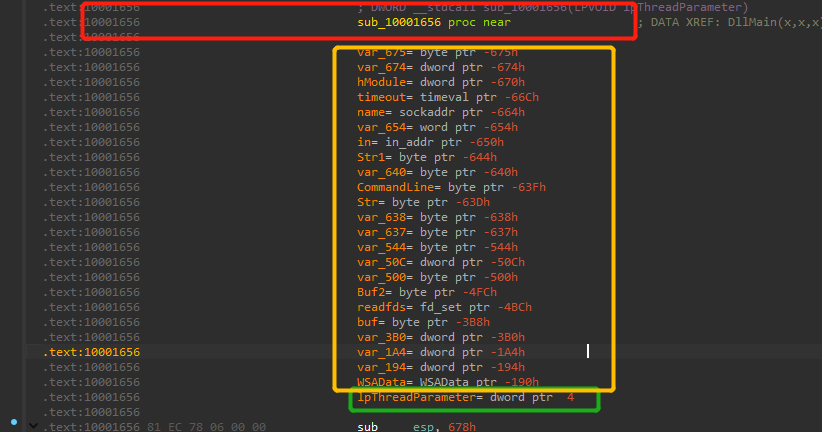


再次点击偏移信息：发现了这个熟悉的网址pics.practicalmalwareanalysis.com

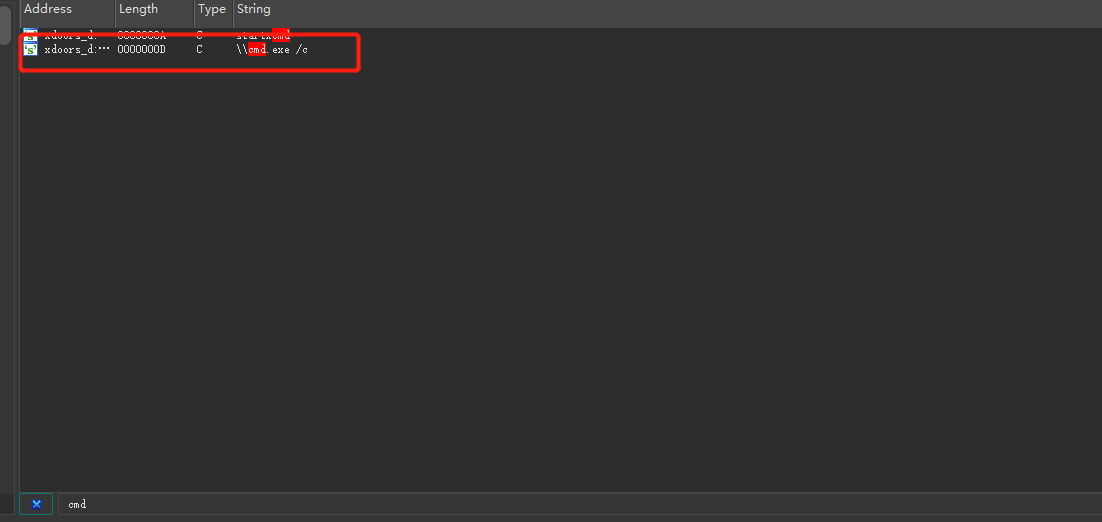


想要分析0x10001656处的代码，就点击g并输入想要前往的地址即可：

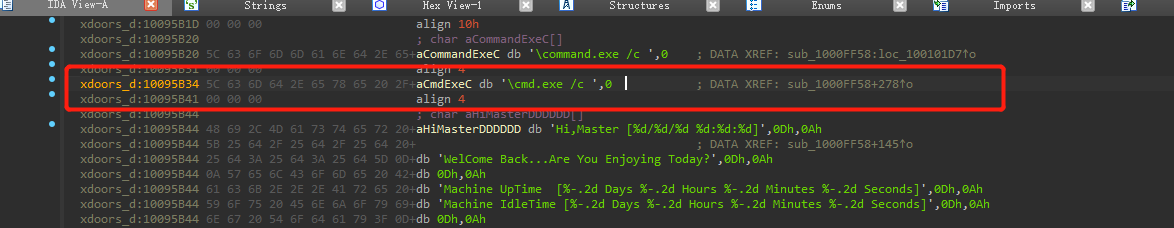
在0x10001656，IDAPro已经帮我们分析好了局部变量以及参数，简单的数一下可以发现总共有23个临时变量和1个参数



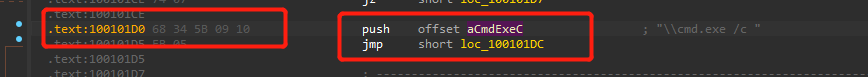
查找特定的string还是和之前的策略一致，利用ida提供的搜索机制进行搜索，可以轻松找到需要的cmd.exe字符串



点击进去就可以看到这个字符串的位置位于0x10095B34的位置

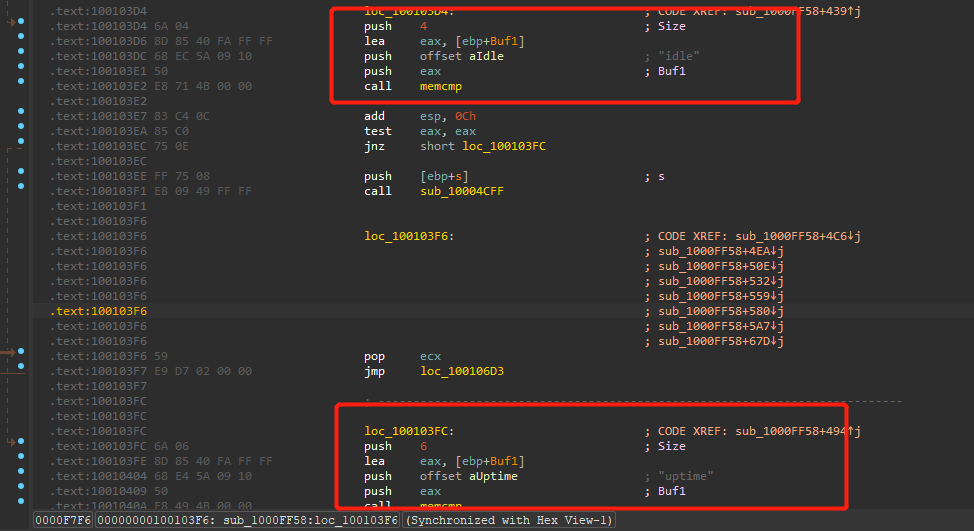


为了分析对这个字符串的引用还是按住ctrl+x随后来到了如下位置：字符串被压栈后跳转到别的地方进行继续执行



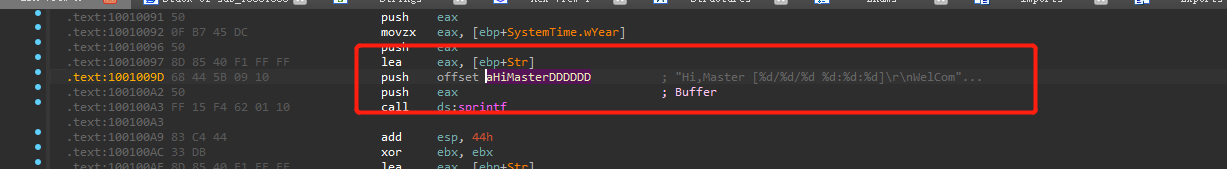
按照题目要求观察上下文：首先查看下文，发现代码进行了一系列的比较操作：包括与quit、exit、cd、idle、uptime、minstall、inject等进行比较，非常像在操控命令行程序。



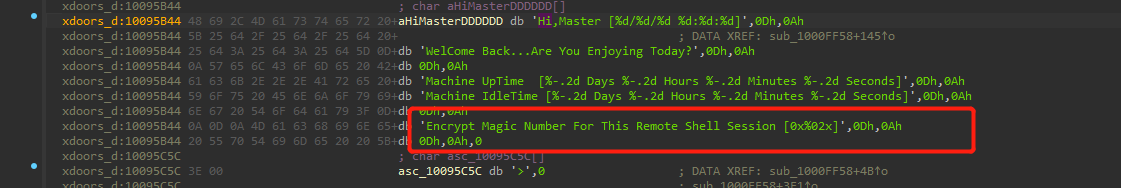




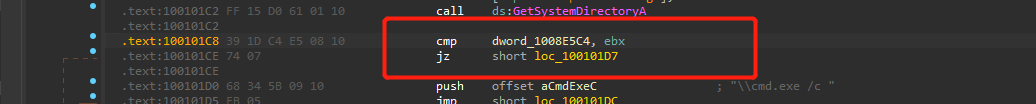
查看上文：发现引用了一大段字符串，我们还是点进去查看一下完整的字符串内容



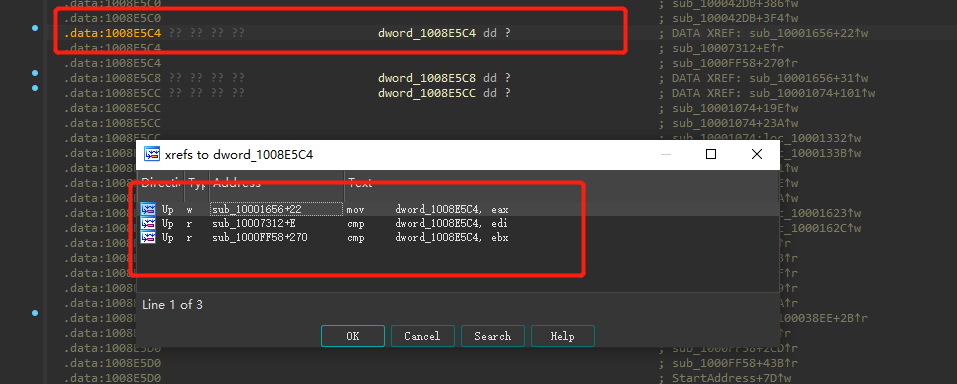
虽然程序我看不太懂，但是英文还是认识一些，可以判断这个阶段应该是想远程连接一个会话（remote shell session）



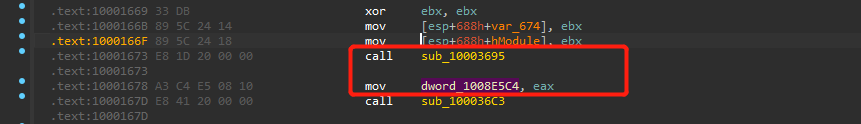
找到对应的全局变量的位置，随后双击进入查看详情。



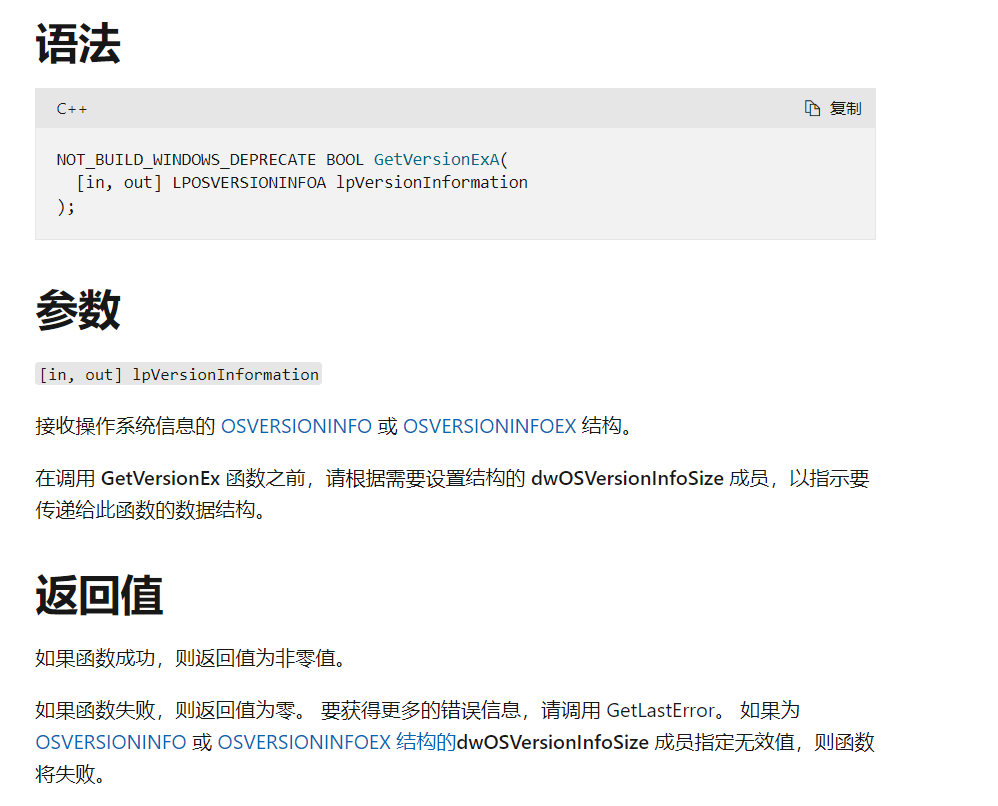
点击ctrl+x进行交叉引用，发现有三个地方对这个变量进行了引用，但是只有一个地方对其进行了改值操作，我们点击进入

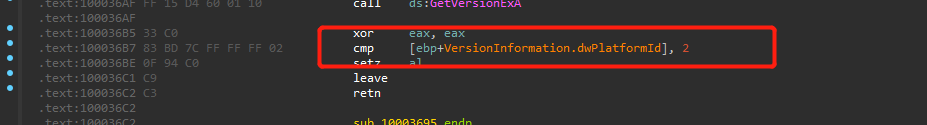


可以看到，程序将上一个调用的函数的值赋值给了这个全局变量，于是我们点击进入该函数：



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。



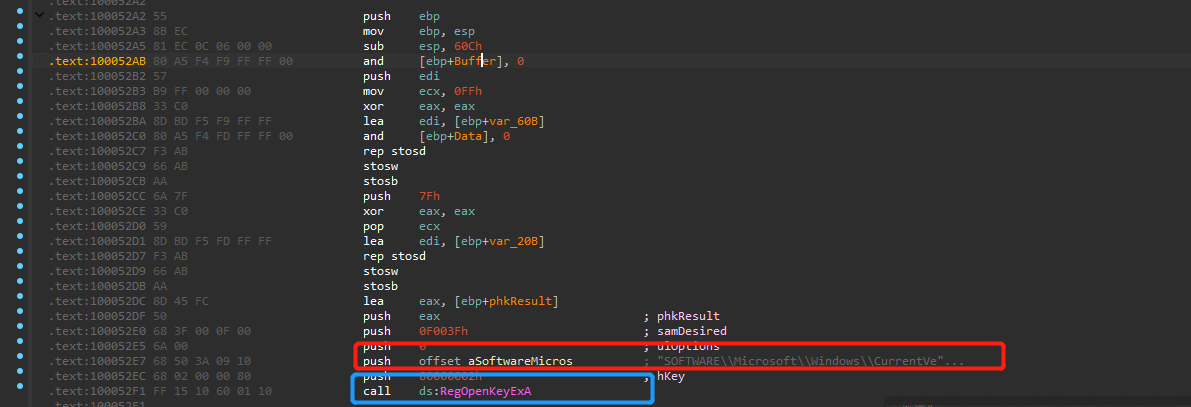


然后，我们来分析这个函数一系列比较中的一个：通过ida给我们的注释找到robotwork我们发现，如果字符串等于robotwork，就会跳转到绿色的位置并执行相应函数，我们点击进入该函数：

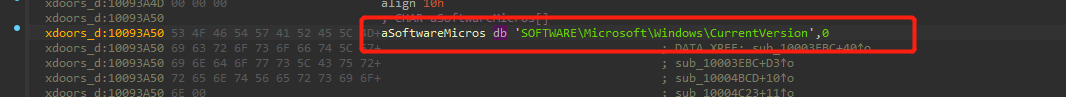


发现它将一个字符串压入了栈并调用了函数，随后就又调用了RegOpenKeyExA，同样通过查看微软官方文档来获取其定义：这是为了打开一个注册表的句柄。





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。

