**恶意代码分析与防治技术实验报告**

**Lab11**

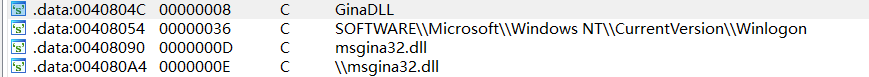
**学号：2013018 姓名：许健 专业：信息安全**

1. **实验内容**

完成课本Lab11的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

1. **问题解答**
2. Lab11-1

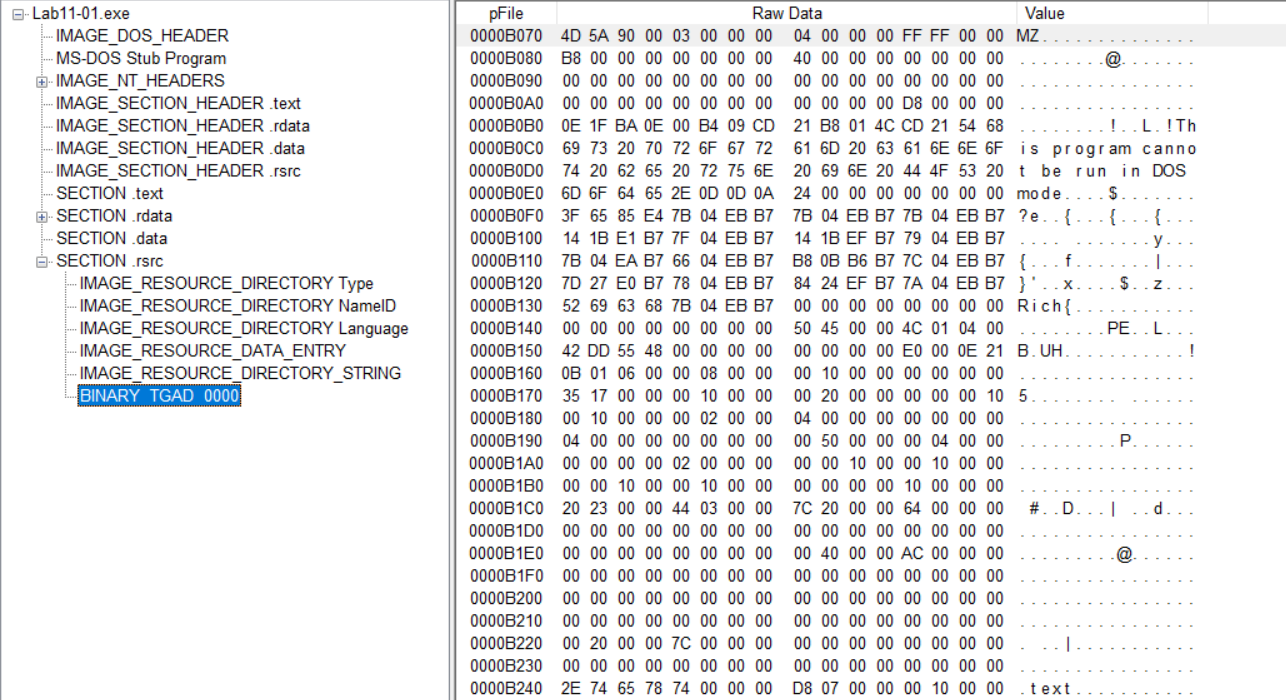
**基础静态分析**



猜测恶意代码要拦截GINA

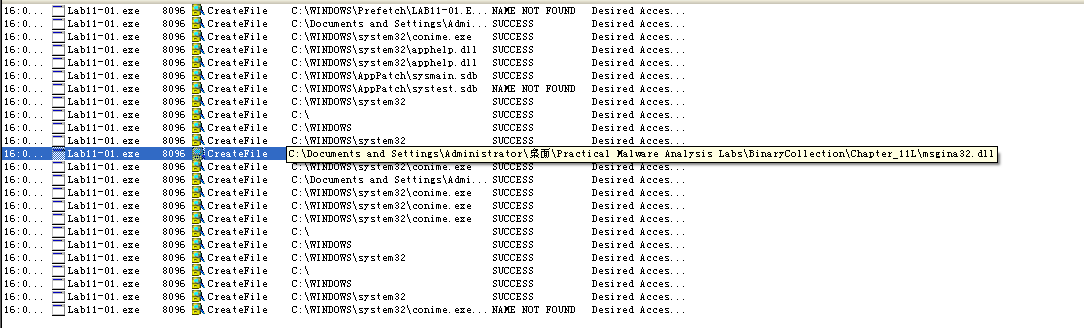


操纵注册表和提取资源节的导入函数



名为TGAD的资源节包含一个内嵌的PE文件

**基础动态分析**

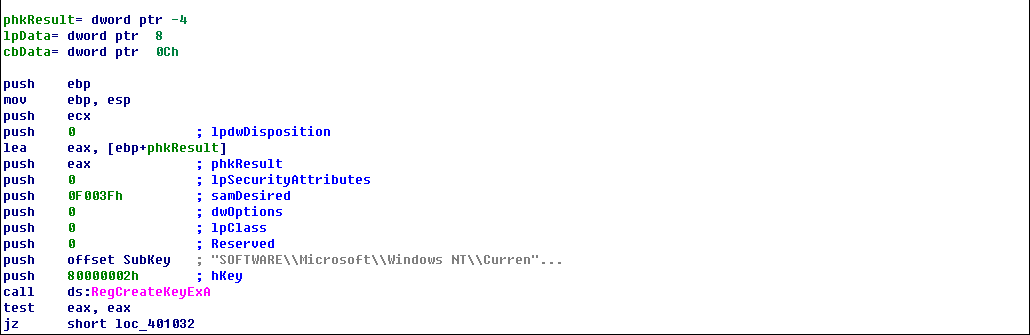


使用procmon监视恶意代码，发现创建文件msgina32.dll在硬盘的启动目录，恶意代码插入msgina32.dll的路径到注册表，系统重启时WinLogon会加载这个DLL

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL

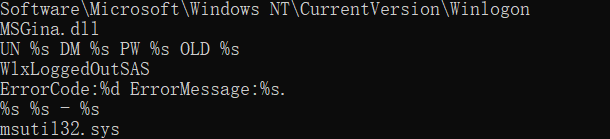
经过比较，msgina32.dll中的内容与TGAD资源节相同

**高级静态分析**

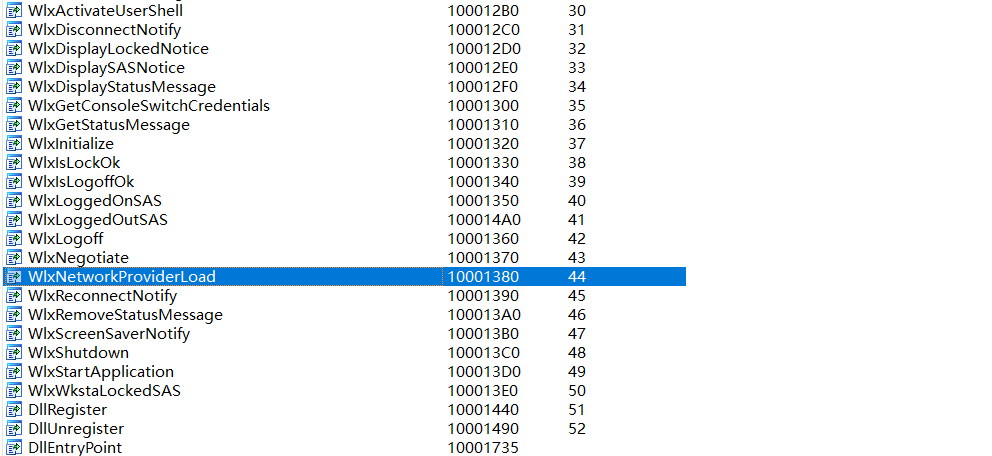


main函数调用了sub\_401080(提取TGAD资源节到msgina32.dll)和sub\_401000(设置GINA注册表的值)，Lab11-01.exe是针对msgina32.dll的安装器，它在系统启动期间由Winlogon加载。

**msgina32.dll分析**



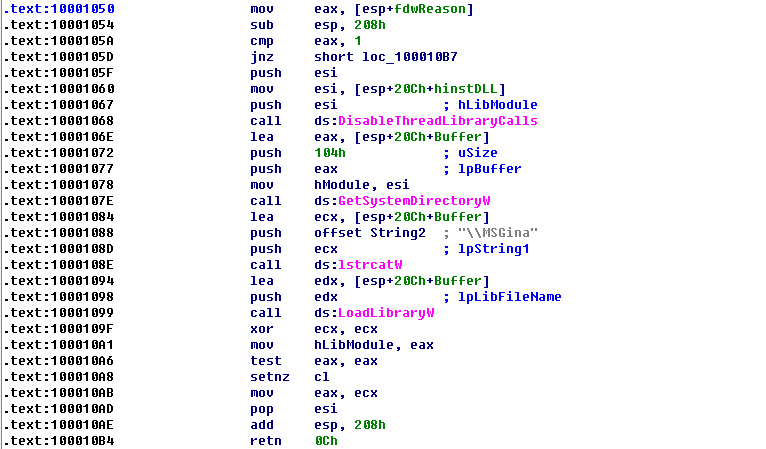
记录用户登录凭证的日志信息



检查msgina32.dll的导出函数表，包含很多以前缀Wlx开始的函数，GINA需要Wlx开始的函数，因此GINA拦截的恶意代码必须包含这些函数。

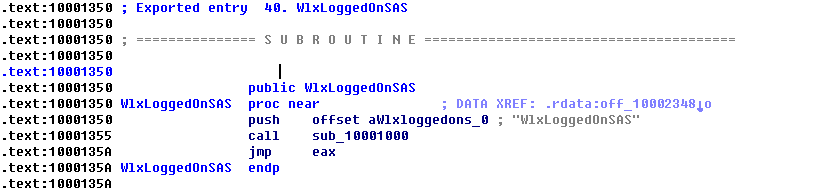
**高级静态分析**

分析msgina32.dll函数的DLLmain，它会检查参数fdwReason，该参数表示DLL入口函数被调用的原因。恶意代码检查传入参数是否是DLL\_PROCESS\_ATTACH，在进程启动或者使用LoadLibrary函数加载DLL时，带有这一参数的入口函数将会被调用。如果这个特殊的DLL在DLL\_PROCESS\_ATTACH期间被调用，那么恶意代码将通过调用LoadLibraryW函数获取Windows系统目录中msgina.dll的句柄。这里被命名为hLibModule。



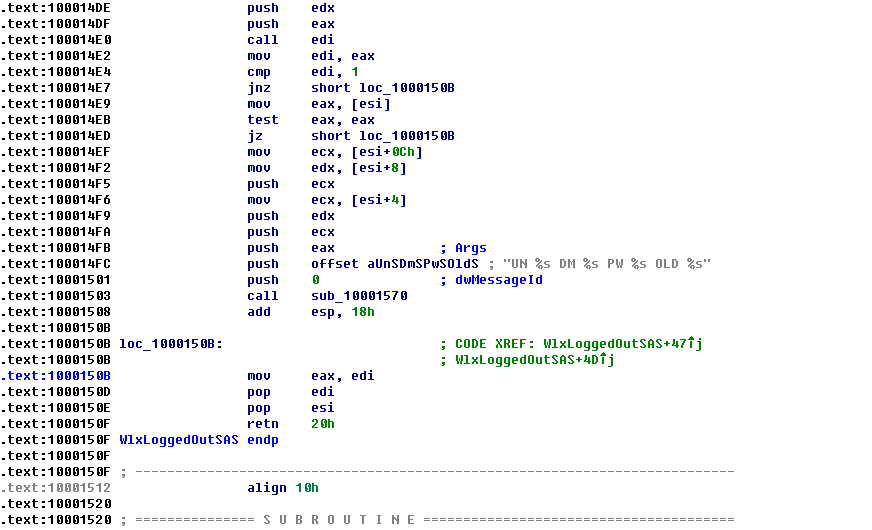


使用hLiibModule可以让这个DLL的导入函数恰当地调用Windows DLL msgina.dll中的函数。既然msgina32.dll拦截Winlogon与msgina.dll之间的通信，则它必须恰当地调用msgina.dll中的函数，从而使系统操作继续正常运行。

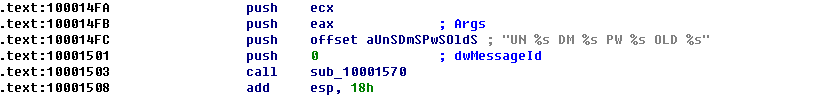


分析导出函数WlxLoggedOnSAS，sub\_10001000只是简单地传递给msgina.dll中真正的WlxLoggedOnSAS函数地址，然后跳转到此处执行。

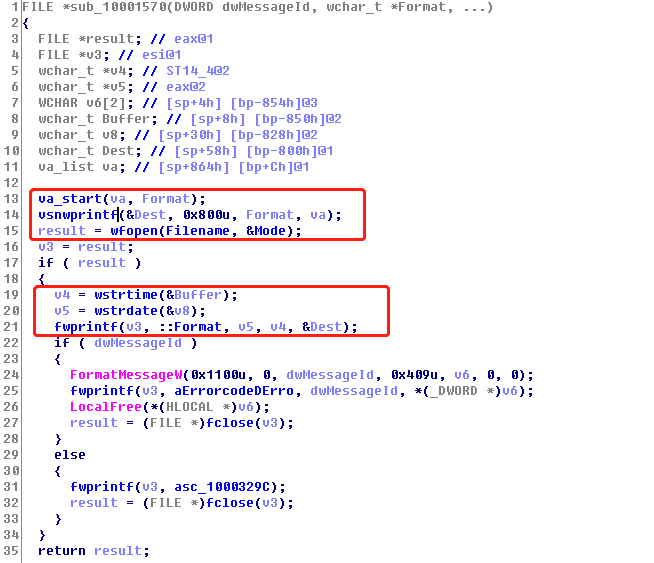




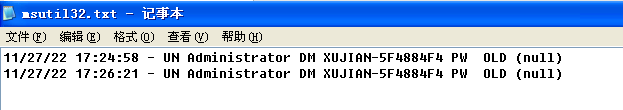
WlxLoggedOutSAS函数包含一些额外的代码（当系统注销时调用WlxLoggedOutSAS）



sub\_10001570是一个登录凭证记录函数，恶意代码打开C:\Windows\System32\msutil32.sys，记录日期、时间、登陆凭证信息等，因此msutil32.sys是一个用来存储记录的日志文件。



重启机器，登陆系统并注销系统，可以看到msutil32.dll中记录用户登录凭证



1. 这个恶意代码向磁盘释放了什么？

这个恶意代码会从名叫TGAD的资源节提取一个文件并命名为msgina32.dll，释放到磁盘

1. 这个恶意代码如何进行驻留？

这个恶意代码会在注册表中添加一个键值来安装这个DLL，系统重启之后，依旧会加载这个DLL

1. 这个恶意代码如何窃取用户登录凭证？

恶意代码用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

1. 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

1. 如何在你的测试环境让这个恶意代码获得用户登录凭证？

重启系统

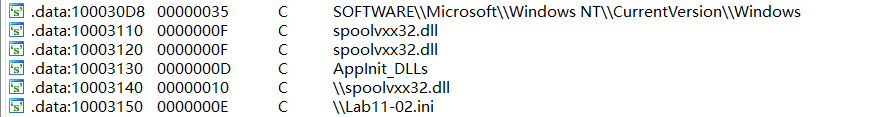
1. Lab11-2

**对Lab11-02.dll做基础的静态分析**

包含一个导出函数installer，导入函数包括操作注册表、修改文件系统、搜索一个进程或者线程列表(CreateToolhelp32Snapshot)



恶意代码使用AppInit\_DLLs来永久安装自身，\\Lab11-02.ini表明恶意代码使用本实验提供的INI文件



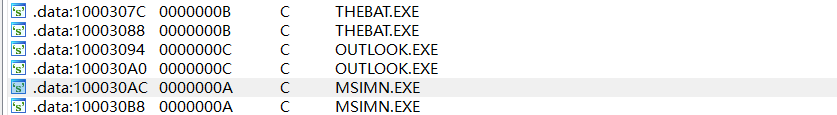
检查INI文件的内容，发现它被编码或加密

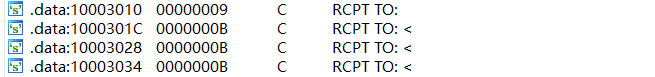


恶意代码可能使用了网络功能，目前我们不清楚执行了什么行为



进程名(THEBAT.EXE、OUTLOOK.EXE、MSIMN.EXE)是邮件客户端，猜测恶意代码对邮件做了某些处理。RCPT是一个SMTP命令，用来创建一个电子邮件的收件人。





**基本动态分析**

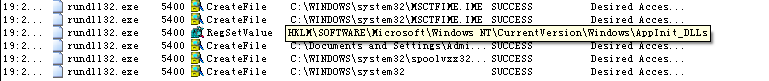
使用installer安装恶意代码



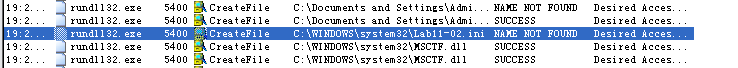
procmon设置过滤器，可以看到恶意代码在windows系统目录下创建了一个名为spoolvxx32.dll的文件，该文件与Lab11-02.dll相同。



恶意代码将spoolvxx32.dll添加到AppInit\_DLLs列表中，导致恶意代码被加载到所有装载User32.dll的进程中。

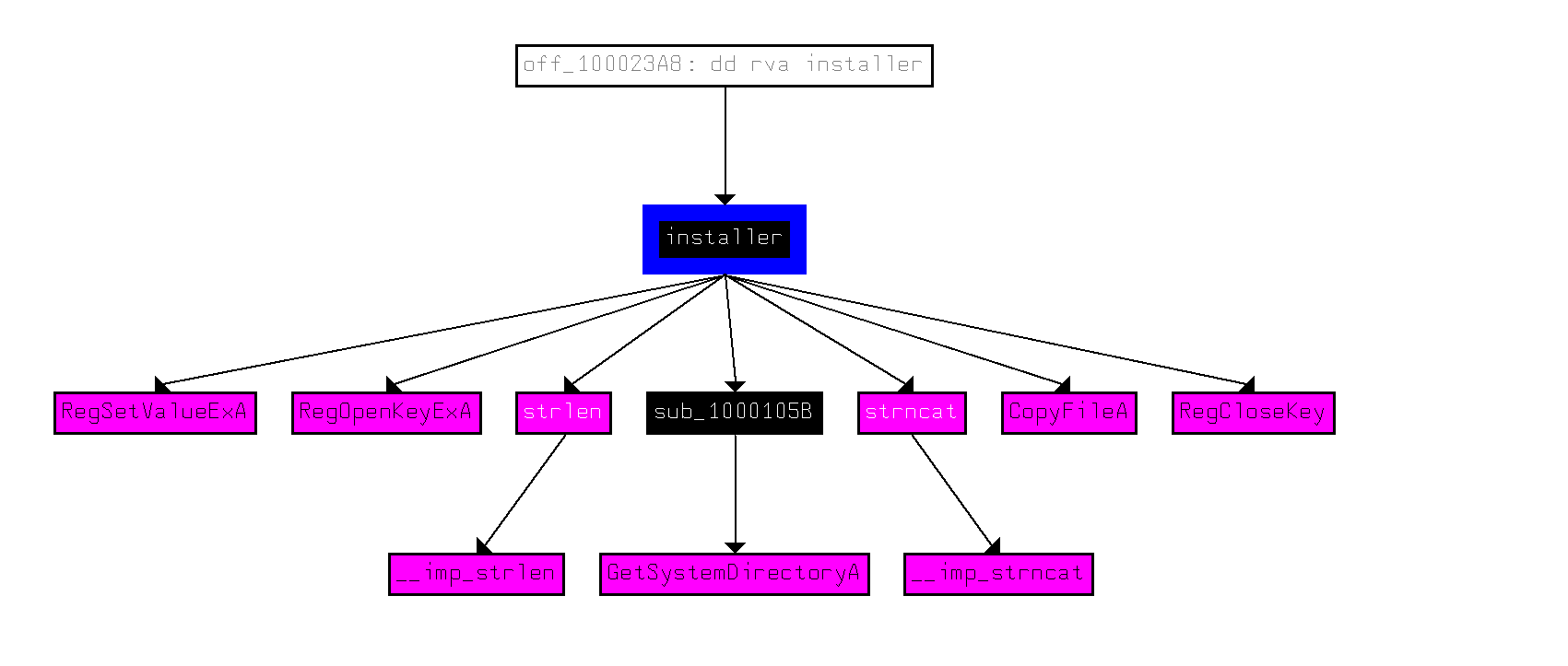


恶意代码从windows系统目录中试图打开Lab11-02.ini文件，为了恶意代码可以访问该文件，我们应该将其放入C:\Windows\System32目录下。



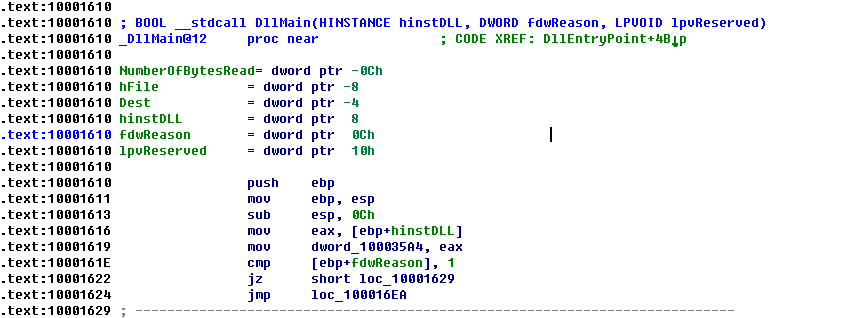
**使用IDA Pro高级静态分析**

查看installer导出函数的交叉引用图，installer在注册表中设置了一个值，并且将一个文件复制到Windows的系统目录，目的就是将恶意代码复制到spoolvxx32.dll，并将其设置为一个APPInit\_DLLs值。

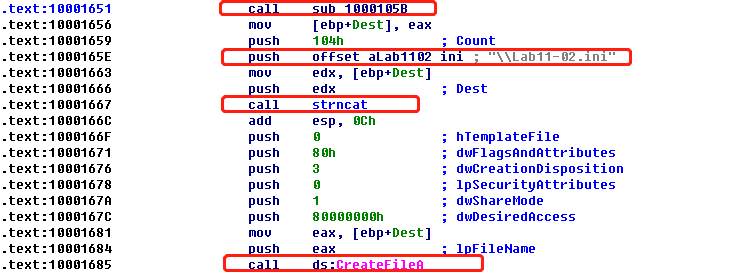


查看DLLmain函数，它首先检查DLL\_PROCESS\_ATTACH

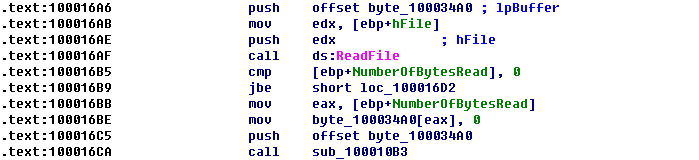
恶意代码仅在DLL\_PROCESS\_ATTACH状态下运行，如果不是则直接返回



恶意代码试图打开C:\Windows\System32\Lab11-02.ini



恶意代码将文件读入到一个全局变量缓冲区中，ReadFile调用后，恶意代码检查确保文件的大小大于0，然后将包含文件内容的缓冲区传递给sub\_100010B3，猜测该程序是解密程序



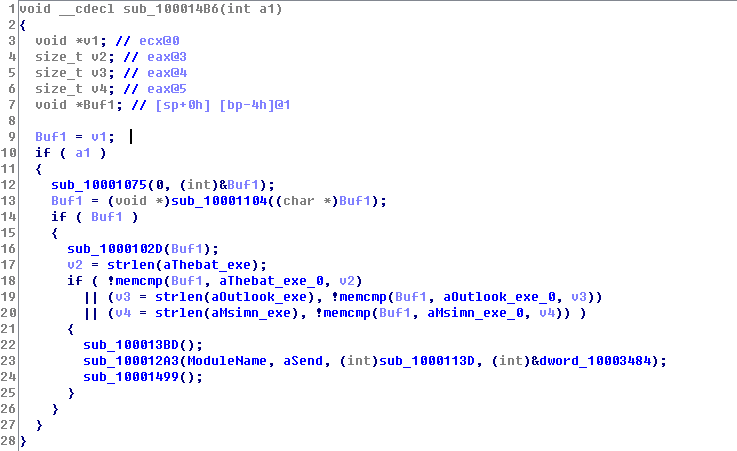
使用ollydbg动态分析，在100016CA处下断点，step over得到如下结果



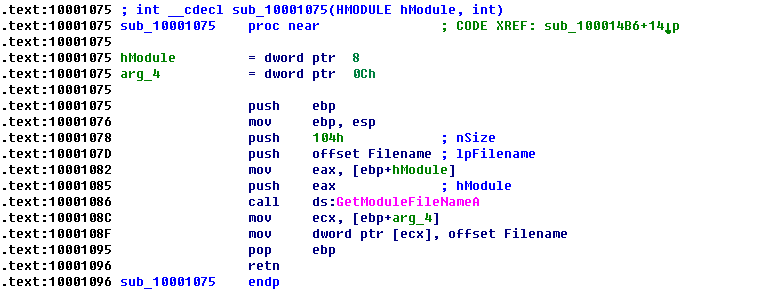
[解密内容为一个邮箱地址](mailto:解密内容为一个邮箱地址billy@malwareanalysisbook.com)billy@malwareanalysisbook.com

该邮箱地址存储在byte\_100034A0处

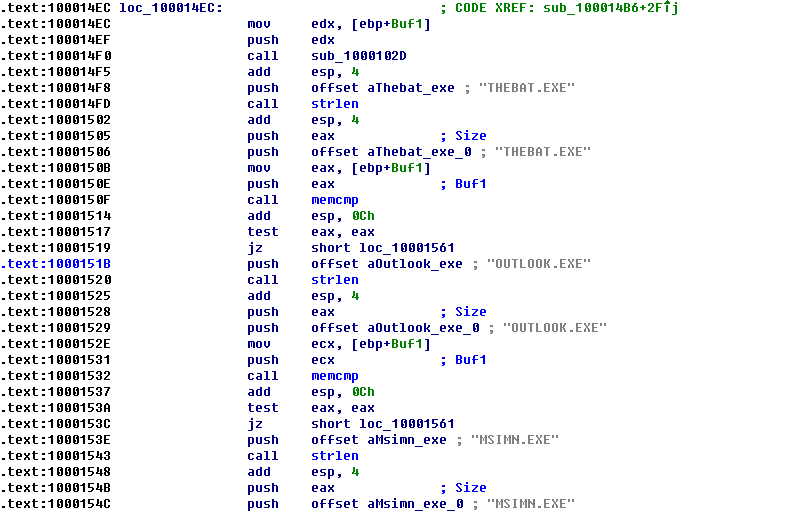
DLLmain中还有函数sub\_100014B6需要分析



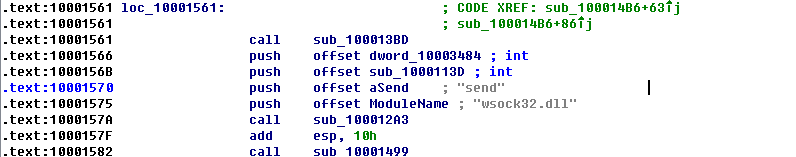
sub\_10001075调用了GetModuleFileNameA函数，参数hModule被设置为0，函数会返回加载这个DLL进程的绝对路劲，接下来恶意代码在arg\_4中返回路径名(字符串指针被传递给函数)，这个字符串被传入至少两个函数，这些函数解析文件名并将文件名中的所有字符转为大写。



接下来，将当前进程大写字母的进程名与THEBAT.EXE、OUTLOOK.EXE、MSIMN.EXE作比较，如果进程名不等于其中的任何一个，恶意代码就会退出



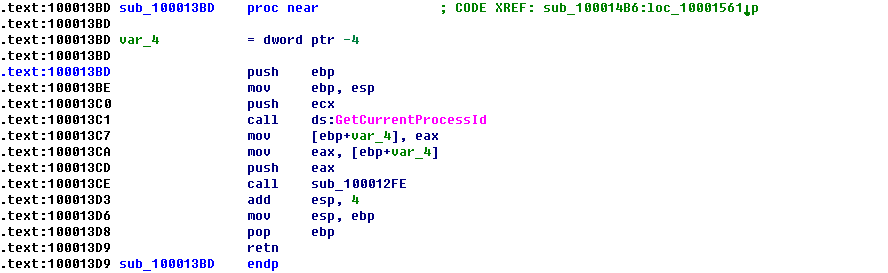
如果恶意代码加载入到这三个进程中的任意一个，就会执行这段代码



函数sub\_100013BD会先调用GetCurrentProcessId函数，然后是sub\_100012FE函数，该函数调用了GetCurrentThreadId，它返回当前运行线程的线程标识符(TID)；

然后调用CreateToolhelp32Snapshot，并且使用结果循环遍历当前进程的所有TID，如果TID不是当前的线程，则用TID调用SuspendThread。

因此函数sub\_100013BD的目的是挂起当前进程的所有运行线程。

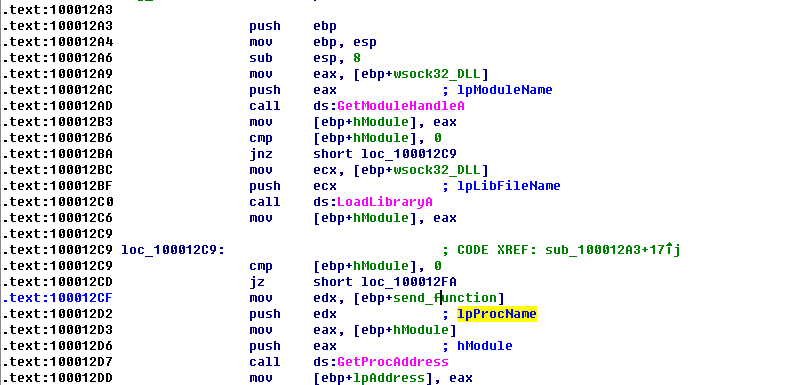




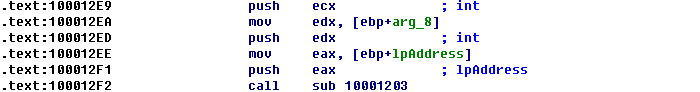
函数sub\_10001499的作用则与之相反，使用ResumeThread来恢复所有的线程

而函数sub\_100012A3应该是恶意代码执行一些影响改变当前进程运行行为的动作，比如修改内核或是安装一个inline挂钩。

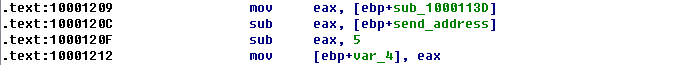
查看函数sub100012A3处的代码，重命名其参数



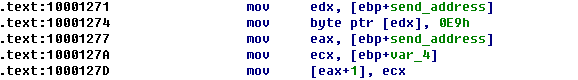
GetModuleHandleA函数获得wsock32.dll的句柄，恶意代码最后传递send函数地址和另外两个参数到sub\_10001203函数(sub\_1000113D和dword\_10003484)



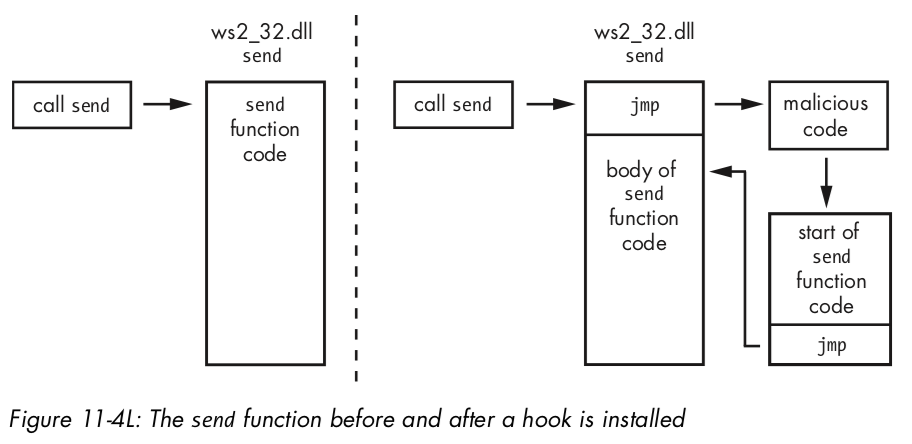
查看sub\_10001203函数，该函数首先计算send函数的内存地址和sub\_1000113D开始的内存地址之间的差，在将差值jmp sub\_1000113D之前，需要从中减去额外的5个字节。



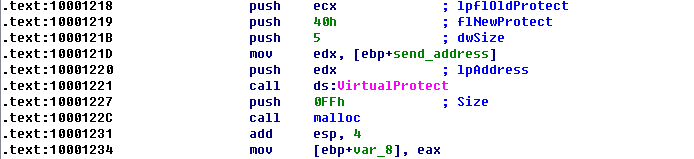
这一部分用来安装inline挂钩，jmp指令0xE9被复制到send函数的开头，之后代码复制var\_4到jmp指令后的内存，作为相对跳转地址，从而让代码跳转到sub\_1000113D处，该函数可以重命名为hook\_function



安装inline挂钩的示意图

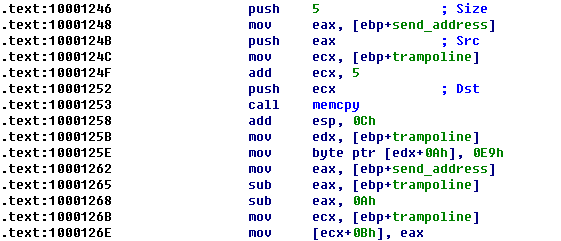


place\_hook在send函数代码的开始调用VirtualProtect函数，修改了内存的运行、读以及写等保护权限，在函数尾部调用另外一个VirtualProtect函数恢复原始的内存保护设置。

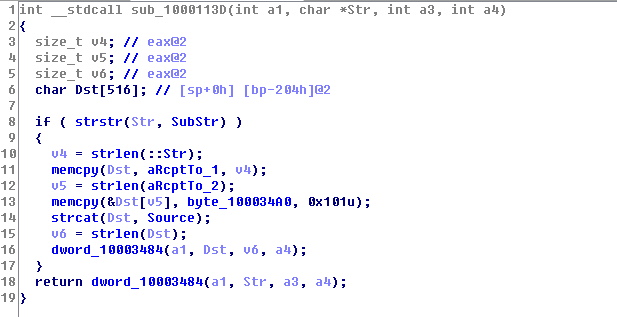


调用VirtualProtect函数后，恶意代码立即使用malloc分配0xFF字节的内存，并将结果保存到var\_8中，这个动态分配的内存作为一个trampoline

为inline挂钩创建trampoline，将send函数的前5个字节复制到trampoline，接下来恶意代码添加了跳转指令，跳转回send函数。最后全局变量dword\_10003484被设置为trampoline的地址，将其重命名为trampoline\_function。

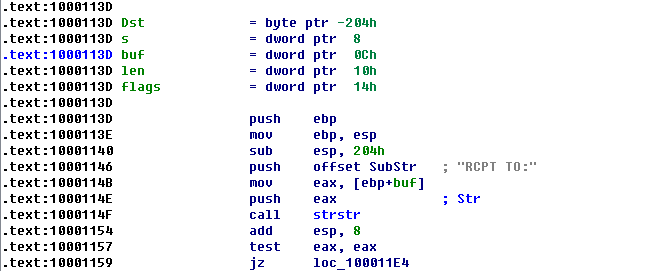


接下来分析hook\_function(sub\_1000113D)，是恶意代码安装的挂钩函数。



挂钩函数在buf中查找字符串RCPT TO:

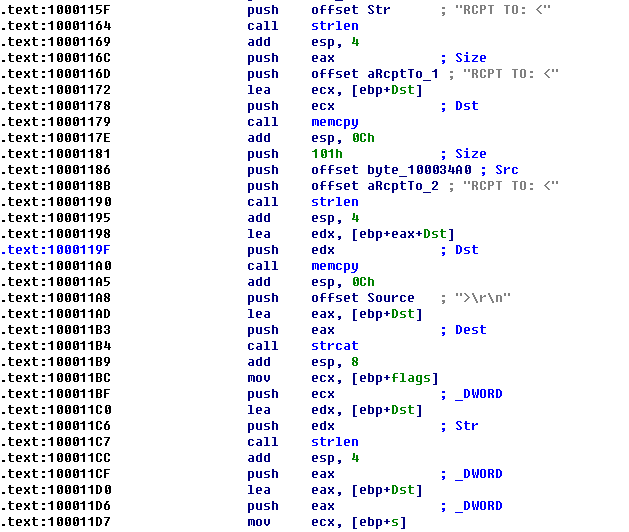
如果没有发现这个字符串，则只调用trampoline\_function



否则将执行这段代码

创建一个添加到向外传输的缓冲区中的字符串，该字符串以RCPT TO: <开头

随后是email\_address，最后以>\r\d结束，这段代码向所有的发出邮件添加了一个收件人



**挂钩操作总结**

* 程序调用send函数。
* 函数的第一条指令send将执行转移到sub\_1000113D.
* sub\_1000113DRCPT TO仅当传出缓冲区包含字符串时才对其进行操作。
* sub\_1000113D调用位于堆上并由 指向的蹦床代码dword\_10003484。
* 蹦床代码执行send函数的前三个原始指令（它覆盖以安装钩子）。
* trampoline 代码跳回send5 个字节的函数，这样就send可以正常运行了。

**捕获网络流量**

要捕获此恶意软件并查看其如何操纵网络流量，请按如下方式设置安全环境：

1. 在虚拟机中打开仅主机网络。
2. 使用命令在您的虚拟机上安装恶意软件rundll32.exe Lab11-02.exe,installer。
3. 将Lab11-02.ini复制到C:\Windows\System32\中
4. 启动Wireshark并开始在虚拟机网络接口上捕获数据包。
5. 设置Outlook Express以将电子邮件发送到主机系统。
6. 使用命令在您的主机上运行一个假邮件服务器python -m smtpd -n -c DebuggingServer IP:25，其中IP是主机的 IP 地址。
7. 从Outlook Express发送电子邮件。
8. 查看Wireshark中的数据包捕获并在电子邮件中选择**Follow TCP Stream 。**
9. 这个恶意DDL导出了什么？

Lab11-02.dll包含一个名为installer的导出函数

1. 使用rundll32.exe安装这个恶意代码后，发生了什么？

恶意代码会作为spoolvxx32.dll将自身复制到系统目录中，并且在APPInit\_DLLs键值下永久安装。此外恶意代码尝试从系统目录中打开Lab11-02.ini，但是并没有这个文件。

1. 为了使这个恶意代码正确安装，Lab11-02.ini必须放置在何处？

C:\Windows\System32系统目录下

1. 这个安装的恶意代码如何驻留？

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows下的AppInit\_DLLs在这个键值中写入DLL的路径，可以实现持久化驻留。

1. 这个恶意代码采用的用户态Rookit技术是什么？

Inline hook

1. 挂钩代码做了什么？

修改了send函数，所有向外发送包含RCPT TO：几个字符的邮件会被额外添加一行，发送给恶意账户。

1. 哪个或者哪些进程执行这个恶意攻击，为什么？

THEBAT.EXE、OUTLOOK.EXE、MSIMN.EXE，查询发现，它们都是邮件进程

1. .ini文件的意义是什么？

存储加密过后的恶意电子邮件地址，用来读取解密。

1. 你怎样用wireshark动态抓获这个恶意代码的行为？

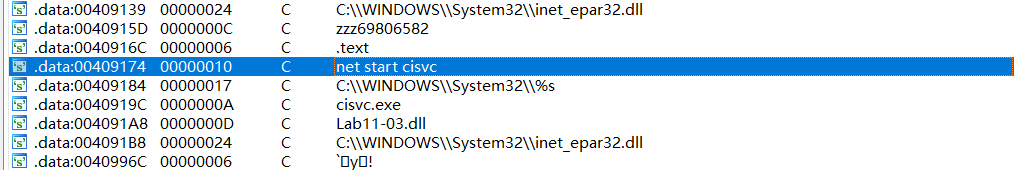
查看与邮件相关的数据包来进行分析。

1. Lab11-3

**基本静态分析**

Lab11-03.exe包含字符串inet\_epar32.dll和net start cisvc

net start命令用于在Windows机器中启动一个服务



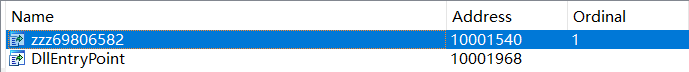
Lab11-03.dll包含字符串C:\Windows\System32\kernel64x.dll

它导入API函数GetForegroundWindow和GetAsyncKeyState，猜测是一个击键记录器，它将击键记录到kernel64x.dll





观察到Lab11-03.dll包含命名奇怪的导出函数zzz69806582

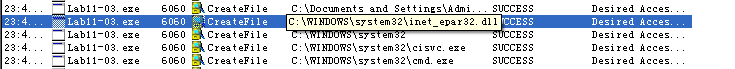


**基本动态分析**

启动procmon运行Lab11-03.exe，恶意代码创建了C:\WINDOWS\system32\inet\_epar32.dll

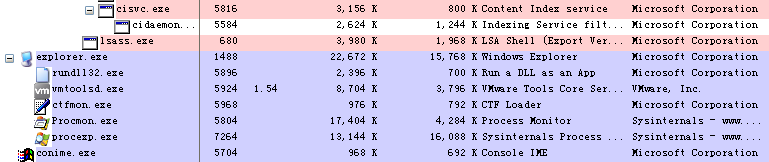
比较文件的md5值，发现inet\_epar32.dll和Lab11-03.dll的内容相同

说明恶意代码复制Lab11-03.dll到Windows系统目录下



之后恶意程序打开了cisvc.exe的句柄，却没有任何WriteFile的操作

使用Process Explorer可以看到cisvc.exe在系统中运行



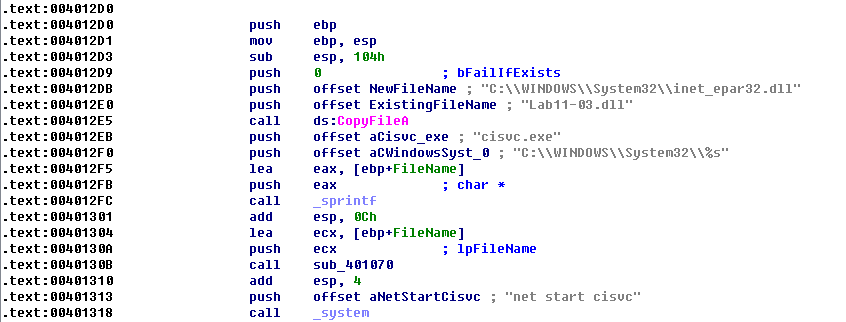
**使用IDA Pro进行高级静态分析**

检查Lab11-03.exe的main函数

main函数首先将Lab11-03.dll复制到C:\Windows\System32中的inet\_epar32.dll

接下来构建字符串C:\WINDOWS\System32\cisvc.exe并将其传递给sub\_4010702处

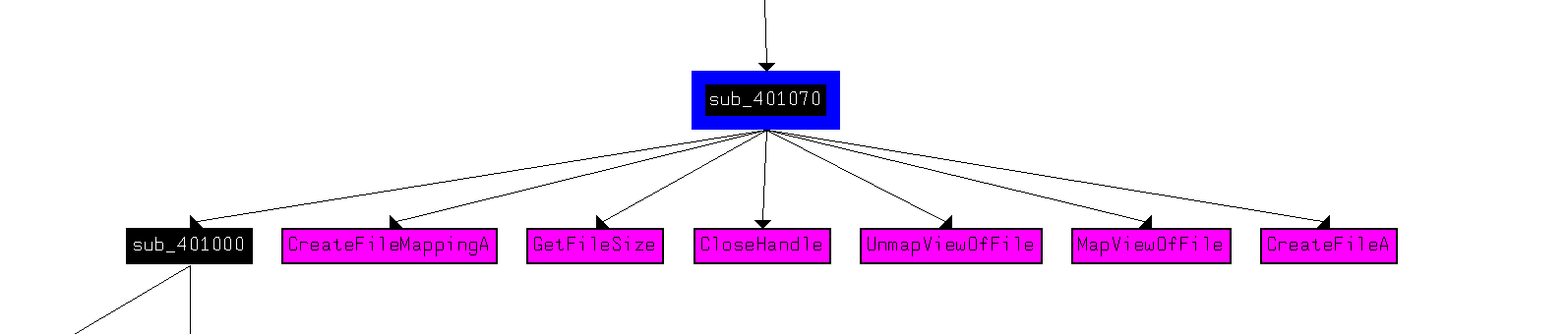
最后，恶意软件通过运行命令net start cisvc启动索引服务



函数sub\_4010702可能对cisvc.exe做了什么，查看其交叉引用图

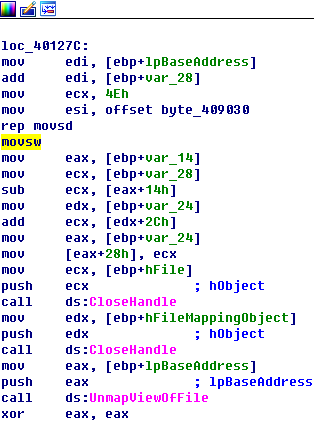
函数调用了CreateFileA、CreateFileMappingA、MapViewOfFile，将cisvc.exe映射到内存中

这些函数都有访问这个文件的读写权限。MapViewOfFile返回的内存映射视图的起始地址可以被读取和写入。在UnmapViewOfFile调用之后，对该文件所做的任何更改都将被写入磁盘，这解释了为什么在procmon输出中没有看到WriteFile函数。

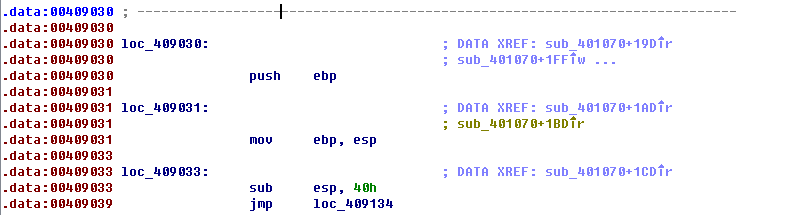


在cisvc.exe的PE头部好像出现了一些计算和检查。

文件的映射位置被移动到EDI中，并使用var\_28调整一些偏移量。 接下来，ECX加载0x4E，即要写入DWORD。因此，总字节数为0x4E \* 4 = 312字节。最后，byte\_409030被转移到ESI, 并将rep movsd数据复制byte\_409030到映射文件中。



查看0x409030处的反汇编代码，它被用于进行进程的注入

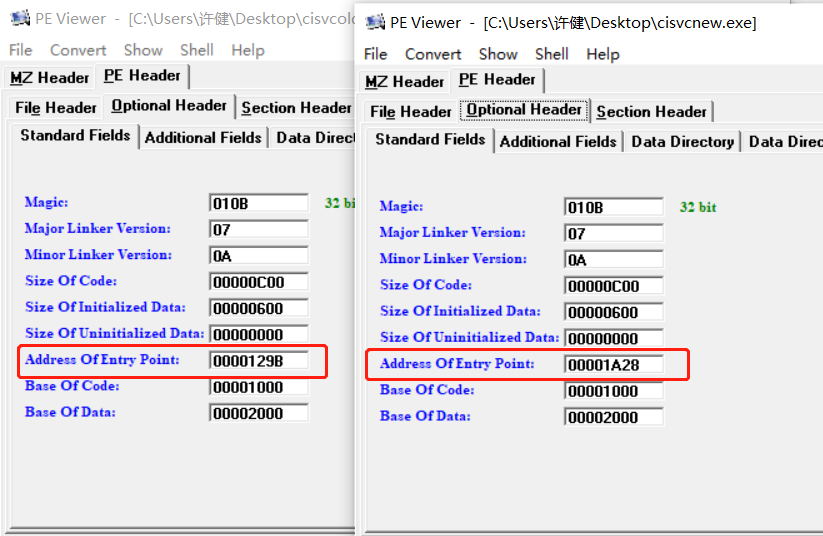


该shellcode包含两个字符串：inet\_epar32.dll路径和导出函数



猜测shellcode加载了这个dll并调用了它的导出函数

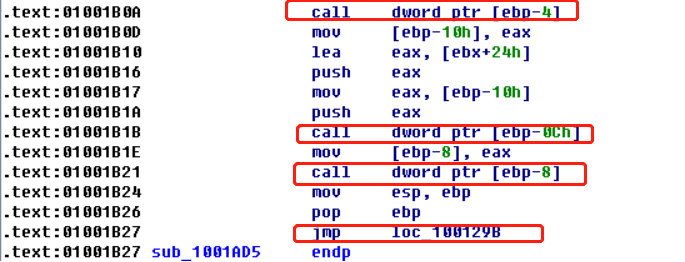
比较进程注入前后cisvc.exe的变化，两个二进制文件的入口点不同



使用IDAPro加载修改后的cisvc.exe

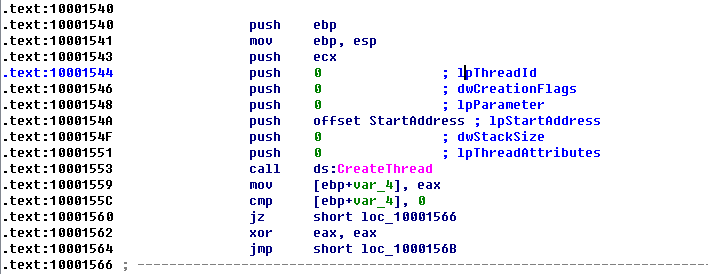
恶意代码的行为包括：

1. 调用LoadLibrary函数，将inet\_epar32.dll载入内存
2. 使用参数zzz69806582调用GetProcAddress，获取导出函数的地址
3. 调用zzz69806582函数
4. 跳转回原始的入口点，使服务正常执行



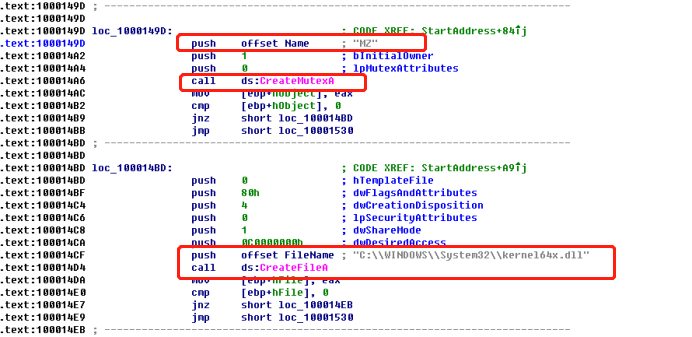
**分析inet\_epar32.dll**

查看导出函数zzz69806582，该函数启动一个线程并返回



重点分析该线程

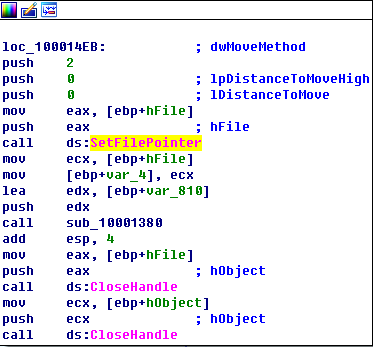
该线程创建一个名为MZ的互斥量，创建一个名为kernel64x.dll的文件，记录击键日志



获得kernel64x.dll的句柄后，恶意软件将文件指针设置为文件末尾，

并调用sub\_10001380，其中包含一个循环。

此循环包含对GetAsyncKeyState、GetForegroundWindow和WriteFile的调用，与“用户空间键盘记录器”中讨论的键盘记录方法一致



1. 使用基础的静态分析过程，你可以发现什么有趣的线索？

可以看到字符串inet\_epar32.dll和net start cisvc，启动了cisvc索引服务，DLL中的导入表有GetAsyncKeyState，进行了键盘记录，还包含字符串C:\WINDOWS\System32\kernel64x.dll，经过静态分析可知这是一个日志文件。

1. 当运行这个恶意代码时，发生了什么？

恶意软件首先将Lab11-03.dll复制到C:\WINDOWS\System32\inet\_epar32.dll，然后将shellcode写入cisvc.exe并启动索引服务，修改cisvc.exe后利用其将击键记录写入C:\Windows\System32\kernel64x.dll

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

将其复制在了system32目录下，然后还向cisvc.exe注入了shellcode，重定向入口点以运行加载 DLL的导出函数实现键盘记录。

1. 这个恶意代码感染Windows系统的哪个文件？

cisvc.exe

1. Lab11-03.dll做了什么？

导出函数zzz69806582实现键盘记录

1. 这个恶意代码将收集的数据放在何处？

C:\WINDOWS\System32\kernel64x.dll

1. **Yara规则**

include ".\includes\IsPE.yar"

include ".\includes\SizeLimit.yar"

rule Lab11\_01

{

    strings:

        $string1 = "msgina32.dll" wide ascii at 00408090

        $string2 = "GinaDLL" wide ascii at 0040804C

        $string3 = "TGAD" wide ascii

    condition:

        IsPE and SizeLimit and (all of ($string\*))

}

rule Lab11\_02

{

    strings:

        $string1 = "RCPT TO:" wide ascii

        $string2 = "spoolvxx32.dll" wide ascii

        $string3 = "\\Lab11-02.ini" wide ascii

        $string4 = "OUTLOOK.EXE" wide ascii

        $string5 = "THEBAT.EXE" wide ascii

    condition:

        IsPE and SizeLimit and (all of ($string\*))

}

rule Lab11\_03

{

    strings:

        $string1 = "\\inet\_epar32.dll" wide ascii

        $string2 = "cisvc.exe"

        $string3 = "zzz69806582"

        $string4 = "\\kernel64x.dll" wide ascii

    condition:

IsPE and SizeLimit and (all of ($string\*))

}

1. **IDAPython脚本**

编写一个查找字符串的IDA Python脚本，实现可以字符串查找，并打印交叉引用地址

import time

# spoolvxx32.dll

str1='73 70 6f 6f 6c 76 78 78 33 32 2e 64 6c 6c'

def GetStrPos(start,end):

    BinaryAddr=FindBinary(start,SEARCH\_DOWN,str1)

    if hex(BinaryAddr)=='0xffffffffL':

        print 'not find'

    else:

        print 'BinaryAddr ',hex(BinaryAddr)

        Jump(BinaryAddr)

    for refhs in XrefsTo(BinaryAddr, flags=0):

        print "refhs: %s refhs.frm 0x%x"%(refhs,refhs.frm)

        time.sleep(2)

        Jump(refhs.frm)

        MakeComm(refhs.frm,"使用了xxx")

for seg in Segments():

    if SegName(seg) == '.rdata':

        GetStrPos(seg,SegEnd(seg))