

**恶意代码分析与防治课程实验报告**

**实验七：分析Windows程序**

****

专 业 信息安全

学 号 2113662

姓 名 张丛

班 级 信安一班

1. **实验目的**

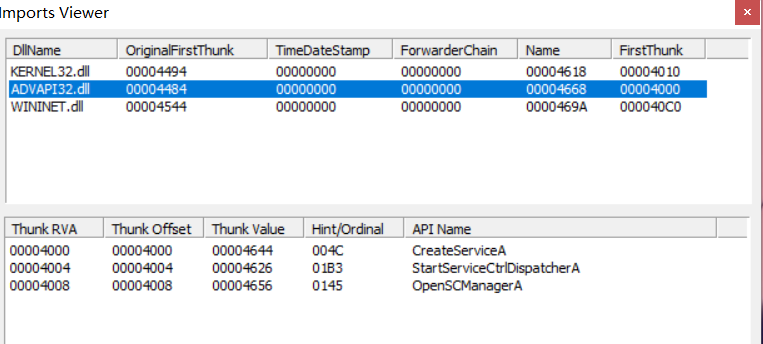
完成课本Lab7的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

1. **实验过程**

### Lab 7-1

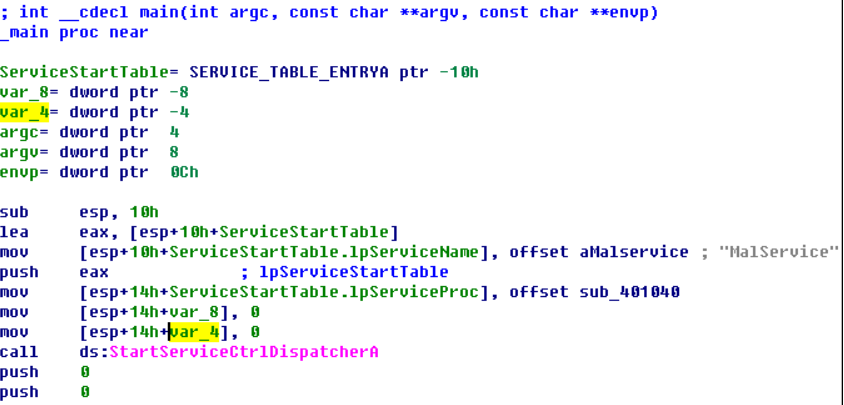
#### **[1.当计算机重启之后，这个程序如何保证它继续运行（达到持久化驻留）？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "1_80" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

查看导入函数：



OpenSCManagerA和CreateServiceA函数暗示着这个恶意代码可能创建一个服务，来保证它在系统被重启后运行，StartServiceCtrlDispatcherA导入函数提示了这个文件是一个服务。

IDA:



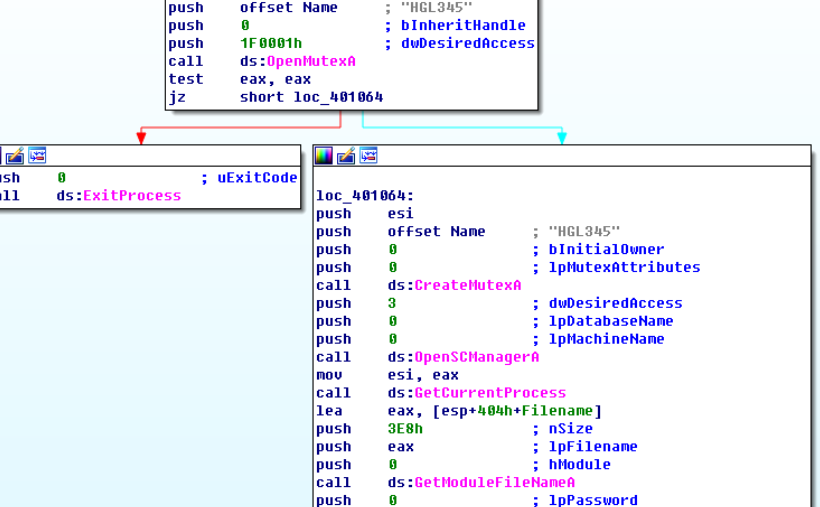
StartServiceCtrlDispatcherA函数用于实现一个服务。

可见，这个程序如何保证它继续运行的措施是：创建了一服务，这个程序创建服务MarlService，来保证它每次在系统启动后运行。

#### **[2.为什么这个程序会使用一个互斥量？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "2_82" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

创建互斥量的函数：





创建了一个名为“HGL345”的互斥量。

如果有一个实例已经在运行了，则对OpenMuteA的第一次调用成功，并且这个程序就会退出。

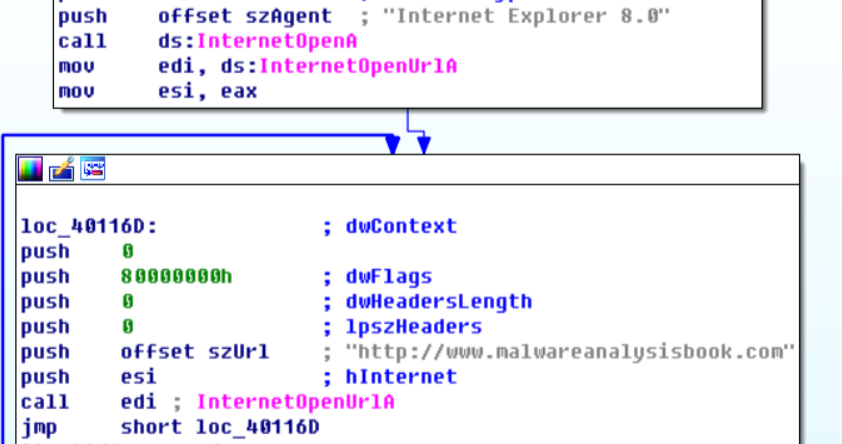
接下来，这段代码调用OpenSCManager，它打开一个服务控制管理器的句柄，以便这个程序可以添加或者修改服务。下一个调用的是GetModuleFileName函数，它返回当前正在运行的可执行程序或一个被夹在DLL的全路径名。

总之，这个程序使用一个互斥量，来保证在同一时间这个程序只有一份实例在运行。

#### **[3.可以用来检测这个程序的基于主机特征是什么？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "3_84" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

名为HGL345的互斥量，以及MalService服务。

#### **[4.检测这个恶意代码的基于网络特征是什么？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "4_87" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**



这个恶意代码使用用户代理 “Internet Explorer 8.0”，并不断的访问 “http://www.malwareanalysisbook.com”。

#### **[5.这个程序的目的是什么？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "5_90" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

这个程序等待到2100年1月1日的半夜，那时发送许多请求到 “http://www.malwareanalysisbook.com” ，带盖是为了对这个网站进行一次分布式拒绝服务（DDoS）攻击。

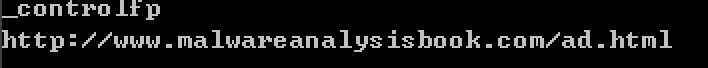
#### **[6.这个程序什么时候完成执行？](https://blog.csdn.net/m0_37442062/article/details/116453999?spm=1001.2014.3001.5502" \l "6_92" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

这个程序永远不会完成。它在一个定时器上等待直到2100年，到时候创建20个线程，每一个运行一个无限循环。

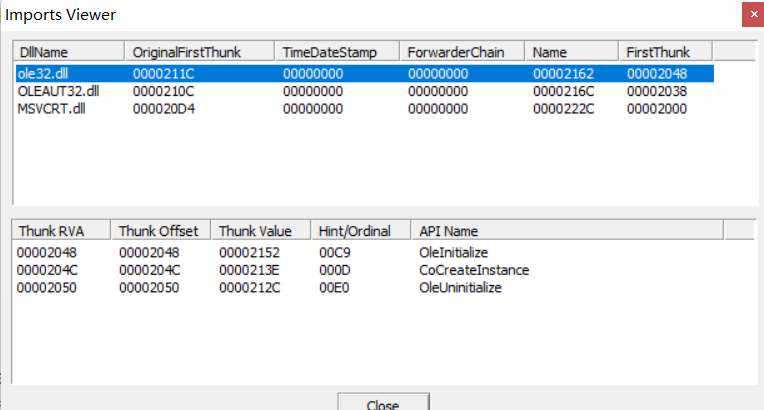
### Lab 7-2

#### **[1.这个程序如何完成持久化驻留？](https://blog.csdn.net/m0_37442062/article/details/116455360?spm=1001.2014.3001.5502" \l "1_32" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

strings查看字符串：



查看导入表：

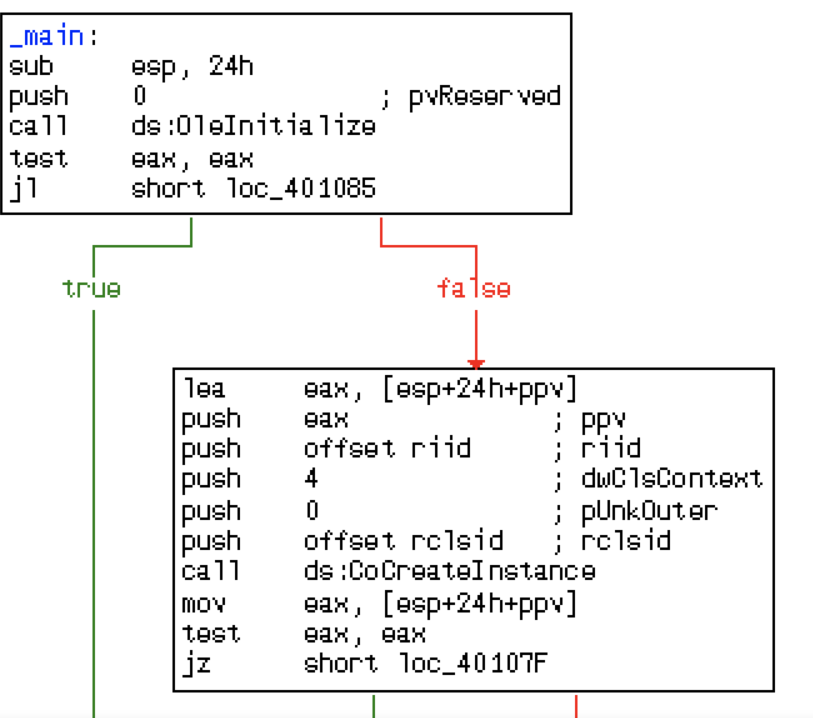


继续查看strings，存在可疑URL：

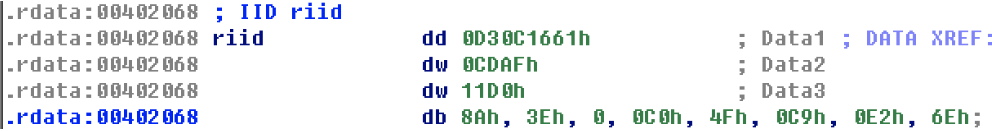
**<http://www.malwareanalysisbook.com/ad.html>**

从ole32库中导入了OleInitialize函数，这个程序可能使用了COM功能。

主函数：

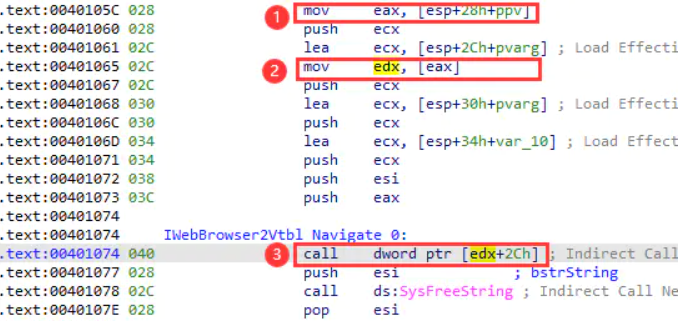


该恶意代码第一件事初始化COM，调用OleInitialize函数和CoCreateInstance获得一个COM对象。返回的COM对象被保存为ppv。



riid和rclsid分别表示接口标识符（IID)和类标识符（CLSID）。

riid为0D30C1661-0CDAF-11D0-8A3E-00C04F0C90E26E，对应IWebBrowser2。  
 rclsid为0002DF01-0000-0000-C000-000000000046，对应Internet Explorer。



返回的COM对象在1处被后面的一些函数访问。

紧跟这条指令，在2处，EAX被解引用并且指向这个COM对象的基址。在3处，这个对象中偏移0x2C是Navigate函数。该函数被调用后，Internet Explorer将导航至：

<http://www.malwareanalysisbook.com/ad.html。>

该函数执行一些清理函数，但是没有持久化驻留，仅简单显示一个广告页面。

总之，该程序没有完成持久化驻留。运行一次后退出。

#### **[2.这个程序的目的是什么？](https://blog.csdn.net/m0_37442062/article/details/116455360?spm=1001.2014.3001.5502" \l "2_34" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

给用户显示一个广告网页：

http://www.malwareanalysisbook.com/ad.html。

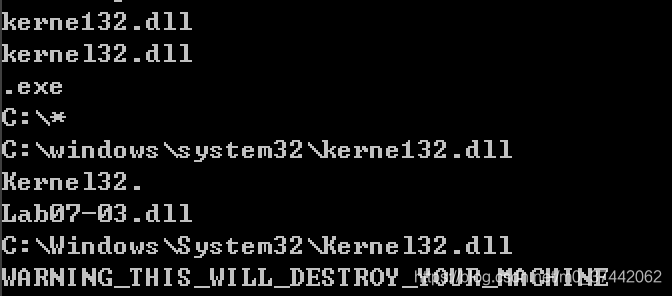
#### **[3.这个程序什么时候完成执行？](https://blog.csdn.net/m0_37442062/article/details/116455360?spm=1001.2014.3001.5502" \l "3_36" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

显示广告页后完成执行。

### Lab 7-3

#### **[1.这个程序如何完成持久化驻留，来确保在计算机被重启后它能继续运行？](https://blog.csdn.net/m0_37442062/article/details/116456223?spm=1001.2014.3001.5502" \l "1_159" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

strings查看字符串：

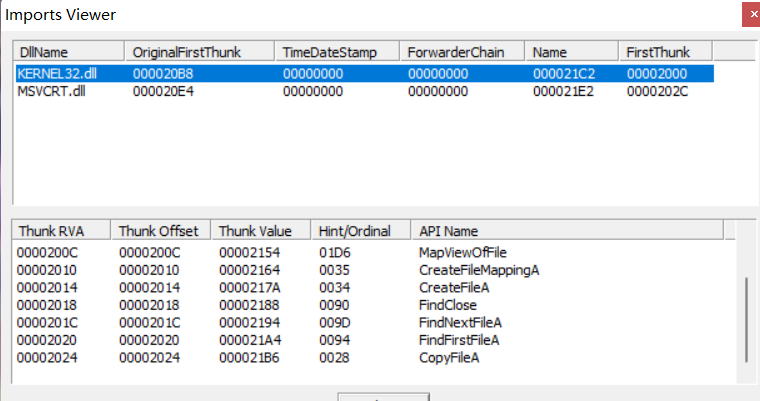


C:\windows\system32\kerne132.dll 是仿冒的dll，说明该文件存在仿冒系统关键dll的可能。

Lab07-03.dll 说明可能调用了Lab07-03.dll。

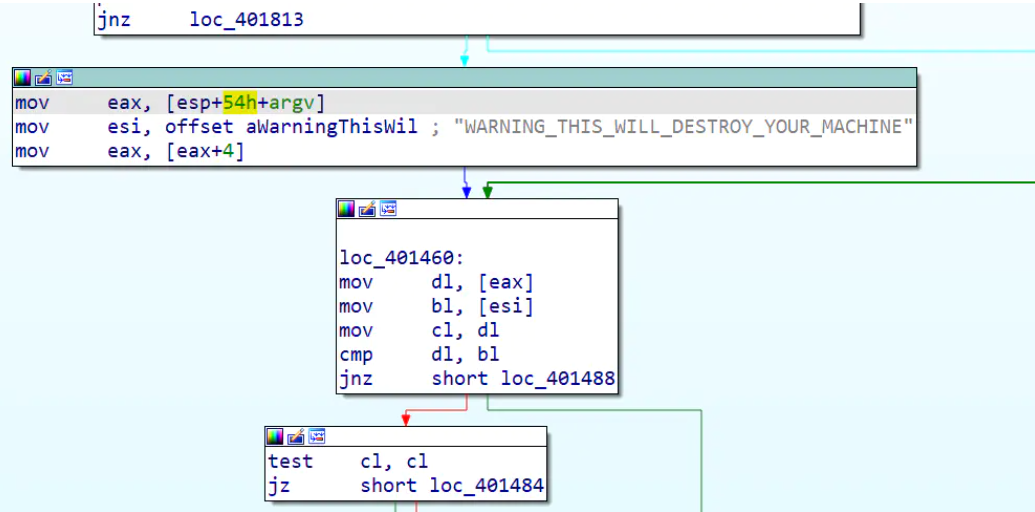


查看导入函数：

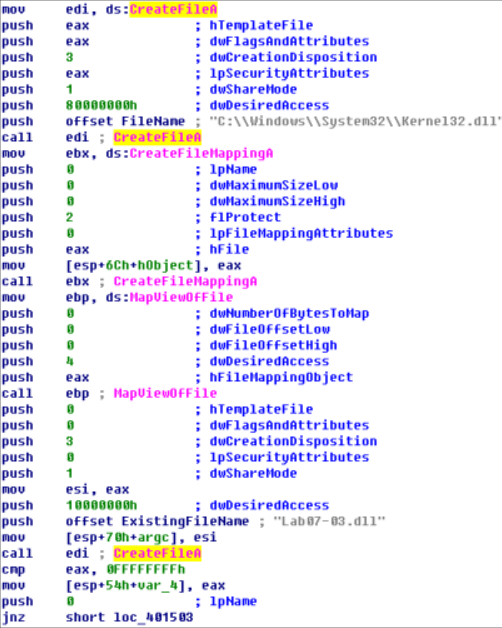


CreateFileA 、 CreateFileMappingA以及MapViewOfFile表明该程序可能打开一个文件，并将其映射到内存中。

FindFirstFileA 和FindNextFileA函数组合，可能会搜索文件，并使用CopyFileA 对找到的文件进行复制操作。



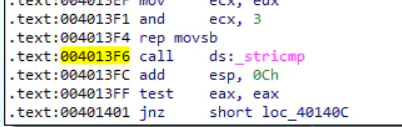
WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE这样保证了恶意代码不会在虚拟机中被自动执行，属于反动态检测的一种手段。  
 打开两个文件C:\Windows\System32\Kernel32.dll 和Lab07-03.dll映射到内存中。



将Lab07-03.dll 复制到C:\windows\system32\kerne132.dll将文件命名为和关键系统库相似的名称，进一步进行恶意行为。并跳转到loc\_401806位置。先将参数C:\\*压栈，调用函数sub\_4011E0。

该函数如下，主要是遍历C盘文件，第29行找到.exe结尾的文件，调用函数sub\_4010A0。

判断文件是不是以.exe结尾的逻辑主要在0x004013F6处。在0x004011E0和0x004013F6处设置断点，F9执行，查看栈上数据变化。可以看到这部分的执行逻辑和静态分析的结论一致。



可见，程序功能就是将对kernel32.dll的调用，转换为kerne132.dll的调用，这样很多程序都会受到影响，很难清除  
 并且分析Lab07-03.dll，查看导入表，有创建互斥体以及网络相关函数，带有后门功能。

总之，这个程序将自己复制到C:\windows\System32，并伪装成重要dll库，实际名称为kerna132.dll来执行持久化。

#### **[2.这个恶意代码的两个明显的基于主机特征是什么？](https://blog.csdn.net/m0_37442062/article/details/116456223?spm=1001.2014.3001.5502" \l "2_161" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

互斥量硬编码SADFHUHF，文件名kerna132.dll。

#### **[3.这个程序的目的是什么？](https://blog.csdn.net/m0_37442062/article/details/116456223?spm=1001.2014.3001.5502" \l "3_164" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

目的是执行持久化操作，并留下后门，可以执行sleep和exec命令。

#### **[4.一旦这个恶意代码被安装，你如何移除它？](https://blog.csdn.net/m0_37442062/article/details/116456223?spm=1001.2014.3001.5502" \l "4_166" \t "https://blog.csdn.net/m0_37442062/article/details/_self)**

这个程序很难被删除，是因它感染系统上每一个.exe文件。

在这个例子中，最好的方法是从一个备份恢复系统。可以留下kernel132.dll文件，删除其中的恶意代码，或者复制kernell32.dll文件，并将它命名为kernel132.dll。

### yara规则

import "pe"

rule UrlRequest {

strings:

$http = "http"

condition:

$http

}

rule Explorer {

strings:

$name = "Internet Explorer"

condition:

$name

}

rule kerne132 {

strings:

$dll\_name = "kerne132.dll"

condition:

$dll\_name

}

rule EXE {

strings:

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

condition:

$exe

}

rule scanC {

strings:

$c = /C:./

condition:

$c

}

### IDA Python

import idautils

print(idautils.Functions())

for i in idautils.Segments():

print(idc.SegName(i),idc.SegStart(i),idc.SegEnd(i))

for i in idautils.Functions():

flags = idc.GetFunctionFlags(i)

if flags&FUNC\_LIB or flags &FUNC\_THUNK:

continue

dism\_addr = list(idautils.FuncItems(i))

for line in dism\_addr:

j = idc.GetMnem(line)

if j == 'call':

op=idc.GetOpType(line,0)

if op == o\_reg:

print("0x%x%s"%(line, idc.GetDisasm(line)))

for i in idautils.Functions():

for j in idaapi.get\_func(here()):

print("Start: 0x%x, End 0x%x" % (func.startEA,func.endEA))

start = idc.get\_func\_attr(here(), FUNCATTR\_START)

end = idc.get\_func\_attr(here(), FUNCATTR\_END)

cur\_addr = start

while cur\_addr <= end:

print hex(cur\_addr), idc.generate\_disasm\_line(cur\_addr, 0)

cur\_addr = idc.next\_head(cur\_addr, end)

1. **实验结论及心得体会**

**对分析windows恶意程序有了更多的经验。**

**更熟悉恶意分析工具的使用。**