

**恶意代码分析与防治课程实验报告**

**实验lab12：常用的隐藏技术**

****

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**

在本章中，我们由浅⼊深地探讨了常⻅的恶意代码隐藏启动⽅法。这些⽅法中的很多都涉及操纵系统中的实际内存。例如DLL注⼊、进程替换以及钩⼦注⼊等。另外⼀些则涉及修改硬盘上的⼆进制⽂件，如向⼀个PE⽂件增加.detour段的例⼦。虽然这些技术各不相同，但是它们的⽬的都是相同的。

为了懂得怎么发现系统中运⾏的恶意代码，恶意代码分析⼈员必须能够识别这些启动技术。识别和分析恶意代码的启动技术，是整个分析过程中必不可少的环节，因为所有的启动器只做⼀件事情：让恶意代码获得运⾏。

在下⾯两章中，我们将介绍恶意代码如何加密数据，以及如何通过⽹络进⾏通信。

**进程注⼊ Process Injector**

把恶意代码注⼊到别的进程中去执⾏，常⽤API：VirtualAllocEx，WriteProcessMemory函数等

**DLL注⼊：**写⼀个DLL加载到⽬标进程中会⾃动执⾏dllmain函数

**代码注⼊：**注⼊shellcode

**进程替换 Puppet process**

创建⼀个合法进程，然后在其内存空间写⼊恶意程序，最后通过SetThreadContext函数来让⼊⼝点指向恶意代码进⾏执⾏，也叫傀儡进程

**Hook注⼊ Hook Injector**

使⽤SetWindowsHookEx来设置消息Hook

**APC注⼊ APC Injector**

每个线程都有⼀个附加的APC队列，在线程处于可警告状态的时候被处理，在这个状态的时候会⼀次调⽤APC队列中的所有函数，可通过编写代码⽤APC抢占可警告状态的线程

**⽤户模式的APC使⽤API**：QueueUserAPC，⼀般会注⼊⽬标进程的所有线程，以确保APC很快会被执⾏

**内核模式的APC使⽤API：**KeInitializeAPC，KeInsertQueueApc，⼀般来注⼊⽤户层shellcode到⽤户空间去执⾏

1. **实验原理**

进程注入、进程替换和APC注入是一些与进程操作和代码执行相关的技术和攻击手段。下面我将为您概述它们的原理以及相关的防治方法及原理。

**进程注入：**

进程注入是一种将恶意代码注入到目标进程中并在其上下文中执行的技术。攻击者通过利用漏洞或特权提升，将恶意代码加载到目标进程的地址空间，并在目标进程的执行流中执行该代码。

**防治方法及原理：**

**代码签名验证：**验证进程中的代码是否被合法签名，防止未经授权的代码注入。

**内存保护：**使用内存保护机制，如写时复制 (Copy-on-Write, COW)、不可执行内存页面等，防止对进程内存的非法修改和注入。

**权限控制：**限制进程的权限，确保只有具有足够权限的进程才能对其他进程进行注入操作。

**进程替换：**

进程替换是指将一个进程的执行内容替换为另一个进程的执行内容。这可以通过多种方式实现，如执行一个新程序，或将进程的执行上下文替换为另一个进程的上下文。

**防治方法及原理：**

代码签名验证：验证被替换进程的代码是否被合法签名，防止替换为未经授权的代码。

进程完整性检查：使用完整性检查机制，如哈希校验，在进程替换前后验证进程的完整性，确保被替换的进程没有被篡改。

权限控制：限制对进程替换操作的权限，仅允许具有足够权限的进程进行替换操作。

**APC注入：**

APC（Asynchronous Procedure Call）注入是一种利用操作系统提供的异步过程调用机制，将恶意代码插入到目标进程的执行流中的技术。攻击者通过将恶意代码注册为APC回调函数，并将其插入到目标进程的执行队列中，使其在目标进程的上下文中执行。

**防治方法及原理：**

**内存保护：**使用内存保护机制，如不可执行内存页面，防止恶意代码被写入可执行内存区域。

**权限控制：**限制对目标进程的操作权限，确保只有具有足够权限的进程才能向目标进程注入APC。

**行为监测：**实施行为监测和入侵检测系统，检测和阻止异常的APC注入行为。

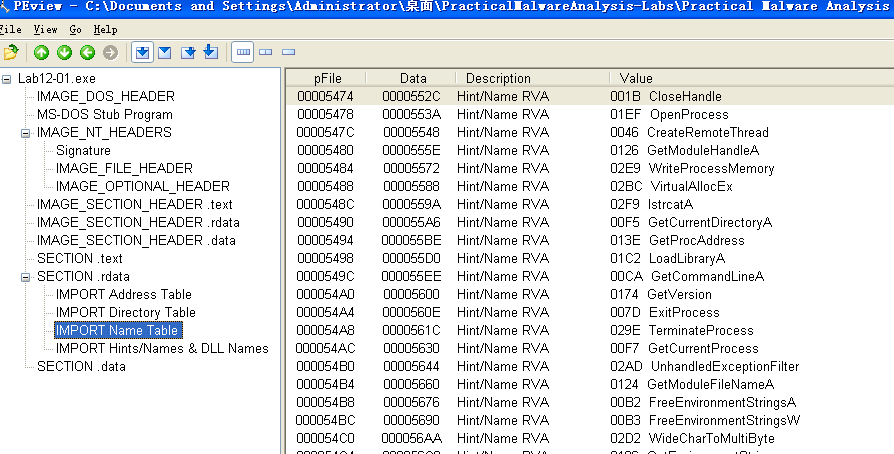
需要注意的是，以上的防治方法并非绝对，攻击者可能会采用新的技术或绕过现有的防护机制。因此，持续的系统更新、安全补丁和综合的防护策略对于保护系统和应用程序的安全性至关重要。

1. **实验过程**



首先使用PEID查壳，发现无壳。

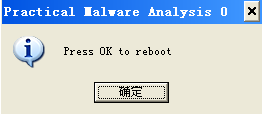
导⼊表导⼊了CreateRemoteThread函数，有点可疑的字符串，如下图所示：



这⾥看到了导⼊表没有的LoadLibraryA函数，这⾥⼤概率是个DLL注⼊。

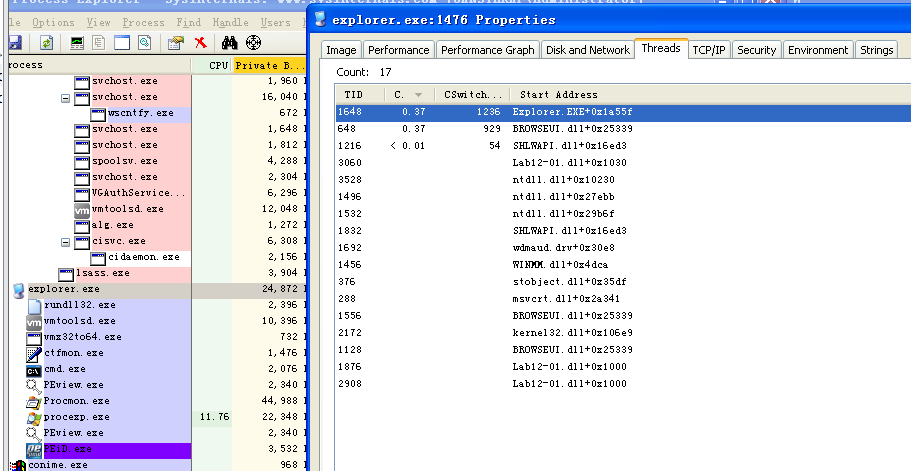
Q1.在你运⾏恶意代码可执⾏⽂件时，会发⽣什么？

运⾏这个恶意代码之后，每分钟在屏幕上显示⼀次弹出消息，⽆限弹窗，⼀关掉就弹出来，⽆穷⽆尽：

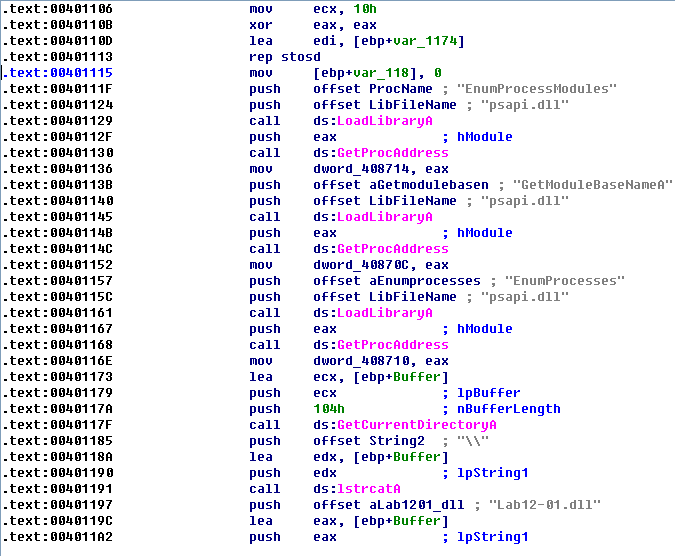


Q2.哪个进程会被注⼊？

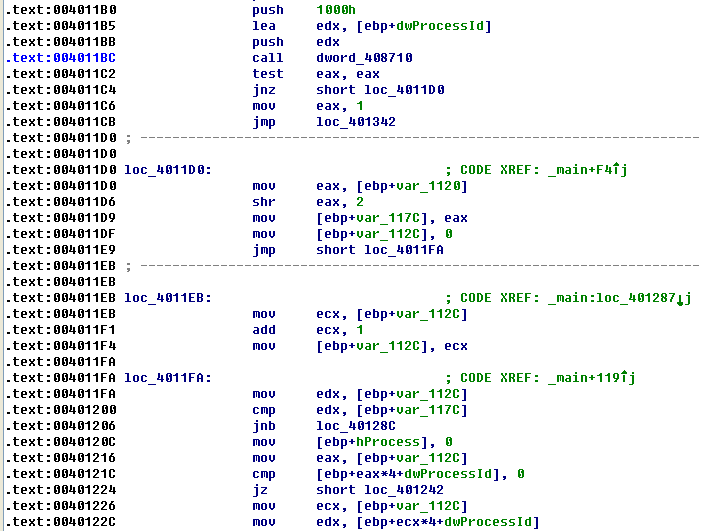
被注⼊的进程是explorer.exe:



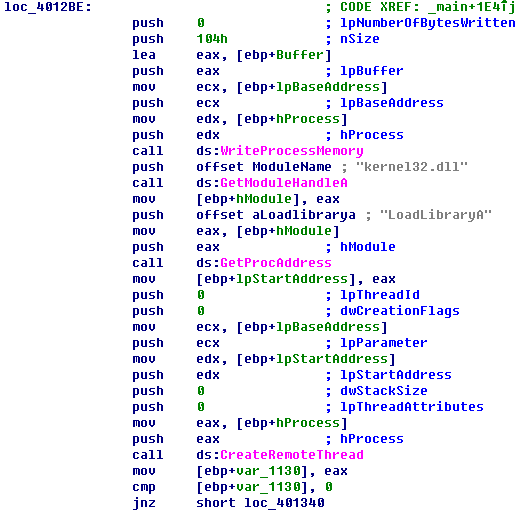
静态分析：exe程序⾸先动态获取⼏个函数地址，如下图所示：



然后进⼊for循环进⾏遍历获取到的进程句柄，如下图所示：



这⾥对每个进程句柄都调⽤了⼀下sub\_401000函数：这个函数的功能是打开进程遍历模块看有没有名字是explorer.exe的模块在，如果有就返回1，然后主程序就会打开进程，跳出循环进⼊下⼀步，如下图所示：

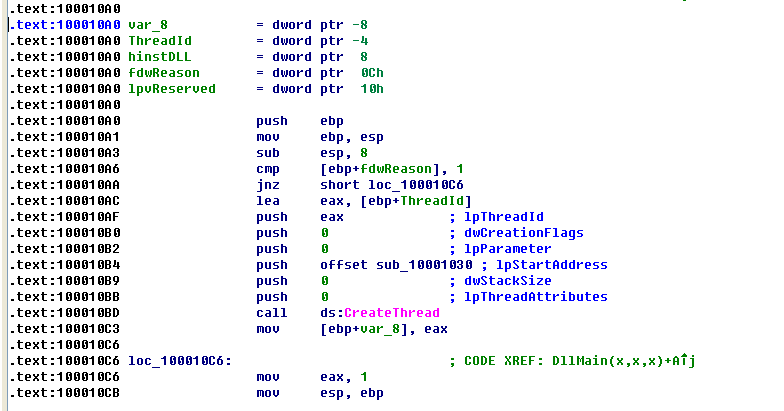


Q3.你如何能够让恶意代码停⽌弹出窗⼝？

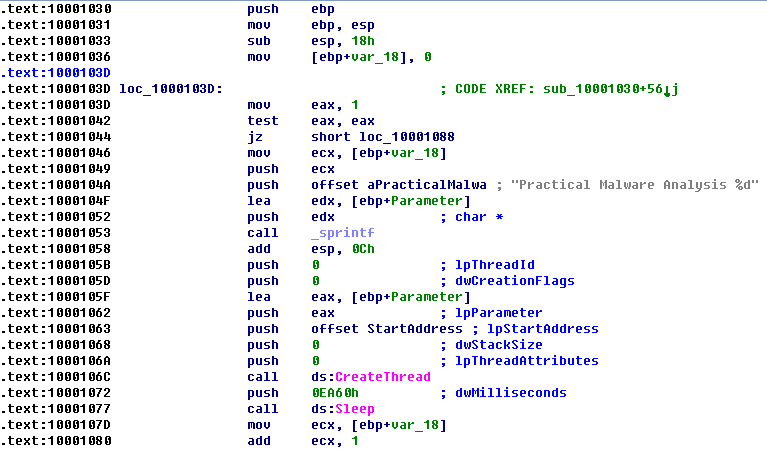
你可以重新启动explorer.exe进程，重启电脑也⾏。

Q4.这个恶意代码样本是如何⼯作的？

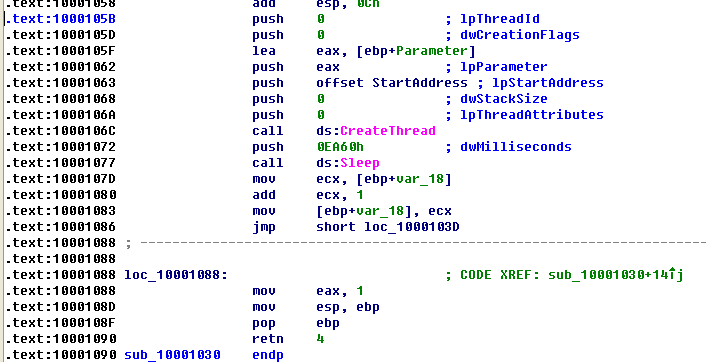
这个恶意代码执⾏ DLL 注⼊，来在explorer.exe 中启动 Lab12-01.dll。⼀旦Lab12-01.dll 被注⼊，它在屏幕上每分钟显示⼀个消息框，并通过⼀个计数器，来显示已经过去了多少分钟，dllmain函数⾥直接创建了线程，如下图所示：



线程⾥是个死循环：不断执⾏弹窗函数:



线程⾥是个死循环：不断执⾏弹窗函数:



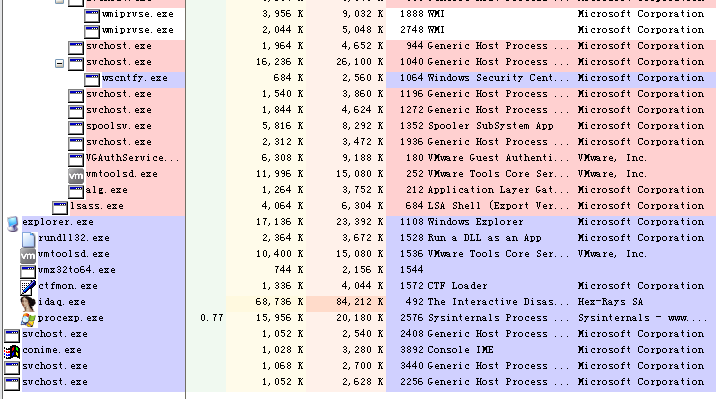
lab12-2

Q1.这个程序的⽬的是什么？

这个程序的⽬的是秘密地启动另⼀个程序（键盘记录器），会把在某个窗⼝下按键的内容记录在exe⽬录下的：practicalmalwareanalysis.log⾥：

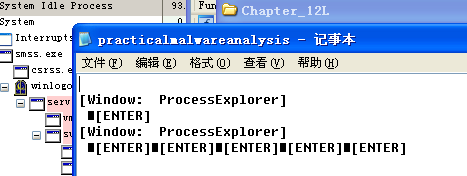
行为分析整体功能：进程替换svchost，做键盘记录器，替换的shellcode是xor加密的资源文件，因此解密的话需要使用xor。

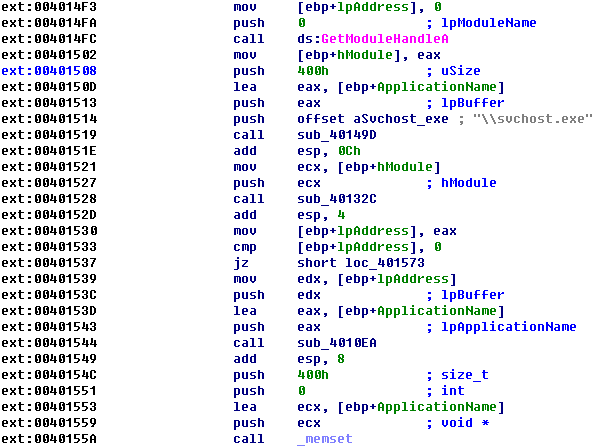
Q2.启动器恶意代码是如何隐蔽执⾏的？



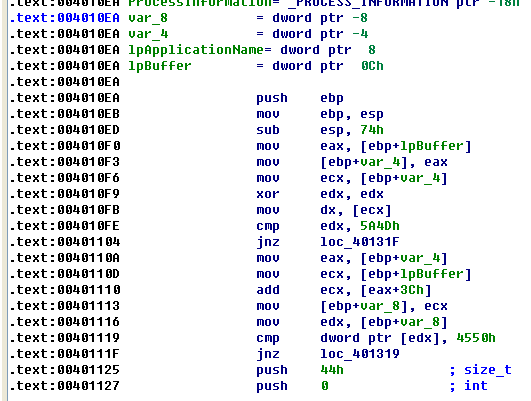
这个程序使⽤进程替换来秘密执⾏，程序exe是个启动器，负责启动svchost.exe，然后修改其内存为资源⽂件的内容，创建傀儡进程来隐蔽执⾏，资源中的PE⽂件才是真正功能模块，提取内存中解密后的资源⽂件拖⼊IDA，分析：

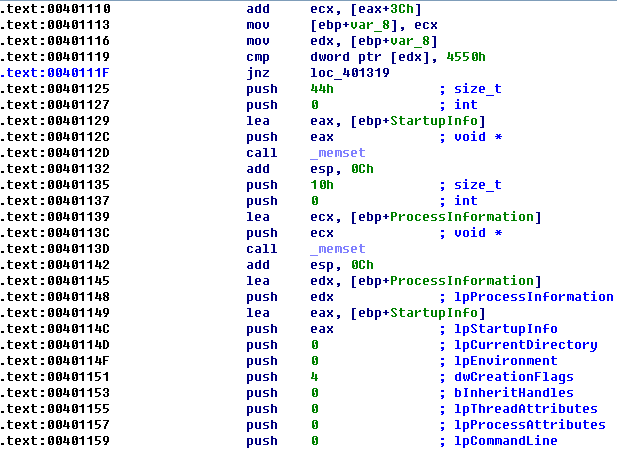
主函数⾸先获取两样东⻄：svchost.exe的绝对路径，资源头⾥解密后的PE⽂件，然后就进去函数创建傀儡进程了，如下图所示：





函数sub\_4010EA：先判断缓冲区是不是PE⽂件，是的话，就创建进程svchost，启动标识为启动后挂起，如下图所示：





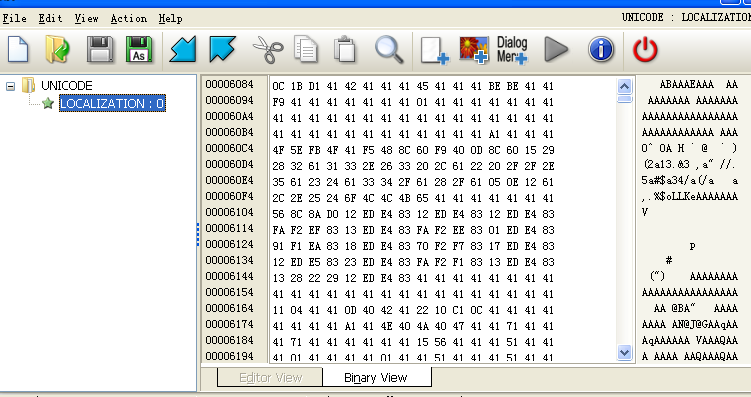
接下来的操作就是加载PE⽂件到内存展开，然后设置线程上下⽂到PE⼊⼝点，然后恢复线程开始执⾏，具体操作就不细看了

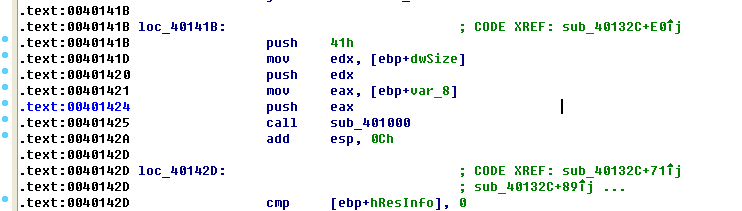
Q3.恶意代码的负载存储在哪⾥？

这个恶意的有效载荷 (payload）被保存在这个程序的资源节中。这个资源节的类型是 UNICODE，且名字是LOCALIZATION，加密存储在资源⾥：

Q4.恶意负载是如何被保护的？

保存在这个程序资源节中的恶意有效载荷是经过 XOR 编码过的。这个解码例程可以在sub 40132C处找到，⽽ XOR 字节在0x0040141B 处可以找到，通过加密存储来进⾏保护，解密程序：





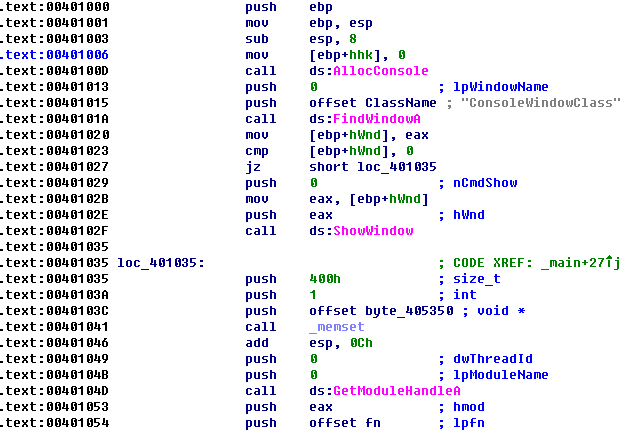


Q5.字符串列表是如何被保护的？ 这些字符串是使⽤在sub\_401000处的函数，来进⾏ XOR 编码的，通过异或进⾏保护，代码同题4。

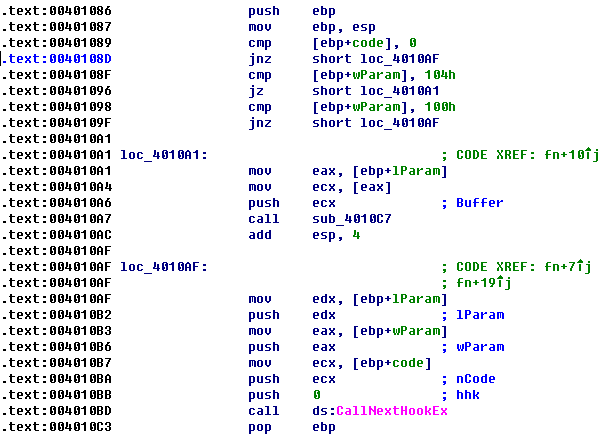
lab12-3

Q1.这个恶意负载的⽬的是什么？

这个程序是⼀个击键记录器，设置键盘钩⼦，监听键盘输⼊事件，如下图所示：



hook函数：向指定⽂件写⼊内容，然后继续传递消息，如下图所示：



Q2.恶意负载是如何注⼊⾃身的？

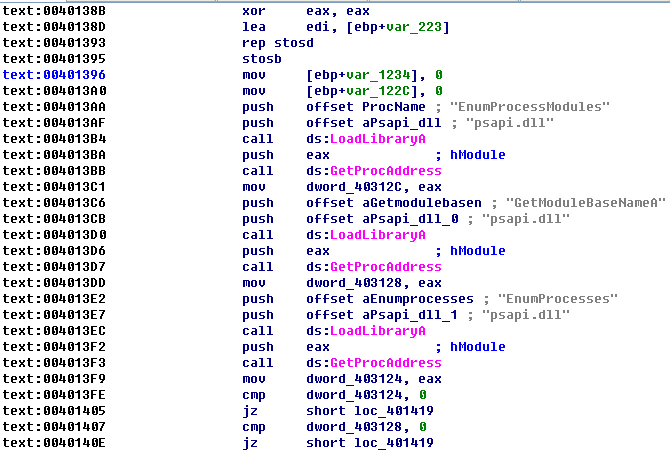
这个程序使⽤挂钩注⼊，来偷取击键记录，通过SetWindowsHookEx函数设置全局消息钩⼦来注⼊⾃身。

Q3.这个程序还创建了哪些其他⽂件？

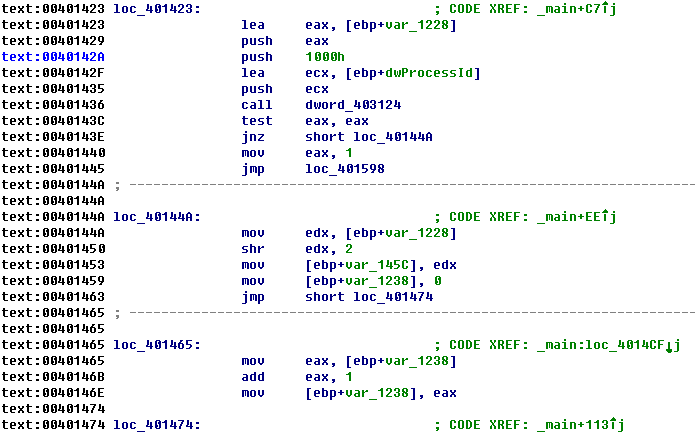
这个程序创建⽂件praticalmalwareanalysis.log，来保存击键记录。

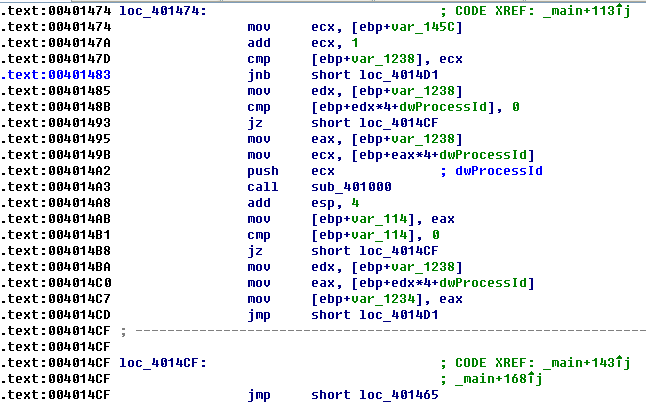
lab12-4

主函数：⾸先动态获取⼏个函数，如下图所示：



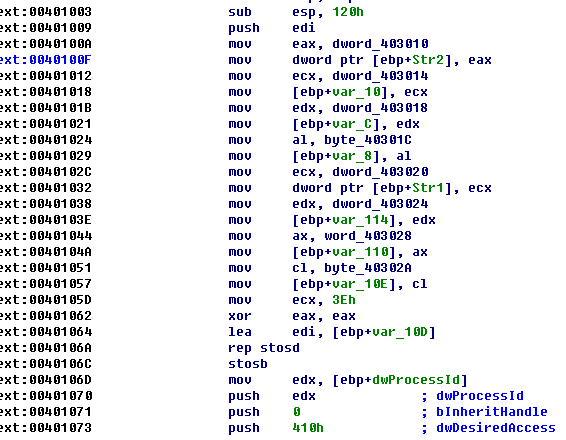
接下来进⼊for循环遍历进程：找到winlogon.exe进程，具体sub\_401000函数分析⻅题1，后⾯的分析写题⾥了就，如下图所示：





Q1.位置 0x401000 的代码完成了什么功能？

恶意代码查看给定 PID 是否为 winlogon.exe 进程，这⾥代码的功能是找到winlogon.exe模块，分析：⾸先填充两个字符串，如下图所示：



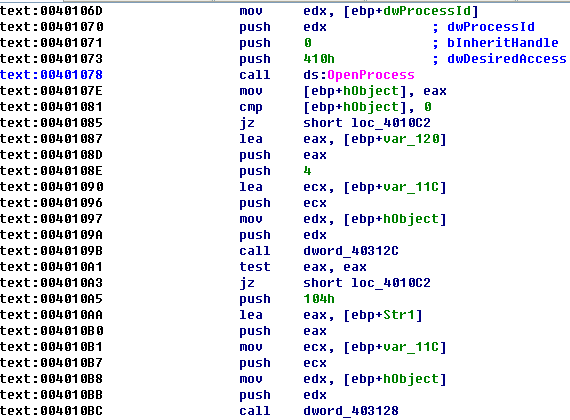
Q2.代码注⼊了哪个进程？

winlogon.exe 是被注⼊的进程，根据题1的分析，注⼊winlogon.exe进程。

Q3.使⽤ LoadLibraryA 装载了哪个DLL程序？

DLL sfc\_os.dll ⽤来禁⽤ Windows 的⽂件保护机制，加载了sfc\_os.dll，紧接着循环遍历进程之后，

⽴⻢调⽤了sub\_401174函数：找到sfc.os.dll的2号函数，然后远程线程到⽬标进程中执⾏2号函数是SfcTerminateWacherThread，在下次启动之前禁⽤windows⽂件保护机制，如下图所示：





Q4.传递给 CreateRemoteThread 调⽤的第4个参数是什么？

传递给 CreateRemoteThread 的第4 个参数是⼀个函数指针，指向sfc\_os.dll中⼀个未命名的序号为2的函数 (SfcTerminateWatcherThread)，和题⽬三类似是从dll中找到的函数地址。

Q5.⼆进制主程序释放出了哪个恶意代码？

恶意代码从资源段中释放⼀个⼆进制⽂件，并且将这个⼆进制⽂件覆盖旧的 Windows 更新程序

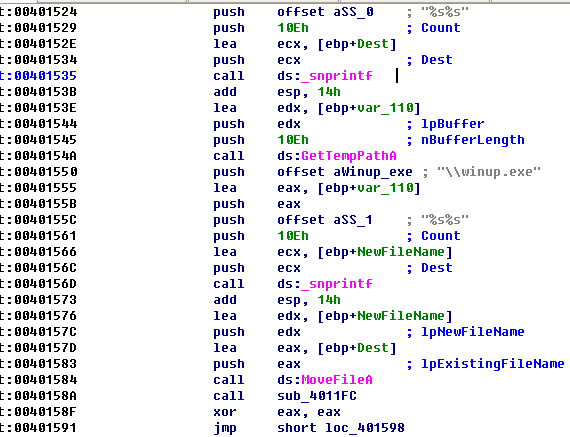
(wupdmgr.exe）。覆盖真实的 wupdmgr.exe之前，恶意代码将它复制到%TEMP%⽬录，供以后使⽤，这⾥先把原本的wupdmgr.exe给移动到临时⽬录下了，然后调⽤函数sub\_4011FC进⾏资源释放，释放假的wupdmgr.exe到原位置，如下图所示：

Q6.释放出恶意代码的⽬的是什么？

恶意代码向winlogon.exe注⼊⼀个远程线程，并且调⽤sfc\_os.dll的⼀个导出函数（序号为2的

SfcTerminateWatcherThread），在下次启动之前禁⽤ Windows 的⽂件保护机制。因为这个函数⼀定要运⾏在进程winlogon.exe 中，所以CreateRemoteThread 调⽤⼗分必要。恶意代码通过⽤这个⼆进制⽂件来更新⾃⼰的恶意代码，并且调⽤原始的⼆进制⽂件（位于%TEMP%⽬录）来特洛伊⽊⻢化 wupdmgr.exe ⽂件。

找到存起来的资源⽂件进⾏分析：执⾏真正的wupdmgr.exe程序，然后下载更新exe程序并执⾏，⽤来特洛伊⽊⻢化wupdmgr.exe⽂件，并通过下载更新⽂件来更新恶意代码，如下图所示：



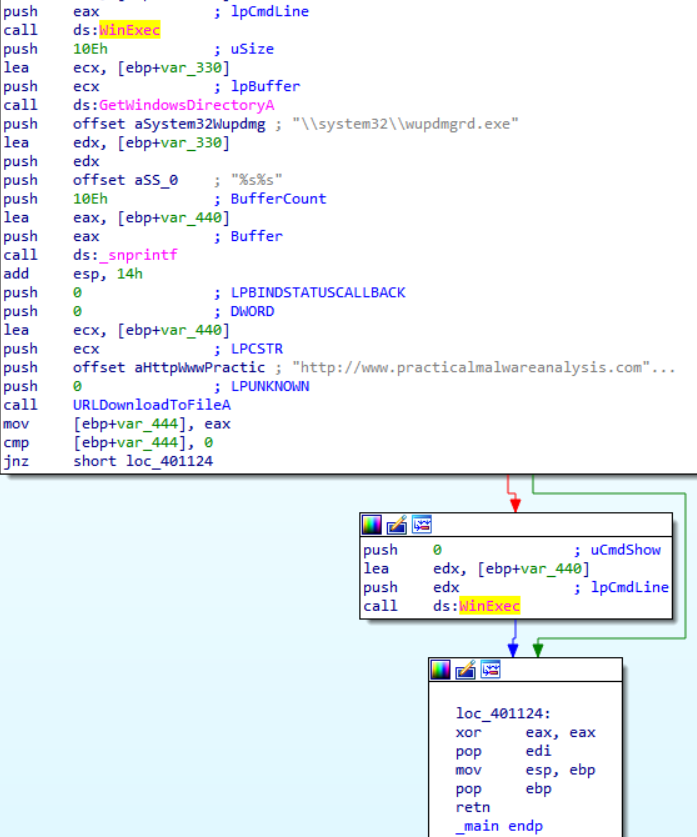
找到winlogon.exe的PID

提升进程权限

创建远程线程执行，sfc\_os.dll的2号导出函数

将wupdmgr.exe移动到%Temp%目录

释放资源到wupdmgr.exe



**YARA**

根据字符串，可以编写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 WriteFile {

strings:

$name = "WriteFile"

condition:

$name }

rule SetHook {

strings:

$SetFunc = "SetWindowsHookExA"

$UnFunc = "UnhookWindowsHookEx"

condition:

$SetFunc or $UnFunc

}

rule URL {

strings:

$Http = "http://" nocase

$Https = "https://" nocase

condition:

$Http or $Https

}

rule UseSource {

strings:

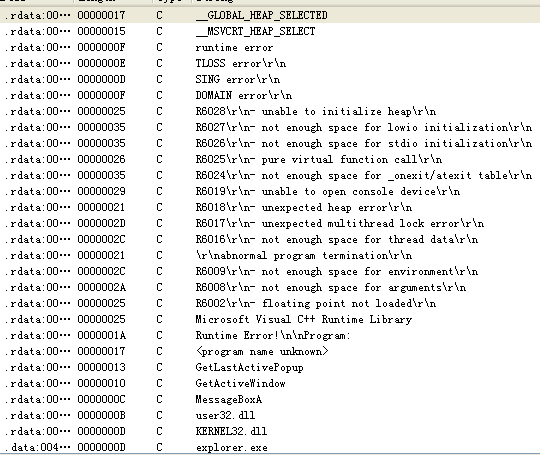
$find = "FineResourceA"

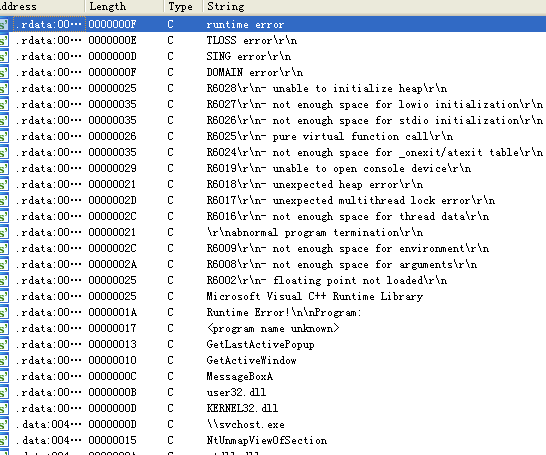
$load = "LoadResource"

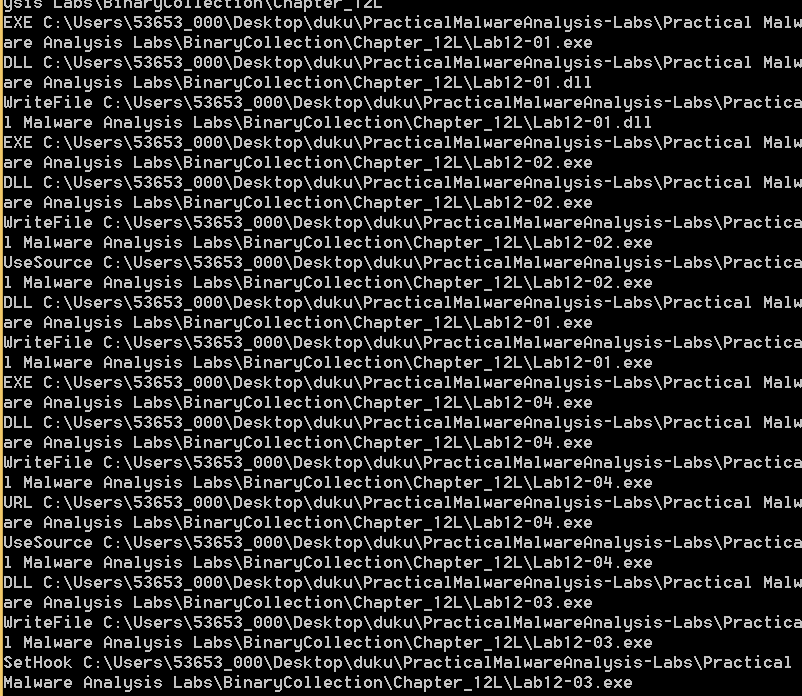
$size = "SizeofResource"

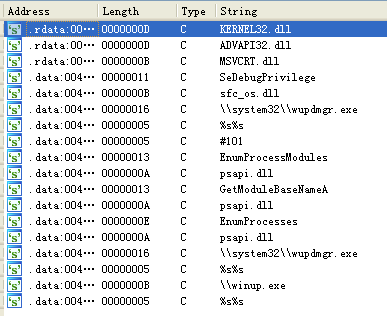
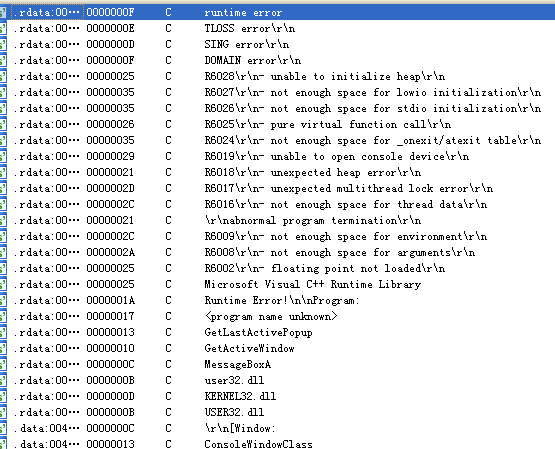
condition:

$find or $load or $size}

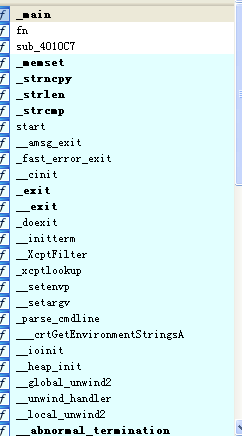
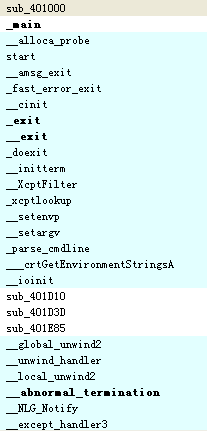


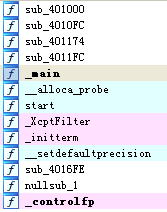






**IDA python**



代码如下：

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**

本次实验，我熟悉了对恶意代码分析工具有了更深入的理解，加深了我对恶意代码的理解和相关知识的掌握。在实验过程中，通过亲手分析恶意代码，我收获了很多，过程是非常快乐的。最后，我对本门课程的实验开始得心应手，做实验的速度越来越快了。

在本次实验当中，我们主要了解到了进程注入，进程注入是一种将恶意代码注入到目标进程中并在其上下文中执行的技术。为了防治进程注入，我们介绍了几种有效的防护方法。其中，代码签名验证是一种验证进程中的代码是否被合法签名的方法，以防止未经授权的代码注入。内存保护机制如写时复制和不可执行内存页面也能有效防止对进程内存的非法修改和注入。此外，限制进程的权限是确保只有具有足够权限的进程才能对其他进程进行注入操作的重要手段。

我们还了解了关于进程替换，APC注入等等的相关技术，这些技术的使用让恶意代码的功能变的会更加丰富，同时我们的防治工作也需要更进一步。作为信息安全专业的学生，更是体会到了未来学习工作道路的艰辛，面对困难迎难而上，砥砺前行。