

**恶意代码分析与防治课程实验报告**

**实验九：各种恶意行为**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2111033

姓 名 艾明旭

班 级 信息安全一班

1. **实验目的**

本章让你快速了解了恶意代码的⼀些常⻅功能。我们以不同类型的后⻔程序作为开始，然后探索了恶意代码。如何窃取受害者的登录凭证。接下来，我看到了悉意代码在系统上获得存活的各种⽅法。最后，我们展示了恶意代码如何通过隐藏它们的踪迹使它们难以被发现。现在，我们⼰经为你介绍了最常⻅的恶意代码的⾏为。

接下来的⼏章将深⼊讨论恶意代码的⾏为。下⼀章，我们讨论恶意代码如何秘密地启动。在剩余章节中，我们将看到恶意代码如何加密数据并且如何通过⽹络进⾏通信。

1. **实验原理**

**各种恶意行为：**

下载器和启动器

下载器用来将恶意代码下载下来进行执行；启动器用来秘密加载恶意代码

后门（Backdoor）

后门程序往往实现了全套功能，不需要额外下载功能代码，有一套通用功能：注册表操作、文件操作等

反向Shell：从目标机器上发起连接来接受控制，可以作为单独的代码，也可以作为组件的一部分存在

RAT：控制管理主机，通常是为了特定目标进行控制

botnet：大范围控制主机，用来进行大规模攻击

登录凭证窃密器

转储Windows口令Hash，用来进行离线破解，或Pass-The-Hash攻击

pwdump：从SAM输出本地账户LM和NTLM口令，通过注入DLL到Lsass进程进行获取，pwdump变种经常会动态获取函数，经常会见到GetProcAddress函数

PTH：也是通过DLL注入进行获取

识别转出技术很重要，但确定恶意代码对哈希做了什么操作更重要，是存在硬盘了还是上传网上了，还是PTH攻击用了

按键记录

内核态常用键盘渠道来进行检测

用户态常用API进行Hook来实现，可能会见到这几个函数：SetWindowsHookEx（设置Hook）、GetAsyncKeyState（判断按键状态）、GetForgeroundWindow（判断当前窗口）

通过字符串列表来识别按键记录器很有用（Up、Num Lock、Down、Right、Left、PageDown等。。）

存活机制

注册表有很多地方能实现存活，Run、AppInit\_DLL、Winlogon、SvcHost DLL等，一般通过procmon等工具去检测访问的注册表、通过sysinternals等工具找出自启动项

特洛伊木马化系统二进制文件，修改系统二进制文件，使其运行时加载恶意代码或DLL劫持

提权

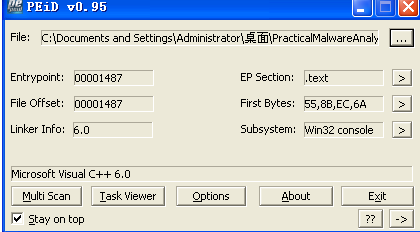
通过访问令牌来提权，据说这种方式在最新的windows上没用了，不知道是不是真的

用户态Rootkit

用来隐藏恶意代码行为的工具称为rootkit，用户态常用的有IAT Hook（过时、容易检测），InlineHook技术

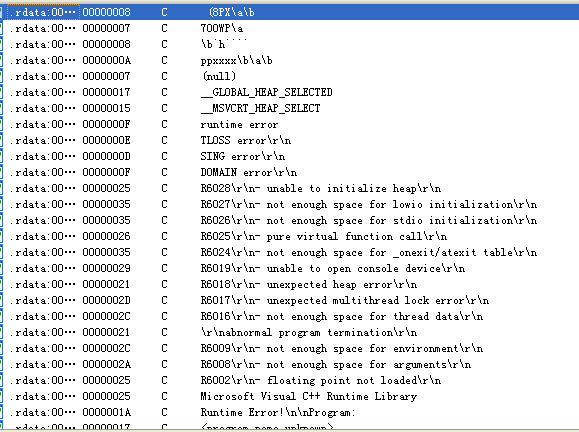
1. **实验过程**

分析目标：Lab11-01.exe 无壳



查导入表发现：设置注册表键值的函数，资源释放的函数，存在资源文件，4D5A开头是个二进制文件

查看字符串：发现了好多Wlx开头没见过的函数，以及其他一些可疑信息：



看下msginal.dll,MSGINA就是系统启动后显示出来的用户名密码窗体，长时间不操作系统桌面进入锁定状态时的窗体，以及2000系统按下CTRL+ALT+DEL后显示出来的窗体。 MSGINA导出了大量的函数，这些是与Winlogon交互必须的。

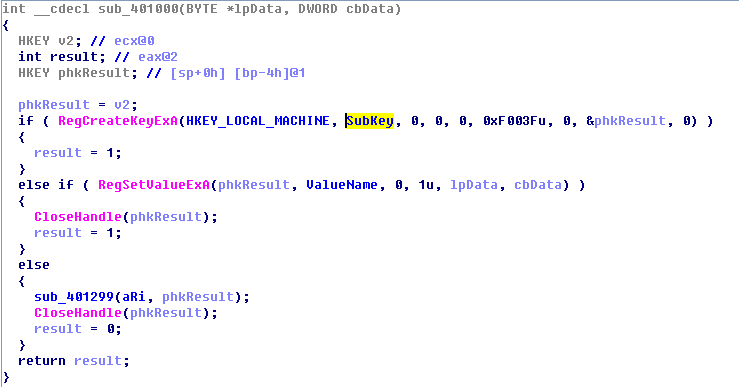
当然了，一开始应该直接看下全局的东西，不能太陷入细节，先来下反汇编：

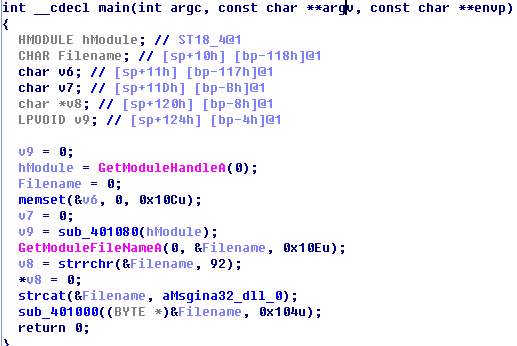
**从这个看，关键的东西在2个 sub里面**

**其中一个在加载资源，另外一个在修改注册表。注册表看下subkey，**

**是winlogon，可以看到是在做持久化，登录的时候就运行。**

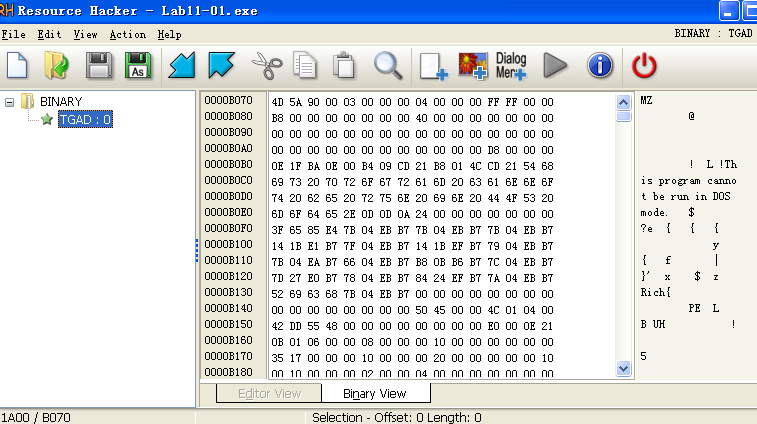
**加载资源的，大概率是在将资源当成二进制exe使用。导出下再反编译看看：**

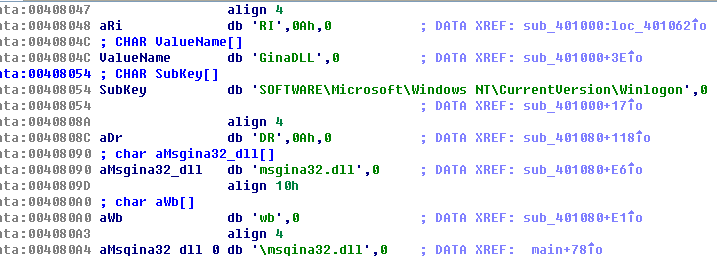


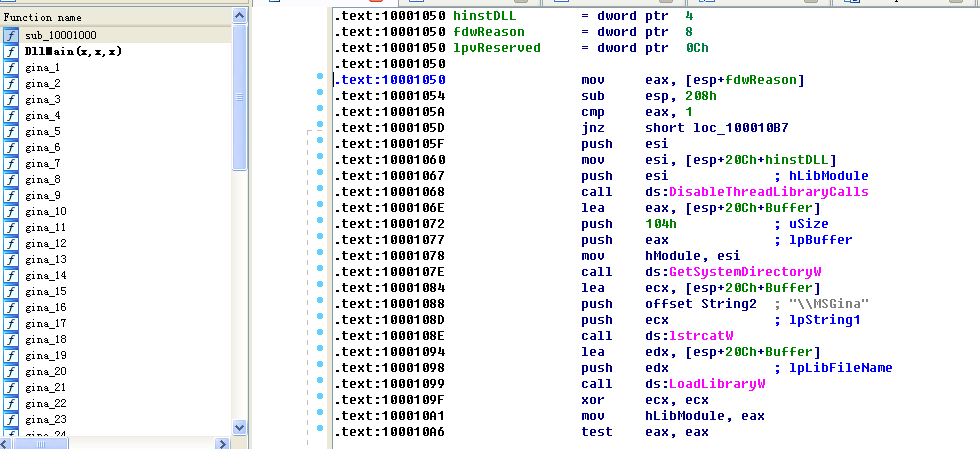


有的gina函数都是在调用sub10001000，估计是在劫持hook。

我们使用resource hacker将相关的可执行文件打开，并且存储到相应的位置。



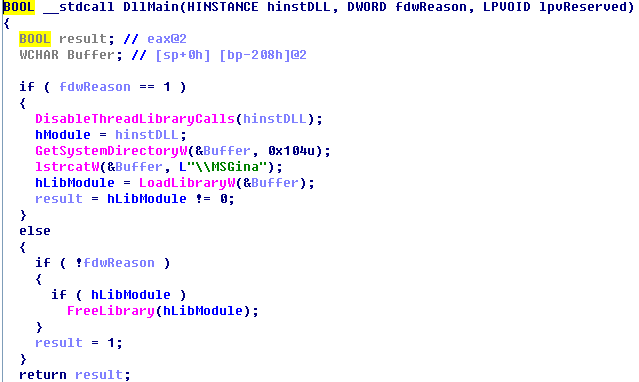




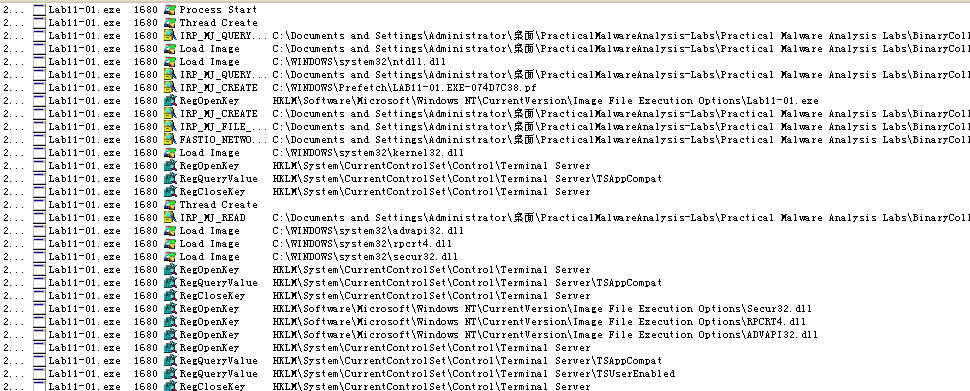
好了有一个大致印象了。我们继续往下：

可以回答实验的问题

1.这个恶意代码向磁盘释放了什么？



在process monitor当中运行监控，可以找得到该恶意代码向进程所在目录释放了msgina32.dll文件，还修改了注册表，设置了GinaDLL的值，是个二进制数据，没有观测到网络行为



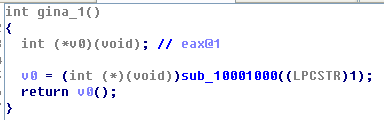
2.这个恶意代码如何进行驻留？

WindowsXP通过注册表HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL来设置需要WinLogon加载的第三方DLL，恶意代码将释放出来的msgina32.dll设置到了这个注册表里

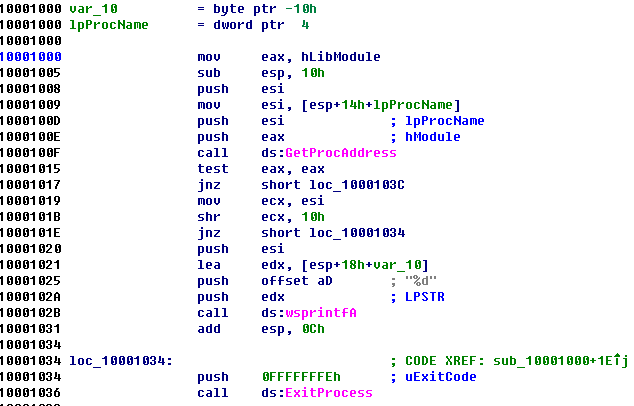
3.这个恶意代码如何窃取用户登录凭证？

静态分析看见exe文件仅仅是做了资源释放和注册表设置两件事，其他功能应该是在资源文件里实现的，分析资源文件，无壳，字符串里是Wlx那堆函数，导入表没啥特别的，有注册表操作相关API，看到一堆Wlx开头函数，以及GinaDLL字符串，这里应该是用了GINA拦截的操作（有点像DLL劫持操作）

这里把系统原本的msgina.dll给加载了，然后把句柄存到全局变量里，然后dllmain就结束了，因为是DLL劫持，所以功能不是全部都在dllmain中实现的，观察旁边的函数列表：



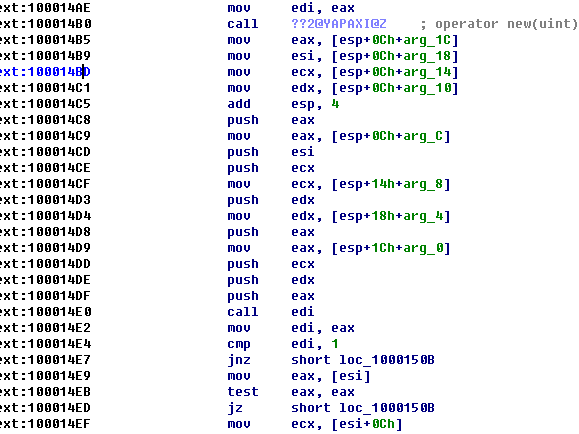
基本上全部都是调用sub\_10001000函数，这应该是个函数转发，查看：



确实是这样，这里是从原本的dll中获取函数地址，然后返回函数地址，然后再返回出来之后直接jmp过去

4.查了下WlxLoggedOutSAS，可以通过他找到用户登录的账户名和密码。

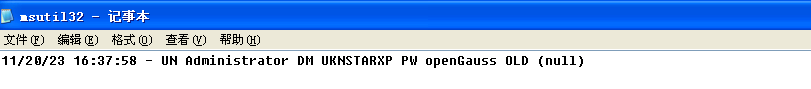
5.前面先是正常调用了函数，然后把参数里的关键信息入栈调用了sub\_10001570函数：



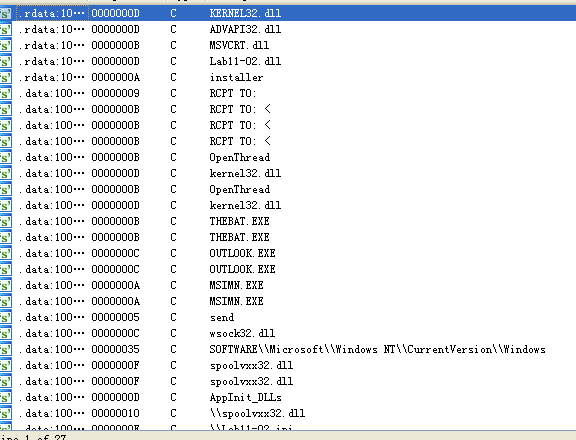
这个函数主要就是打开文件msutil32.sys然后把数据写进去了



1. 至于写的内容是什么？想要了解很简单，就是运行目标文件后就会生成msgina32.dll 文件然后再用IDA分析 msgina32.dll。
2. 这个恶意代码对窃取的证书做了什么处理？把信息记录在了msutil32.sys中
3. 如何在你的测试环境让这个恶意代码获得用户凭证？要重启系统才能触发：



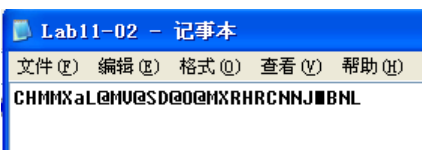
Lab10-02.exe



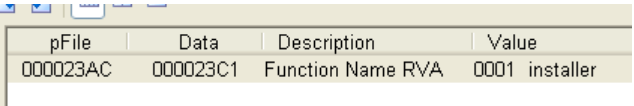
1.这个恶意DLL导出了什么？

导出了installer函数 可以使用dependency walker查看。首先对Lab11-02.dll进行基础静态分析。发现如下字符串，其中尤其要注意AppInit\_DLLs，这个 Appinit\_DLLs会在系统启动之前就把恶意DLL加载在Windows服务中，来达到驻留的目的，字符串中恰好有一 个注册表项。\Lab11-02.ini表明了这个程序有可能使用文件Lab11-02.ini。除此之外，还有一些像 THEBAT.EXE、OUTLOOK.EXE还有MSIMN.EXE的字符串，wsock32.dll说明这个程序可能会使用网络，RCPT为 SMTP协议中的一个命令，说明这个程序可能用了与邮件有关的功能。

然后查看文件Lab11-02.ini的内容，发现是无意义的乱码，猜测这个程序很可能有加解密的功能。

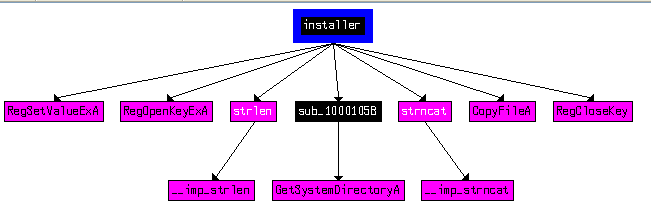


查看Lab11-02.dll的导入导出表，发现只有一个导出函数installer。导入函数有对注册表、文件进行操作的函数，还有一个CreateToolhelp32Snapshot，用于搜索一个进程或者线程列表。

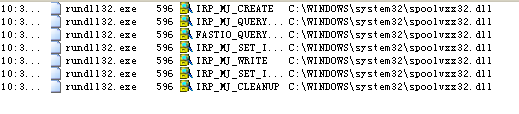


然后看一下交叉引用图

从交叉引用的图中可以更加直观的看出这个installer调用了哪些函数 结合流程框图可以看出，这个install函数在刚刚我们看见的注册表的位置，设置了一个名为 spoolvxx32.dll 的文件，并在最后会复制文件。 经过分析我们可以知道，这个动态链接库导出了一个具有安装恶意代码自身功能的函数。

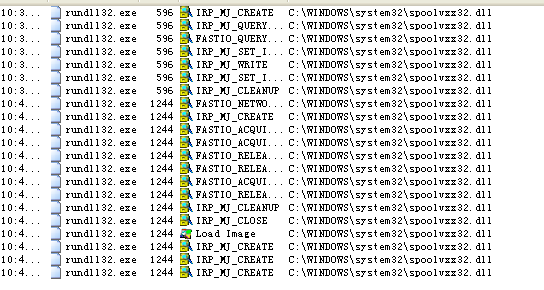


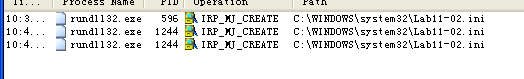
1. 使用rundll32.exe安装这个恶意代码后，发生了什么？==》使用process monitor监控恶意程序的运行，并设置过滤器：



使用命令rundll32.exe Lab11-02.dll,installer运行恶意代码，它会将自己复制到文件 C:\WINDOWS\system32\spoolvxx32.dll中，并且在键值AppInit\_DLLs下永久安装。它尝试在路径 C:\WINDOWS\system32\中打开Lab11-02.ini，但是并没有找到目标文件。

之后发现，该文件尝试在路径C:\WINDOWS\system32\中打开Lab11-02.ini。于是我们将这个文件放在指定 路径下，重新运行，发现这个恶意代码会在最后将自己加载到user32.dll中，使得所有加载了user32.dll的进程也会加载它。





可以看见这个恶意代码创建了 AcGenral.dll

同时这个恶意代码还在系统目录下创建了名为 spoolvxx32.dll 的文件。 并且通过计算文件的MD5值，我们可以发现这个新创建的dll和本次实验的样本dll文件是同一个文件。 综上，这个恶意代码在运行以后会将自己复制到Windows的系统目录下。

问题3

为了使这个恶意代码正确安装，Lab11-02.ini必须放置在何处？

答： 必须放在路径C:\WINDOWS\system32\下。

问题4

这个安装的恶意代码如何驻留？

答： 它将自身的副本spoolvxx32.dll添加到AppInit DLLs列表中，将自己加载到user32.dll中，使得所有加载 了user32.dll的进程也会加载它。



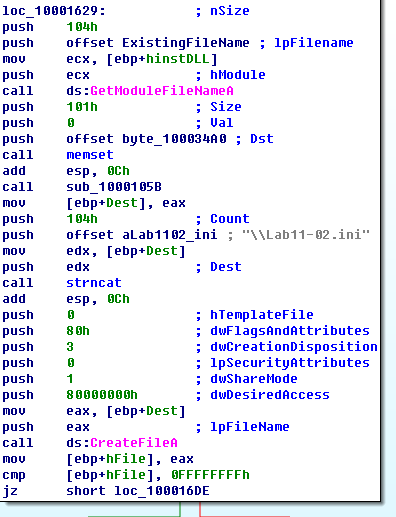
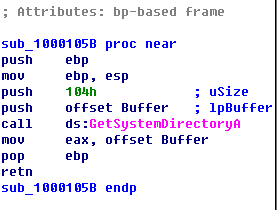
通过边上的详细内容，我们可以看见添加的就是刚刚复制出来的dll文件，由此实现了驻留。这里就可以 学到，之前修改注册表一般是修改RUN中的内容，但是除此之外，还可以修改这个AppInit的内容达到驻留和自启动的目的。

问题5

这个恶意代码采用的用户态 Rootkit 技术是什么？

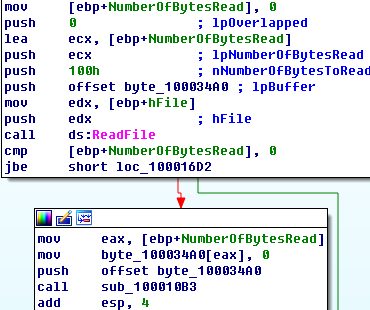
答： 针对wsock32.dll中的send函数安装了一个inline hook

下面使用IDA打开Lab11-02.dll进行分析，首先分析导出函数installer。首先调用了函数RegOpenKeyExA， 打开键SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows，打开成功才会继续执行。调用函数 strlen计算字符串长度并传给下一函数RegSetValueExA当参数，将键AppInit\_DLLs的值改为了 spoolvxx32.dll。



之后会调用sub\_1000105B，进入查看，会发现这个函数是用来查询系统目录的，返回值是查询到的路径。之 后调用strncat拼接字符串，最终会得到字符串C:\WINDOWS\system32\spoolvxx32.dll。之后调用函数 CopyFileA，将Lab11-02.dll复制到上面那个文件中。

总之，installer函数复制恶意代码到spoolvxx32.dll，并将它设置为一个APPInit\_DLLs值。 下面我们开始分析主函数。和之前一样，它首先检查是否是DLL\_PROCESS\_ATTACH状态，是的话才会执行后面 的代码。 调用函数sub\_1000105B返回系统目录的路径，之后依然调用了strncat拼接字符串，这次拼接出来的字符串 C:\WINDOWS\system32\Lab11-02.ini。之后调用CreateFileA创建文件C:\WINDOWS\system32\Lab11- 02.ini，并调用ReadFile来读这个文件。



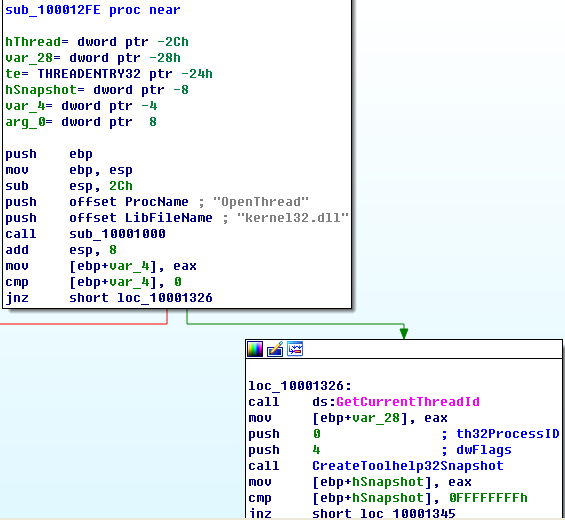
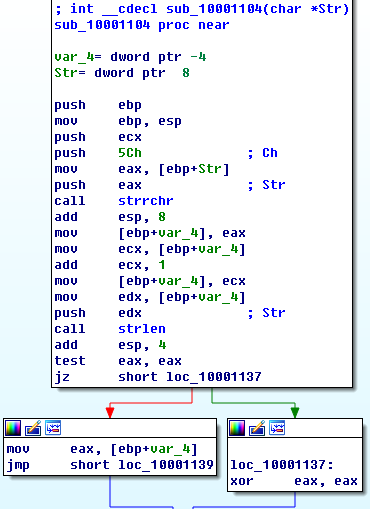
之后调用函数sub\_100010B3，参数是从上面的文件中读到的内容，进入这个函数查看发现像是一个解密的函数，于是通过OD运行并查看结果。

在调用这个函数的位置0x100016CA处设置一个断点，命中断点后，step over，可以看到解密出的内容是一个邮箱地址billy@malwareanalysisbook.com，并存储在全局变量byte 100034A0中，于是我们在IDA中将它 重命名为email\_address，以便于分析。



之后还调用了函数sub\_100014B6，这是一个hook install函数，会在这里安装恶意代码的 hook。 首先，它会比较第一个参数和0是否相等，非0才会继续向下执行。之后调用sub\_10001075返回系统的系统路 径，再调用函数sub\_10001104，我们进入这个函数查看。先调用了函数strrchr，找到最后在字符串中最后出 现某个字符的位置，根据它的参数5Ch（即\）和str，我们知道这个函数的返回值是\system32，之后将这个 字符串向后偏移了一位，变成system32，然后调用strlen计算该字符串的长度，检查该字符串是否为空。 从这个函数返回后，将返回值（system32）赋给了[ebp+Buf1]，并判断是否为0，如果为0就跳转结束；否则会 接着调用函数sub\_1000102D，我们使用OD动态分析这个函数。可以看见，此时入栈的参数是字符串 LOADDLL.EXE，通过动态运行我们可以看出，函数sub\_1000102D实质上就是将字符串中的字母由小写变成大 写，之后再计算该字符串的长度，用于比较。通过memcmp函数来比较该字符串与THEBAT.EXE是否相等，如果 不相等就再与OUTLOOK.EXE和MSIMN.EXE进行比较。一旦发现相等的就继续执行下面的代码，如果都不相等就 跳转到结束的位置。

比较成功的话，首先会调用函数sub\_100013BD，进入这个函数，发现它首先会调用GetCurrentProcessId获 取进程的PID，之后调用sub\_100012FE。



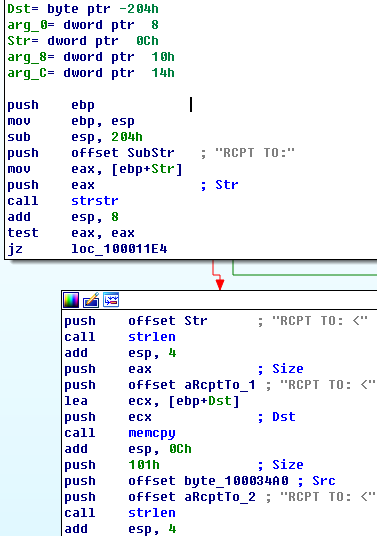
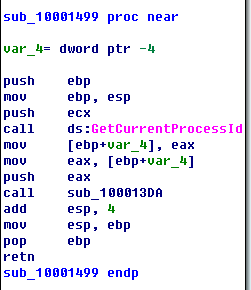
进入函数sub\_100012FE，发现它首先调用sub\_10001000获取kernel32.dll的基地址，之后调用 GetCurrentThreadId获取当前线程的标识符，再调用CreateToolHelp32Snapshot来获取当前进程的快照， 以及这些进程使用的堆，模块和线程，参数dwFlags的值为4，即TH32CS\_SNAPTHREAD，意思就是要获取的快 照包括系统中的所有线程，同时会枚举线程。将返回值保存到hSnapshot中。



之后调用Thread32First、Thread32Next以及SuspendThread遍历并挂起这些线程。至此，函数 sub\_100013BD分析完毕。 这个hook调用的下一个函数是sub\_100012A3，这个函数用来安装hook，进入这个函数进行分析，发现这个函 数首先查找wsock32.dll的地址，之后函数在wsock32.dll中查找send函数的地址，并且把这个地址存在 lpAddress中。之后调用sub\_10001203，进入这个函数分析。

首先会调用函数VirtualProtect，这个函数用来更改调用进程的虚拟地址空间中已提交页的区域的保护。之 后调用malloc分配了一个0FFh大小的空间，再调用memecpy在分配好的空间复制了send函数中的前五个字节， 保证send函数代码的完整性。 之后进行了一系列的堆栈操作，把memcpy函数分配的空间的地址赋值给了edx，将0E9h（这是jmp指令的操作 码）赋值给从偏移量为0Ah的地址。要从恶意函数回到正常的send函数中，我们就需要知道send函数到我们分 配的内存空间的地址差，然后利用这个地址差来跳转到前面复制了5字节send函数代码的空间中，继续执行 send函数，好像什么都没发生过一样。最后会再次调用VirtualProtect来对我们的地址进行保护更改。

函数返回，hook最后调用sub\_10001499，使用ResumeThread恢复所有的线程。 此外，我们发现调用函数sub\_100012A3时还有一个int型的参数为sub\_1000113D，于是我们也进入这个函数 查看一下。它首先调用了strstr，它的两个参数分别是RCPT TO:和函数sub\_1000113D的第二个参数。再查看 后面的函数调用与参数，推测这是一个构造邮件的函数。如果再邮件中发现了RCPT TO:，这个代码会将这台电 脑上发送的邮件都再给我们在.ini文件中解密出的邮箱中发一份，起到窃听邮件的作用。

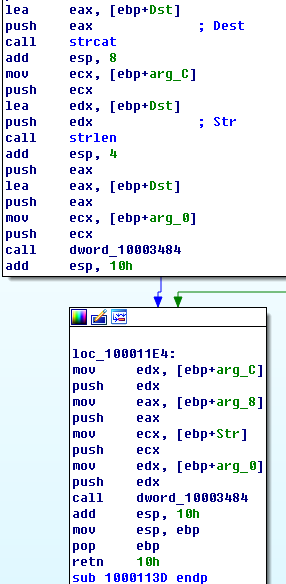


问题6 挂钩代码做了什么？

答： 通过上面的分析，我们知道这个挂钩会检查向外发出的包中是否是包含 RCPT TO:的电子邮件信息，如果是， 它会增加一个恶意代码的RCPT TO邮箱，将这些消息转发到指定的信箱里，起到窃听邮件的作用。

进入到我们认为是Hook函数的1000113D

可以看见是先和RCPT的这个字符串进行比较，如果没有找到这个字符串就退出，找到以后才能执行接下来的功能。



可以看见是先和RCPT的这个字符串进行比较，如果没有找到这个字符串就退出，找到以后才能执行接下来的功能。 在发现了RCPT TO这个字符串以后，恶意代码会再构建一个RCPT TO，这里的emailAddr 是之前在解码后得到的email地址。那么这个hook的功能就是再添加一个恶意的邮箱地址，并进行发送。

问题7 哪个或者哪些进程执行这个恶意攻击，为什么？

答： 这个恶意代码仅针对的是程序MSIMN.exe、THEBAT.exe或者OUTLOOK.exe，它们都是Windows操作系统中默认 的与邮件发送相关的程序。

在这里其实就是查看当前进程的名字是不是这几个，如果是的话才会执行攻击，否则退出。而这几个程 序都是属于电子邮件客户端的程序，这样能够将自己的功能隐藏在这些进程中，不会容易被发现。

问题8

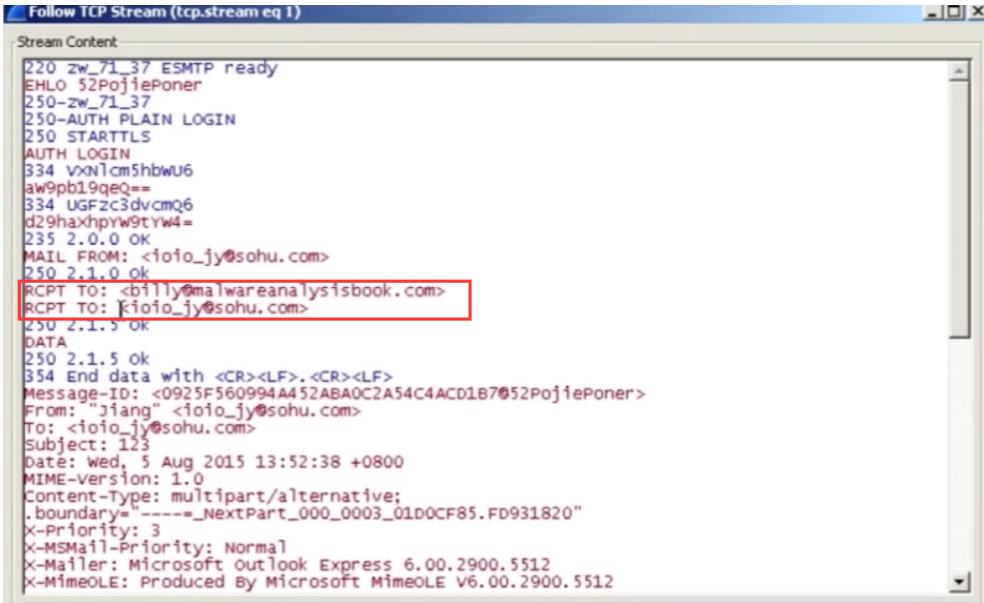
.ini文件的意义是什么？

答： 这个.ini文件中存储了一个加密后的恶意邮箱地址，在运行.dll文件的过程中被解密出来并使用。

问题9

你怎样用 Wireshark 动态抓获这个恶意代码的行为？

答：可以通过Wireshark看到这个恶意代码向指定的恶意邮箱发送邮件信息。



首先我们知道了这个恶意代码是附带在邮件程序中进行的，如果没有这几个程序是没有办法捕获到恶意 代码的行为的， 所以我们安装了一个outlook程序，用来收发邮件。 之后我们使用这个程序随便发送一封邮件，同时使用wireshark捕捉一下网络数据包

我们捕获到了一个SMTP的数据包，在打开以后发现，这里不仅发送了我们输入的邮箱地址，还发送到了 之前解析出来的恶意邮箱

LAB11-3

问题1

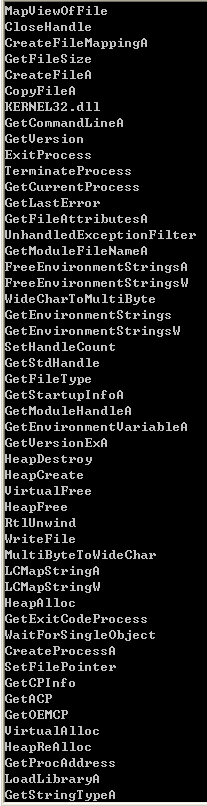
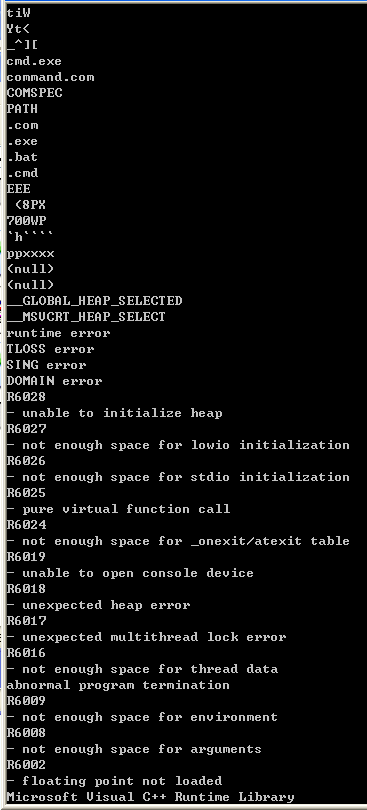
使用基础的静态分析过程，你可以发现什么有趣的线索？

先简单使用srings工具查看一下有没有什么有趣的字符串

lab11-03.exe

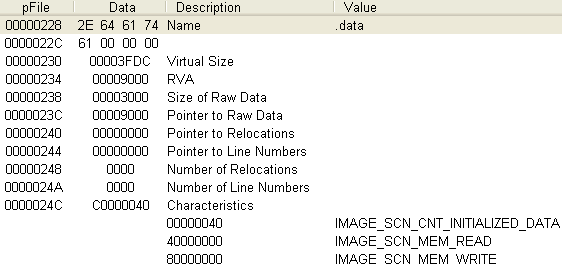
在exe的检查中可以看见有一个字符串是 net start cisvc ，这里就是启动一个名为cisvc的服务，经过 查询可以知道，这个服务是用来检测系统内存的。之后可以看见有一个系统路径的dll文件，说明这个恶 意代码可能会在这个位置创建一个dll文件。

lab11-03.dll





dll中出现的字符串就比较杂乱，其中有一些是关于星期、月份的，还有一个比较值得注意的就是图中框 出来的系统路径下的dll文件。

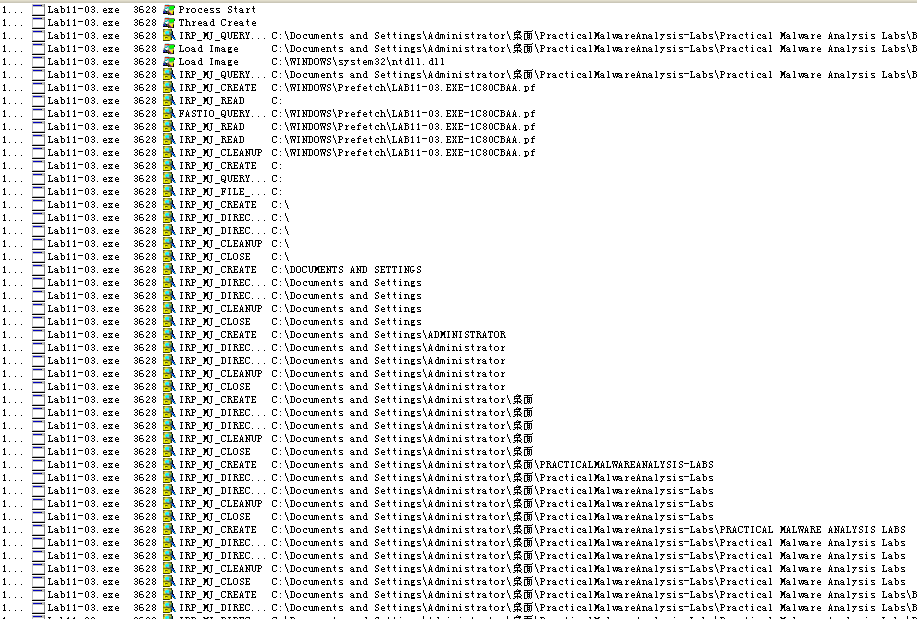


问题2

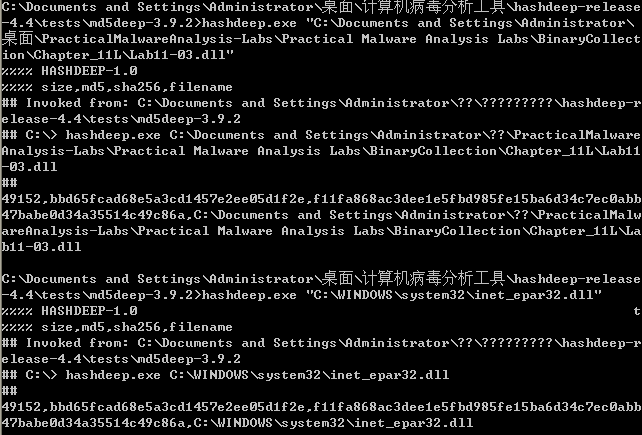
当运行这个恶意代码时发生了什么？

答：它创建了文件C∶\Windows\System32\inet\_epar32.dll，并将Lab11-03.dll复制到该文件中。打开了 cisvc.exe并且启动该索引服务。同时也将击键行为记录到文件C:\Windows\System32\kernel64x.dll中。

同样还是使用procmon进行监视，可以观察到弹出了一个什么内容都没有显示的命令行窗口，然后这个窗口很快就关闭了。

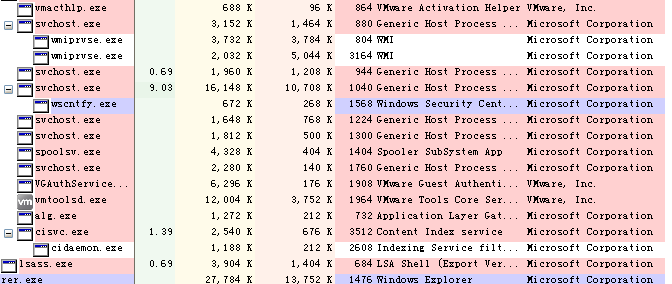


可以看见恶意代码在系统目录下创建了一个inet\_epar32.dll的文件，猜测这个dll文件和lab11-03.dll是同一个文件，所以还是计算一下MD5的值

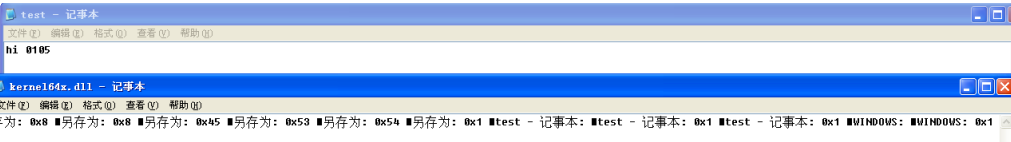


果然是一样的，也就是说这个恶意代码把自己的dll文件复制到了系统目录下，并重命名为了 inet\_epar32.dll 并且发现这个代码试图打开之前说的那个exe文件

之后，恶意代码打开了cisvc.exe，但是并没有进行任何 写文件的操作。最后，恶意代码通过命令net start cisvc来启动该索引服务。通过Process Explorer我们观察到cisvc.exe正在运行。



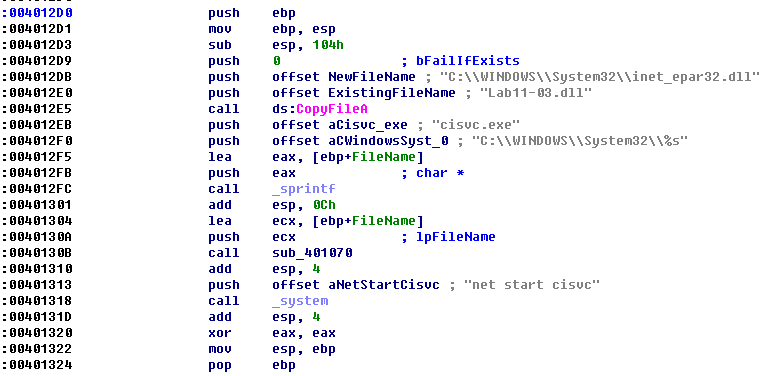
由于我们怀疑该恶意程序是一个击键记录器，于是我们打开 记事本输入一串字符串进行测试。可以看到文件 kernel64x.dll被创建，打开该文件，发现它确实记录了我们的击键行为。

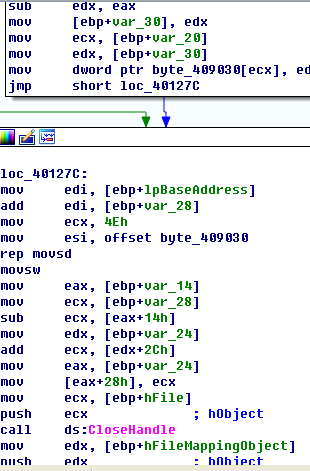


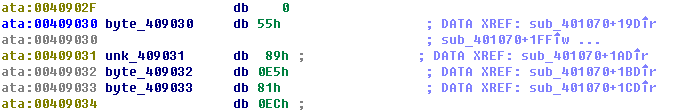
问题3 Lab11-03.exe 如何安装Lab11-03.dll 使其长期驻留？

答： 它首先将Lab11-03.dll复制到inet\_epar32.dll中，然后对cisvc.exe进行了入口重定向，使得无论什么时 候运行cisvc.exe，都将先执行shellcode而不是原始的程序入口点。该shellcode用来加载 inet\_epar32.dll，并且调用它的导出函数，从而使得Lab11-03.dll长期驻留。

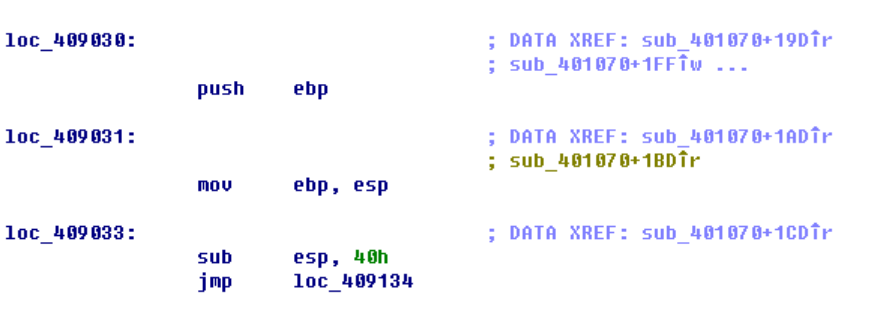
接下来我们用IDA打开Lab11-03.exe，从main函数开始分析。 首先调用了函数CopyFileA，根据参数我们知道确实是将Lab11-03.dll复制到inet\_epar32.dll中。接下 来，它创建了字符串C:\WINDOWS\System32\cisvc.exe并且将这个字符串传给sub\_401070作参数。最后使用 命令net start cisvc，来启动索引服务。



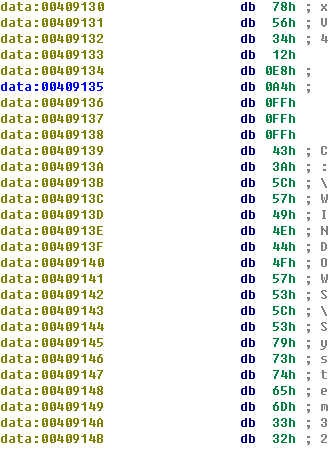
下面我们重点分析函数sub\_401070。首先，它调用了包括CreateFileA、GetFileSize、 CreateFileMappingA和MapViewOfFile的一系列文件操作函数，创建并将cisvc.exe映射到内存上。之后调 用UnmapViewOfFile停止该程序的一个内存映射，这解释了我们为什么没有在procmon中看到WriteFile操 作。 之后有一系列的赋值和计算操作，跳过这些，我们将重点放到写入文件的数据上，然后提取硬盘上的 cisvc.exe来进行分析。lpBaseAddress中记录着文件的映射位置，它被赋值给edi寄存器，加上var\_28进行 偏移，之后将0x4E赋值给ECX，循环进行写操作movsd，总共写入了0x4E\*4=312字节。最后，byte\_409030被 赋值给esi，其中的数据也被到映射到文件中。



反汇编结果如下，这就是写入cisvc.exe的shellcode。

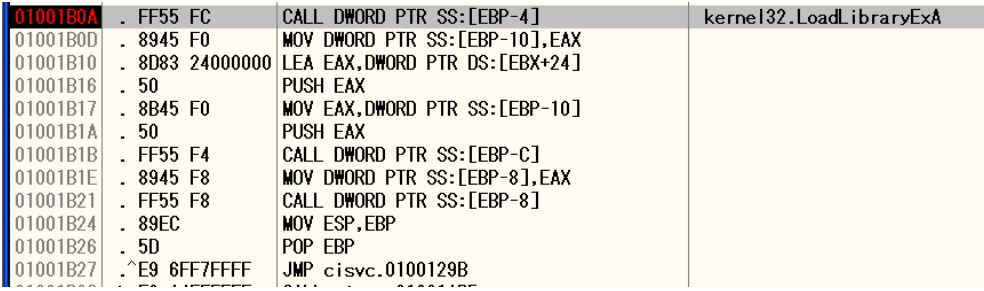


在shellcode的末尾，我们看到字符串C:\WINDOWS\System32\inet\_epar32.dll和zzz69806582，说明这个shellcode加载了这个DLL，并且调用了它的导出函数。

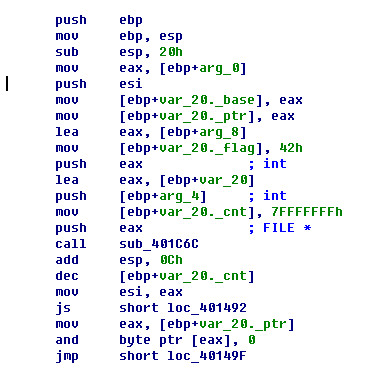


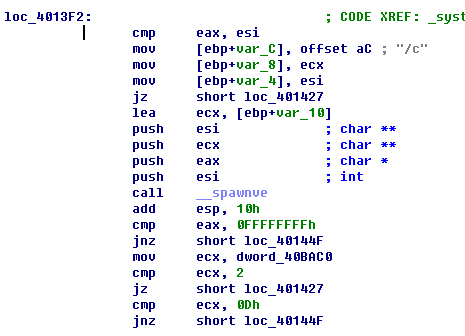
比较被感染前后的cisvc.exe，可以看到程序的入口点发生了变化，并且添加了大量的代码。恶意代码执行了 入口重定向，使得无论什么时候运行cisvc.exe，都将先执行shellcode而不是原始的程序入口点。  
下面使用IDA和OD打开被感染的cisvc.exe进行分析，我们发现恶意代码调用LoadLibrary载入 inet\_epar32.dll，之后调用GetProcAddress获取导出函数zzz69806582的地址，然后根据得到的地址调用 导出函数。最后跳转到程序的原始入口点，使得服务正常执行。





总之，这个shellcode用来加载inet\_epar32.dll，并且调用它的导出函数。 接下来我们分析一下inet\_epar32.dll即Lab11-03.dll做了什么。我们用IDA打开Lab11-03.dll开始分析， 发现DLLMain很短，并没有做什么，于是我们分析导出函数。发现它调用CreateThread创建了一个线程然后就 返回了，下面我们去分析这个线程。



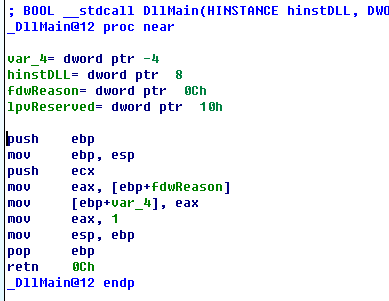


问题4 这个恶意代码感染 Windows 系统的哪个文件？

答： 为了每次都加载inet\_epar32.dll（即Lab11-03.dll），该恶意代码感染了cisvc.exe，对cisvc.exe进行 了入口重定向，使得无论什么时候运行cisvc.exe，都将先执行shellcode而不是原始的程序入口点，从而加 载inet\_epar32.dll和它的导出函数zzz69806582。

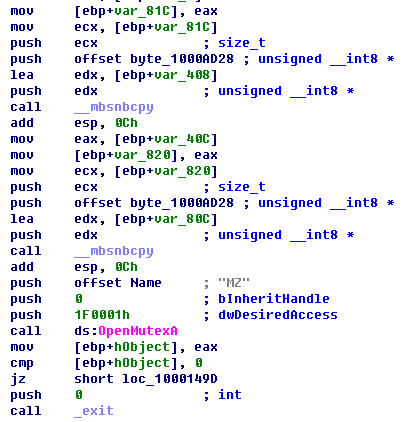
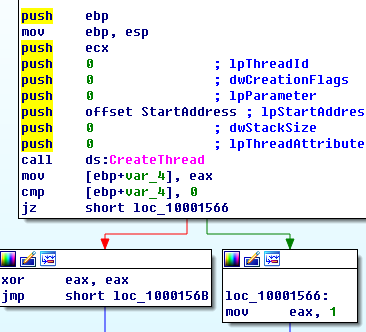
问题5 Lab11-03.dll做了什么？

答： Lab11-03.dll是一个轮询的击键记录器，这用它的导出函数 zzz69806582中实现。



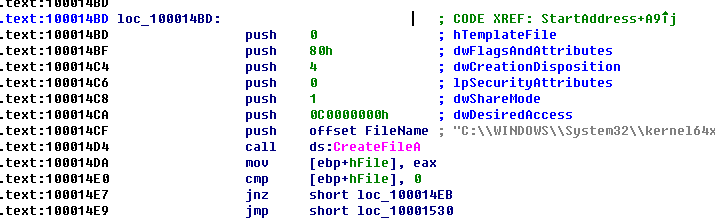
我们使用IDA分析这个dll文件，同时通过之前的分析可以知道这个dll文件主要的不是dllmainentry，而是他的另一个导出函数，所以这里就只分析这个导出函数。

可以发现这个导出函数先创建了一个线程，然后这个函数就结束了，但是在创建线程的时候，其中设置了一个启动函数的地址。



进入这个函数可以发现，这个函数首先查看系统中有没有一个名为MZ的互斥量，如果没有就会创建。

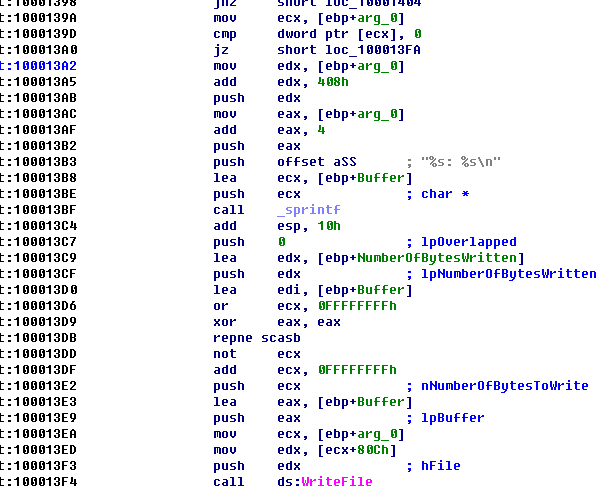
这个互斥量的作用自然就是防止多个程序的运行，确保每个时刻最多只有规定数量的程序在运行。



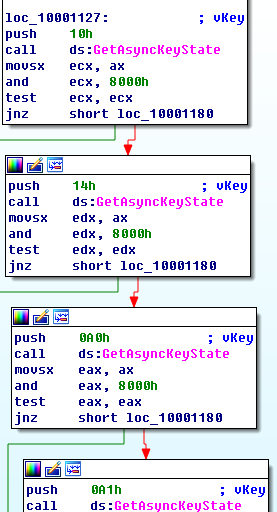
然后可以看见这个代码创建了系统目录下的一个dll文件。



之后通过 SetFilePointer 函数，将指针移动到文件的末尾（dwMoveMethod的参数为2） 进入到后面调用的sub\_10001380函数，可以看见



这里调用了写文件的函数，以及上面出现了一些格式化字符串，那么整体来说这个函数的功能就是像刚刚打开的文件的末尾写入一些内容。



然后函数多次调用了 GetAsyncKeyState 这个函数，来识别一个按键是否被按下或者弹起（对比按键的 状态），也就是轮询来获取键盘状态的变化，也就获得了键盘的输入。 综上，这个dll文件会创建一个线程，并通过创建互斥量来保证同时只运行一个线程，然后打开系统目录 下的一个dll文件，通过对比键盘状态来记录当前窗口的输入。

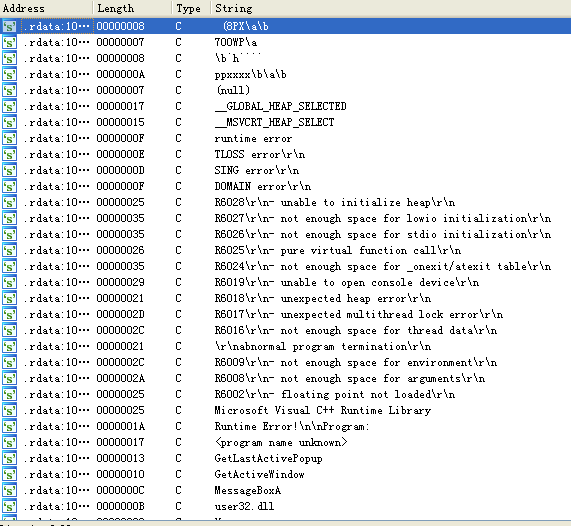
问题6

这个恶意代码将收集的数据存放在何处？

答： 这个恶意代码会存储击键记录和输入记录，收集的数据都被存放在 C:\Windows\System32\kernel64x.dll 中。

**YARA规则撰写**

根据我们观察到的字符串，编写yara规则



import "pe"

rule EXE { strings:

$exe = ".exe" nocase

condition:

$exe}

rule DLL { strings:

$dll = /[a-zA-Z0-9\_]\*.dll/

condition:

$dll}

rule Wlx { strings:

$WlxFuncs = /Wlx[a-zA-Z]\*/

condition:

$WlxFuncs }

rule Gina { strings:

$name = "Gina"

condition:

$name}

rule Regedit { strings:

$system = "NT"

$software = "SOFTWARE"

$winlogon = "Winlogon"

condition:

$system or $software or $winlogon }

rule INI { strings:

$name = /[a-zA-Z0-9\_]\*.ini/

condition:

$name}

rule Service { strings:

$start = "net start"

condition:

$start }

rule RCPT { strings:

$name = "RCPT TO"

condition:

$name}

优化这个规则，做出如下改写：

rule Lab11 { meta:

description = "rules for Lab11"

strings:

$a0 = "msgina32.dll"

$a1 = "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\winlogon\\GinaDLL"

$a2 = "msutil32.sys"

$a3 = "spoolvxx32.dll"

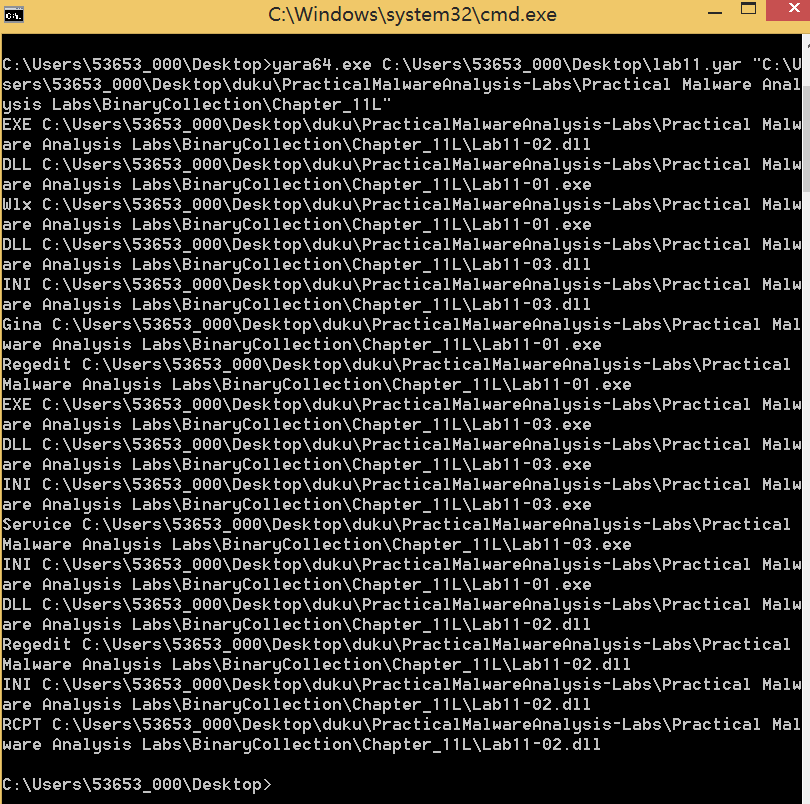
$a4 = "billy@malwareanalysisbook.com"

$a5 = "inet\_epar32.dll"

$a6 = "kernel64x.dll"

$a7 = " zzz69806582"

condition: any of them }



**IDAPython**

首先对某字符串进行搜索，找到后返回字符串地址：

print hex(FindBinary(MinEA(),SEARCH\_DOWN,'HGL345'))

print hex(FindBinary(MinEA(),SEARCH\_DOWN,'http://www.malwareanalysisbook.com'))

从当前地址查找第一个指令并返回指令地址，从当前地址查找第一个数据项并返回数据地址。

print hex(FindCode(MinEA(),SEARCH\_DOWN))

print hex(FindData(MinEA(),SEARCH\_DOWN))

获取代码段中的所有函数、函数中的参数、函数名及函数中调用了哪些函数。

for seg in Segments():

#如果为代码段

if SegName(seg) == '.text':

for function\_ea in Functions(seg,SegEnd(seg)):

FunctionName=GetFunctionName(function\_ea)

print FunctionName

nextFunc=NextFunction(function\_ea)

print nextFunc

遍历所有函数，并查找所有对每个函数执行的调用，引用将存储在两个字典中。

from sets import Set

ea=ScreenEA()

Par=dict()

son=dict()

for fun in Functions(SegStart(ea),SegEnd(ea)): #遍历函数

f\_name=GetFunctionName(fun)

Par[f\_name]=Set(map(GetFunctionName,CodeRefsTo(fun,0))) #创建一个集合，其中包含调用（引用）的所有函数的名称

for fun\_son in CodeRefsTo(fun,0): #遍历所有的引用

fname\_son=GetFunctionName(fun\_son) #获取引用函数的名称

son[fname\_son]=son.get(fname\_son,Set())

son[fname\_son].add(f\_name); #将当前函数添加到函数列表中

functions=Set(Par.keys()+son.keys()) #获取所有函数的列表

for per in functions:

print "%d %s %d" % (len(Par.get(per,[])),per,len(son.get(per,[])))

得到如下的部分结果

0x401000, RegSetValueExA

0x401040, RegOpenKeyExA

0x401070, strlen

0x4010a0, \_\_imp\_strlen

0x4011e0, sub\_1000105B

0x401440, \_main

0x401820, installer

0x401930, GetSystemDirectoryA

0x401936, strncat

0x40193c, \_\_imp\_strncat

0x40194e, CopyFileA

0x401951, RegCloseKey

0x401960, \_retcopylen

0x401966, \_strat

1. **实验结论及心得体会**

**后门攻击是我们网络安全和信息安全的主要攻击方式之一，很多的软件和网络应用都存在着各种各样的后门，本次实验我们学习利用到的后门攻击，是经过隐藏，需要我们跟随ida不断向下分析才能找得到的后门，需要我们努力寻找努力纠错。在实验的过程当中，我体会到了ida,ollydbg,process monitor,peview，resourcehacker，dependency walker等等软件的协同操作解决本次实验的目标问题，感到非常困难，挑战性很强。**

**其中我们还和r77实验当中利用网络传输进行验证的方法类似，学习到了利用网络传输协议进行后门攻击和恶意软件的验证，是一种很独特，但又很有意思的验证方式，值得我们进一步学习了解。**

**本次实验还进行了yara规则优化以及ida python的编写，熟悉了上述的操作，在学习本课程的过程不断进步。**