Lab7实验报告

专业：信息安全 学号：2011395 姓名：魏伯繁

### ***一、实验工具：***

IDAPro7.0 IDAPro7.7 win10虚拟机 winxp虚拟机

Process monitor

### ***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：Lab7-1

1、当计算机重启后，这个程序如何确保他继续运行？

**这个程序将自己注册为服务，并制定开机后自动加载运行来保证持续运行**

2、为什么这个程序会使用一个互斥量？

**用来保证这个程序只会被加载运行一次**

3、可以用来检测这个程序基于主机特征是什么？

**该恶意代码创建了一个名为MALService的服务，并且创建了一个互斥量HGL345**

4、检测这个恶意代码的网络特征是什么？

**szagent为Internet Exploer8.0以及访问的URL为<http://www.malwareanalysisbook.com>**

5、这个程序的目的是什么？

**在2100年1月1日零点创建20个线程不停的访问<http://www.malwareanalysisbook.com>进行DDOS攻击**

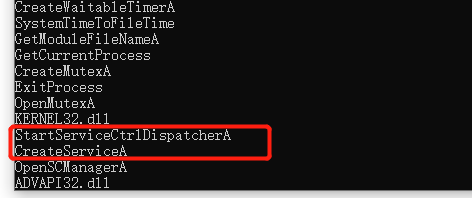
6、这个程序什么时候完成执行？

**这个程序会一直运行直到2100年1月1日不断的完成攻击，并且攻击也是一个无线循环，永远不会停止。**

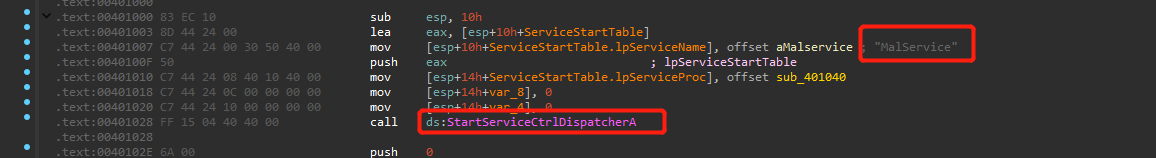
分析过程：

其实保证计算机重启后运行恶意代码的方法大体只有两种：就是修改注册表的启动项或者将这个恶意程序注册为服务进行持久化驻留。

在简单的静态分析中我并没有找到它修改了注册表的证据，但是却观察到了一些关于服务的信息：下图是在strings工具中运行得到的结果



并且，当使用IDAPro打开代码后会在程序的开头观察到如下语句：很明显，该程序将自己注册为了一个服务

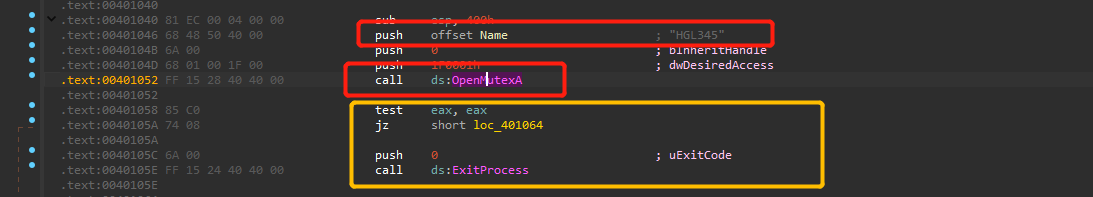


查看微软官方文档给出的定义，确实是将这个进程与服务关联，并且指定了服务名为Malware，同时设置了服务控制函数为sub\_401040,也就是这个服务要执行的函数。



随后，我们进入sub\_401040函数查看其执行过程：

在一开始，这个服务试图去使用一个句柄获取一个互斥量HGL345，并且根据获取的结果在黄色的方框内进行判断，如果获取成功，也就是返回值不为零就退出程序，反之如果没有获取到就去到loc\_401064继续执行。



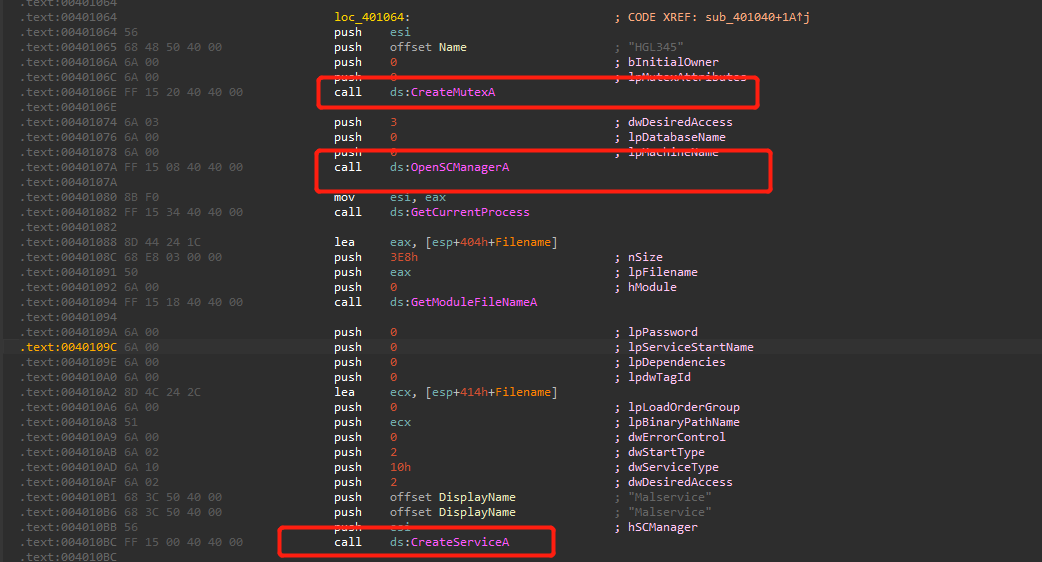
查看在loc\_401064部分的代码：这个函数的核心由五个函数组成，其中最重要的函数就是最后的CreateService用来注册一个服务。

但在这之前要先调用CreateMMutexA函数来创建一个互斥量，以便以后再执行该进程时不会重复注册服务。

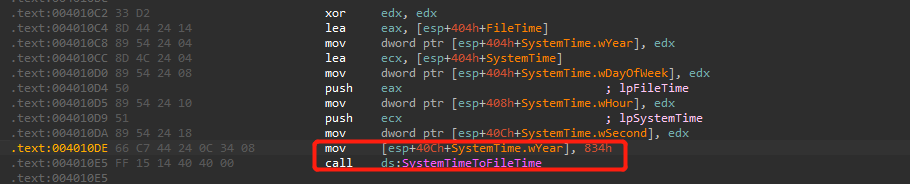
随后调用OpenSCManagerA函数打开服务控制管理器句柄，这样该进程就可以添加服务。

随后调用GetCurrentProcess来获取当前的进程，随后利用返回值调用GetModuleFileNameA来加载进程的路径名

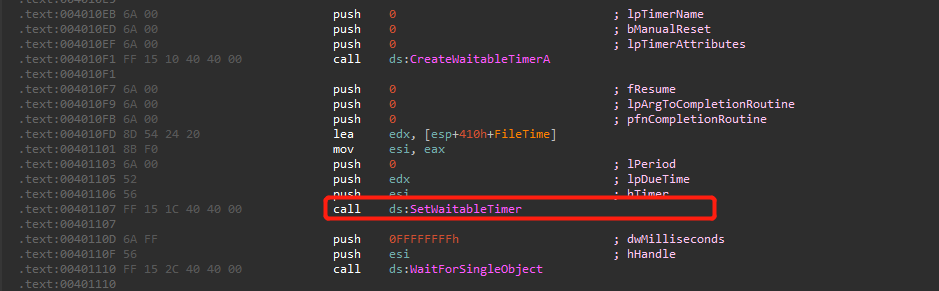
在获取足够多的信息之后，就可以将这些信息作为参数压栈并调用CreateServiceA来创建一个进程。将刚刚获取到的路径作为LpBinaryPathName作为参数就可以指定可执行文件的路径，dwStartType设置为0x02意为SERVICE\_AUTO\_START也就是开机自启动

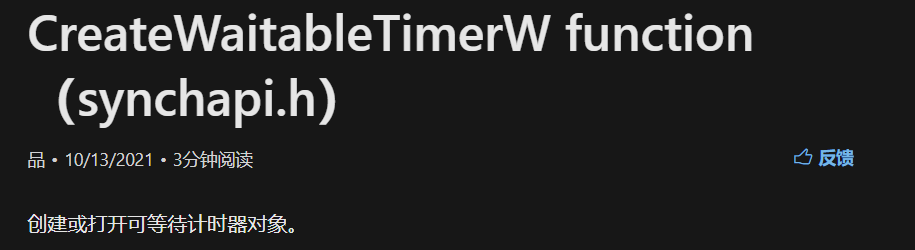


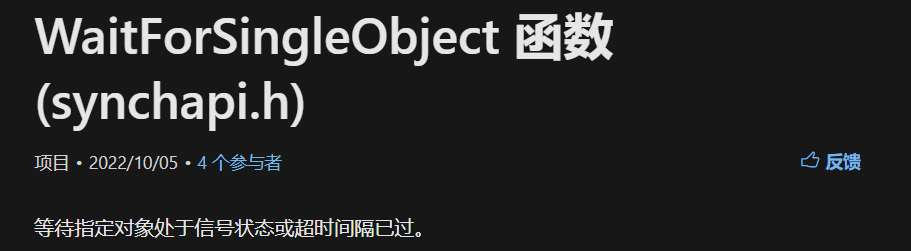
接下来，代码设置了一个时间，这个时间首先全部被设置为0，然后将year位置设置为了16进制数834h，也就是10进制的2100，最后进行了将SystemTime转换为FileTime。也就是说恶意代码设置了一个时间点，也就是2100年1月1日的0点



随后是三个与时间有关的函数，第一个函数用来创建或打开可等待计时器对象，通过这个函数我们获得了一个计时器句柄，随后调用SetWaittableTimer并使用之前的FileTimer作为参数调用该函数设置一个定时器，在调用WitForSingleObject后进入等待，也就是等待指定的时间后在做出动作



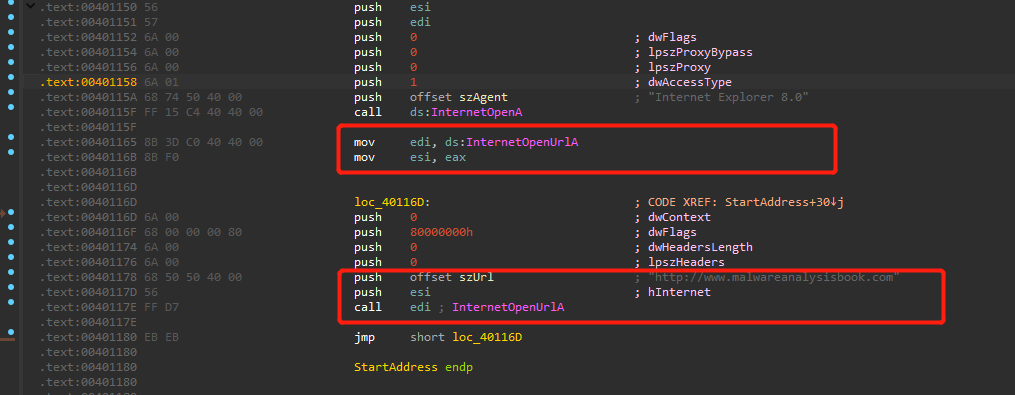




然后是一个简单的for循环结构，循环的次数在结构开头指定了为20次，每一次执行都会创建一个新的线程，线程的执行起始位置在0x00401050



然后，我们来到线程执行的线程位置，该位置使用了一个InternetOpenUrlA用来使用指定的浏览器版本Internet Exploer8.0来打开指定的url，并且注意到，虽然整个过程没有任何判断条件，但是在代码的最后有一个jmp语句并且是向前跳抓紧，于是不难分析这个程序是一个死循环，他会不停的访问该网页，也就是我们熟悉的DDos攻击



第二部分：Lab7-2

1、这个程序如何完成持久化驻留？

**并没有完成持久化驻留，该文件没有修改注册表或注册服务或无线循环的迹象**

2、整个程序的目的是什么？

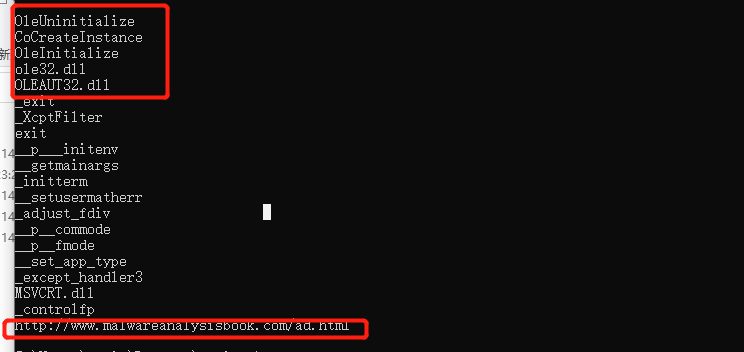
**使用COM对象弹出一个网页**

3、这个程序什么时候完成执行？

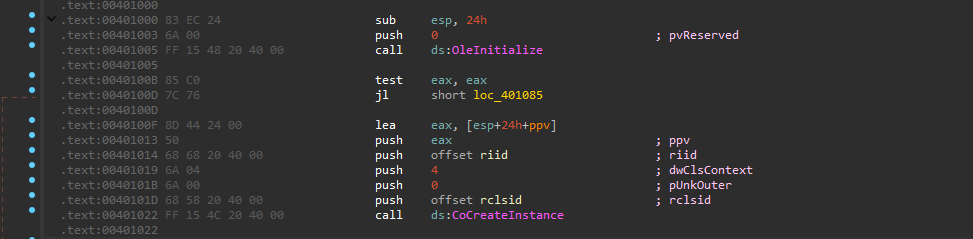
**在将网页弹出后会自动退出完成执行**

分析过程：

首先利用strings工具查看其字符串内容，可以比较清晰的看到这个恶意代码首先进行了网络访问，其次他还使用了COM对象



进入主函数，我们将看到两个函数，其中第一个函数对COM对象进行了初始化，第二个函数则是创建了一个COM对象的实体，返回的对象也就是eax中的内容即是ppv中保存的内容

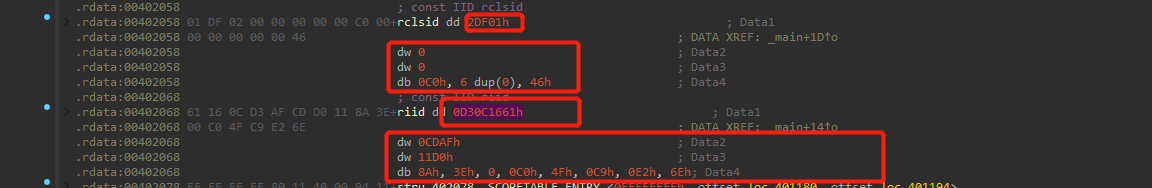


检查两个标识符riid和rclsid：，通过详细查看其内容为：

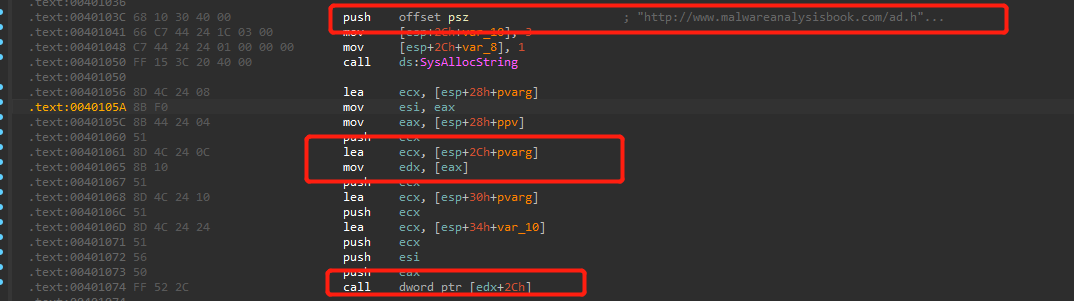
riid的内容为：根据格式填补上必要的0之后为：0D30C1661-CDAF-11D0-8A3E-00C04FC9E26E

rclsid的内容为：0002DF01-0000-0000-C000-000000000046

不难发现，这个id的内容与课本上的例子是一样的，也就是调用IWebBrowser和Internet Exploer访问特定的网站。



而网站的内容也在字符串中体现了：其中2C就是Navigate的函数偏移，当这个函数被调用时，Internet Explorer将访问指定的url：<http://www.malwareanalysisbook.com/ad.html>



第三部分：LAB07-03

1、这个程序如何完成持久化驻留？来确保计算机重启后它能继续运行？

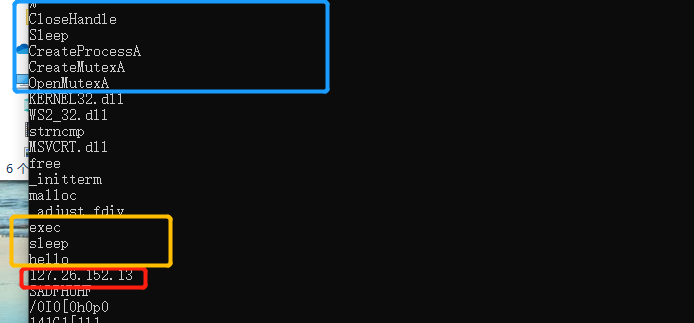
2、这个恶意代码的两个明显的基于主机特征是什么

3、这个程序的目的是什么

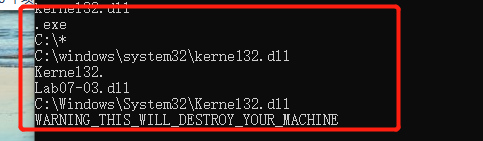
4、一旦这个恶意代码被安装，你如何移除它？

分析过程：

首先采用基本动态分析进行检测，在dll中发现了一系列联网迹象，比如包括一个ip地址。并且有三个似乎和选择的操作有关（用黄色框表示），还有一部分和IAT有关，基本的操作包括休眠、新建一个进程、创建一个互斥量以及访问一个互斥量。

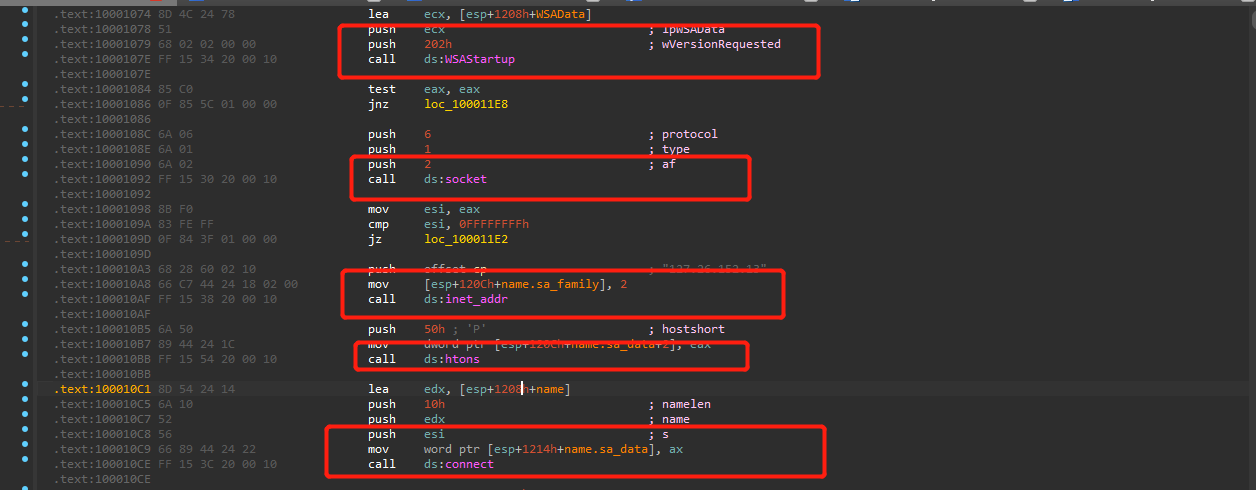


马上对exe文件进行基本动态分析：在这张照片中很明显我们可以知道这个可执行文件访问了C盘下的文件，并且一个路径似乎可以包括C盘下所有的路径，于是这个可执行文件大概率包括是搜索一个文件，并且很意外的其中还包括本次实验所用的dll，并且还有两个很相似的路径只不过一个是kernel32.dll另外一个是kerne132.dll，其实这个时候我们可以猜想一下这个恶意代码的功能，很有可能就是要以假乱真，而替换的对象就是Lab07-03.dll，被替换的对象就是kernel32.dll，Lab07-03.dll会被加载成kerne132.dll，当然这些只是猜想，我们可以将分析思路引向这边。

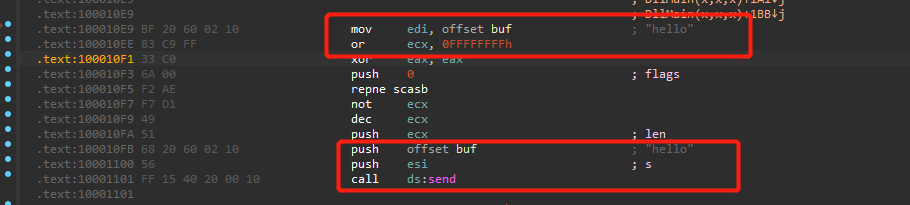


其实根据书中的提示，已经大概可以猜想到这个恶意代码的内容可能是非常复杂的，所以我们才要尽可能的抓住本质，不要陷入到局部的细节中。

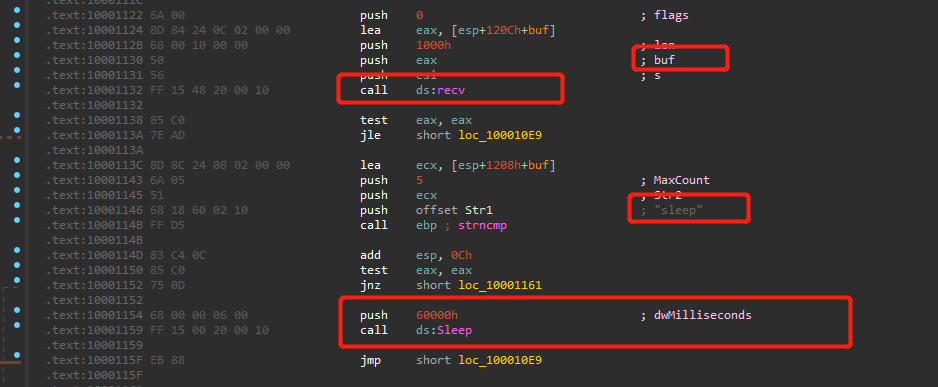
我们首先对dll文件进行分析：一进来首先看到的就是一系列的和网络有关的API函数，在本学期刚刚学过计网的基础上理解这些函数并不困难，其实就是初始化dll、绑定套接字的过程，说明dll实现了交互功能，即与远程进行通信，这很有可能就与之前出现过的ip地址密切相关



我们再往下看看到了一个send函数，也就是说被感染的主机要向远程发送消息，可以看到发送的缓冲区由buf表示，发送的内容为一个hello字符串。这里的作用应该是向恶意端报告这台主机已经被控制了。



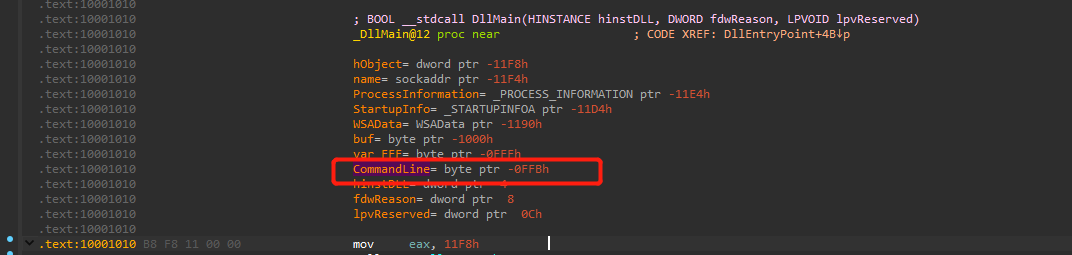
然后接下来是一个recv函数用来接收远程发来的消息，并且根据远程发来的消息不同本机会做出不一样的反应，比如如果远程发来了sleep指令那么就休眠60秒



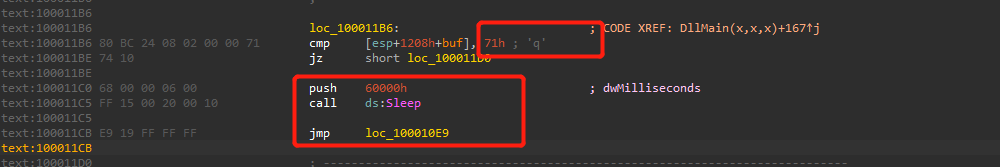
如果发来的字符串是exec那么就执行一系列的动作，这个动作的终极目的是启动一个进程，要创建的进程由lpcommandline给出，通过期传递方式我们发现他是一个临时变量存储在栈中



果然在函数开头时我们看到了这个临时变量，那么这个临时变量的值是什么呢？不妨仔细观察一下临时变量的分布状态，buf的位置为-0FFFh而CommandLine的位置为-0FFBh，很明显CommandLine在读取buf中的内容，所以说，要被执行的命令的一部分参数就来源于接受缓冲区中保存的东西

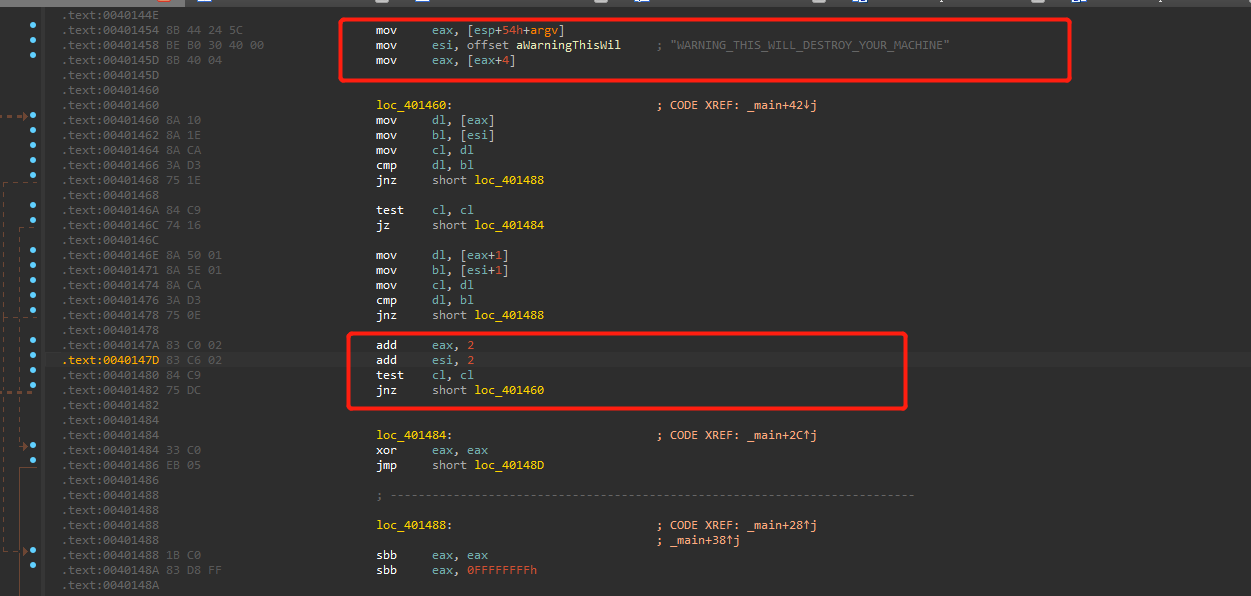


最后，海域一个q指令，应该是代表着退出，程序会休眠60秒然后跳转到退出位置的代码进行执行

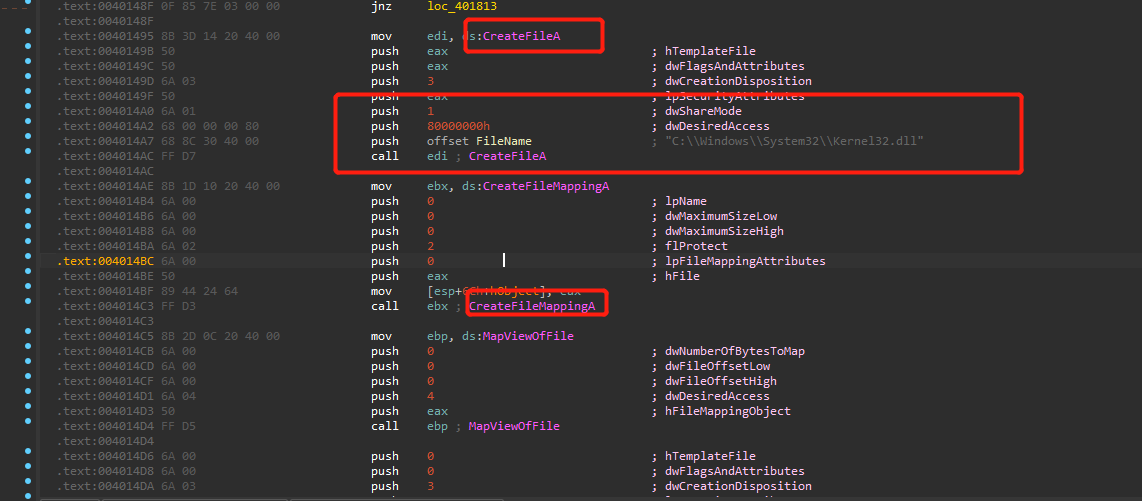


dll文件基本分析结束了，我们马上进入exe文件的分析，不出所料，exe文件被大块大块的函数分割，有两个大块的函数甚至都很少有API函数调用，于是我们可以分析得知，该块函数很可能是实现了特定的恶意功能，是恶意代码编写者实现的重点工作，但由于缺乏提示，直接分析这段程序会效率很低，所以我们依然从右线索的部分开始分析。

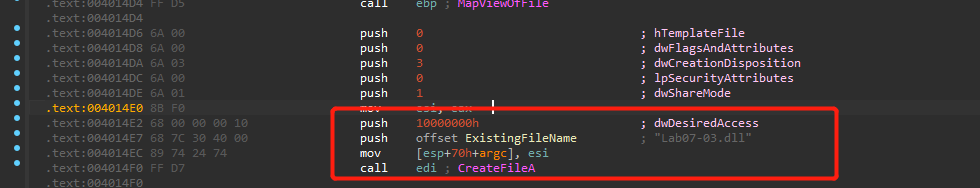
首先我们可以看到，如果想要正确的运行该程序，必须要输入相应的参数，空参的情况下程序会自动退出，而参数正是使用strings工具时查看到的字符串，通过输入参数来让exe程序正常执行，后面的大端程序其实就是将输入的参数与内存中的字符串逐字节比较，如果吻合那么就进入程序执行。



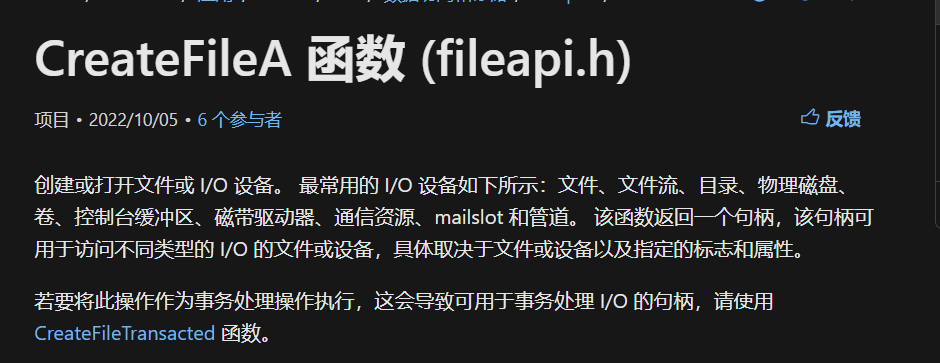
接下来，恶意程序对本地的文件进行了操作，这也符合我们在使用strings工具后得到的假设，恶意代码首先加载并映射如内存的文件是kernel32.dll



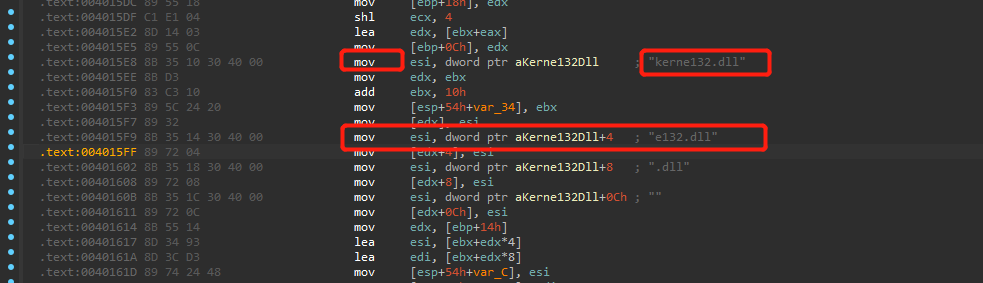
随后也对本进程文件Lab07-03.dll进行了操作



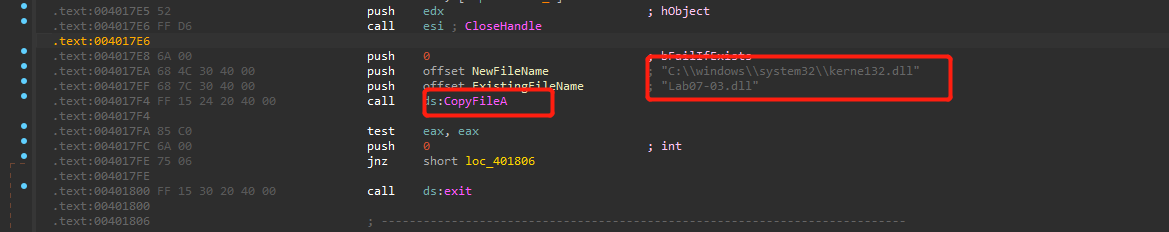
通过阅读微软官方文档可以得知，以上语句打开了一个IO设备，并且准备对该进程相对应的Lab07-03.dll进行操作



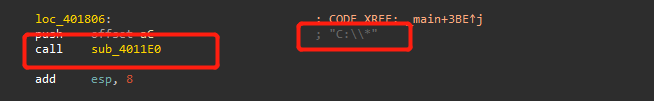
接下来进入一些函数的主题，我们可以看到程序对指定的字符串“kerne132”进行了mov操作，这是一个很怪异的字符串，他和本身windows自带的动态链接库kernel32.dll非常相近，并且恶意代码还将这个字符串赋值给了esi，esi最后将其写入了一个内存地址中



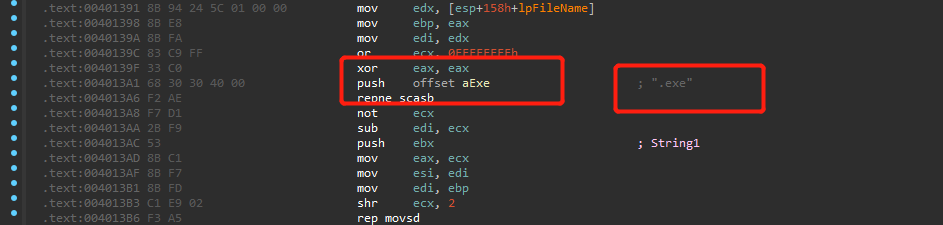
随后的一个API调用让我们成功得知这个假冒的kernel32是由谁产生的，恶意代码通过一个CopyFile函数将Lab07-03.dll的内容复制到指定的新建的冒牌动态链接库中

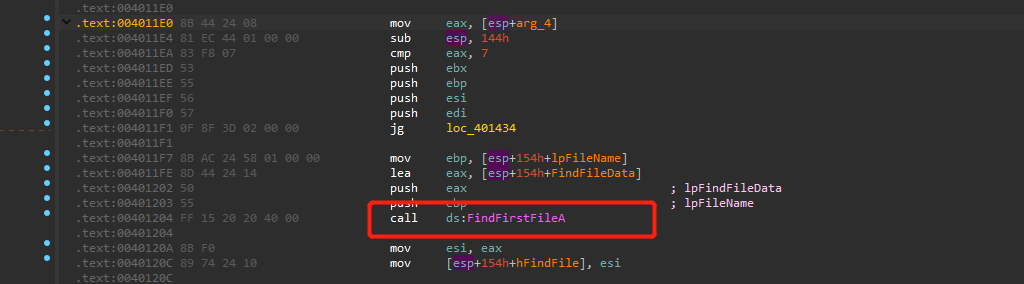


紧接着，一个代表着C盘下所有路径的字符串出现了，并且恶意代码还将该字符串压栈并调用了一个函数

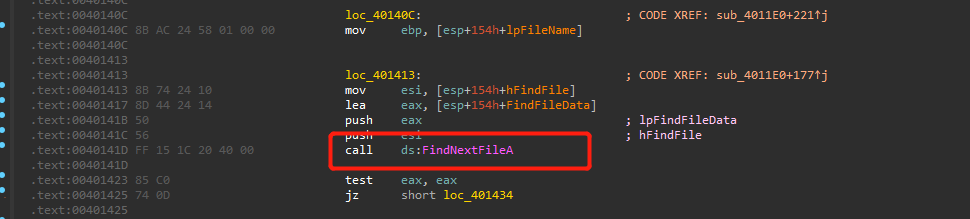


而在接下来的函数过程中，我们可以发现一些基于主机的信息，FindFile函数证明这个恶意代码在搜索什么符合规定的文件，而其中唯一出现的字符串就是.exe，也就是说，恶意代码正在遍历C盘下所有的.exe文件

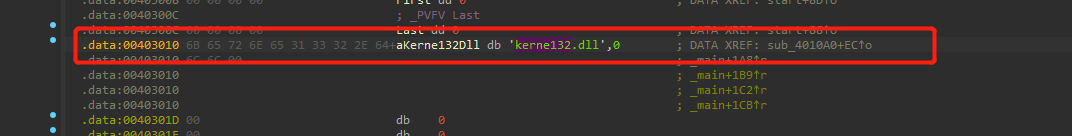




因为它由一个FindNext函数，所以他会一直递归搜索下去直到遍历完C盘所有文件为止

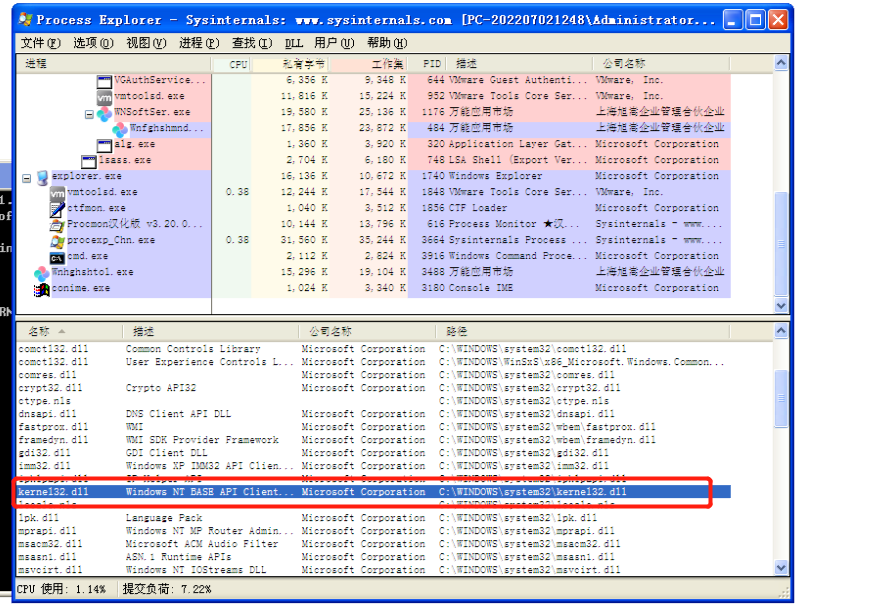


那么要遍历做什么呢，结合前面的分析以及对字符串的解码，我们不难得出结论：恶意代码会遍历C盘下所有的exe文件，当他们试图去连接连接kernel32.dll时这个操作不会直接进行，而是会连接到由Lab07-03.dll函数所复制生成的kerne132.dll，由kerne132.dll负责相应动态链接库的加载。



那么，如何验证这个结论是否正确呢？我在xp上真的运行了这个程序，并且使用process explorer进行监视，查看其加载的动态链接库的变化情况：

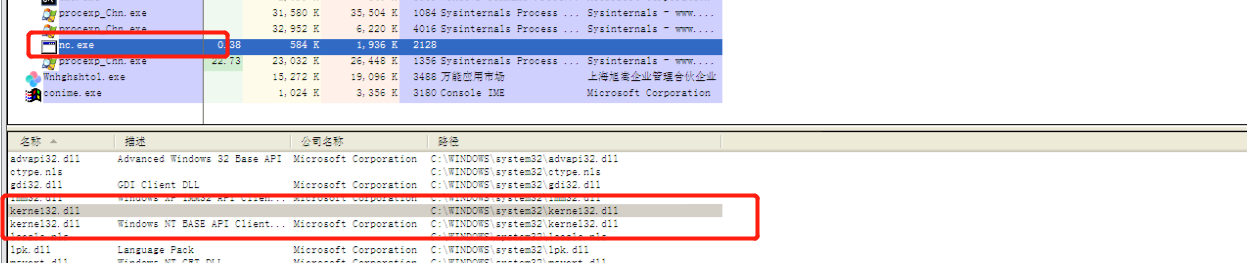
首先，在执行恶意代码之前，查看C盘文件下的exe的动态链接库的引用情况，毫无意外的，kernel32动态链接库被征程引用，没有异常



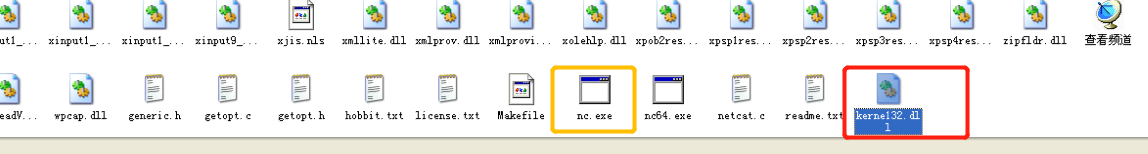
我们再挑选一个系统dll发现确实如此 其中的描述WINDOW NT BASE API更加验证这个dll是干净的



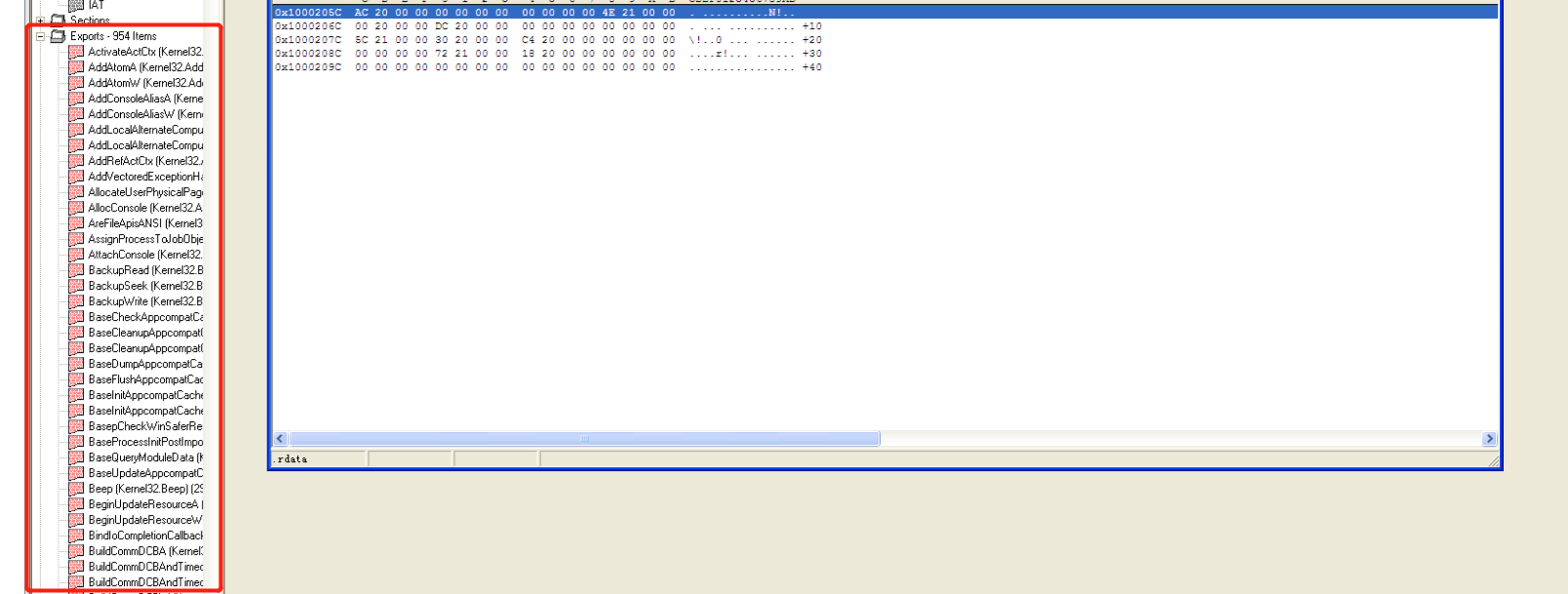
随后，使用Lab07-03.exe WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE调用exe，再打开我们在上上个实验中使用过的配置在C盘下的nc.exe网络连接程序，果然观察到有一个缺少描述的kerne132.dll被加载了，但是正常的kernel32.dll依然存在在引用的动态链接库中



可以看到，新生成的kern132.dll已经在System32目录下了，黄色框框内是测试使用的exe文件以证明其确实在C盘目录下



使用PEBroswer打开Kerne132.dll，发现他的导出表非常非常庞大，也就是说冒名顶替的dll无法完成exe文件要求的API函数加载动作，依然需要映射到原有的Kernel32中进行执行，这也是为什么干净的kernel32.dll依然会存在在进程正在使用的动态链接库列表中原因。



第二部分：Yara规则编写：

Lab07\_01的yara规则：

import "pe"  
rule IsPE{  
 meta:  
 description = "检查文件是否为PE文件"  
 author="wbf"  
 date="2022-11-8"  
 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="2022-11-8"  
 condition:  
 filesize<500KB  
}  
  
rule mayBeMaleware{  
 meta:  
 description = "检查文件小是否大概率为maleware文件"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = { E8 00 00 00 00 }  
 $b = "malware" nocase  
 condition:  
 $a at pe.entry\_point or $b  
}  
  
rule Lab0701exe{  
 meta:  
 description = "检查改文件是否大概率与Lab07-01的文件属于同一类型的恶意代码"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = "SystemTimeToFileTime"  
 $b = "CreateWaitableTimerA"  
 $c = "CreateMutexA"  
 $d = "StartServiceCtrlDispatcherA"  
 $e = "InternetOpenUrlA"  
 $f = "MalService" nocase  
 $g = "HGL345"  
 $h = "http://www.malwareanalysisbook.com"  
 $i = "Internet Explorer 8.0"  
 condition:  
 (6 of them) and (IsPE and smallFileSize) or (mayBeMaleware)  
}

Lab07-02的yara规则：

import "pe"  
rule IsPE{  
 meta:  
 description = "检查文件是否为PE文件"  
 author="wbf"  
 date="2022-11-8"  
 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="2022-11-8"  
 condition:  
 filesize<500KB  
}  
  
rule mayBeMaleware{  
 meta:  
 description = "检查文件小是否大概率为maleware文件"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = { E8 00 00 00 00 }  
 $b = "malware" nocase  
 condition:  
 $a at pe.entry\_point or $b  
}  
  
rule Lab0702exe{  
 meta:  
 description = "检查改文件是否大概率与Lab07-02的文件属于同一类型的恶意代码"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = "http://www.malwareanalysisbook.com/ad.html"  
 $b = "OleUninitialize"  
 $c = "CoCreateInstance"  
 $d = "OleInitialize"  
 $e = "ole32.dll"  
 $f = "OLEAUT32.dll"  
 $g = "HGL345"  
 condition:  
 (4 of them) and (IsPE and smallFileSize) or (mayBeMaleware)  
}

Lab07-03.dll的yara规则

import "pe"  
rule IsPE{  
 meta:  
 description = "检查文件是否为PE文件"  
 author="wbf"  
 date="2022-11-8"  
 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="2022-11-8"  
 condition:  
 filesize<500KB  
}  
  
rule mayBeMaleware{  
 meta:  
 description = "检查文件小是否大概率为maleware文件"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = { E8 00 00 00 00 }  
 $b = "malware" nocase  
 condition:  
 $a at pe.entry\_point or $b  
}  
  
rule Lab0703dll{  
 meta:  
 description = "检查改文件是否大概率与Lab07-03的dll属于同一类型的恶意代码"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = "127.26.152.13"  
 $b = "hello"  
 $c = "sleep"  
 $d = "exec"  
 $e = "OpenMutexA"  
 $f = "WS2\_32.dll"  
 $g = "OpenMutexA"  
 condition:  
 (5 of them) and (IsPE and smallFileSize) or (mayBeMaleware)  
}

lab07-03.exe的yara规则

import "pe"  
rule IsPE{  
 meta:  
 description = "检查文件是否为PE文件"  
 author="wbf"  
 date="2022-11-8"  
 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="2022-11-8"  
 condition:  
 filesize<500KB  
}  
  
rule mayBeMaleware{  
 meta:  
 description = "检查文件小是否大概率为maleware文件"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = { E8 00 00 00 00 }  
 $b = "malware" nocase  
 condition:  
 $a at pe.entry\_point or $b  
}  
  
rule Lab0703exe{  
 meta:  
 description = "检查改文件是否大概率与Lab07-03的exe属于同一类型的恶意代码"  
 author="wbf"  
 date="2022-11-8"  
 strings:  
 $a = "WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE"  
 $b = "C:\Windows\System32\Kernel32.dll"  
 $c = "Kernel32."  
 $d = "C:\windows\system32\kerne132.dll"  
 $e = "C:\\*"  
 $f = "kernel32.dll"  
 $g = "kerne132.dll"  
 $h = "CopyFileA"  
 $i = "FindFirstFileA"  
 $j = "FindNextFileA"  
 $k = "IsBadReadPtr"  
 condition:  
 (7 of them) and (IsPE and smallFileSize) or (mayBeMaleware)  
}

第三部分：IDA Python代码编写：

获取函数的汇编代码

ea = here()  
start = idc.get\_func\_attr(ea, FUNCATTR\_START)  
end = idc.get\_func\_attr(ea, FUNCATTR\_END)  
cur\_addr = start  
while cur\_addr <= end:   
 print hex(cur\_addr), idc.generate\_disasm\_line(cur\_addr, 0)  
 cur\_addr = idc.next\_head(cur\_addr, end)

找到对应的函数：包括一系列的socket初始化以及操作函数，帮助我们准确识别可执行文件的功能。

python>for func in idautils.Functions(): # 获取已知函数list  
 flags = idc.get\_func\_attr(func,FUNCATTR\_FLAGS)   
 if flags & FUNC\_LIB or flags & FUNC\_THUNK:   
 continue  
 dism\_addr = list(idautils.FuncItems(func)) # 函数指令地址  
 for line in dism\_addr:  
 m = idc.print\_insn\_mnem(line)  
 if m == "send" or m == "recv" or m == "bind" or m == "connect" or m == "accept":  
 op = idc.get\_operand\_type(line,0)  
 if op == o\_reg:  
 print "0x%x %s" % (line,idc.generate\_disasm\_line(line,0))

找到对应字符串出现的地址：因为Lab07-03.exe的文件非常长，使用脚本帮助我们定位该函数

例如想找到含有“output”的地址  
while cur\_addr < end\_addr:  
 cur\_addr = idc.find\_text(cur\_addr,SEARCH\_DOWN,0,0,"kernel32")#FindText  
 # 注意第五个参数需要写成字符串的形式，使用字符串变量的话会报错  
 if cur\_addr == BADADDR:  
 continue  
 else:  
 print(cur\_addr)  
 cur\_addr = idc.next\_head(cur\_addr)

检索每一个函数的函数名称：

ea = BeginEA()  
for funcea in Functions(SegStart(ea), SegEnd(ea)):  
 functionName = GetFunctionName(funcea)  
 print functionName

在Lab07-03中也有需要将16进制转换为字符串的需求：

def GetAns(start,end):  
 flag=''  
 for addr in range(start-4,end):  
 #当前地址的下一个地址  
 #判断下一个地址所在的指令是否为mov [esp+xxh], al  
 if GetOpnd(addr,1)=='al' and 'esp' in GetOpnd(addr,0):  
 #判断当前地址所有的指令是否是'mov eax, xxh'  
 if GetOpnd(addr-5,0)=='eax':  
 #获取十六进制数  
 hex=GetOpnd(addr-4,1)[:2]  
 #转换成10进制数  
 Int=int(hex,16)  
 flag+=chr(Int)  
 if chr(Int)=='}':  
 break  
 print flag  
# 遍历所有的段  
for seg in Segments():   
 #如果为代码段，则调用GetAns  
 if SegName(seg) == '.text':  
 GetAns(seg,SegEnd(seg))