

**恶意代码分析与防治课程实验报告**

**实验11**

****

专 业 信息安全

学 号 2113662

姓 名 张丛

班 级 信安一班

1. **实验目的**

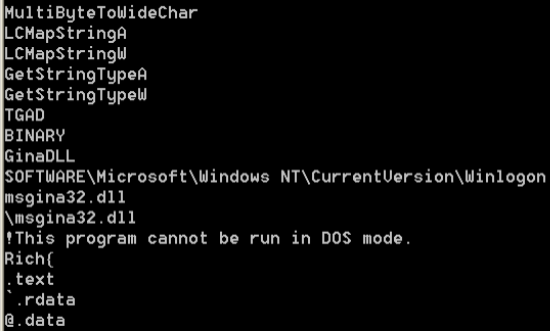
完成课本Lab 11的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

1. **实验过程**

## Lab 11-1

**分析：**

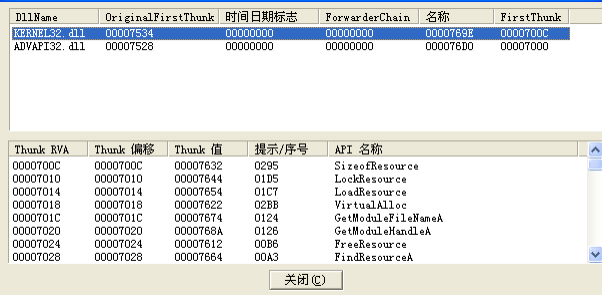
查看字符串：



对于Software\Microsoft\Windows NT\CurrentUersion\Winlogon

和GinaDLL ，猜测这个样本会对图形化界面和windows的登录界面有一些行为，有可能就是拦截Gina，窃取用户的凭证。

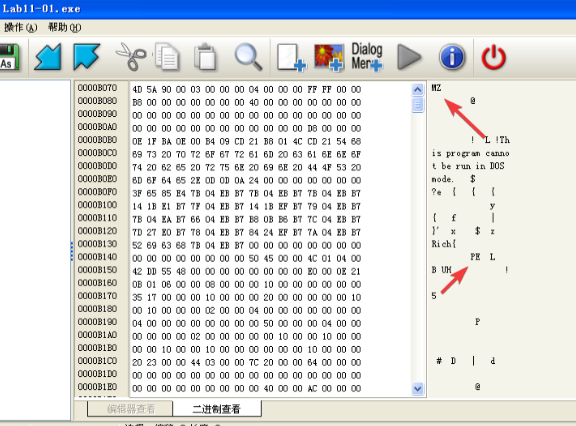
查看导入表：





可见有关于注册表的函数。

使用resource\_hacker工具查看一下样本的资源节 ：

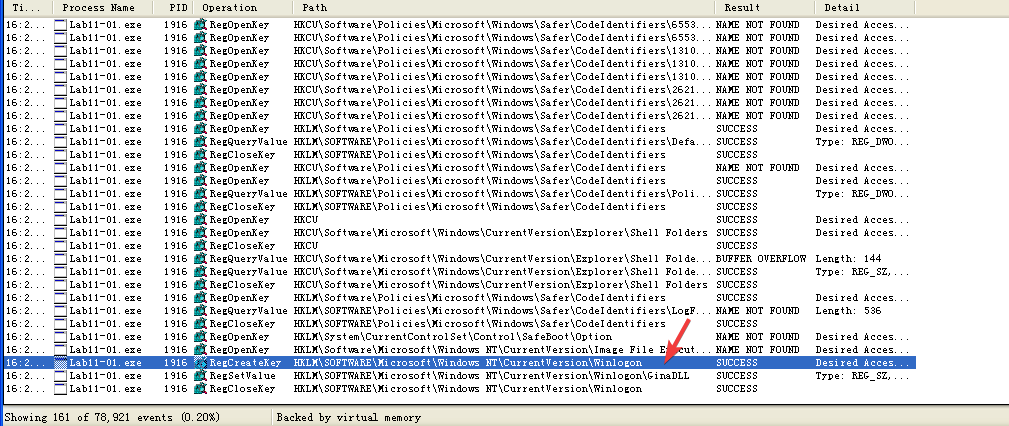


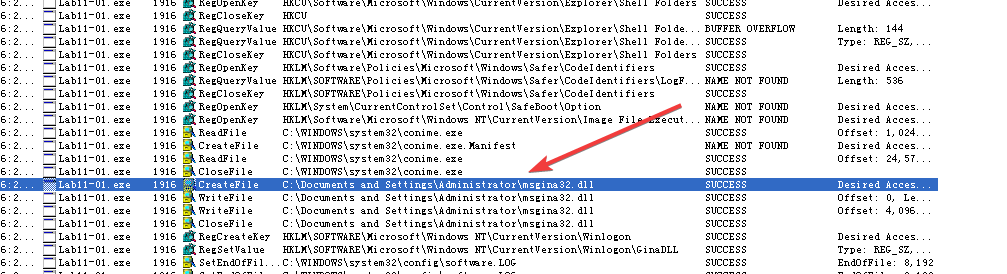
可以看见开头的字节依旧是 4D 5A ，验证了我们之前的猜想：这个资源节里藏有PE格式的文件。

使用PEiD：

 可以看见这个资源是一个dll类型的文件。

使用Procmon查看样本行为 ：



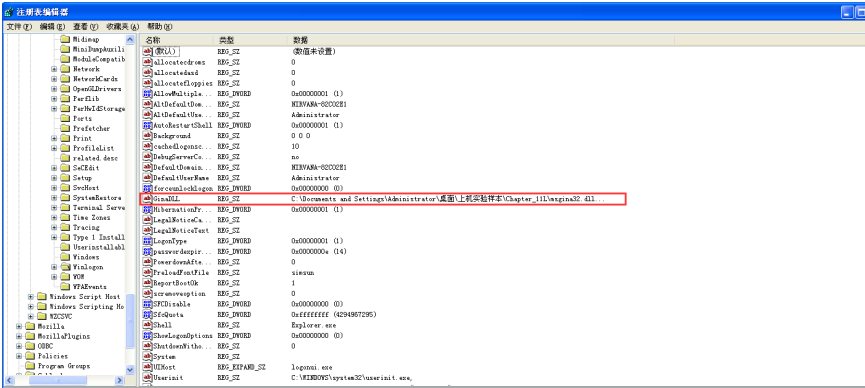


可以看见这个样本在同目录下创建了一个名为msgina32.dll的文件，并对这个文件进行了写文件的操作。

合理猜测这个就是资源节中的文件。

通过计算两个文件的Hash值，可以知道这两个文件是否相同。

查看注册表：

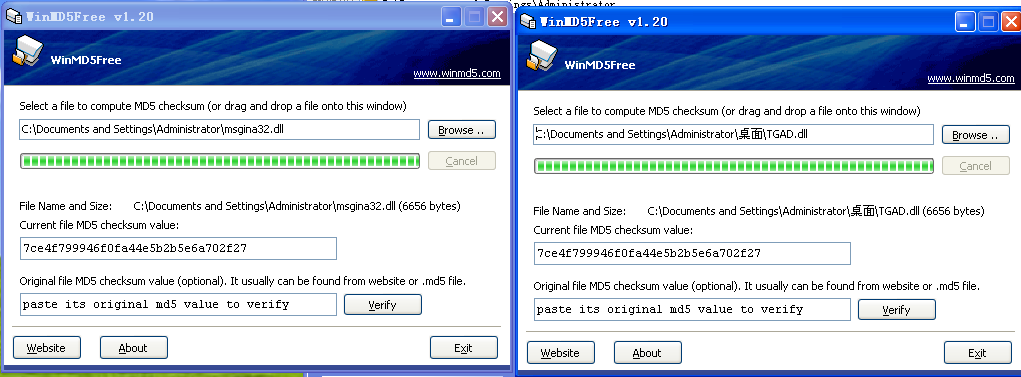


可以看见这里添加了GinaDLL的表项，并且路径设置为了之前释放的dll文件的路径。

接下来我们分析main函数调用下一个主要函数：sub\_401000。可以看到它创建并设置了一个注册表项，与我 们在动态分析时看到的一样，设置了GINA注册表的值。

### 这个恶意代码向磁盘释放了什么？

向磁盘释放了一个msgina32.dll。使用WinMD5发现这个dll和我们在资源中提取出来的文件是同一个文件。



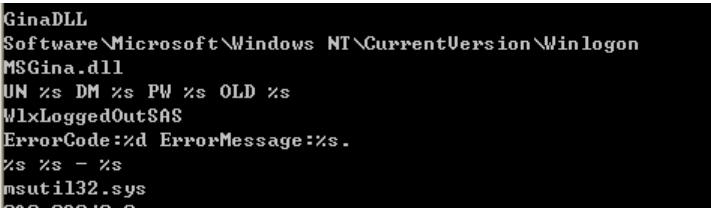
### 这个代码如何进行驻留？

代码将自己添加到注册表中，使得系统被重启后，msgina32.dll会被重新加载。

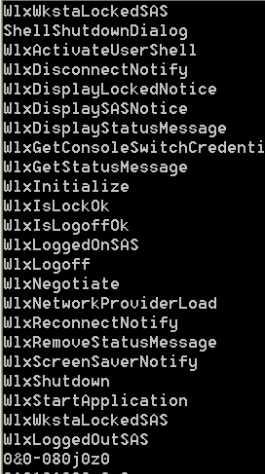
### 这个恶意代码如何窃取用户登陆凭证？

对msgina32.dll进行分析。

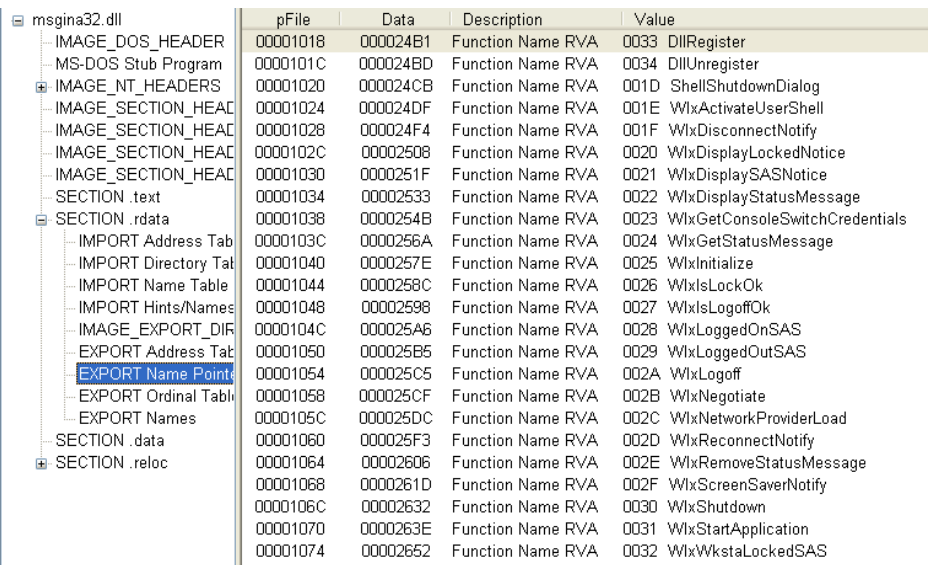
字符串如下：



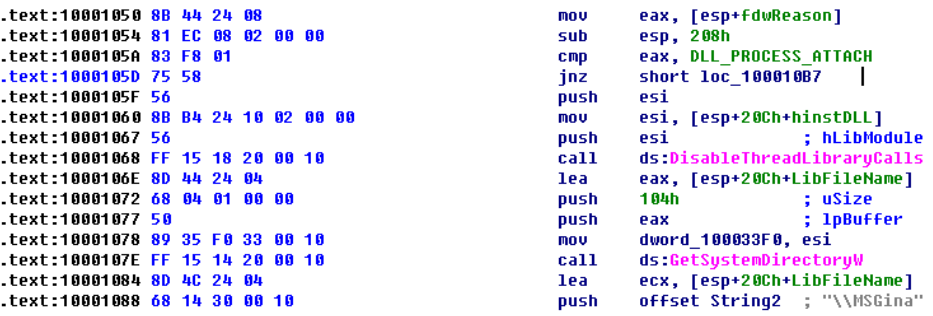
推测这是一个用来拦截GINA的恶意代码，则这些字符串似乎是用来记录用户登录凭证的日志信息。



在导出函数表中有很多以Wlx为前缀的函数，而GINA需要以wlx开头的函数，所以拦截GINA的恶意代码也必须包含这些函数。

IDA分析：

首先，DllMain首先检查参数fdwReason是否为DLL PROCESS ATTACH，即检查该DLL是否是在进程启动或者使 用 LoadLibrary函数时被加载的。如果是的话才会执行后面的代码。



首先调用DisableThreadLibraryCalls禁用DLL\_THREAD\_ATTACHE和DLL\_THREAD\_DETACH通知，减少工作集大小。

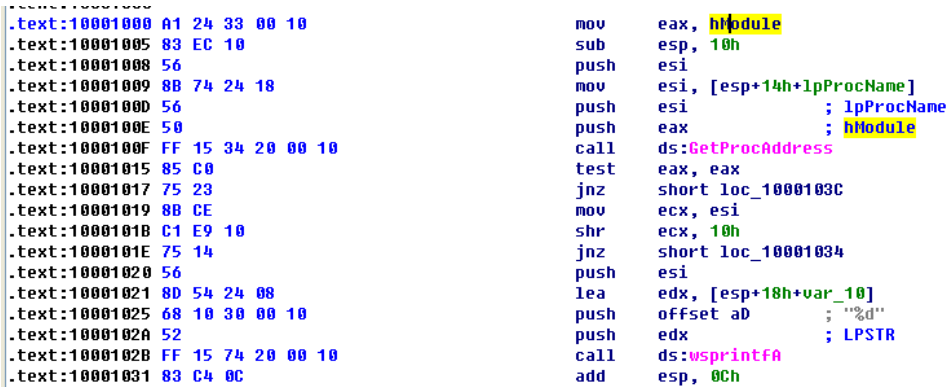
然后调用GetSystemDirectoryW检查系统目录的路径，主要是为了兼容。

之后调用lstracatW拼接字符 串，其参数是ecx和\\MSGina，ecx中是GetSystemDirectoryW的返回值，生成了一个路径。

之后调用LoadLibraryW获取msgina.dll的句柄，之后将打开的句柄存在hLibModle中，这是个全局变量。

每一个导出函数的格式都是先将一个值为导出函数名的参数压栈，然后调用函数sub\_10001000。

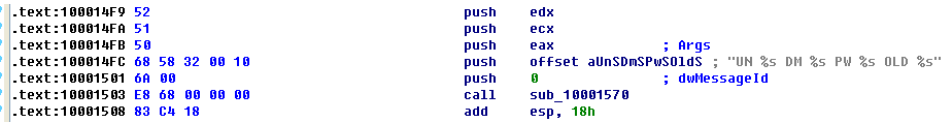
这个函数调用GetProcAddress，参数是全局变量hLibModule和lpProcName（这是调用sub\_10001000时传入 的参数），因此会在msgina.dll中检索当前导出函数。如果成功找到该导出函数的地址才会继续执行下面的代码。



在调用完sub\_10001000之后就会跳转到eax，而sub\_10001000在调用完GetProcAddress后并没有再修改eax的值，eax中存储的就是这个导出函数的真实地址，也就是这里只是劫持了该函数。

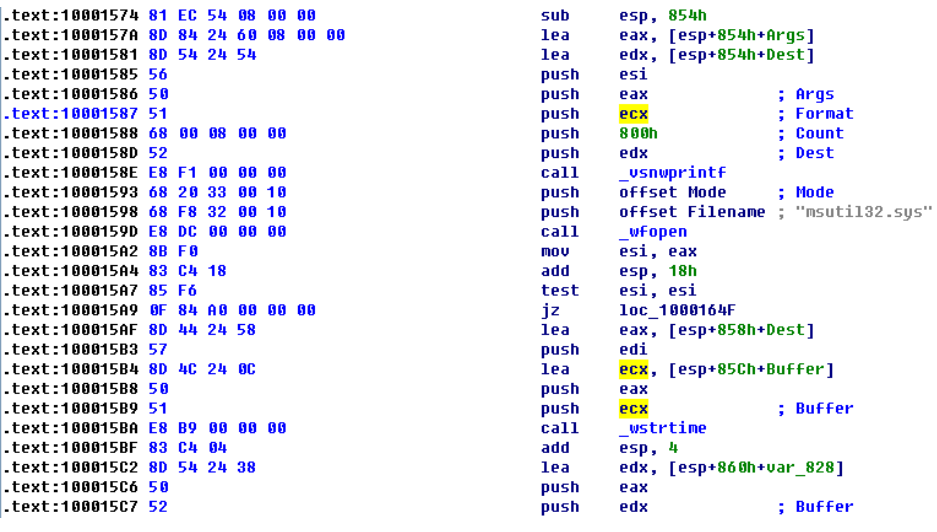
在执行完恶意程序中的代码sub\_10001000之后就又会跳转到真实的WlxLoggedOnSAS函数中去执行。有一个例外是WlxLoggedOutSAS，它包含一些额外的代码。

之后调用了sub\_10001570，传入一堆参数和一个格式字符串。



进入这个函数查看，发现它在记录登录凭证。首先调用了\_vsnwprintf填充之前传入的格式化字符串。

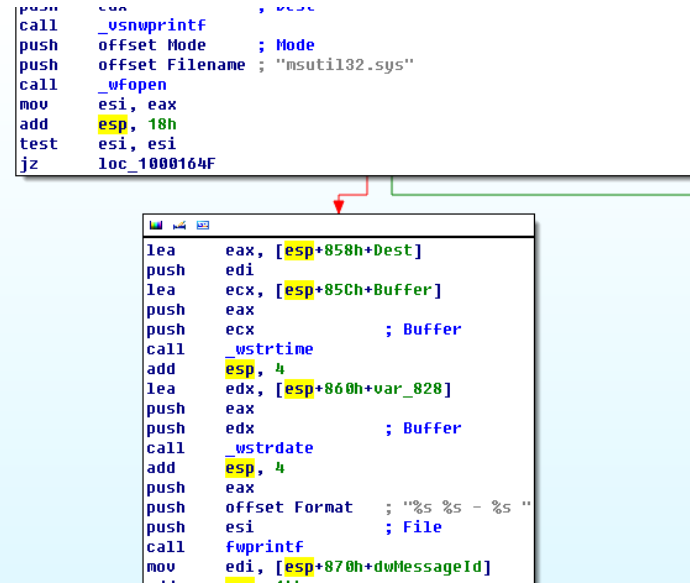
接下 来，调用\_wfopen打开msutil32.sys。调用\_wstrtime和\_wstrdate记录当前的时间和日期。调用fwprintf 记录登录凭证信息。因此，msutil32.sys并不是一个驱动，而是被用来存储记录的日志文件。



总之，恶意代码使用GINA机制来拦截用户的登录凭证，msgina32.dll会拦截所有提交到系统认证的用户登录凭证。

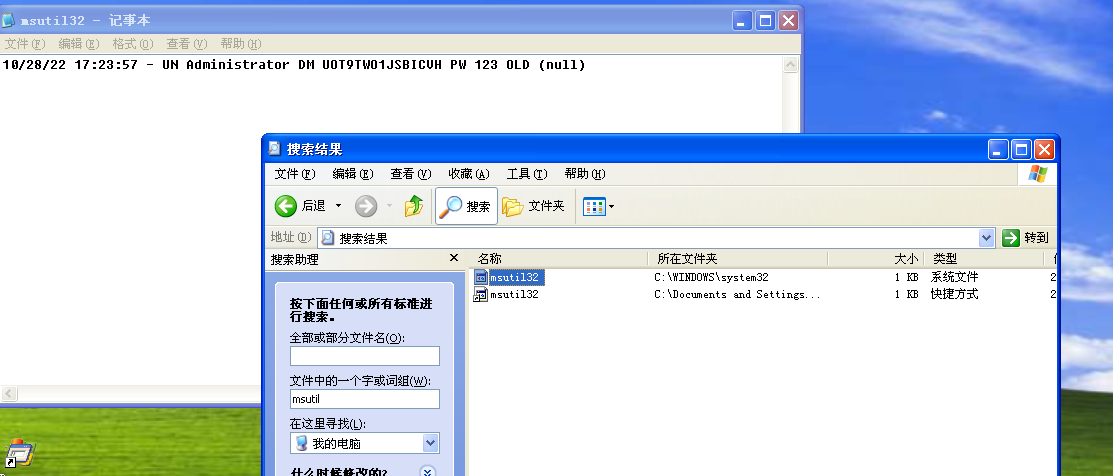
### 这个恶意代码对窃取的证书做了什么处理？

这个恶意代码会将凭证保存在C:\WINDOWS\system32\msutil32.dll中。



### 如何在你的测试环境让这个恶意代码获得用户登录凭证？

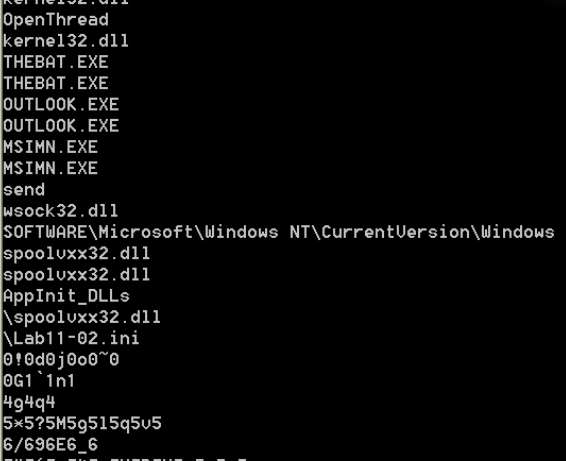
运行Lab11-01.exe后重启，就可以在msutil32.sys中发现我们的用户和密码。



## Lab 11-2

**分析：**

查看字符串：



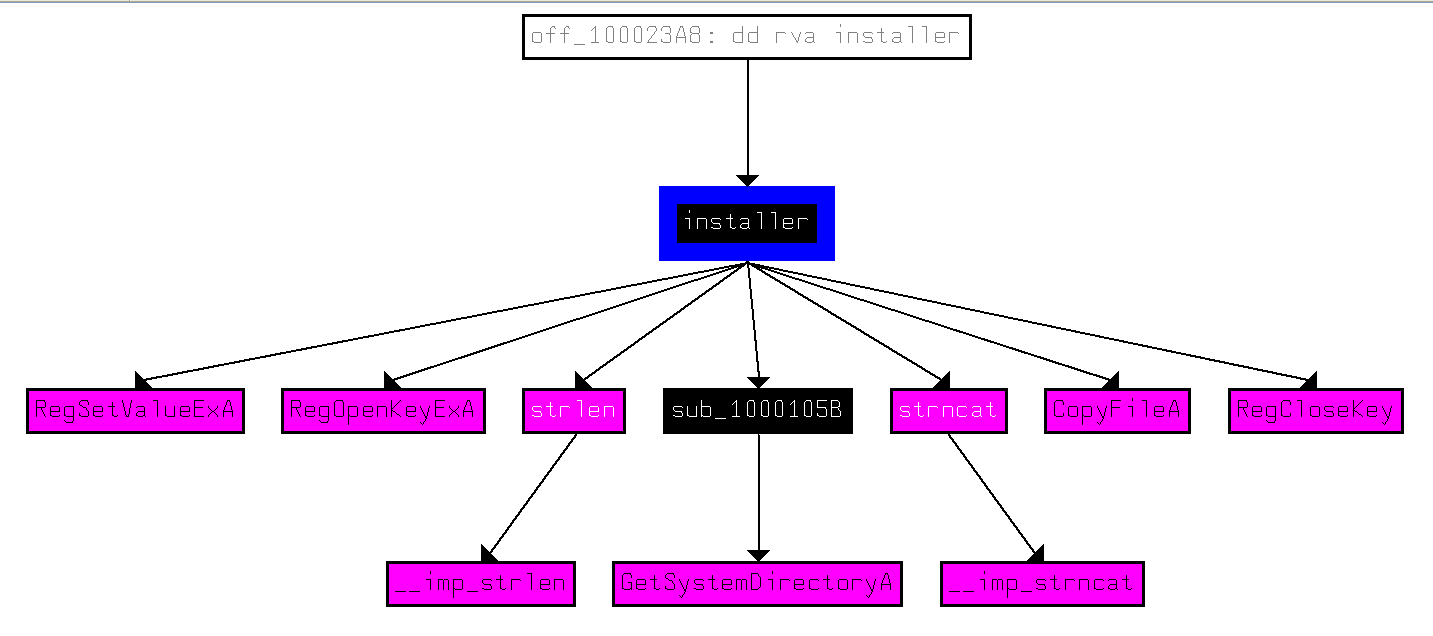
存在注册表的位置和AppInit\_DLLs，表明程序可能通过注册表来达到永久驻留自身的功能。

出现了Lab11-02.ini则表明程序可能使用了这个文件。

send和wsock32.dll，表明程序可能有联网的操作。

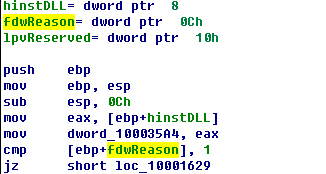
还有OUTLOOK.EXE这种字符串，可能是对邮件进行了操作。

IDA分析：

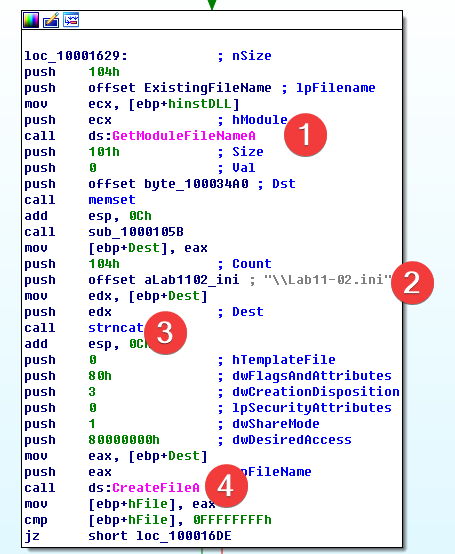


installer注册了一个键值，将文件复制到其他目录。

首先检查DLL\_PROCESS\_ATTACH。



接下来，DLLMAIN试图打开Lab11-02.ini。



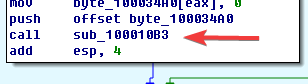
①处程序试图提取Windows系统目录.

②处的字符串Lab11-02.ini。

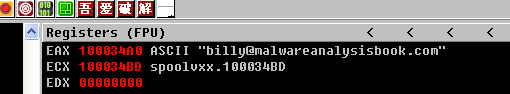
③处程序使用strncat将它们组成一个路径。

恶意代码在④处打开一个用来读取的INI文件，如果不能，则退出。

恶意代码如果成功打开了INI文件，就会生成一个缓冲区，最后将内容传入到一个解密函数中去。



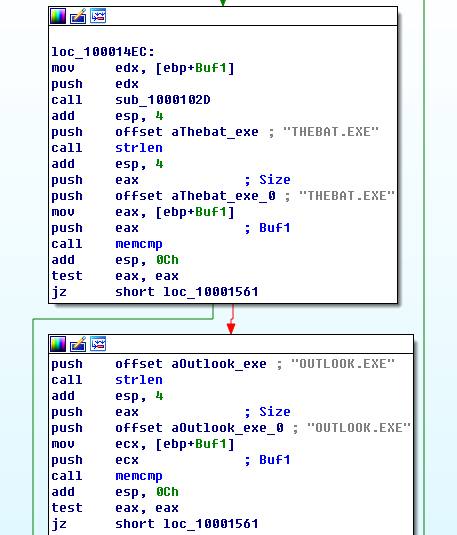
使用OllyDbg来查看解密的内容:



是一个邮箱，有可能是我们的邮件会给这个邮箱发一份。

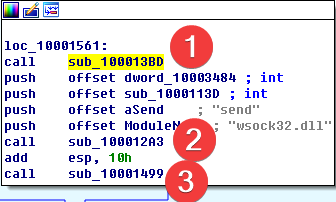
分析挂钩函数:

hook\_installer函数安装挂钩之前，会检查恶意代码正在哪个进程中进行。它调用了三个函数来获取当前的进程名。



接下来恶意代码检查当前进程大写字母的进程名是否与这四个相同，不同的话就会退出。

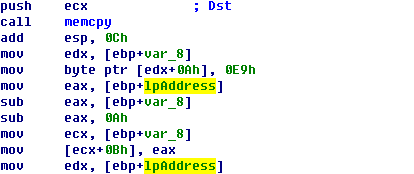
接下来，分析安装inline挂钩的函数。



在①中我们看到它首先调用GetCurrentProcessId函数，然后是sub\_100012FE函数，我们将这个函数命名为suspend\_threads。suspend\_threads函数调用了GetCurrentThreadId，它返回当前运行线程的线程标识符（TID）。接下来，suspend\_threads调用CreateToolhelp32Snapshot，并且使用结果循环遍历当前的所有TID，如果TID不是当前的线程，则用TID调用SuspendThread。我们可以得出结论，在①处的函数调用挂起了进程的所有运行线程。

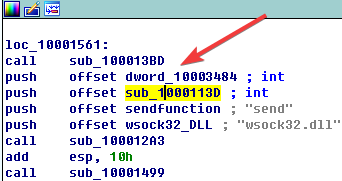
③处函数调用做的工作则是恢复所有线程，使用ResumeThread函数，两个函数的意图是，先挂起，然后恢复执行。恶意代码执行一些影响改变当前进程运行行为的动作比较常见，例如修改内核或者是安装一个inline挂钩。

接下来，检查②处调用的代码。将该函数的参数修改成合适的名字。



可以发现此过程是在修改地址，首先将0E9赋值到edx中，接着用参数覆盖ecx寄存器中的值，这一过程就是实现jmp执行，因为jmp指令的机器码就是E9。

此过程执行完毕以后就会跳转到sub\_1000113D函数执行位置：



因为此函数的地址是作为参数传递进去的。现在将此函数重命名为：hook\_function现在我们返回到钩子函数执行过程函数：



VirtualProtect函数可以修改进程的运行，保护其读和写的权限，下面又调用了此函数的目的是恢复原始进程权限。在上图中我们发现其使用malloc函数分配的新的空间并将其保存在var\_8这个参数里面，目的是为了以后在构造E9（jmp指令）时所需的空间。

### 这个恶意DLL导出了什么？

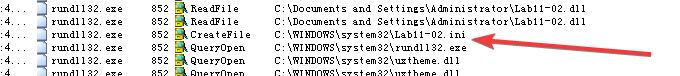
一个名为installer的具有安装恶意代码自身功能的导出函数。

### 使用rundll32.exe安装这个恶意代码后，发生了什么？

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。

### 为了使这个恶意代码正确安装，Lab11-02.ini必须放置在何处？

必须放在路径C:\WINDOWS\system32\下。



### 这个安装的恶意代码如何驻留？

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。

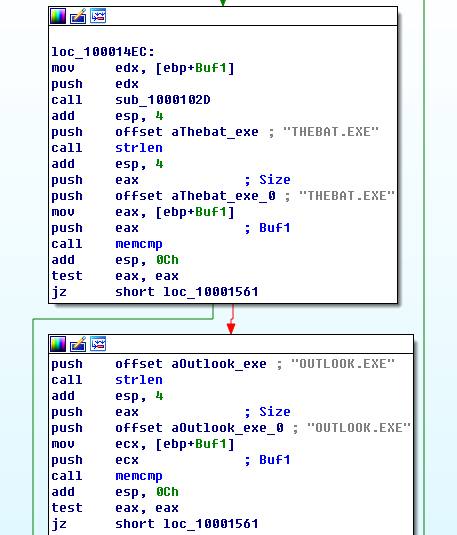
### 这个恶意代码采用的用户态Rootkit技术是什么？

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

### 挂钩代码做了什么？

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。

### 哪个或者哪些进程执行这个恶意攻击，为什么？



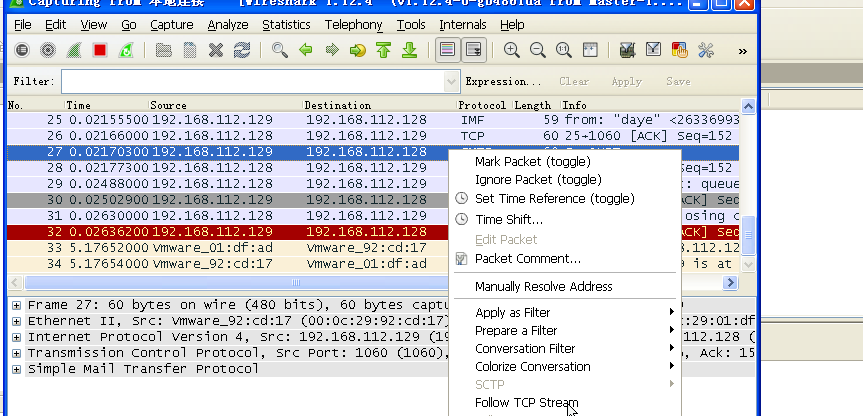
这个恶意代码仅针对的是程序MSIMN.exe、THEBAT.exe或者OUTLOOK.exe，它们都是Windows操作系统中默认的与邮件发送相关的程序。

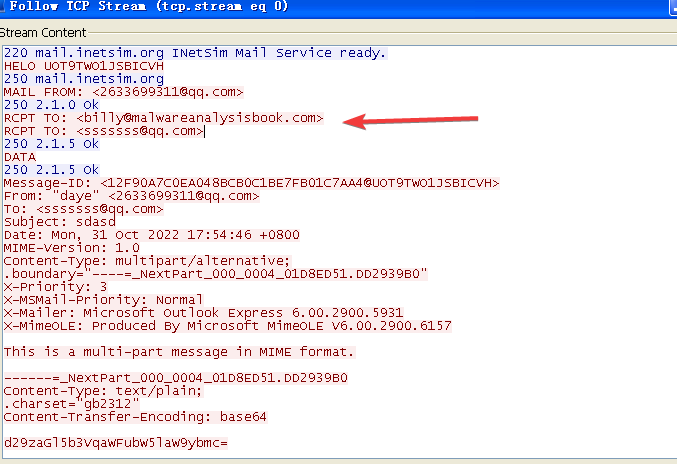
### ini文件的意义是什么？

这个.ini文件中存储了一个加密后的恶意邮箱地址，解密之后发现就是一个邮箱 billy@malwareanalysisbook.com。

### 你怎么用wireshark动态抓获这个恶意代码的行为？

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。





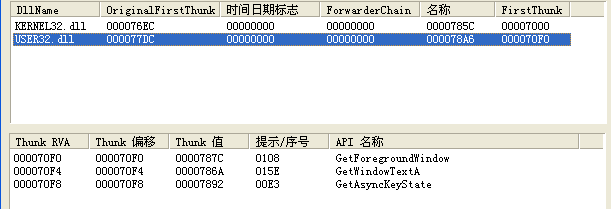
## Lab 11-3

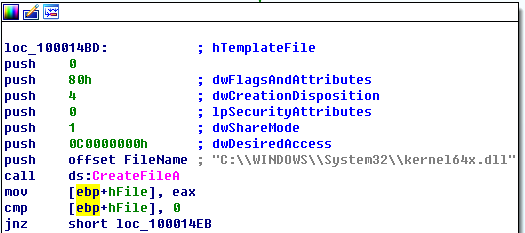
### 使用基础的静态分析过程，你可以发现上面有趣的线索？



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，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。

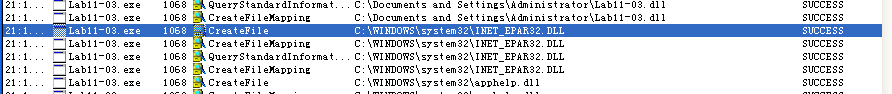


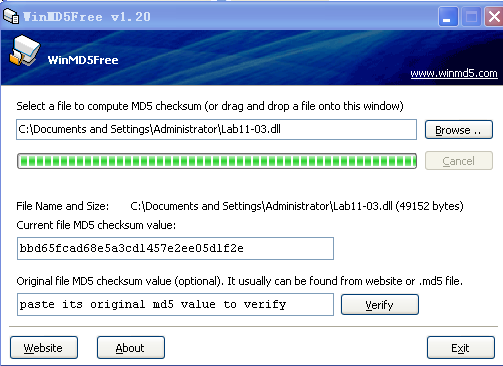
这里的GetForegroundWindow；GetiindowTextA；GetAsyncKeyState三个导入函数。初步推断是要做键盘记录。  
 

怀疑它是一个记录到文件kernel64x.dll的击键记录器。

### 当运行这个恶意代码时，发生了什么？

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。

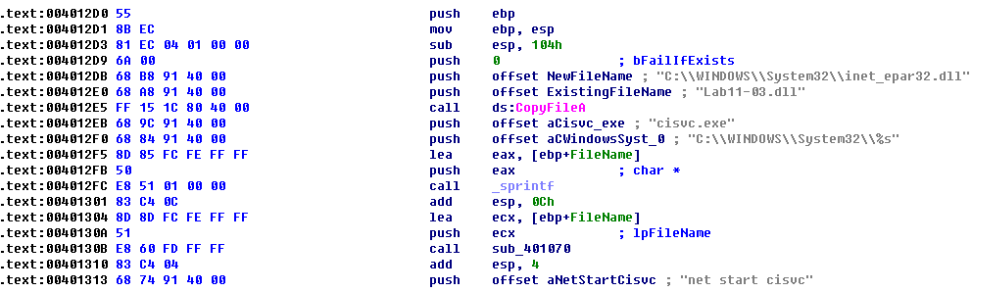




### Lab11-03.exe如何安装Lab11-03.dll使其长期驻留

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。

IDA调试lab11-03.exe:



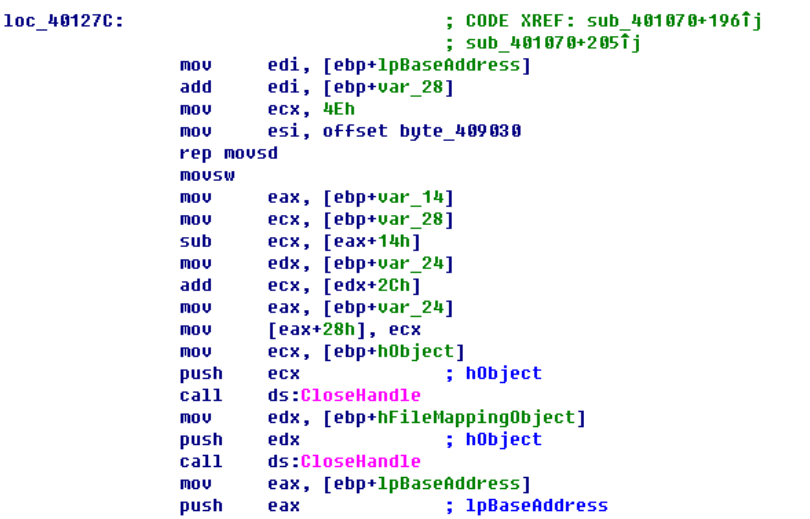
在main函数中一开始是复制lab11-03.dll到system32系统目录中的inet\_epar32.dll。之后创建一个字符c:\windows\system32\cisvc.exe,将其作为参数传递给了sub\_401070.之后是使用系统运行命令net start cisvc，来启动索引服务。

分析函数sub\_401070:

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。

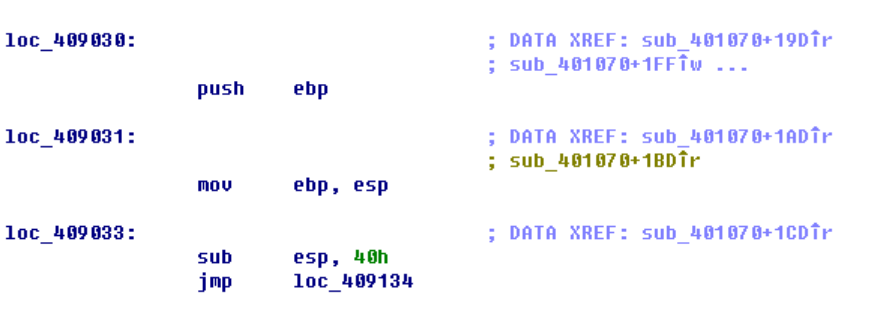
然后提取硬盘上的 cisvc.exe来进行分析。lpBaseAddress中记录着文件的映射位置，它被赋值给edi寄存器，加上var\_28进行 偏移，之后将0x4E赋值给ECX，循环进行写操作movsd，总共写入了0x4E\*4=312字节。最后，byte\_409030被赋值给esi，其中的数据也被到映射到文件中。



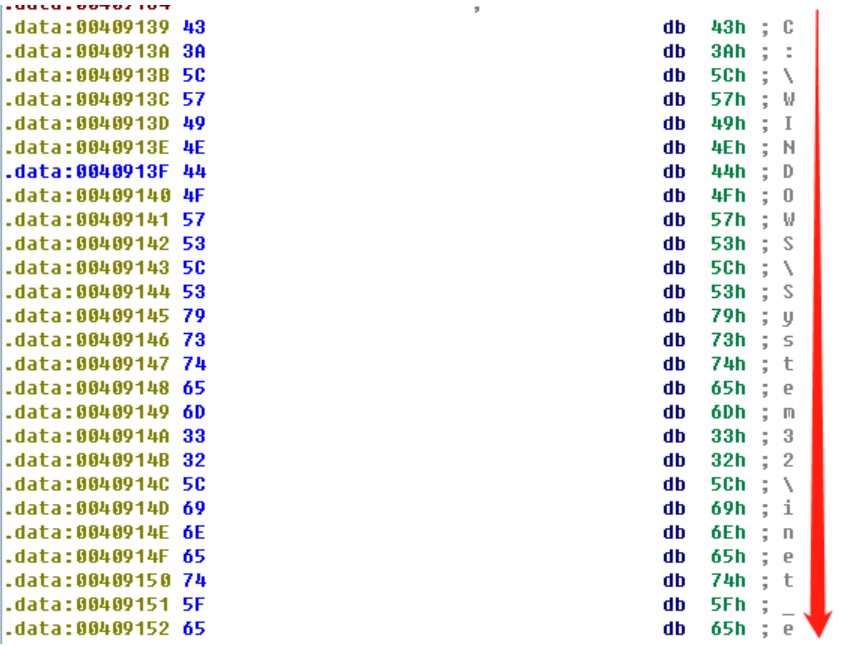
我们查看byte\_409030处的数据，如下：



反汇编结果如下，这就是写入cisvc.exe的shellcode。



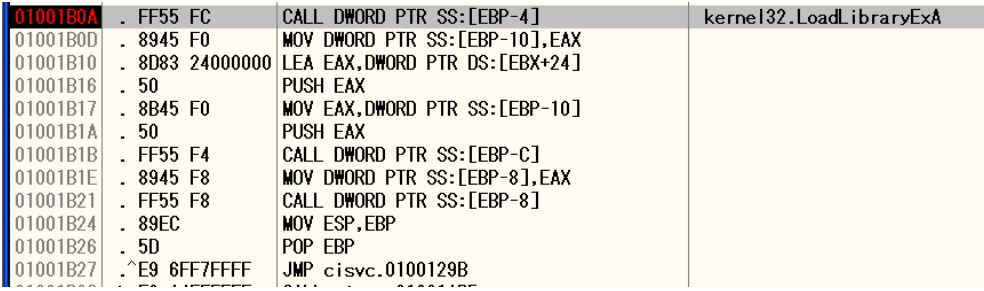
在shellcode的末尾，我们看到字符串C:\WINDOWS\System32\inet\_epar32.dll和zzz69806582，说明这个shellcode加载了这个DLL，并且调用了它的导出函数。



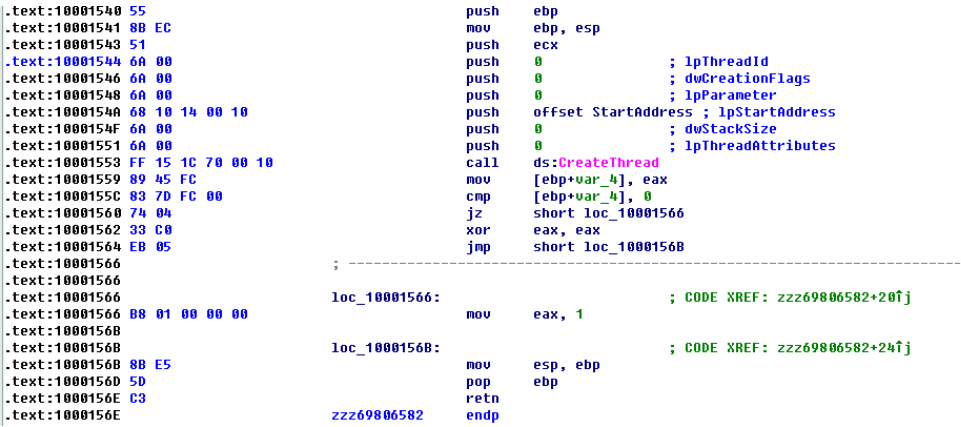
比较被感染前后的cisvc.exe，可以看到程序的入口点发生了变化，并且添加了大量的代码。恶意代码执行了 入口重定向，使得无论什么时候运行cisvc.exe，都将先执行shellcode而不是原始的程序入口点。



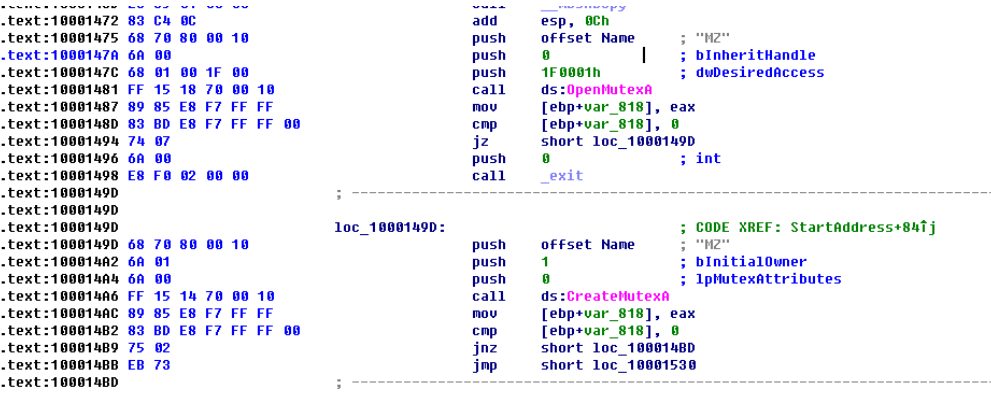
下面使用IDA和OD打开被感染的cisvc.exe进行分析，我们发现恶意代码调用LoadLibrary载入 inet\_epar32.dll，之后调用GetProcAddress获取导出函数zzz69806582的地址，然后根据得到的地址调用 导出函数。最后跳转到程序的原始入口点，使得服务正常执行。

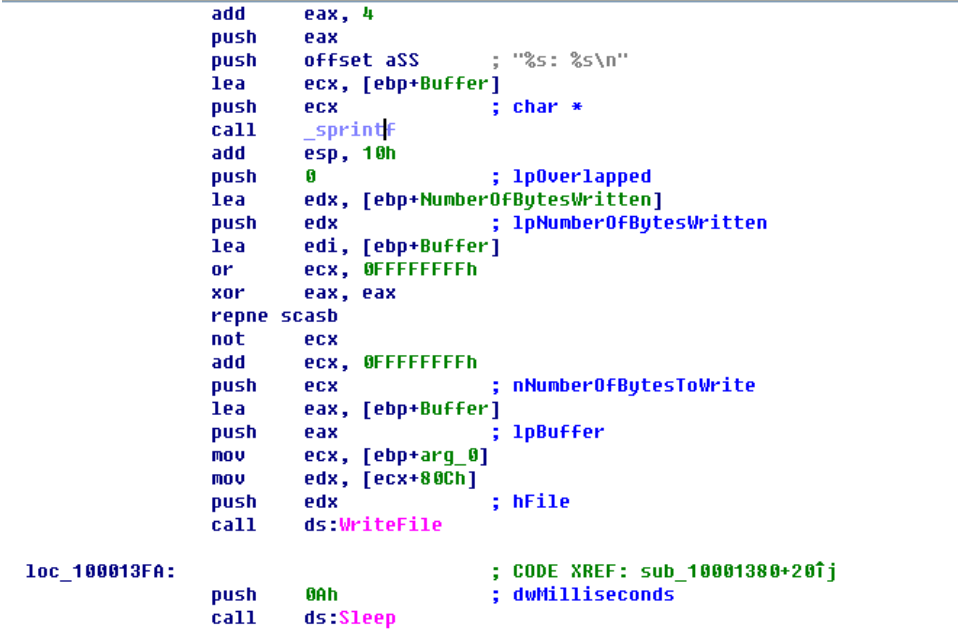


总之，这个shellcode用来加载inet\_epar32.dll，并且调用它的导出函数。接下来我们分析一下inet\_epar32.dll即Lab11-03.dll做了什么。我们用IDA打开Lab11-03.dll开始分析， 发现DLLMain很短，并没有做什么，于是我们分析导出函数。发现它调用CreateThread创建了一个线程然后就 返回了，下面我们去分析这个线程。



首先调用OpenMutex打开一个名为MZ的互斥量，如果成功则退出，否则继续调用CreateMutex创建这个互斥 量，保证只运行一个实例。接下来，调用CreateFile打开或者创建C:\WINDOWS\System32\kernel64x.dll 的文件来写日志。获得该文件的句柄之后，它会将文件指针指向文件的末尾，然后调用函数sub\_10001380,循 环调用GetAsyncKeyState、GetForegroundwindow和WriteFile，进行击键记录。





### 这个恶意代码感染Windows系统的哪个文件?

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。

### Lab11-03.dll做了什么？

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。

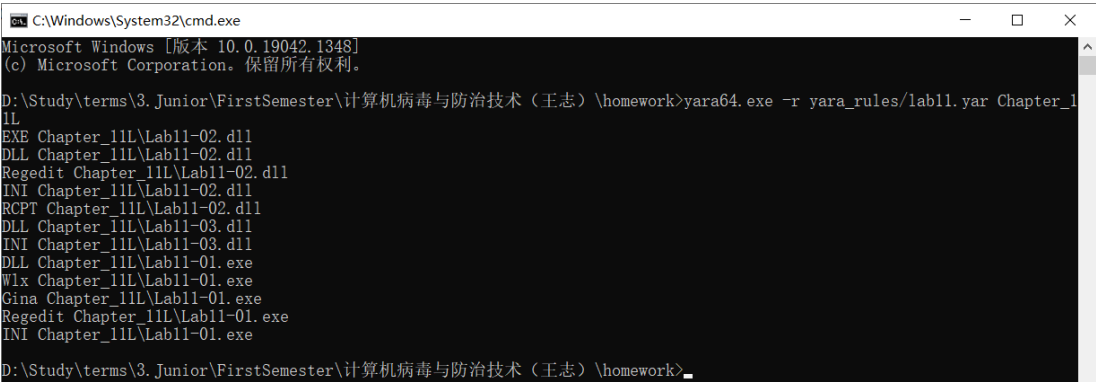
### 这个恶意代码将收集的数据存放在何处？

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

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  
}

扫描结果：



## IDA Python

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,[]))) |

1. **实验结论及心得体会**

复习到课上的知识点。

熟练使用恶意代码分析工具。