## 课本第十一章实验报告

专业：信息安全 姓名：魏伯繁 学号：2011395

### **一、实验工具：**

IDAPro7.0 IDAPro7.7 win10虚拟机 winxp虚拟机、Wireshark、Ollydbg、strings、ProcessMonitor、Process Explorer、RegShot等

### 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：

Lab11-01

1、这个恶意代码向磁盘释放了什么？

**恶意代码从TGAD资源节中提取出动态链接库msgina32.dll，然后把他释放在硬盘上，目录为exe文件的目录**

2、这个恶意代码如何进行驻留？

**msgina32.dll作为GINA DLL安装，恶意代码将自己添加到注册表HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\ginaDLL中，这使得系统重启之后misgina.dll就会被重新加载**

3、这个恶意代码如何窃取用户的登录凭证？

**msgina.dll能够拦截所有提交到系统认证的用户登录凭证**

4、这个恶意代码对窃取的证书做了什么处理？

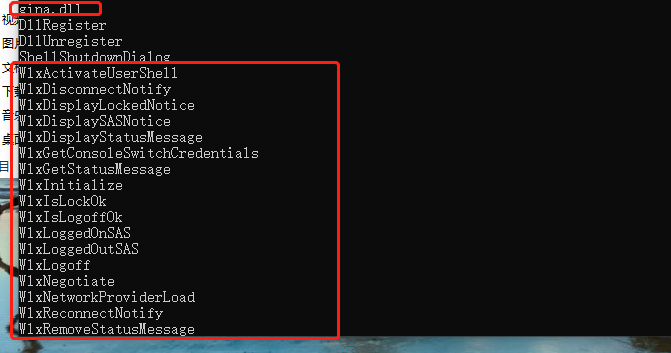
**恶意代码将被盗窃的登录凭证保存到系统目录下的msutil32.sys中，包括登录的时间、日期、用户名以及密钥等等**

5、如何在你的测试环境让这个恶意代码获得用户的登录凭证？

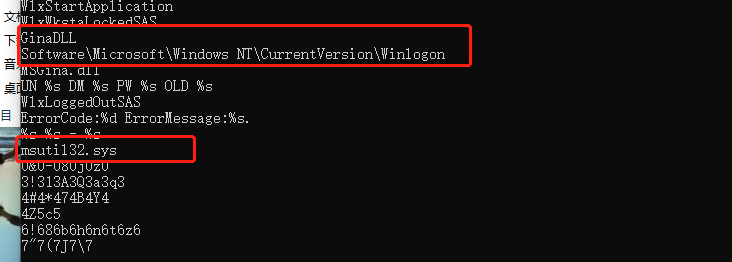
**先在本机运行exe文件然后重启系统让msgina32被成功加载之后，并且仅当用户注销时恶意代码才会记录登录凭证，之后就可以去系统目录下去查看相应的记录了。**

分析过程：

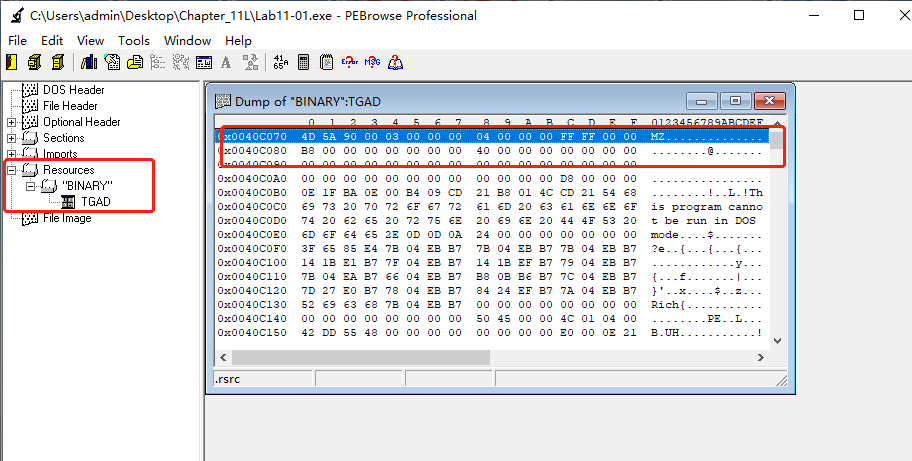
基本静态分析：使用strings查看exe文件中对应的字符串可以看到存在第一个gina.dll并且由很多已Wlx开头的字符串，所以说他很有可能是一个拦截GINA的恶意代码



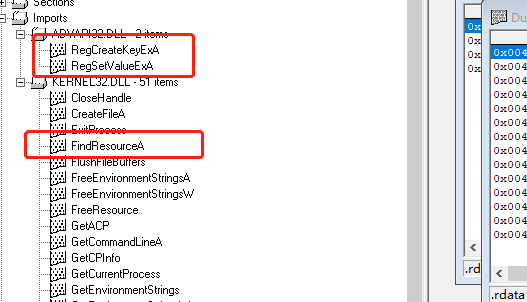
在这张图片中，同样我们也看到了一个注册表的路径以及相应的硬件驱动，于是根据这些我们可以猜测该恶意代码通过修改注册表完成拦截GINA的操作，并且配合一些驱动完成相应工作

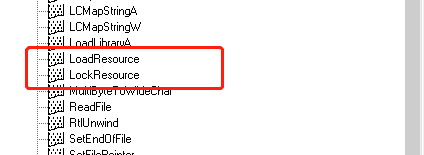


通过PEbrowser可以看到，他在资源节存在一个TGAD，点开看发现是一个PE文件，那么这个PE文件很可能就是exe文件想要下载导入的文件：



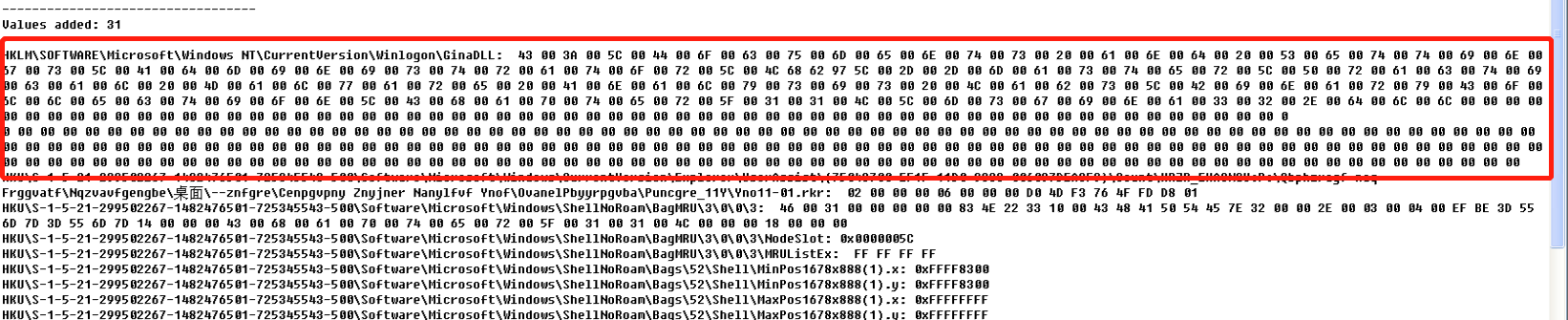
再通过查看该文件的导入表，我们也可以确认上面的猜想，导入表中存在对资源节的加载以及对注册表的修改函数：



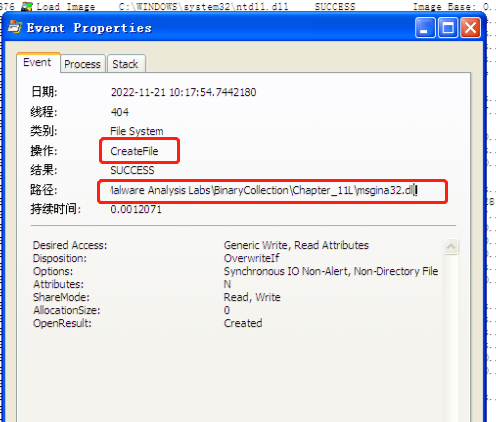


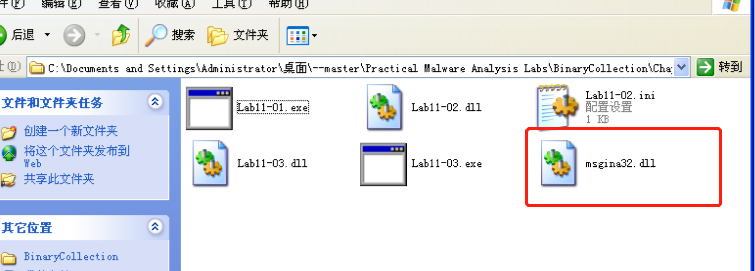
第二部分：基本动态分析

我们在xp虚拟机上使用process Explorer和process monitor监视进程的改变并监视Lab11-01的行为，并通过RegShot对注册表进行快照，首先观察RegShot的结果：发现他确实更改了注册表，恶意代码插入了msgina.dll的路径到注册表GINA，所以当系统自启动时WinLogon会加载这个DLL

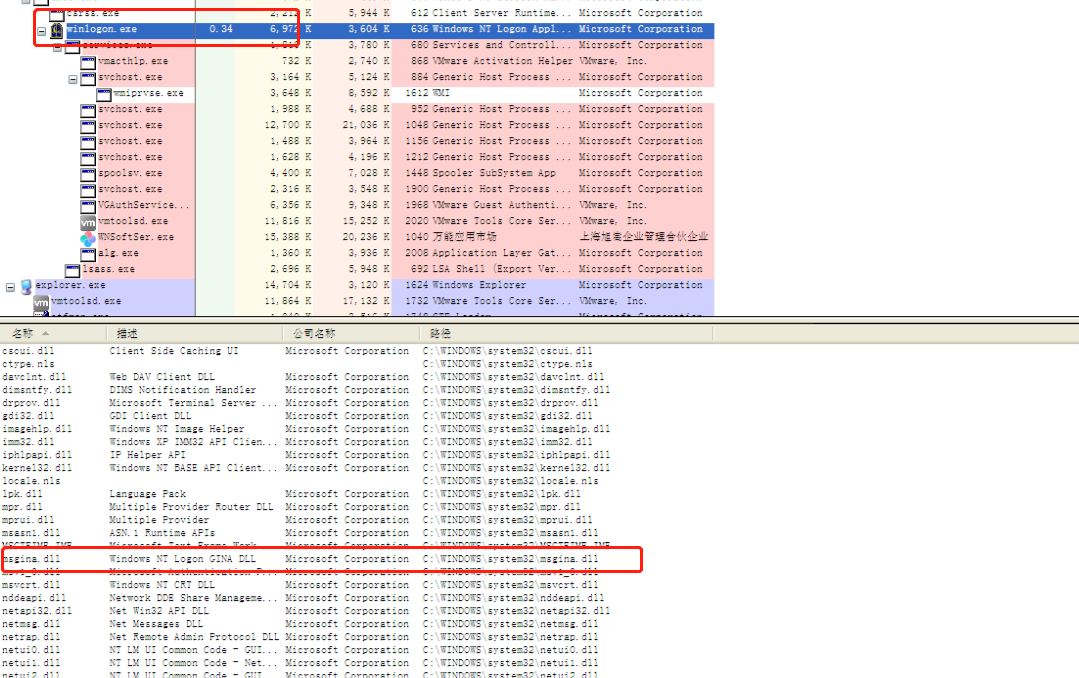


再查看Process Monitor的内容，成功发现了一个CreateFile的动作，他创建了一个msgina32.dll的动态链接库，具体的位置就在和Lab11-01.exe同级的目录下





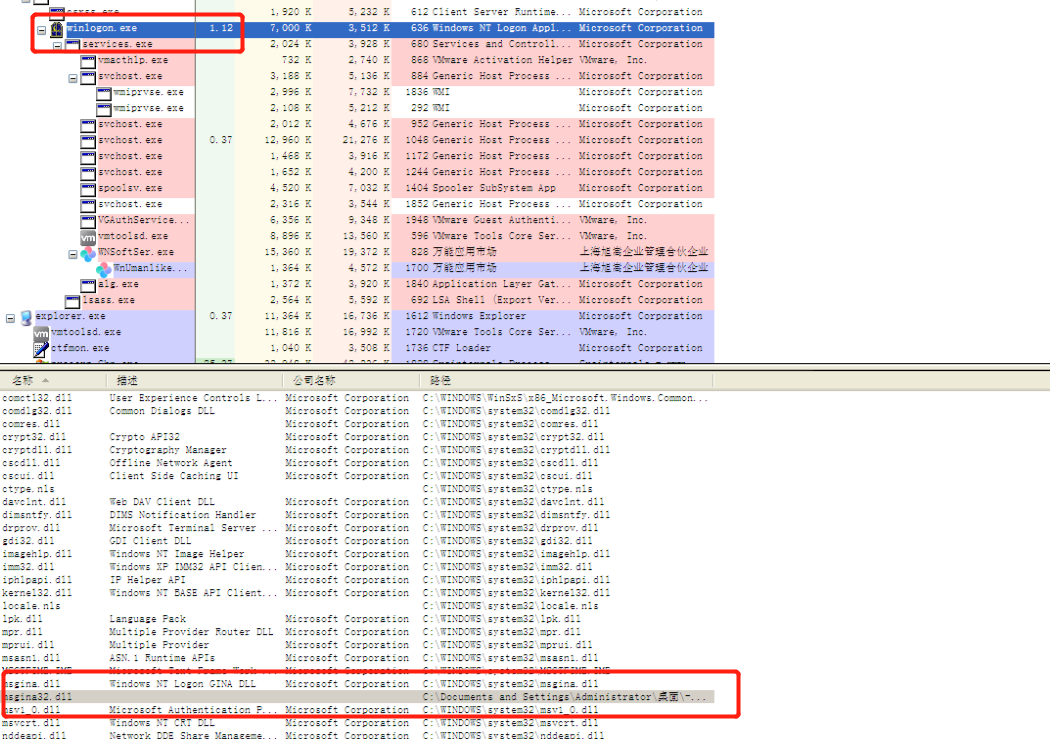
再查看Winlogon.exe下的dll，我们发现确实是存在一个msgina.dll的动态链接库的，但是这个dll是在system32下的，也就是干净的dll，说明这个时候恶意代码的dll还未被加载



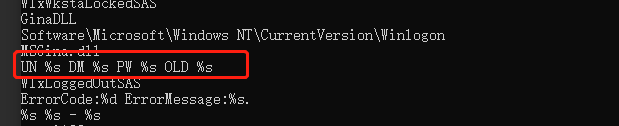
于是我们关机重启，发现他给我弹了一个弹窗，也就是弹出了一个开机启动输入密码的界面，所以我们可以大概知道这个恶意代码要记录用户的笔记本上的的账号和密码



这个时候我们再打开Process Explorer发现，一个被恶意代码生成的msgina32就被生成并动态加载了

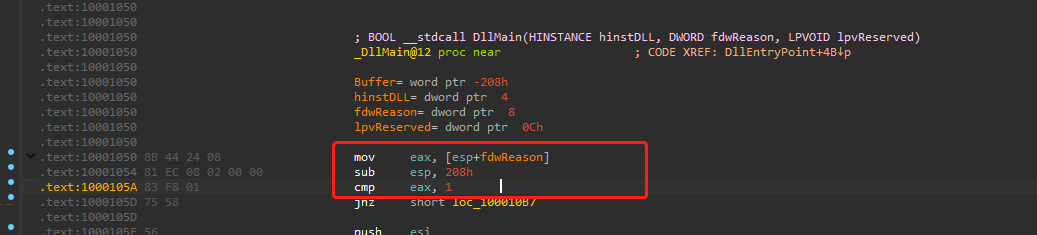


然后我们通过ResourceHacker来分析其导出的dll，发现他和在xp机上生成的msgina32是一致的，所以我们分析该dll，再次分析dll的strings，我们大概可以判断，格式化的输出其实是记录了用户的账号和密码信息

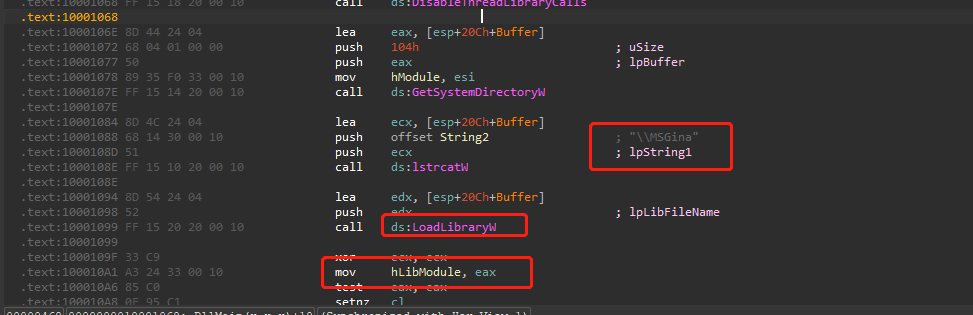


第三步：使用高级静态分析

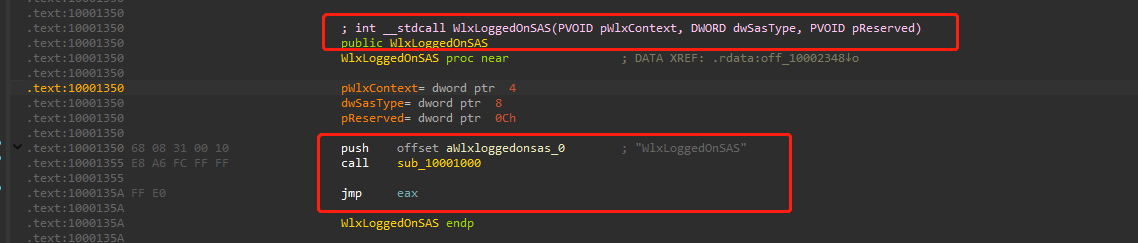
使用IDAPro分析msgina.dll文件，首先可以看到该dll对参数进行检查：即对第二个参数进行检查，如果不是1的话就退出



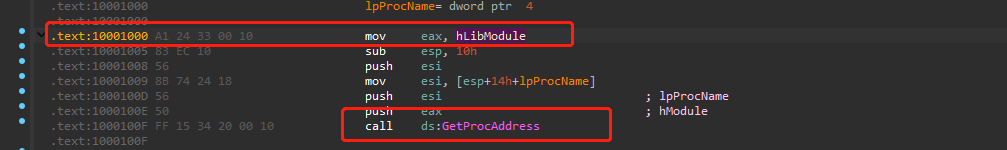
接下来，如果参数正确，就执行后面的代码，通过调用LoadLibraryW函数获取Windows系统中msgina.dll的句柄，然后再将这个句柄存入hLibMoudle中，这样的话就可以让恶意代码可以正常的响应响应的对msgina.dll的请求，因为msgina32拦截了Winlogon和msgina的通信，所以msgina32相当于要帮助分发正常的系统调用



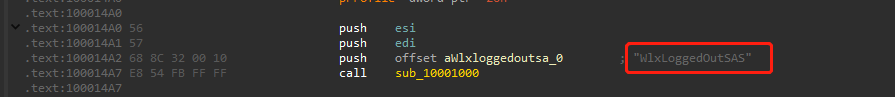
然后我们查看该dll的导出函数，我们以WLxLoggdOnSAS进行说明：这个函数非常简单，首先传递字符串WLxLoggdOnSAS到sub\_10001000，然后跳转到程序入口地址，函数sub\_10001000传入msgina.dll的句柄和想要调用的函数WLxLoggdOnSAS来使用GetProcessAddress解析msgina中的函数地址，然后跳转到该函数，通过跳转并调用WLxLoggdOnSAS，这段代码并不会创建一个栈帧或者将返回地址压栈，所以他将直接在Winlogon上运行，就像什么都没发生过一样



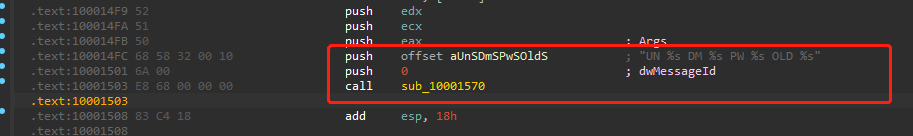
下图是对应的10001000处的代码，通过函数名寻找对应应该跳转的地址



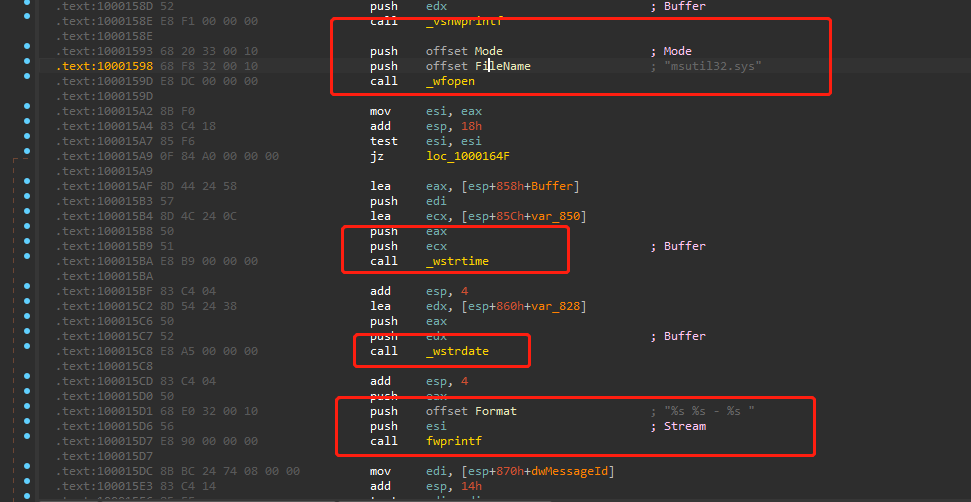
其他大部分的导出函数都与WLxLoggdOnSAS的操作过程基本一致，除了WLxLoggdOutSA函数以外：他在完成这些常规操作之后又进行了后续的很多操作：



后面的代码将那个格式化字符串压栈当做参数后调用了一个其他位置的函数：



10001570位置的代码看起来像是一个记录窃取登录凭证的函数，首先传入被当做参数传入的格式化字符串，然后打开msutil32.sys文件，因为Winlogon在C:\Windows\System32的目录下，所以这个文件也被创建在该目录下，接下来记录日期时间最后记录登录的凭证，所以说，这个sys文件并不是一个驱动，只是记录记录凭证信息的文件。



然后我们可以查看具体的该文件下的记录：发现和我们分析的一致



Lab11-02

1、这个恶意代码DLL导出了什么？

**Lab11-02包含一个名为installer的导出函数**

2、使用rundll32.exe安装这个恶意代码后，发生了什么？

**installer会从命令行启动恶意代码，恶意代码会作为spoolvxx32.dll将自身复制到系统目录中，并且从AppInit\_DLLs键值下永久安装，并且该恶意底阿妈希望从系统录下读取配套的ini文件**

3、为了使这个恶意代码正确安装，Lab11-02.ini必须放置在何处？

**就像上一问所说的，此时ini文件必须被放置在系统目录下**

4、这个恶意代码如何驻留？

**恶意代码将自身安装到AppInit\_DLLs的注册表键值中，这样这个恶意代码就会加载到所有装载User32.dll的进程中**

5、这个恶意代码才用的Rootkit技术是什么？

**这个恶意代码针对send函数进行了挂钩攻击**

6、挂钩代码做了什么？

**这个挂钩会检查哈向外发出的包，如果向外传递的包包含了RCPT:TO的电子邮件信息那么他会将这样的电子邮件加入一个新的收件人billy**

7、哪个或者哪些进程执行这个恶意攻击，为什么？

**恶意代码的攻击目标仅仅针对MSIMN.exe THEBAT.exe以及OUTLOOK.exe，因为他们都是电子邮件客户端软件所以才只选择他们**

8、.ini的意义是什么？

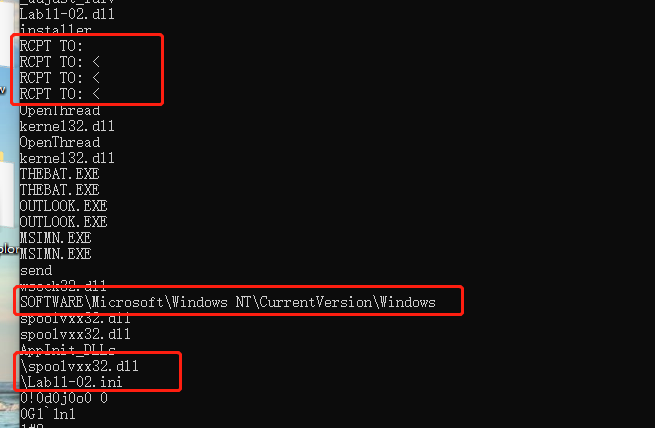
**ini文件包含了一个加密的电子邮件地址，使用ollydbg解密后可以得到他的明文**[**billy@malwareanalysis.com**](mailto:billy@malwareanalysis.com)

9、你怎么样使用wireshark动态抓获这个恶意代码的行为？

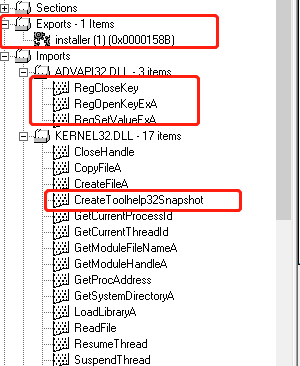
**使用网络数据抓取流量包就可以看到一个假冒的服务器以及OutlookPress客户端**

分析过程：

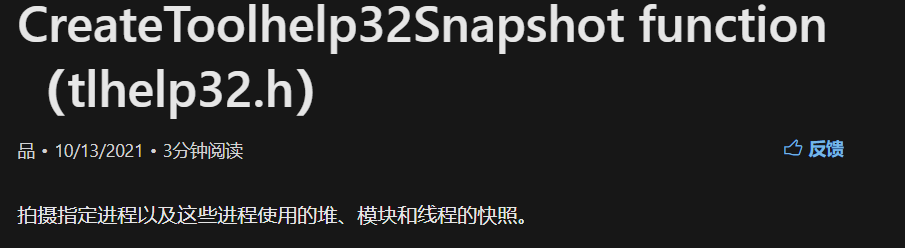
第一部分：基本静态分析，使用strings工具可以观察到，其中APPInit\_DLLs和后面的一个注册表的路径用来永久安装自身，后面的ini文件说明该动态链接库使用了ini文件，其中RCPT TO这个字符串说明了：RCPT是一个命令，用来创建一个电子邮件的接收人



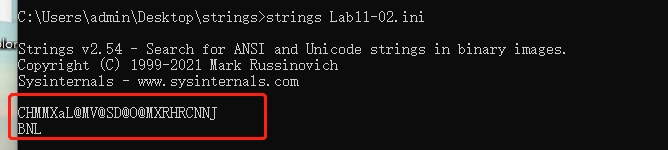
然后再通过PEBrowser查看他的导入导出表：他具有一个导出表installer，应该是要安装一个可执行文件或一些可执行的进程，导入表包括对注册表的修改以及搜索一个进程线程列表的导入函数。



根据查找微软官方文档我们可以名曲该函数的使用：

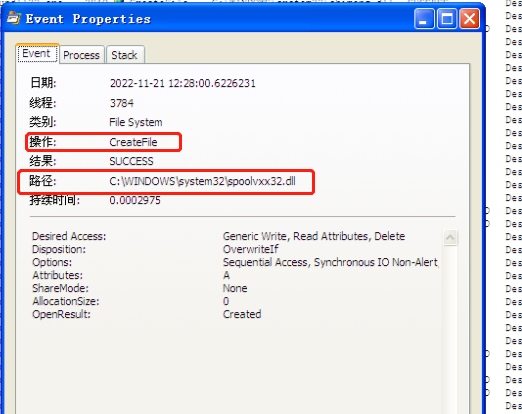


然后我们对ini文件进行静态分析，发现他并没有提供什么有用的信息，并且他似乎被加密了

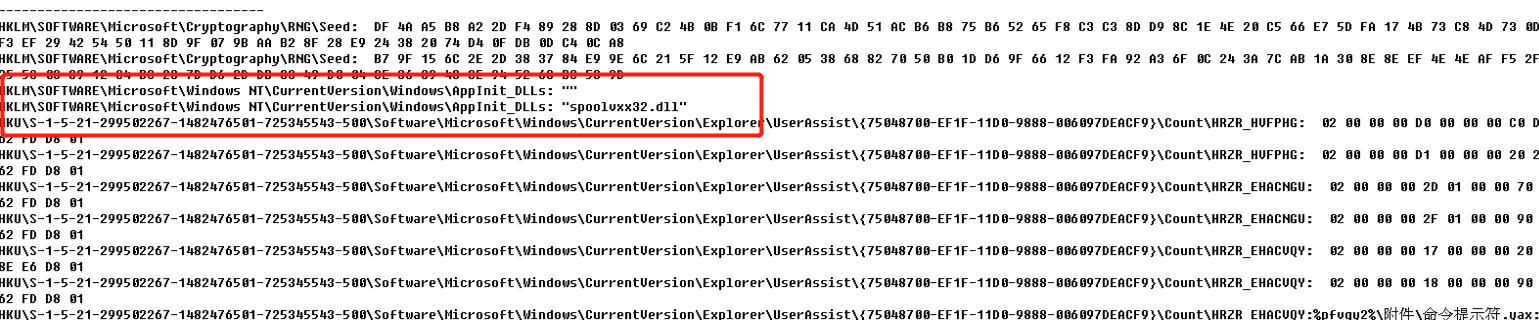


第二步：基本动态分析

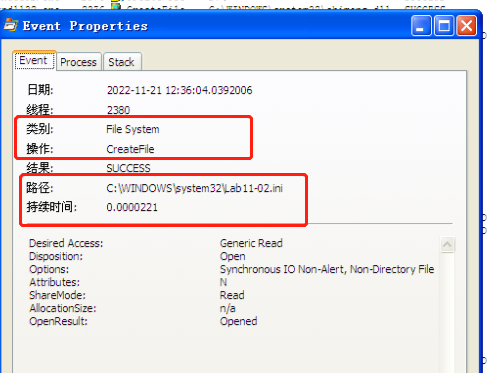
使用命令rundll32.exe Lab11-02.dll,install 执行dll文件，通过procmon可以观察到恶意代码在系统目录下创建了一个spoolvxx32.dll的文件，并且这个dll文件其实就是Lab11-02.dll文件



然后我么根据regshot可以看到恶意代码将spoolvxx32,dll添加到AppInit\_Dlls列表中，这样的话恶意代码就会被加载到所有装载User32.dll的进程中

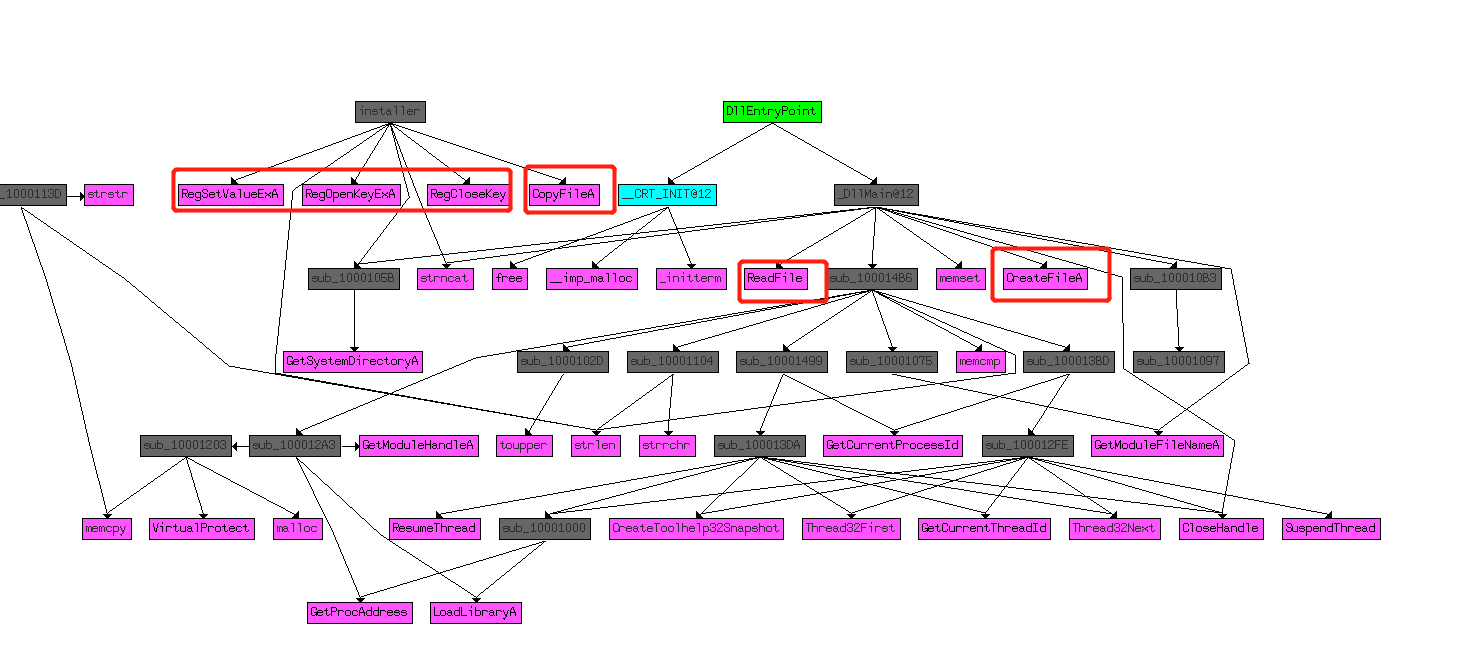


然后，我们在procmon中也可以看到在系统目录下打开ini文件的请求，这表明我们需要将ini文件移动到系统目录下

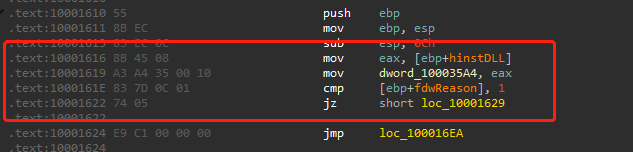


第三步：使用高级静态分析

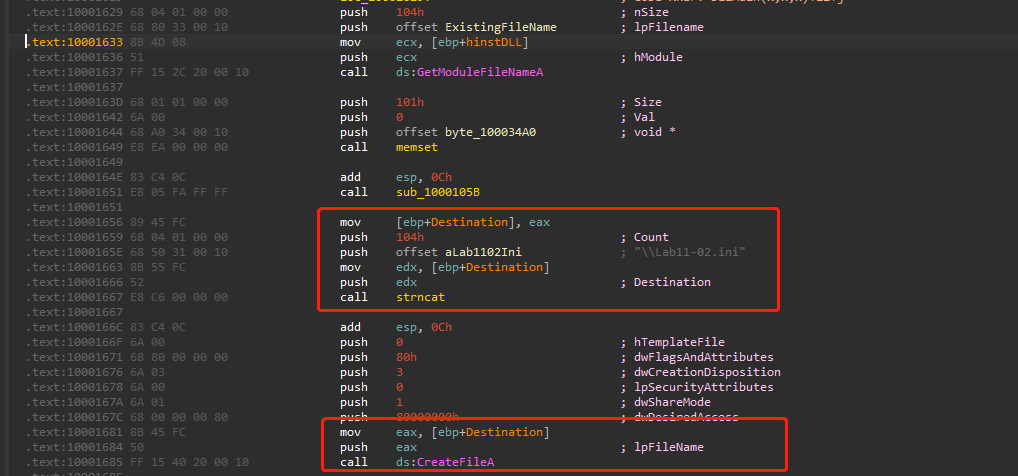
使用IdaPro打开dll文件，然后分析其调用的函数：我们发现这个installer的具体功能和我们之前观察过的基本一致，也就是修改注册表的值然后对文件进行读取、写入



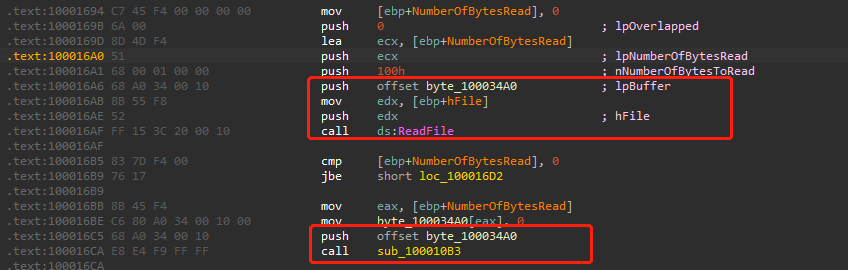
然后我们进入函数的内部进行分析：首先和上一个程序一致的是他在一开始的时候检查了参数的状态，如果参数不符合要求则直接退出



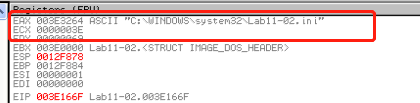
接下来的程序实现的功能是：首先获取Windows的系统目录以及一个字符串代表的接下来的目录lab11-02.ini，最终使用strncat将他们组合成一个完整的路径，并且在最后打开这个文件，如果打开失败也直接退出



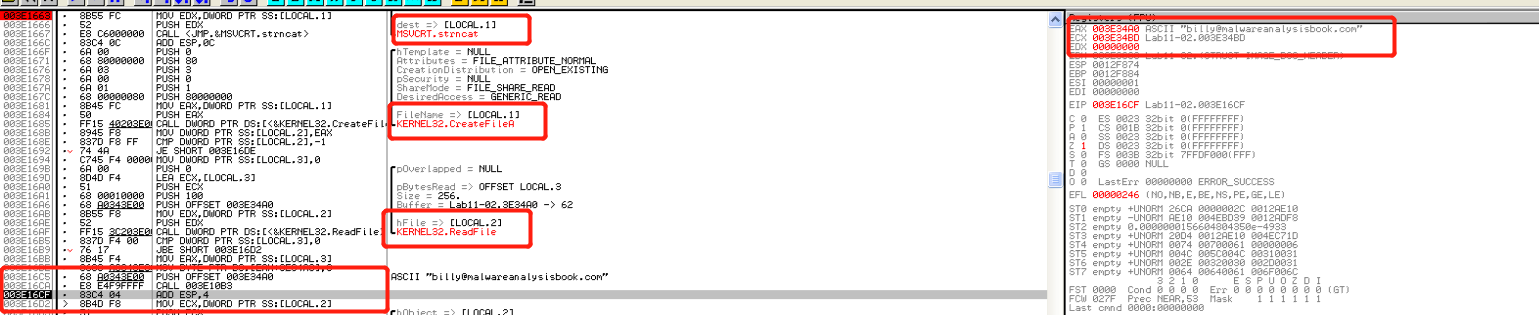
如果成功获取了ini文件，然后就需要将他读入缓冲区，但是我们通过strings程序读取ini文件中的字符串时看到的是里面是一堆乱码，所以很有可能里面的内容需要解密才能获取到其中的内容。那么读入后调入的第一个函数很可能就是对应的解密函数



为了获取其解密之后的结果，我们利用ollydbg对其进行动态调试：在读取文件的位置设置断点并观察其执行过程，可以看到，当读取完ini函数的地址后再使用strcat，eax的值变成了整个lab11-02.ini路径的值的字符串，于是我们可以读取该位置的文件

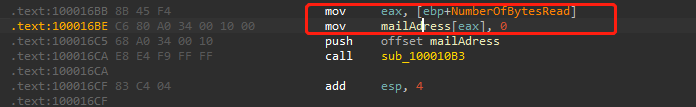


接下来，我们使用F8进行step over的形式进行调试，因为我们并不关注其内部是怎么执行的，于是我们得到的解密结果为一个emai的地址：放大后我们可以看到他的具体内容：[billy@malwareanalysisbook.com](mailto:billy@malwareanalysisbook.com),此时eax的值正在指向他，这个邮箱的内容存在全局变量中

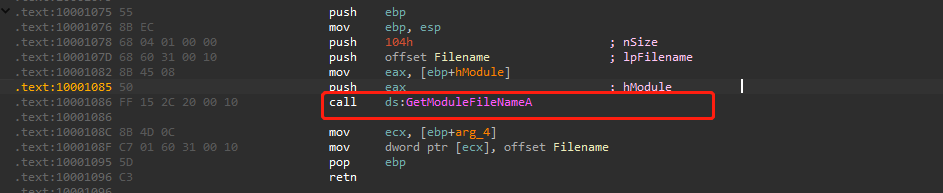




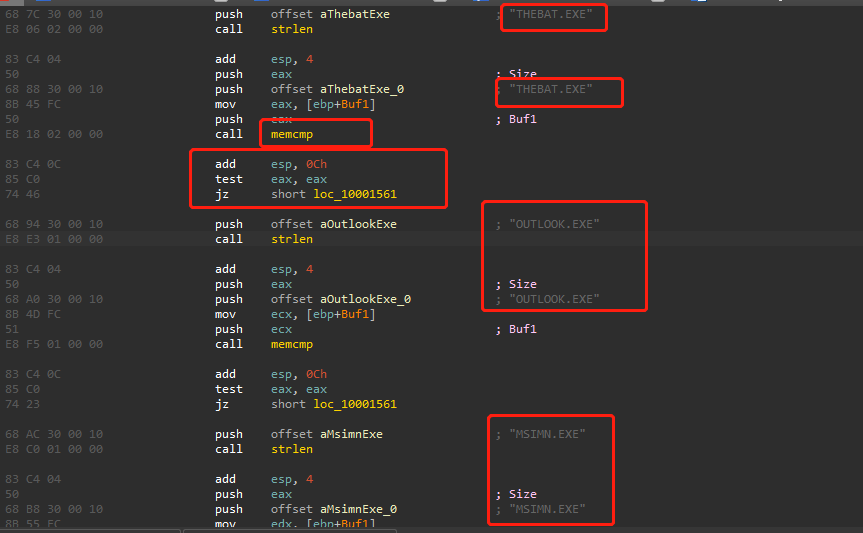
于是，我们可以将byte\_100034A0的变量值的名称修改为mailAdress便于后面的识别



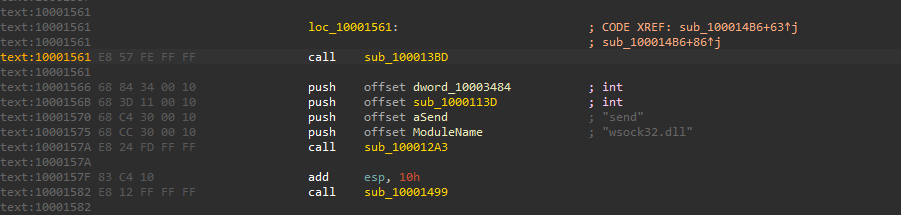
接下来，恶意代码调用GetModuleFileNameA函数，因为在调用这个参数之前，它的参数hMoudle被置为0，所以函数会加载这个DLL的绝对路径，接下来恶意代码在arg\_4中返回路径名



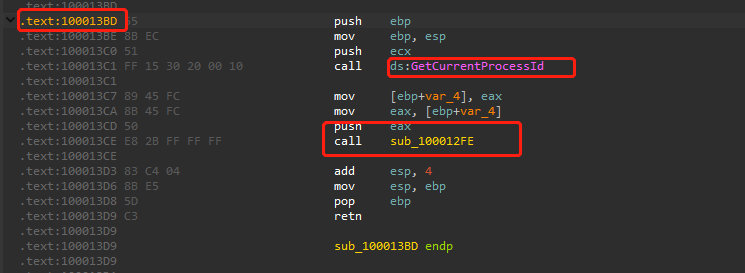
接下来，将当前进程的大写字母与THEBAT.EXE和OUTLOOK.EXE以及MSIMN.EXE互相比较，如果进程名不等于其中任何一个就退出，如果恶意代码已经载入了这三个中的一个那么后面的代码就会运行

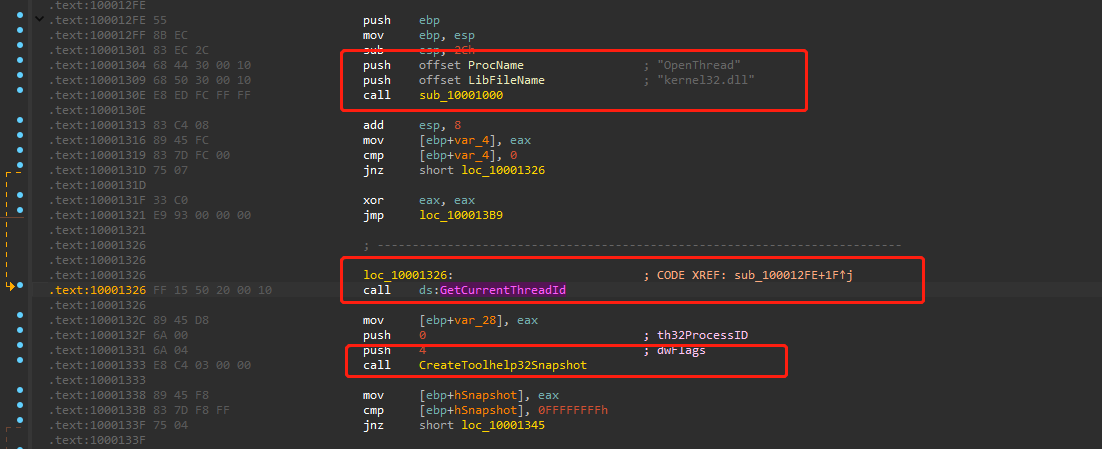


接下来的代码sub\_10010BD处的代码进行了一系列操作，其具体的流程在下面两张图中有所体现。根据下面的分析，我们可以知道在sub\_100013BD中暂停了其他的所有线程。通过后面的分析我们又知道，1499位置的函数将之前所有挂起的线程全部都恢复了，那么我们可以合理猜测，在这中间可能要做一些可能会改变进程运行状况的动作，比如改变内核或者安装一个inline挂钩。

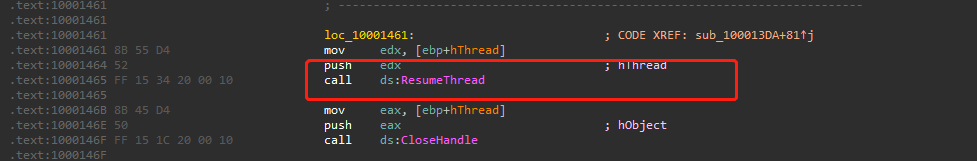


下面两张图详细展示了100013BD的函数具体流程，首先该函数调用了GetCurrentProcessId函数用来返回当前进程的标识符，然后紧接着调用了100012FE函数，这个函数调用了GetCurrentThreadId函数用来返回当前进程的标识符，接下来调用CreateToolhelp32SnapShot，并且遍历当前进程的所有Tid，如果Tid不是当前的线程那么就将这些进程暂停，所以说这个函数刮起了当前其他的进程。





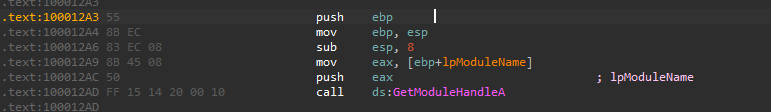
接下来，我们可以看到位于1499的函数最终调用了ResumeThread函数将之前暂停的线程全部恢复了。



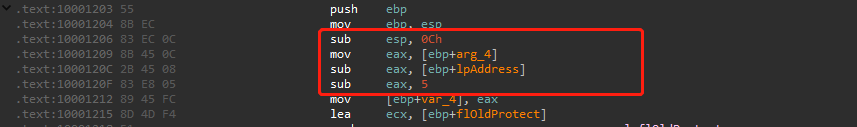
接下来，我们分析中间的函数，观察在所有的TID中断的这段时间里发生了什么，首先他的两个参数我们是已知的，分别是send和wsock32.dll，



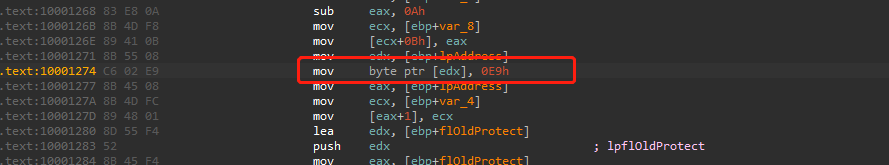
并且通过这句GetMoudleHandleA获得了所需要的dll的句柄，接下来马上传递相应的参数到1203的位置继续执行相应函数



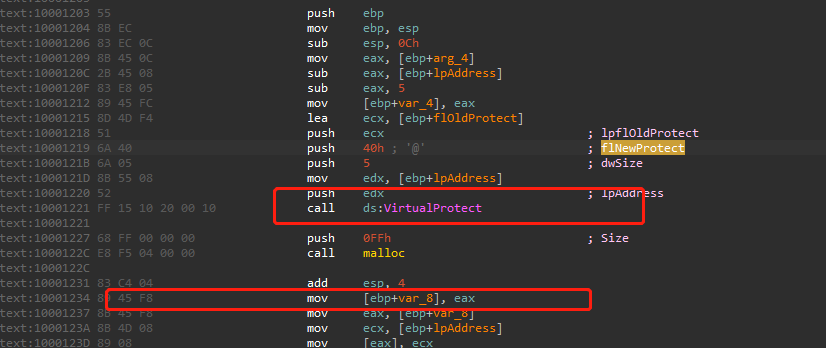
那么这个函数究竟是怎么完成挂钩函数的呢？代码通过计算send函数和内存地址sub\_1000113D开始的内存地址之间的差，再将差值移到var\_4变量之前，需要从中再减去5个字节，随后的代码在使用var\_4的时候可以通过在前面直接加入jmp然后跳转到sub\_1000113D函数的位置。



然后在后面我们就看到：在send函数的开头复制了0E9h，这个机器码的意思就是jmp，之后就将代码赋值var\_4到0xe9之后的内存，于是我们就可以通过这样一个简单的指令设置跳转到DLL中的sub\_1000113D函数



在接下来的动作中：VirtualProtect函数的调用修改了内存的运行。读以及写权限，因此可以修改恶意代码中的send函数的可执行性，在函数末尾的VirtualProtest则恢复原有的保护，随后使用alloc函数分配256字节的内存并将结果保存在Var\_8中

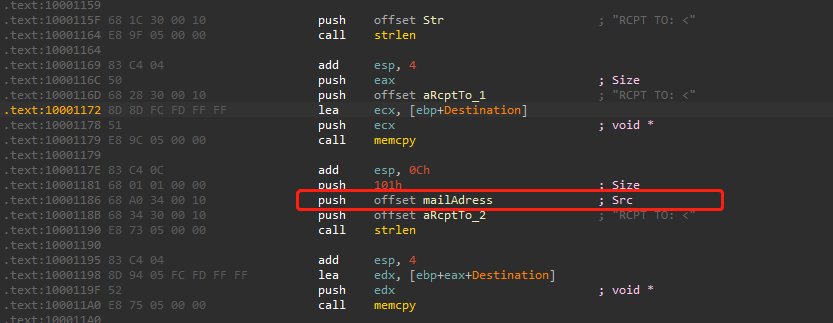


然后我们再次查看恶意代码是如何填充这256个字节的代码的：首先，使用memcpy进行复制send的前五个字节，因为恶意代码覆盖了前5个字节，因此需要确保保存了原始指令。恶意代码假设send函数前几条指令严格按照5个字节对齐

接下来在0E9h和后面的添加跳转的地址，这样一个跳转指令就成功的进行了初始化，最后在函数的末尾将全局变量10003484设置为了这个缓冲区的地址



挂钩函数在buf中查找字符串RCPT TO，如果没有发现这个字符串那么一切如常，就是正常的去send函数中去调用相应的代码，反之会向mailaddress位置的客户发送一个邮件，而这个用户就是[billy@malwareanalysisbook.com](mailto:billy@malwareanalysisbook.com),这段代码向所有发出的邮件都添加了一个收件人



Lab11-03

1、使用基础静态分析，你可以发现什么有趣的线索

**可执行文件包含inet\_epar32.dll以及net start cisvc这意味着他可能启动了CiSvc索引服务，包含字符串一个位于系统目录的kernel64x.dll的Lab11-03.dll导入了API函数GetAsyncKeyState和GetForegroundwindow两个函数，这可能是一个击键记录器**

2、当运行这个代码时发生了什么

**恶意代码首先赋值Lab11-03.dll到Windows系统目录下的inet\_epar32.dll中，他向cisvc.exe写入数据并且启动索引服务，恶意代码会向kernel64x.dll中记录击键记录**

3、Lab11-03.exe如何安装Lab11-03.dll并使其长期驻留

**恶意代码通过入口重定向来进行木马索引服务使他永久性的安装Lab11-03.dll他通过入口重定向来加载这个dll的shellcode**

4、这个恶意代码感染Windows系统的哪个文件？

**为了加载inet\_epar32，恶意代码感染了cisvc然后调用了inet\_epar32.dll的导出函数zzz69806582**

5、lab11-03.dll做了什么？

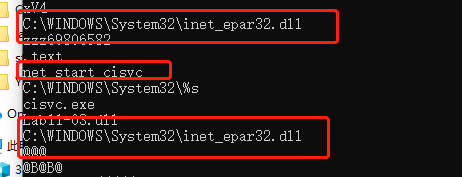
**是一个轮询的击键记录器，他的导出函数在zzz69806582中实现**

6、这个恶意代码收集的数据放在何处？

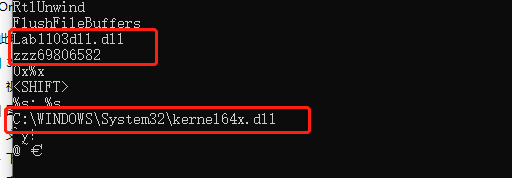
**恶意代码的击键记录和窗口输入记录被记录在了系统目录下的kernel64x.dll中**

第一步：基本静态分析

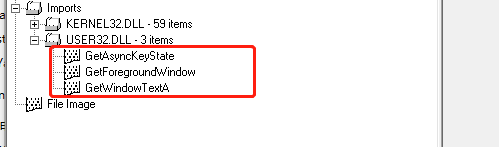
首先，对exe文件进行字符串扫描，可以看到其中引用了两个非常重要的dll分别是inet\_epar32.dll，以及一个服务启动命令net start cisvc，这个命令用来在windows上启动一个服务



然后，我们对dll文件进行字符串扫描。我们可以观察到一个非常奇怪的字符串zzz69806582,目前我们并不明确他的功能，但是可以发现其引用了一个非常重要的动态链接库kernel64x.dll，但是这并不是一个通常意义上合法的常用的dll



接着，我们对dll的导入表进行分析，我们观察到了GetAsyncKeyState和GetForegroundwindow两个函数，第一个函数用来确定确定用户当前是否按下了键盘上的一个键的函数。如果按下，则返回值最高位为1；第二个函数用于获取目前系统最前端的窗口，和第一个函数搭配使用的效果是记录目前用户正在向哪个窗口进行输入。

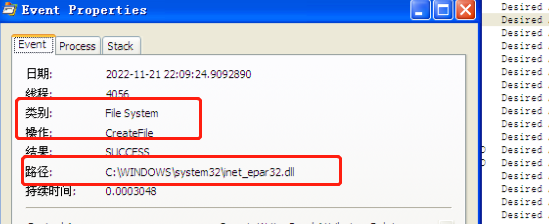


第二步：基本动态分析

然后，我们在xp虚拟机上进行恶意代码的运行来查看其具体的攻击方式，在设置好procmon以及Process Explorer来观察恶意代码的行为，在procmon中我们观察到，一个新的进程cisvc被执行，并且这个进程加载了恶意代码的加载inet\_epar32.dll，经过比较后我们可以发现inet\_epar32.dll与Lab11-03.dll是相同的，于是我们可以判定这个动态链接库把自己放在了系统目录下。



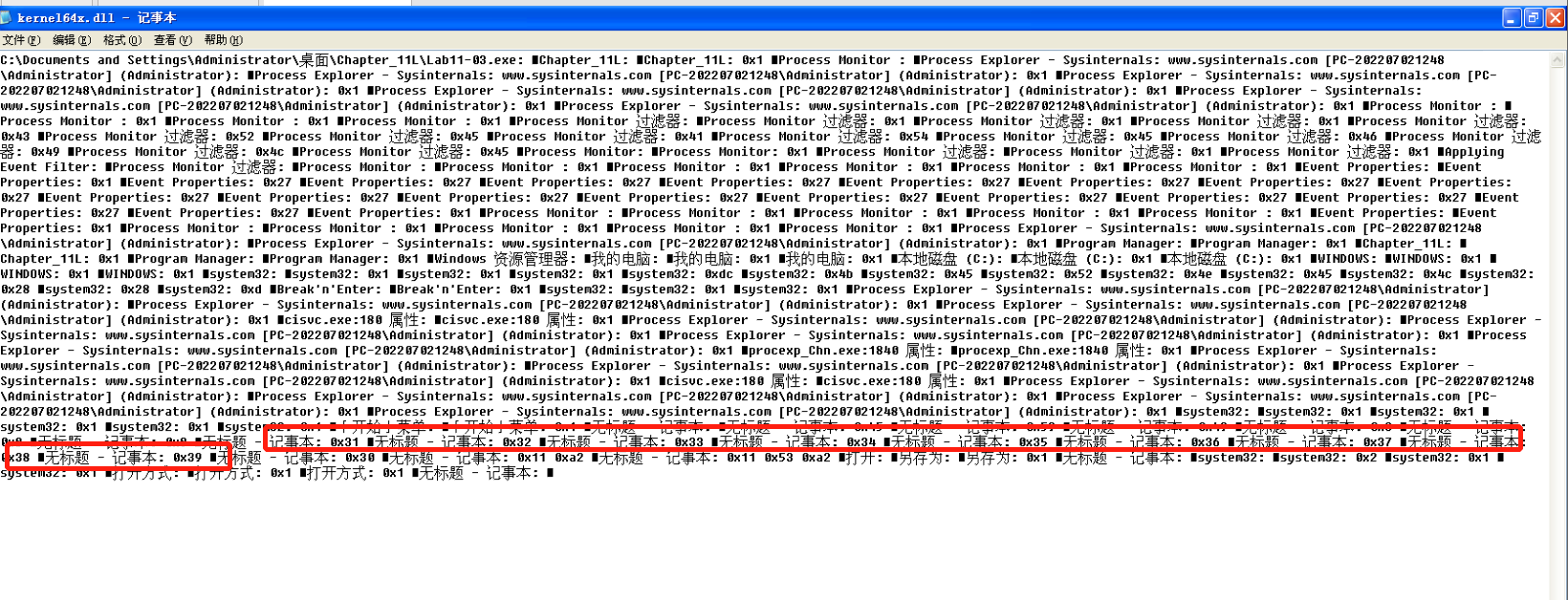
然后，我们在Process Explorer中观察到乐意个CreateFile的操作，这个操作在系统目录下创建了一个文件，也就是刚才我们观察到的被cisvc加载的dll



并且除此之外，一个新的dll文件也被加载了，就是刚才我们提到的，一个和系统文件很像但是不是系统文件的kernel64X.dll，这个文件是用来记录东西的，并不被当做传统的动态链接库使用

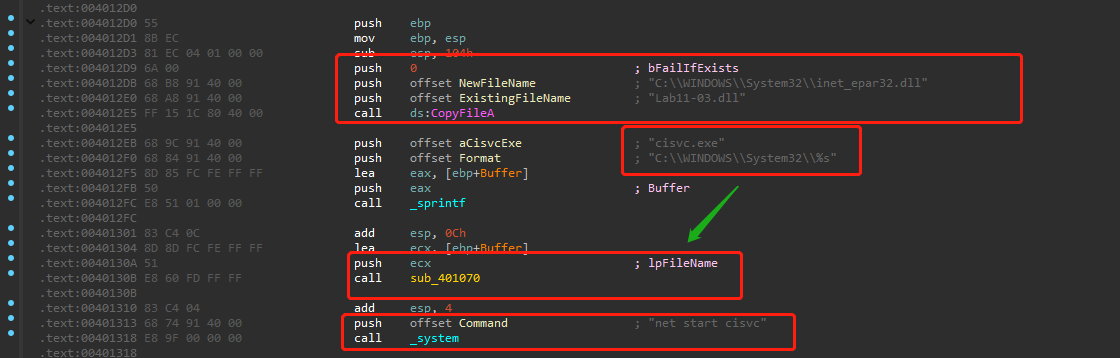


我们打开记事本并敲击一些内容发现我们的敲击内容被完整的记录了下来：让我们用记事本的方式打开kernel64x.dll可以观察到下列内容：0x31-0x39代表着1-9的ascii码，所以到此为止我们基本可以判定这个恶意代码的功能--通过将自己连接到cisvc上来记录用户的键盘敲击记录并且记录到kernel64x.dll中

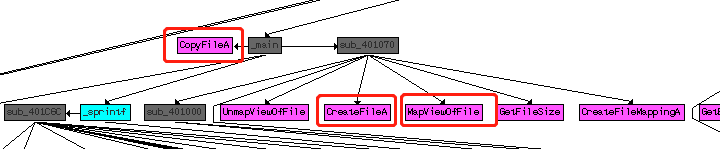


第三部分：高级静态分析

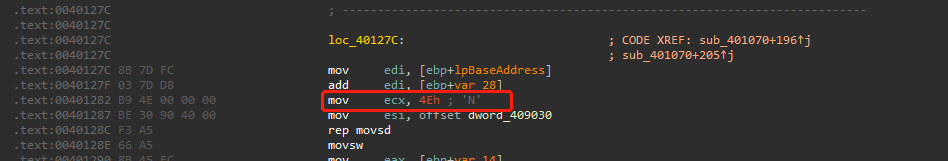
首先，我们进入exe文件一开始执行的位置，首先这个可执行文件做的第一件事就是将dll文件拷贝到系统目录下当做inet\_epar32.dll，然后就将系统目录下的cisvc.exe当做参数传递给了sub\_401070函数，然后就通过系统命令net start cisvc来启动索引服务



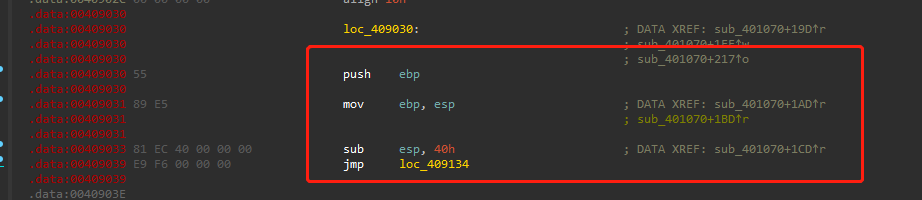
然后，进入函数sub\_401070并且查看sub\_401070的调用树，我们可以观察到，这个函数调用了CreateFileA、CreateFileMappingA和CreateViewOfFile，将cisvc.exe映射到内存中，CreateFileMapping()函数创建一个文件映射内核对象，通过参数hFile指定待映射到进程地址空间的文件句柄（该句柄由CreateFile()函数的返回值获取），下一个函数的作用是将文件映射的视图映射到调用进程的地址空间，并且返回内存映射视图的基地址。在UnmapViewOfFile调用后，对这个文件做的任何修改都会被写入到磁盘中，所以不需要再显示的调用WriteFile函数。



然后在进行了一些列创建文件、加载文件到内存并且unmap、得到他的基地址之后我们分析写入到内存映射文件缓冲区，首先，文件的映射基地址被一如了edi，并且使用var\_28调整了一些偏移量之后将0x4E载入ECX并且持续写入，最后将byte\_409030写入ESI，并且使用rep movsd复制该位置内容到映射文件中，进入检查409030处的内容



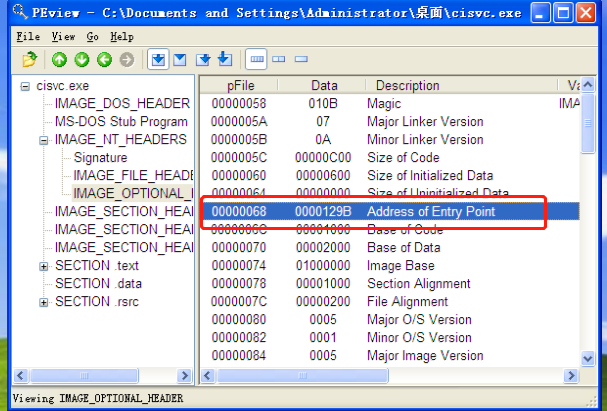
在该位置的是用来进行shellcode注入的作用，得到的反汇编代码如下：其核心其实是一个jmp函数，我们点击进入该函数

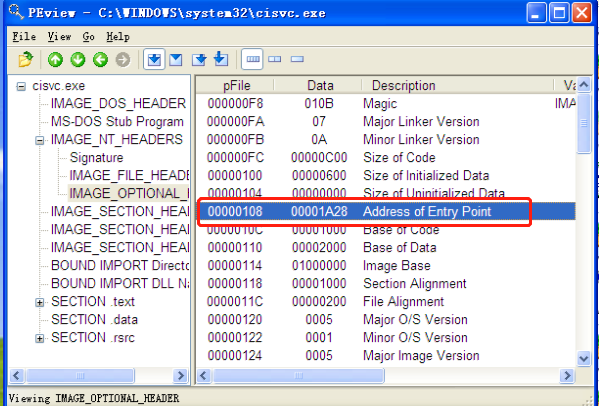


发现该函数其实是两部分，第一部分是一个路径"C:\WINDOWS\System32\inet\_epar32.dll"，第二个是一个字符串，字符串的内容是我们之前疑惑不知道有什么用的zzz69806582，说明这个shellcode加载了这个dll并且调用了他的导出函数

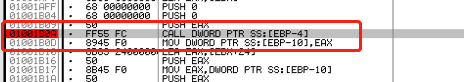


接下来，可以比较恶意代码运行前后的cisvc的区别，使用PEView可以观察到两个恶意代码的二进制文件入口点不同，也就是说被感染过的cisvc被恶意代码进行了入口重定向，从而无论cisvc何时启动，shellcode都会咸鱼原始的入口点执行，下面这两张图片清楚的反应了两个exe文件的入口点不同

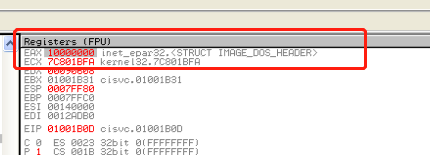




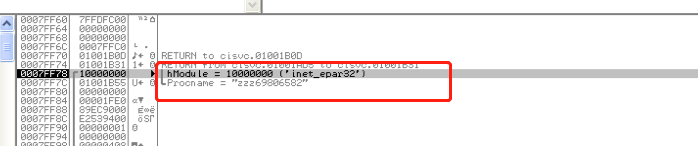
接下来，我们可以使用ollydbg对被感染后的cisvc进行动态调试，在01001B0A处调用了LoadLibrary函数将inet\_epar32.dll载入内存



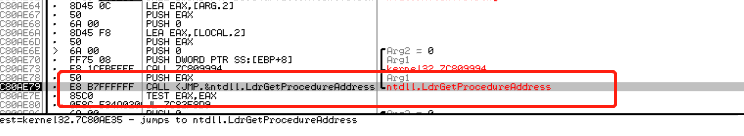
可以清楚的看到这里的EAX就是inet\_epar32.IMAGE\_DOS\_HEADER



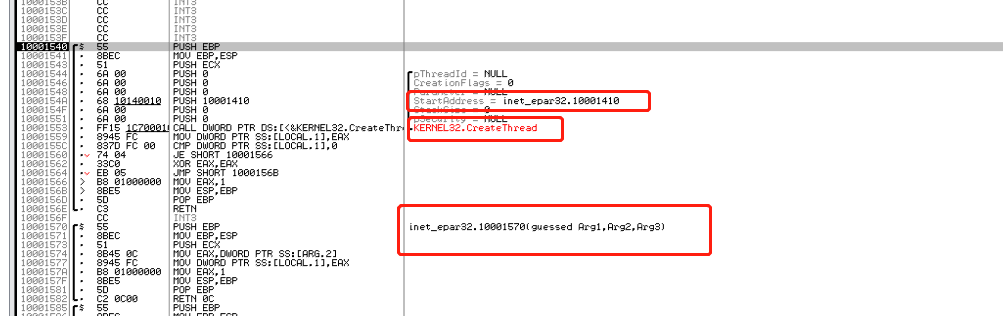
具体的加载过程我们可以看到，将上面返回的值载入到hmoudle中，然后那一堆奇怪的字符串就是procname，要去这个dll中调用指定的process



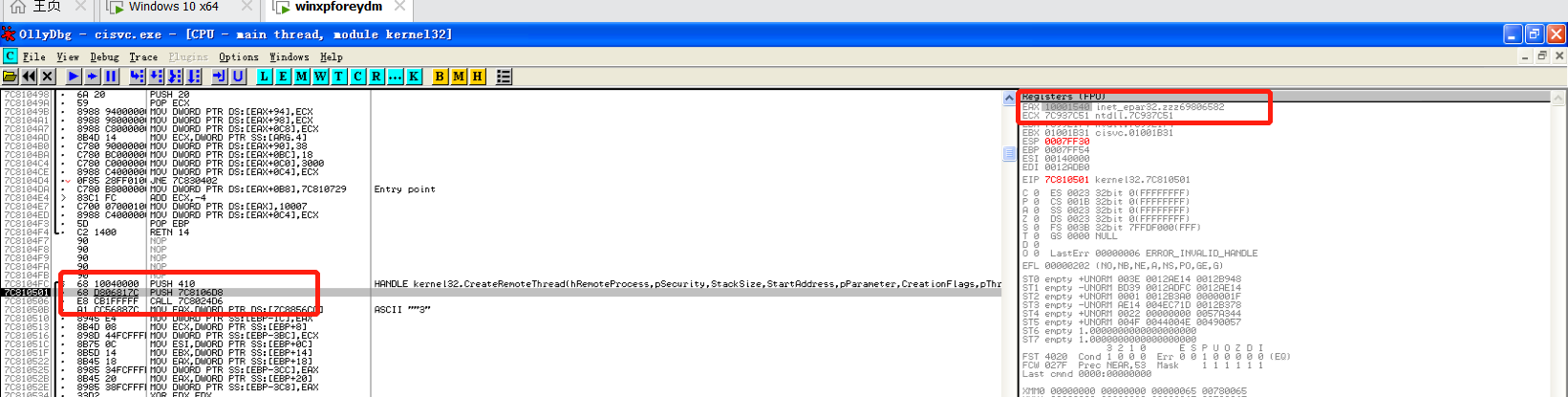
后面马上使用参数zzz69806582调用GetProcessAddress



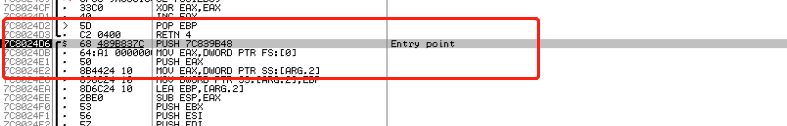
开启了一个线程，这个线程也是时该dll中的一个函数，但是这个时候我们还是不知道这个东西是被如何创建的



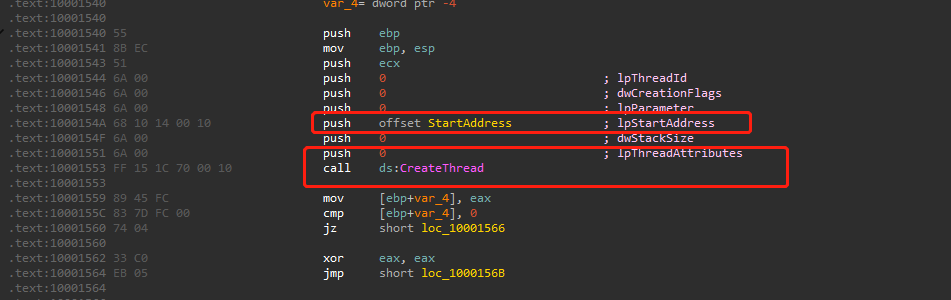
可以看到上一个函数的返回地址到这里就是导出函数的地址即inet那个dll.zzzz这个函数作为函数返回值返回



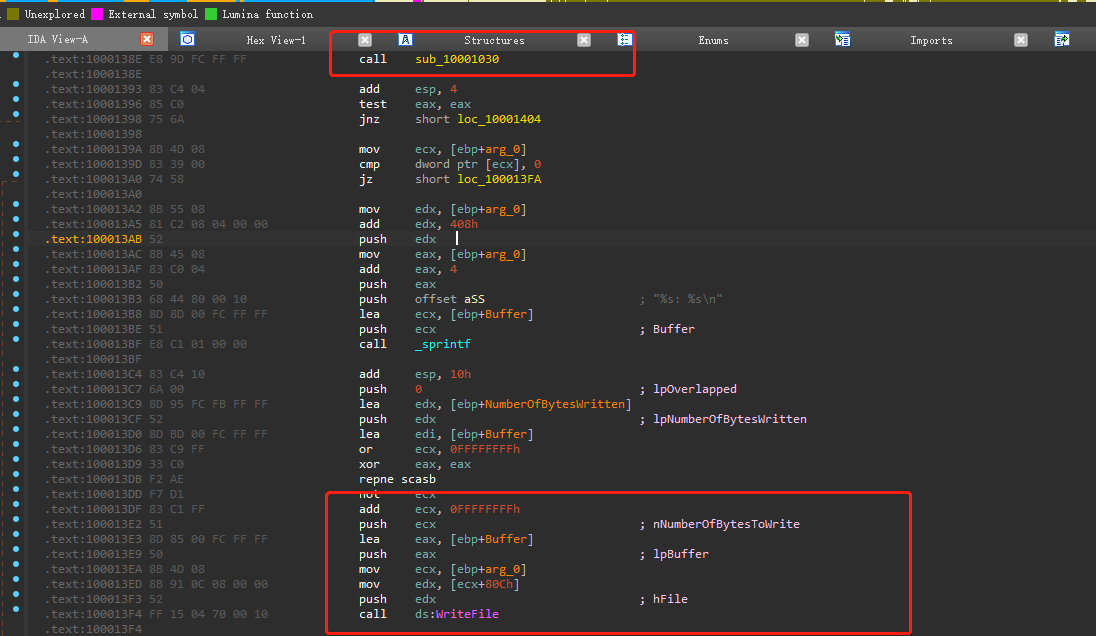
最后，这个函数跳转到了原始的入口点从而使服务能够正常执行



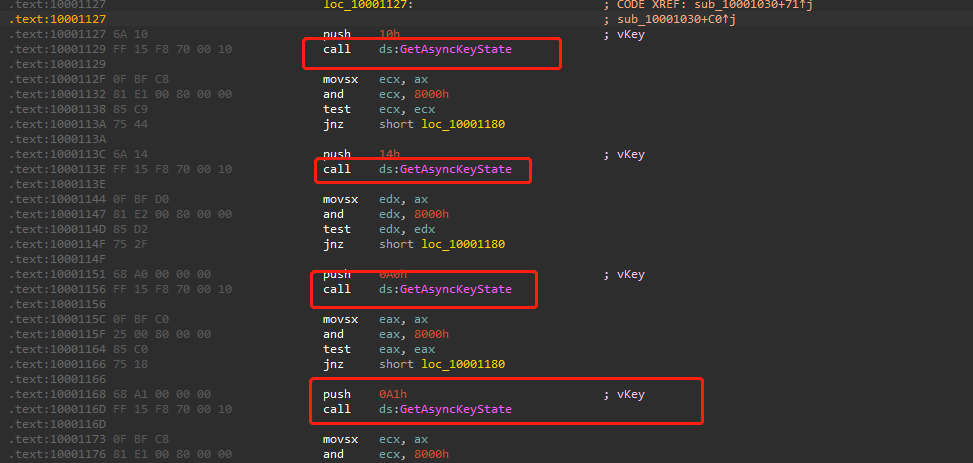
然后在刚才的上面的分析来说，最重要的就是那个thread的创建，我们需要知道被创建的thread调用了哪个函数，我们顺着他的参数10001410去idapro中进行寻找：首先在这里也看到了CreateThread这说明我们的动态分析是基本无误的



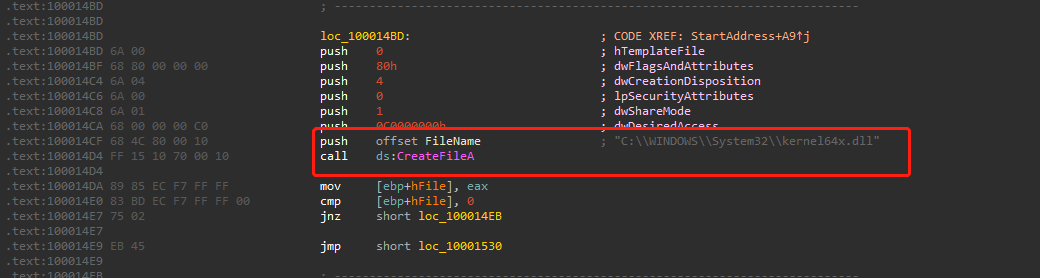
在相应的位置他先调用了10001030函数，这个函数的作用在后面一张图中体现了，这个函数中的函数调用了之前我们在导入表中看到的GetAsyncKeyState来获取键盘的输入，然后存储在buffer中并在最后由buffer统一写入最后的file中，那么file是什么呢，在下面也有说明



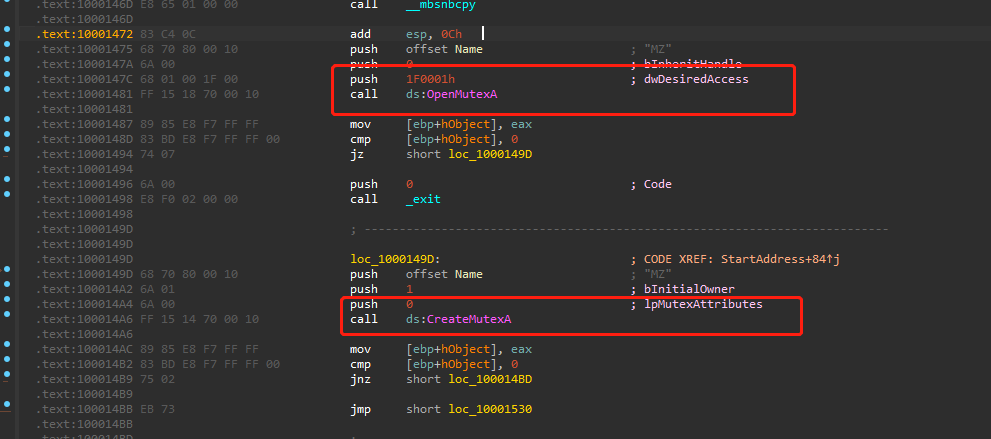
调用GetAsyncKeyState来获得键盘的输入



在这里也是可以看到对于我们刚才分析的kernel64x.dll的分析，需要将这个文件打开并向其中输入内容，最终得到的键盘输入的结果以及向哪个窗口输入都会输入得到这个文件中



可以看到这个恶意代码还创建了一个互斥量，这样只会有一个线程一直在监视对键盘的输入，避免开启很多线程消耗客户机的性能



第二部分：Yara规则编写：

Lab11-01.exe的YARA规则

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 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="2022-11-23"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule Lab1101exe{
  
 meta:
  
 description = "检查改文件是否大概率与Lab011-01exe的文件属于同一类型的恶意代码"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = "GetSystemDirectoryW"
  
 $b = "\_wstrdate"
  
 $c = "\_wstrtime"
  
 $d = "RegSetValueExW"
  
 $e = "RegCreateKeyW"
  
 $f = "gina.dll"
  
 $g = "DllRegister"
  
 $h = "WlxLoggedOnSAS"
  
 $i = "\MSGina"
  
 $j = "Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
  
 $k = "ErrorCode:%d ErrorMessage:%s."
  
 condition:
  
 (8 of them) and (IsPE and smallFileSize) or (mayBeMaleware)
  
}

Lab11-01导出的msgina的YARA规则：

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 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="2022-11-23"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule Labmsginadll{
  
 meta:
  
 description = "检查改文件是否大概率与masgina.dll的文件属于同一类型的恶意代码"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = "msutil32.sys"
  
 $b = "MSGina.dll"
  
 $c = "ShellShutdownDialog"
  
 $d = "DisableThreadLibraryCalls"
  
 $e = "WlxLoggedOnSAS"
  
 $f = "gina.dll"
  
 $g = "DllRegister"
  
 $h = "WlxLoggedOnSAS"
  
 $i = "\MSGina"
  
 $j = "Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
  
 $k = "ErrorCode:%d ErrorMessage:%s."
  
 condition:
  
 (8 of them) and (IsPE and smallFileSize) or (mayBeMaleware)
  
}

Lab11-02.dll的YARA规则

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 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="2022-11-23"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule Lab1101exe{
  
 meta:
  
 description = "检查改文件是否大概率与Lab011-02dll的文件属于同一类型的恶意代码"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = "\Lab11-02.ini"
  
 $b = "\spoolvxx32.dll"
  
 $c = "AppInit\_DLLs"
  
 $d = "spoolvxx32.dll"
  
 $e = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows"
  
 $f = "wsock32.dll"
  
 $g = "sock"
  
 $h = "MSIMN.EXE"
  
 $i = "OUTLOOK.EXE"
  
 $j = "THEBAT.EXE"
  
 $k = "RCPT TO:"
  
 $l = "installer"
  
 $m = "CreateToolhelp32Snapshot"
  
 $n = "GetCurrentThreadId"
  
 $o = "RegSetValueExA"
  
 condition:
  
 (10 of them) and (IsPE and smallFileSize) or (mayBeMaleware)
  
}

Lab11-03.exe的YARA规则

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 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="2022-11-23"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule Lab1101exe{
  
 meta:
  
 description = "检查改文件是否大概率与Lab011-03exe的文件属于同一类型的恶意代码"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = "C:\WINDOWS\System32\inet\_epar32.dll"
  
 $b = "Lab11-03.dll"
  
 $c = "cisvc.exe"
  
 $d = "C:\WINDOWS\System32\%s"
  
 $e = "net start cisvc"
  
 $f = "zzz69806582"
  
 $g = "C:\WINDOWS\System32\inet\_epar32.dll"
  
 $h = "xV4"
  
 $i = "GetOEMCP"
  
 $j = "GetACP"
  
 $k = "GetCurrentProcess"
  
 condition:
  
 (7 of them) and (IsPE and smallFileSize) or (mayBeMaleware)
  
}

Lab11-03.dll的YARA规则

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 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="2022-11-23"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule Lab1103dll{
  
 meta:
  
 description = "检查改文件是否大概率与Lab011-03dll的文件属于同一类型的恶意代码"
  
 author="wbf"
  
 date="2022-11-23"
  
 strings:
  
 $a = "C:\WINDOWS\System32\kernel64x.dll"
  
 $b = "zzz69806582"
  
 $c = "SunMonTueWedThuFriSat"
  
 $d = "JanFebMarAprMayJunJulAugSepOctNovDec"
  
 $e = "Sleep"
  
 $f = "CreateFileA"
  
 $g = "CreateMutexA"
  
 $h = "OpenMutexA"
  
 $i = "GetCurrentProcess"
  
 $j = "%s: %s"
  
 $k = "0x%x"
  
 condition:
  
 (7 of them) and (IsPE and smallFileSize) or (mayBeMaleware)
  
}

IDA-PYTHON脚本编写：

1、取出字符串

def decrypt(data):
  
 length = len(data)
  
 c = 1
  
 o = ""
  
 while c &lt; length:
  
 o += chr(ord(data[0]) ^ ord(data[c]))
  
 c += 1
  
 return o

2、找到函数的参数

def find\_function\_arg(addr):
  
 while True:
  
 addr = idc.PrevHead(addr)
  
 if GetMnem(addr) == "mov" and "esi" in GetOpnd(addr, 0):
  
 print “We found it at 0x%x” % GetOperandValue(addr, 1)
  
 break

3、综合程序，辅助获得对应参数的字符串并获得函数汇编代码

def find\_function\_arg(addr):
  
 while True:
  
 addr = idc.PrevHead(addr)
  
 if GetMnem(addr) == "mov" and "esi" in GetOpnd(addr, 0):
  
 return GetOperandValue(addr, 1)
  
 return ""
  
  
def get\_string(addr):
  
 out = ""
  
 while True:
  
 if Byte(addr) != 0:
  
 out += chr(Byte(addr))
  
 else:
  
 break
  
 addr += 1
  
 return out
  
  
def decrypt(data):
  
 length = len(data)
  
 c = 1
  
 o = ""
  
 while c &lt; length:
  
 o += chr(ord(data[0]) ^ ord(data[c]))
  
 c += 1
  
 return o
  
  
print "[\*] Attempting to decrypt strings in malware"
  
for x in XrefsTo(0x00405BF0, flags=0):
  
 ref = find\_function\_arg(x.frm)
  
 string = get\_string(ref)
  
 dec = decrypt(string)
  
 print "Ref Addr: 0x%x | Decrypted: %s" % (x.frm, dec)

4、判断是否是一个函数

def main():
  
 # Print banner
  
 printBanner()
  
  
 # First let's get the function object for the current function.
  
 functionObject = idaapi.get\_func(here())
  
 try:
  
 if (functionObject == None):
  
 print("This is not a function! Exiting!")
  
 return -1
  
 except:
  
 print("Looks like this is a function. Continuing...")

5、找到所有返回地址块的位置及汇编代码

# Locate which basic block within the flow chart array is the one that is the basic block containing the ret/retn instruction.
  
 returnBlockOffset = findReturnBlockOffset(flowChart)
  
 if (returnBlockOffset == -1):
  
 return -1
  
  
 # Get the address in virtual memory where the basic block containing the ret/retn instruction is located.
  
 returnBlockStartEA = flowChart[returnBlockOffset].startEA
  
  
...
  
  
def findReturnBlockOffset(flowChart):
  
 returnBlockOffset = 0
  
 while (returnBlockOffset < flowChart.size):
  
 if flowChart[returnBlockOffset].type == idaapi.fcb\_ret: # Taken from https://www.hex-rays.com/products/ida/support/sdkdoc/gdl\_8hpp.html#afa6fb2b53981d849d63273abbb1624bd
  
 # which shows that block type 2 is fbc\_ret, or a return block.
  
 break
  
 else:
  
 returnBlockOffset += 1
  
  
 if (returnBlockOffset > flowChart.size):
  
 print("Something is seriously wrong! Couldn't find the basic block that returns execution to the caller!")
  
 return -1
  
 else:
  
 print("Found return block at offset %d in the flowChart array!" % returnBlockOffset)
  
 return returnBlockOffset