

**恶意代码分析与防治课程实验报告**

**实验10**

****

专 业 信息安全

学 号 2113662

姓 名 张丛

班 级 信安一班

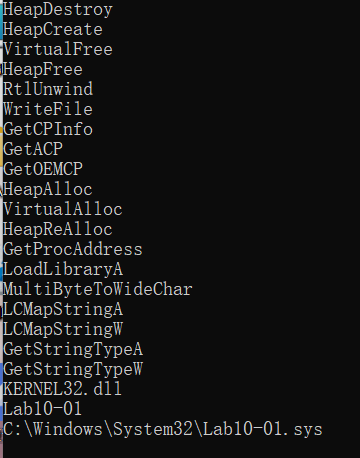
1. **实验目的**

完成课本Lab 10的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

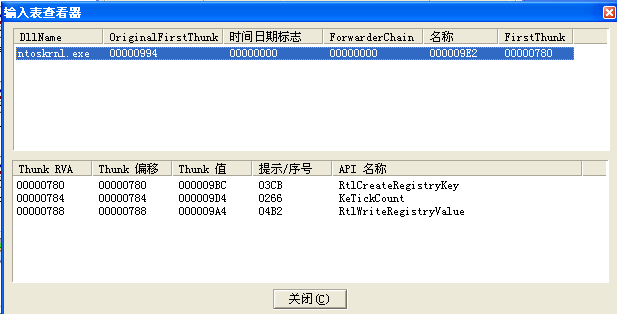
1. **实验过程**

### Lab 10-1

字符串：



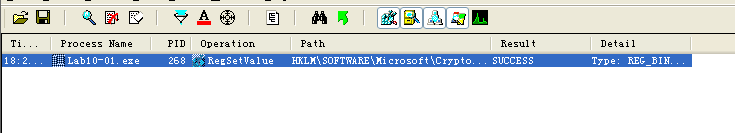
导入函数：



驱动可能访问了注册表



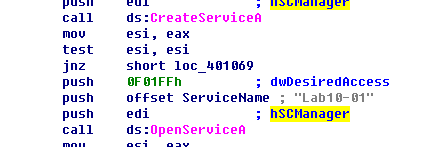
这些字符串预示着恶意代码写了注册表，所以我们打开procmon来验证我们的假设：





程序调用了OpenSCMangagerA获取服务管理器的句柄，然后在调用CreateServiceA，创建一个名为Lab 10-01的服务。 CreateServiceA调用告诉我们服务使用了"C:\Windows\System32\Lab10-01.sys"中的代码，服务类型为3，意味着这个文件将被加载到内核。

如果CreateServiceA调用失败，代码会使用相同的服务名调用OpenServiceA。如果因为服务已经存在而导致调用CreateServiceA失败，这将打开Lab 10-01服务的句柄。

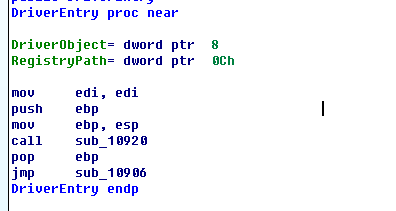


接下来，程序将调用StartServiceA来启动服务，最后调用ControlService。

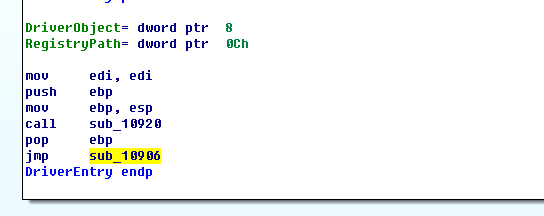
ControlService的第二个参数是发送控制消息的类型。本例中。它的值是1，查询文档，我们发现他是SERVICE\_CONTROL\_STOP，这将会卸载驱动，并调用驱动的卸载函数。

**分析Lab 10-01.sys：**

在试图使用WinDbg分析这个驱动之前，我们可以用IDA Pro打开这个驱动，来检查它的DriverEntry函数。当打开驱动文件，移动到它的入口时，看到如下代码。



这个函数是驱动的入口点，但是它不是DriverEntry函数。编译器在DriverEntry的周围插入封装代码。真正的DriverEntry函数位于sub\_10906。



函数的主体部分似乎将一个偏移量转移到另一个内存位置。除此之外，它没有进行任何函数调用，也没有与系统进行交互。

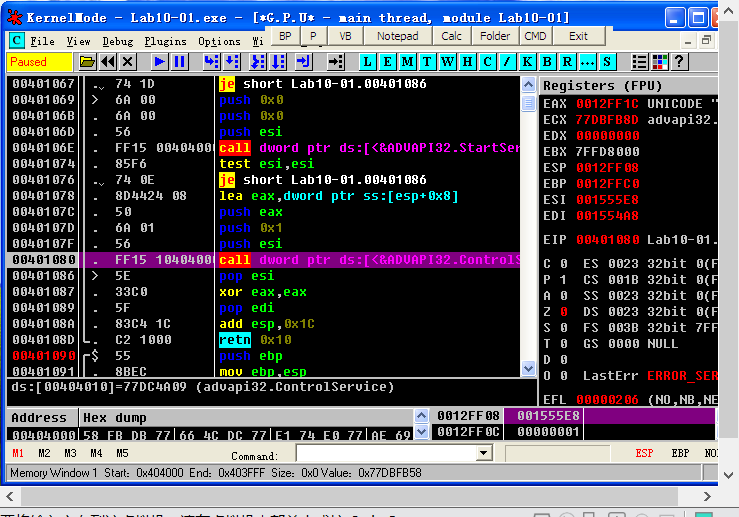
使用WinDbg：

使用WinDbg检查一下Lab10-01.sys，来查看调用ControlService卸载Lab10-01.sys时会发生什么事。

用户空间可执行的代码加载Lab10-01.sys，然后立即卸载它。如果我们在运行恶意可执行程序之前使用内核调试器，因为此时驱动还未在内存中，所以我们还不能检查它。但是，如果我们等待应用程序运行完成，那时驱动又已经从内存中卸载了。

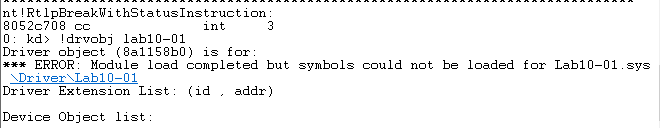
为了在Lab10-01.sys载入内存后，使用WinDbg分析它，在虚拟机中，我们将可执行程序载入到WinDbg中。使用下列命令，我们在驱动加载和卸载之间设置一个断点（在ControlService调用上）：

使用OllyDbg下一个断点：



启动WinDbg进行调试，输入命令来获取驱动对象：

**!drvobj Lab10-01**



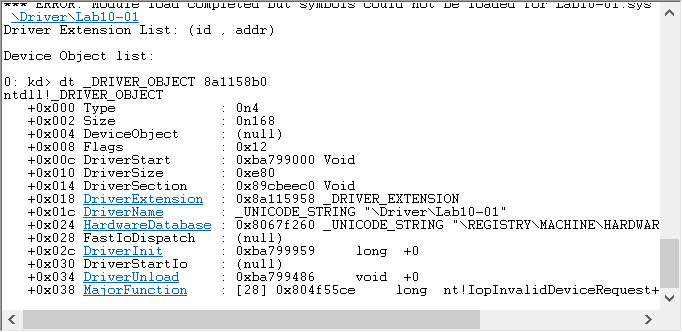
在设备对象列表中没有设备列出，所有以我们明白这个驱动没有供用户空间中应用程序访问的设备。

获取当前内核中驱动对象列表：

**!object \Driver**

查看：

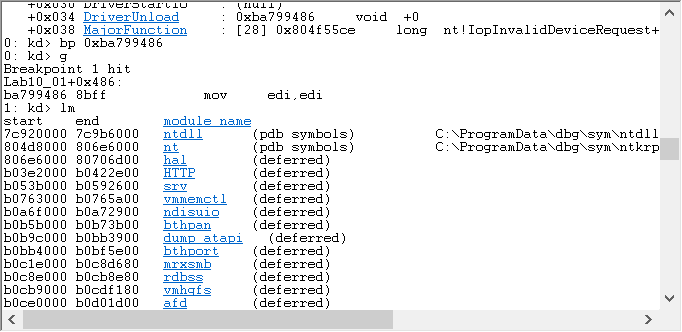
**dt \_DRIVER\_OBJECT 8a1158b0**



设置一个断点：

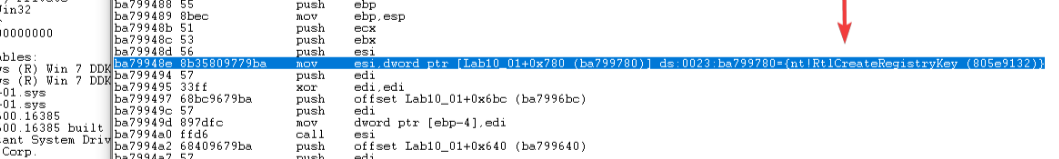
**bp 0xba799486**

然后恢复运行：



文件被加载到ba799000。从上文中，可以得知，卸载函数位于0xba799486。我们从0xba799486减去0xba799000得到偏移量。然后在IDA中跳转到卸载函数。例如，如果IDA中架子基地址0性00000，那么我们在IDA Pro中地址0星00486处找到卸载函数。

继续运行会触发断点，单步运行可以看到调用了nt!RtlCreateRegistryKey。



### 这个程序是否直接修改了注册表(使用procmon来检查)？

写入一个随机值。

### 用户态的程序调用了ControlService函数，你是否能够使用WinDbg设置一个断点，以此来观察由ControlService的调用导致内核执行了怎么样的操作？

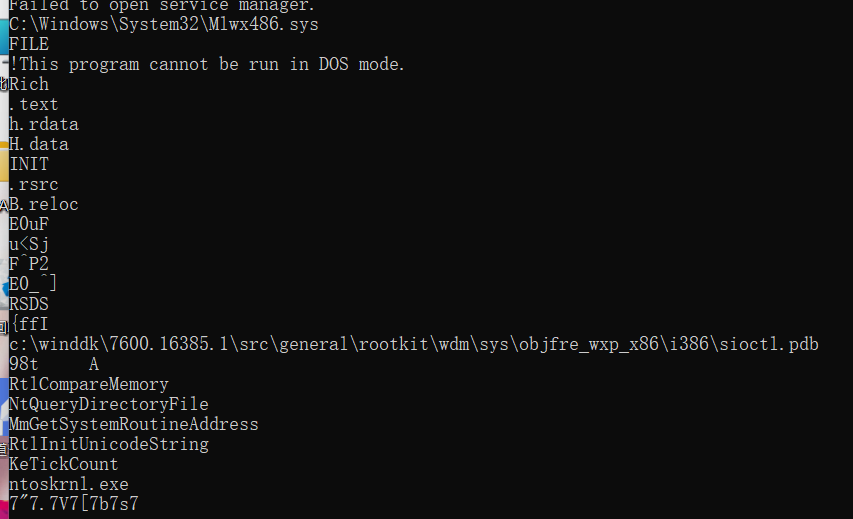
使用OllyDbg将可执行文件下断点到函数前，然后使用另外一个 操作系统的WinDbg，使用!drvobj命令获得驱动设备的句柄，它 包含一个卸载函数的指针。接下来，在驱动的卸载函数上设置一 个断点。重启可执行文件之后，断点将会被触发。

### 这个程序做了些什么？

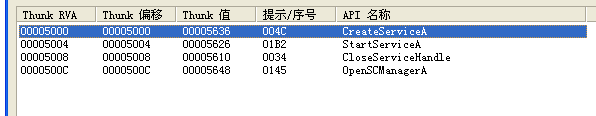
修改了注册表，并关闭了防火墙。

### Lab 10-2

strings:

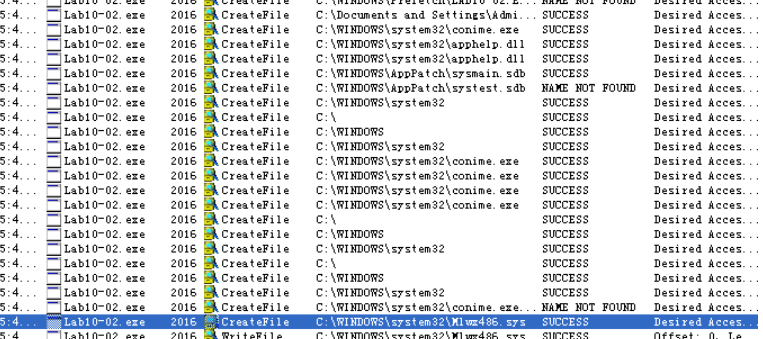


导入表：



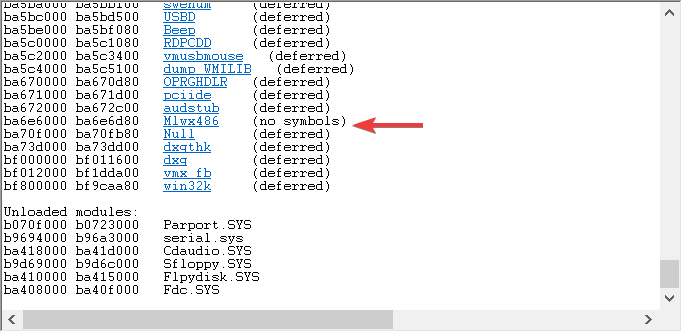
程序调用了CreateFile和WriteFile，说明这个程序可能创建一个文件，而LoadResource和FindResource则说明了该程序对Lab10-02.exe的资源节做了某些处理。

使用ProcessMon进行监视:



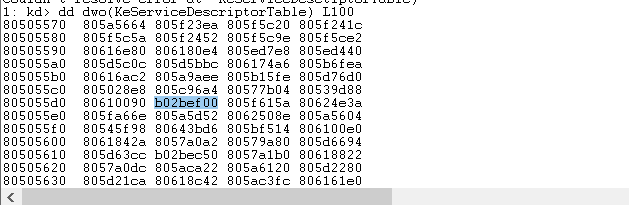
发现该程序创建了一个文件，但我们在当前目录下却并不能查找到该文件。

使用WinDbg机型调试:



使用lm命令，查看驱动是否被真正加载。我们看到一个与Lab10-02.exe创建文件名匹配的条目。

我们确定了文件名为Mlwx486.sys的驱动被载入到内存，但是文件没有在硬盘上显示，这暗示了它可能是一个Rookit。  
 接下来，我们检查SSDT的所有修改项。



我们可以看到有一处条目所在的内存位置很明显位与ntoskrn1模块的范围之外，位与Mlwx486.sys驱动内。为了确定替换了哪个函数，我们回复虚拟机安装Rootkit之前的状态，以便于我们查看存储在SSDT中的哪个函数被覆盖了。

### 这个程序创建文件了吗？它创建了什么文件？

创建了文件，C:\Windows\System32\Mlwx486.sys。

### 这个程序有内核组件吗？

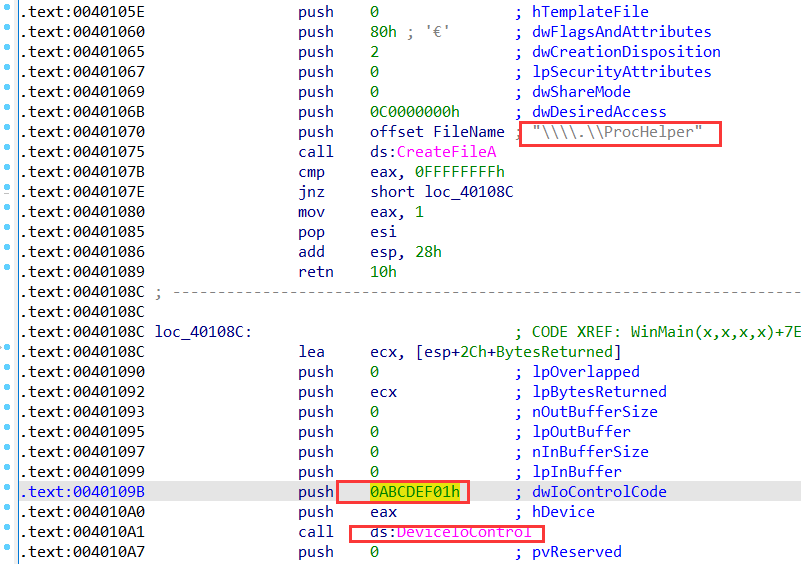
有内核组件，就在Lab10-02.exe的资源节中。

### 这个程序做了些什么？

通过修改SSDT的NtQueryDirectoryFile为FakeNtQueryDirectoryFIle，来隐藏文件。

### Lab 10-3

### 这个程序做了些什么？



首先是安装驱动，驱动位置在：

C:\Windows\System32\Lab10-03.sys

然后接着打开设备对象。

通过IO通信给驱动发送了控制码0ABCDEF01h，接下来的几个函数初始化了COM对象，并在每30s执行一次某个函数。

这个函数有一个参数是这个网址字符串，经过在虚拟机运行程序可知，这是每30s弹出一个广告网页。

### 一旦程序运行，怎么停止？

重启系统。

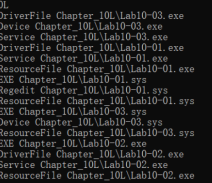
### 它的内核组件做了什么操作？

内核组件负责响应，从进程链表中摘除进程的DeviceIoControl请求。

## yara

import "pe"  
  
rule UrlRequest {  
 strings:  
 $http = "http"  
 $com = /[a-zA-Z0-9\_]\*.com/  
 condition:  
 $http or $com  
}  
  
rule EXE {  
 strings:  
 $exe = /[a-zA-Z0-9\_]\*.exe/  
 condition:  
 $exe  
}  
  
rule Regedit {  
 strings:  
 $system = "Registry"  
 $software = "SOFTWARE"  
 condition:  
 $system or $software   
}  
  
rule DriverFile {  
 strings:  
 $name = ".sys"  
 condition:  
 $name  
}  
  
rule Device {  
 strings:  
 $name = "Device"  
 condition:  
 $name  
}  
  
rule Service {  
 strings:  
 $create = "CreateService"  
 $start = "StartService"  
 condition:  
 $create or $start   
}  
  
rule ResourceFile {  
 strings:  
 $name = ".rsrc"  
 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**

对分析windows恶意程序有了更多的经验。

学习了Windbg的使用，加深了对内核调试的理解。