2024-1-14

北京理工大学计算机学院

计算机病毒防治

实验九 恶意软件隐蔽启动实验报告

目录

[实验准备 3](#_Toc156145900)

[一、 实验环境和所用到的软件 3](#_Toc156145901)

[二、 基本知识 3](#_Toc156145902)

[Lab 12-1 3](#_Toc156145903)

[一、 当运行恶意软件时发生了什么 3](#_Toc156145904)

[二、 哪个进程被注入了 4](#_Toc156145905)

[三、 如何让恶意软件停止弹窗 9](#_Toc156145906)

[四、 这个恶意软件是如何运作的 10](#_Toc156145907)

[Lab 12-2 11](#_Toc156145908)

[一、 这个程序的目的是什么 11](#_Toc156145909)

[二、 启动器程序如何隐蔽执行 12](#_Toc156145910)

[三、 恶意负载存储在哪里 16](#_Toc156145911)

[四、 恶意负载如何被保护 17](#_Toc156145912)

[五、 字符串如何被保护 17](#_Toc156145913)

[Lab 12-3 18](#_Toc156145914)

[一、 这个恶意负载的目的是什么 18](#_Toc156145915)

[二、 恶意负载是如何自我注入的 18](#_Toc156145916)

[三、 这个程序创建了什么文件系统残留物 19](#_Toc156145917)

[Lab 12-4 21](#_Toc156145918)

[一、 0x401000处的代码完成了什么 21](#_Toc156145919)

[二、 哪个进程被注入了代码 23](#_Toc156145920)

[三、 使用LoadLibraryA加载了什么DLL 24](#_Toc156145921)

[四、 传递给CreateRemoteThread调用的第四个参数是什么 24](#_Toc156145922)

[五、 主可执行文件生成了什么恶意软件 25](#_Toc156145923)

[六、 这个和产生的恶意软件的目的是什么 27](#_Toc156145924)

[实验总结 29](#_Toc156145925)

# 实验准备

1. 实验环境和所用到的软件

* Windows 10 x86
* IDA Pro、Resource Hacker、WinHex、Netcat、Procmon等

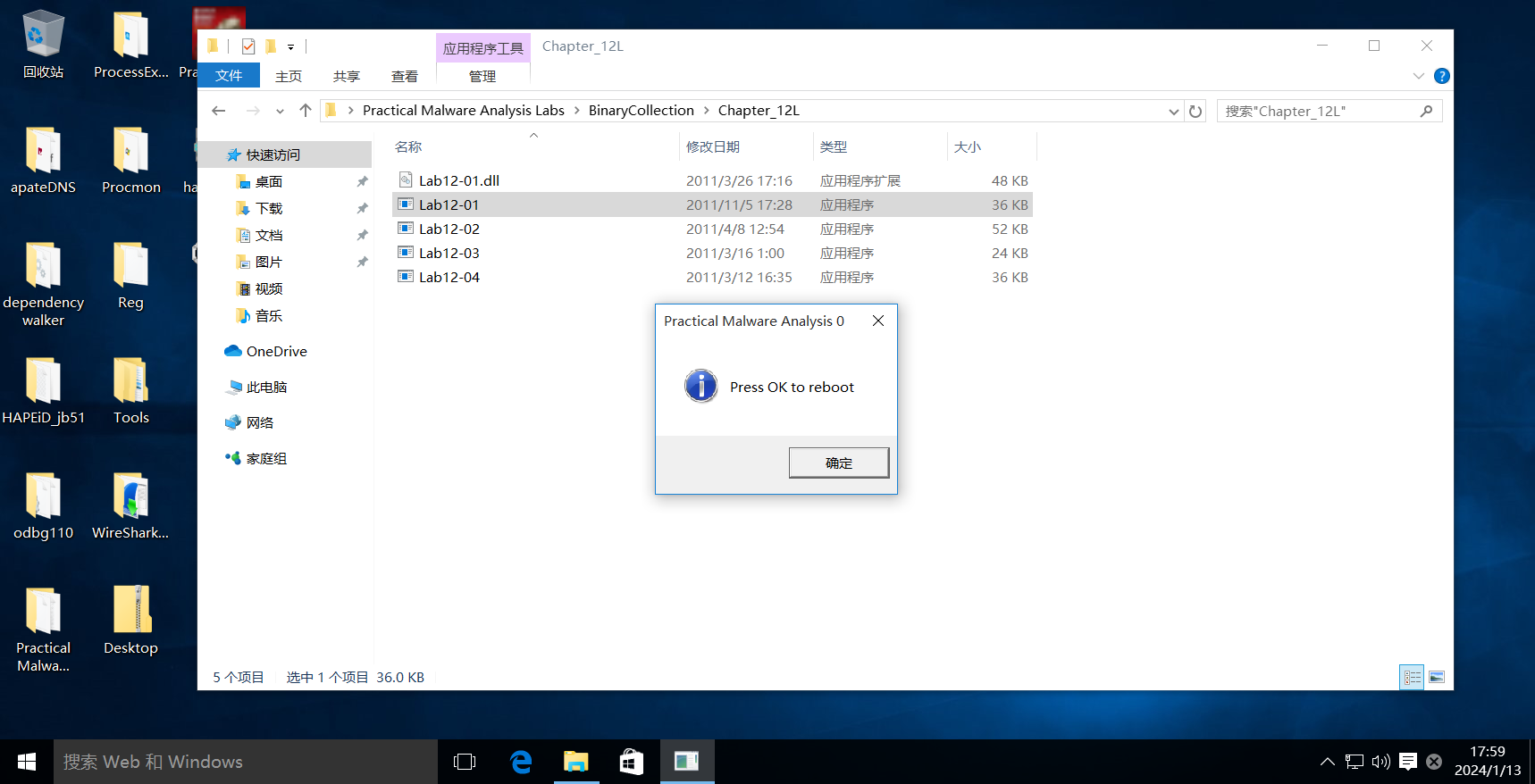
1. 基本知识
2. 常用的隐藏技术：

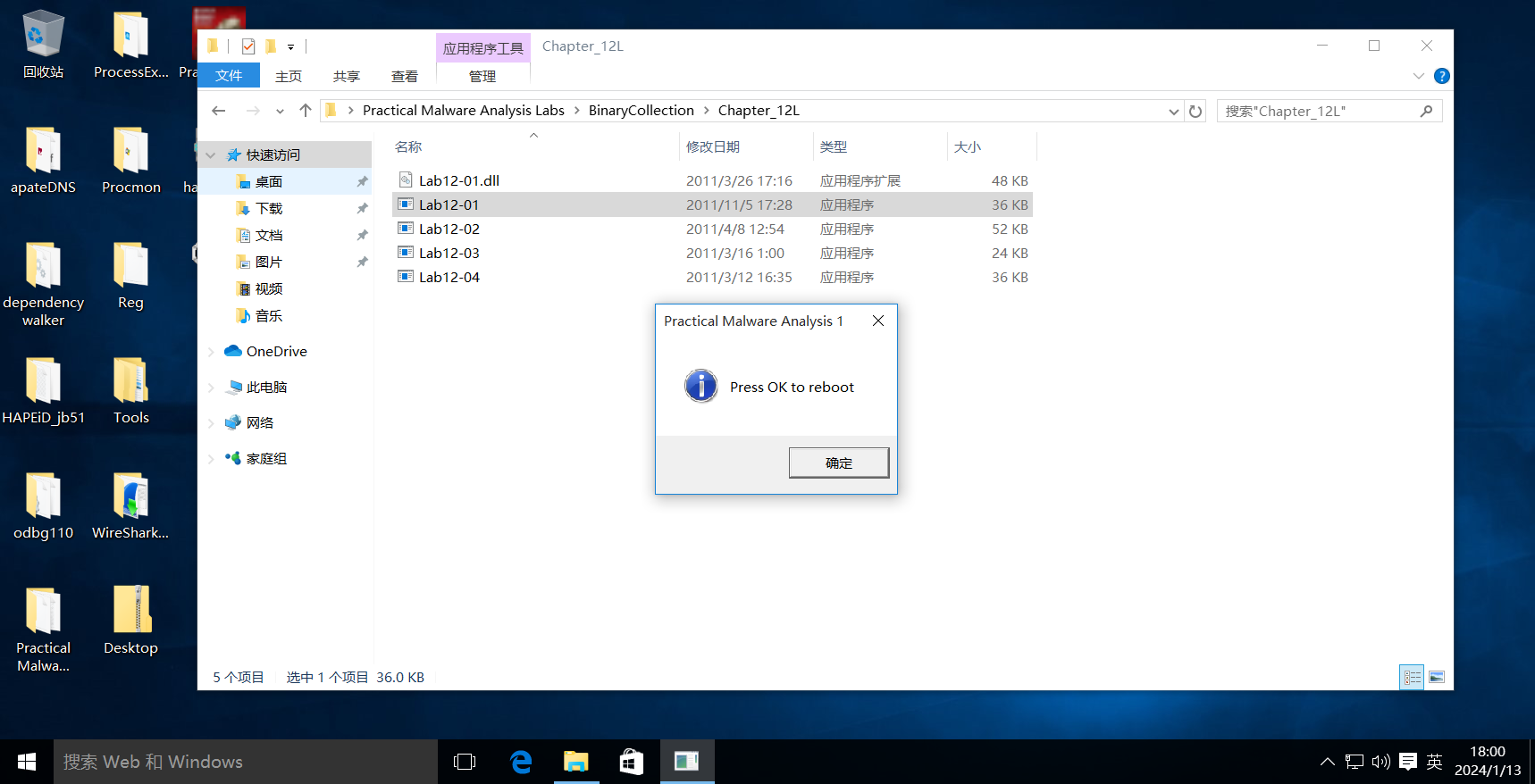
* 启动器
* 进程注入： CreateRemoteThread、WriteProcessMemory、VirtualAllocEx函数暗示该恶意软件使用了进程注入
* 进程替换: UnMapViewOfSection、WriteProcessMemory、VirtualAllocEx、SetThreadContext函数暗示该恶意软件使用了进程替换
* Hook注入：SetWindowsHookExA函数暗示该恶意软件使用了Hook注入
* Detours
* APC注入

# Lab 12-1

1. 当运行恶意软件时发生了什么

在运行了恶意软件后，发现该恶意软件每分钟会在屏幕上进行弹窗。





1. 哪个进程被注入了

接下来，使用IDA Pro打开Lab12-01.exe文件。可以发现在程序开头使用了psapi.dll库来获取其中的EnumProcessModules、GetModuleBaseNameA、EnumProcesses函数地址，并将其分别保存在dword\_408714、dword\_40870C、dword\_408710中，为了方便以后分析，将其重命名为容易识别的名字。

图形用户界面, 文本, 应用程序

描述已自动生成

之后，恶意软件拼接出Lab12-01.dll的地址并存放在Buffer中。并调用EnumProcesses函数来检索系统中每个进程对象的进程标识符PID，并将其存储在dwProcessId所指向的数组中。

图形用户界面, 应用程序

描述已自动生成

之后，恶意程序对PID数组进行循环遍历，并在每次循环中调用sub\_401000函数。

图形用户界面, 文本, 应用程序

描述已自动生成

接下来我们分析sub\_401000函数。在该函数中调用了OpenProcess函数来打开PID所表示的进程，之后调用了EnumProcessModules函数来获取进程的地址。接着调用了GetModuleBaseNameA函数来从PID获取进程名。

图形用户界面, 应用程序, 表格

描述已自动生成

接下来将获取到的进程名和“explorer.exe”进行比较，如果找到就返回1，并在主线程中调用OpenProcess函数去打开这个进程。

图片包含 图形用户界面

描述已自动生成

在打开这个进程后，在main函数中调用了VirtualAllocEx函数在进程explorer.exe中动态分配内存，动态分配的内存地址保存在lpParameter中。之后调用了WriteProcessMemory函数去向分配的内存中写入数据。结合之前的内容，我们可以发现它将Lab12-01.dll的地址写入explorer.exe进程中。

图形用户界面, 应用程序

描述已自动生成

之后调用了CreateRemoteThread函数来创建在另一个进程的虚拟地址空间中运行的线程。在恶意程序中，它用来在explorer.exe中创建线程调用LoadLibraryA函数，其参数为Lab12-01.dll的地址。综上，该恶意软件执行DLL注入，在explorer.exe进程中运行Lab12-01.dll。

图片包含 日程表

描述已自动生成

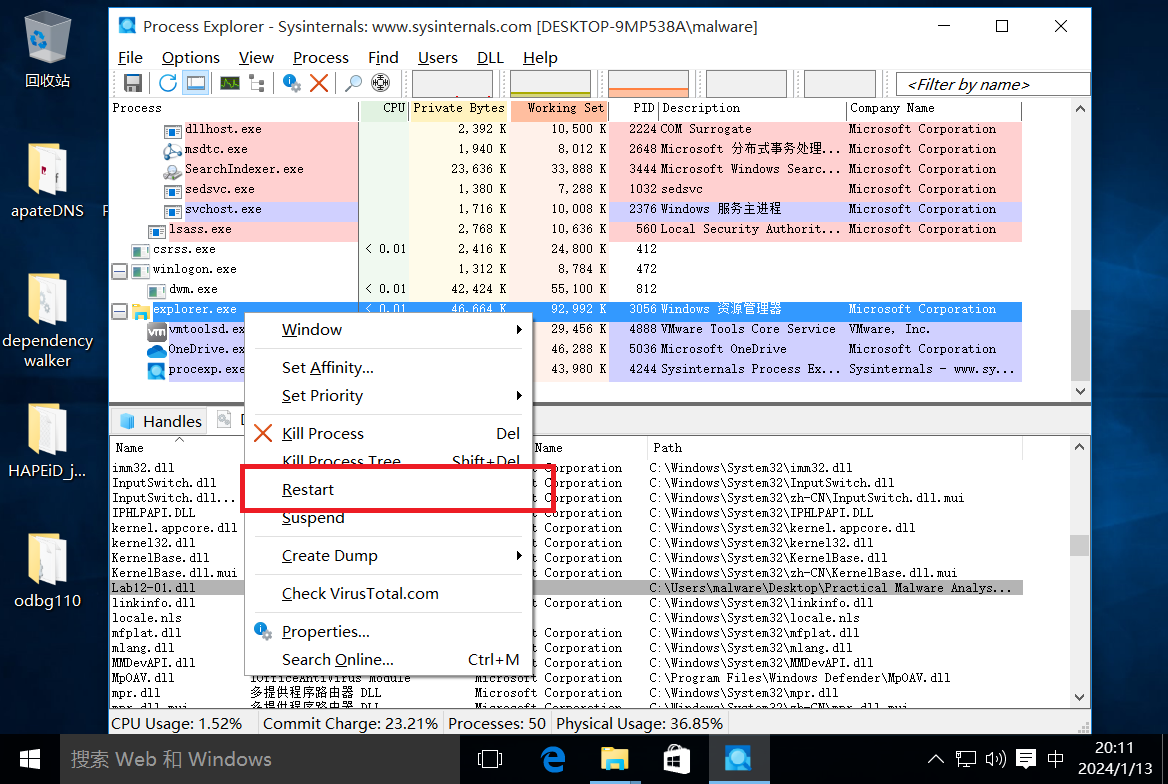
接下来使用Process Explorer来验证我们的想法。选中explorer.exe进程，并选择View->Show Lower Pane和View->Lower Pane View->DLLs，从中可以看到Lab12-01.dll。

图形用户界面

中度可信度描述已自动生成

1. 如何让恶意软件停止弹窗

重启explorer.exe进程，这里可以通过Process Explorer软件来方便完成的重启，具体操作方法如下：选中explorer.exe并右键选中Restart。之后检查explorer.exe进程的DLLs，并发现Lab12-01.dll已经不存在了。



1. 这个恶意软件是如何运作的

接着使用IDA Pro分析Lab12-01.dll。我们发现该dll调用了CreateThread函数来创建一个线程，然后在该线程中每隔60s创建了一个线程。该线程执行的函数为StartAddress。

图片包含 文本

描述已自动生成

StartAddress函数中通过调用MessageBoxA函数来实现弹窗。

图形用户界面, 文本, 应用程序

描述已自动生成

# Lab 12-2

1. 这个程序的目的是什么

通过IDA Pro打开Lab12-02.exe，定位到第一个调用CreateProcessA函数的地方0040115F。

图形用户界面, 文本, 应用程序

描述已自动生成

通过查阅MSDN手册，我们可以知道调用参数dwCreationFlags为4代表CREATE\_SUSPENDED，表示进程创建后不会立即运行，直到调用ResumeThread函数。

文本

描述已自动生成

所以这个程序的目的是隐蔽启动另一个程序。

1. 启动器程序如何隐蔽执行

之后，恶意程序调用了GetThreadContext函数来获取刚才创建进程的主线程的上下文。接着调用了ReadProcessMemory函数将指定地址范围中的数据从指定进程的地址空间复制到当前进程的指定缓冲区中。

图形用户界面, 应用程序

描述已自动生成

接着打开Structures面板，按下Insert键打开添加结构体窗口，然后点击AddStandard Structure窗口来插入CONTEXT结构体。然后将0x004011C3处的0xA4重定义为CONTEXT结构体。

图形用户界面, 文本

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

通常情况下，CONTEXT结构体中的\_Ebx指向新创建进程的PEB（进程环境块），所以ReadProcessMemory函数读取的是PEB的地址加8处。查询后可以得知PEB结构体的偏移8处指向ImageBaseAddress，即加载可执行文件的起始处。之后调用了UnMapViewOfSection函数来该节的映射。

图片包含 应用程序

描述已自动生成

接着调用了VitrualAllocEx函数来分配空间，其基地址为PE文件的ImageBase值，大小为ImageSize，它在PE头中定义。然后调用WriteProcessMemory函数来向分配的地址空间中写入数据，其中数据为PE文件。之后程序进入一个循环中，将PE文件中的每节复制到新创建进程的地址空间。

图形用户界面, 文本, 应用程序

描述已自动生成

接下来，恶意程序调用SetThreadContext函数来设置线程上下文，同时将eax寄存器设置为可执行文件的入口点。之后调用ResumeThread函数来让进程运行。至此，成功实现了进程替换。所以这个程序使用进程替换来实现隐蔽执行。

文本

描述已自动生成

1. 恶意负载存储在哪里

接下来，我们分析之前分析函数之外的内容。首先通过交叉引用来判断什么地方调用了刚才我们分析的函数。可以发现main函数调用了sub\_4010EA函数。而ApplicationName参数是通过调用sub\_40149D函数来生成的，其值为%SystemRoot%\System32\svchost.exe。

文本

中度可信度描述已自动生成

接着，我们查看sub\_4010EA的第二个参数lpBuffer，可以发现它调用了函数sub\_40132C才得到。接下来查看sub\_40132C函数，这个函数调用了FindResource、LoadResource、LockResource、SizeofResource、VirtualAlloc和memcpy等函数。

图示

描述已自动生成

从这里我们得知，恶意负载被存储在了资源节部分。

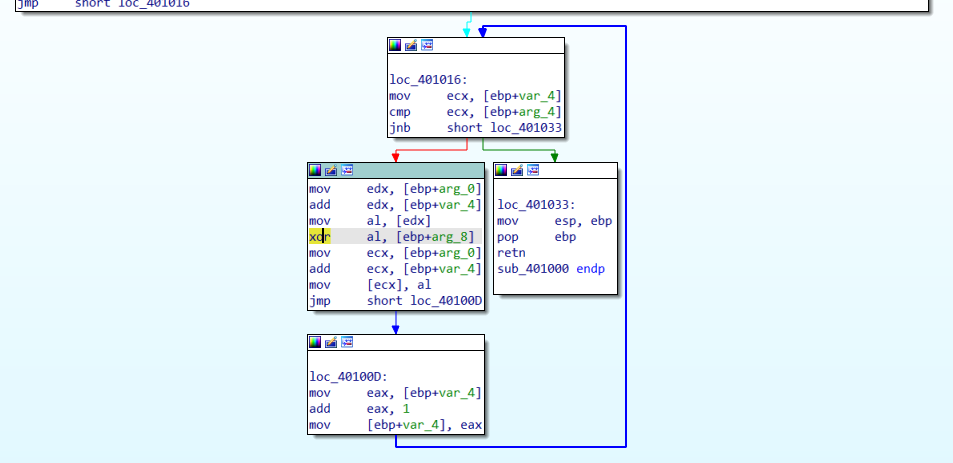
1. 恶意负载如何被保护

在sub\_40132C函数中，我们可以看到在从资源节加载出内容后，又调用了sub\_401000函数，这时可以猜测该函数进行了解密操作。

图形用户界面

中度可信度描述已自动生成

通过分析sub\_401000函数，我们可以发现该函数使用了xor指令，所以该函数是异或解密，每一位都异或’A’的ASCII码。



1. 字符串如何被保护

字符串被包括在资源节中，其加密方法与第四问相同，也通过异或来保护的。

# Lab 12-3

1. 这个恶意负载的目的是什么

这个恶意程序在基础动态分析实验中分析过，当时判断这个程序是一个键盘记录器。而且实际该恶意负载就是上面一个实验解密出来的内容。使用IDA Pro打开该恶意文件。

图形用户界面, 应用程序, Excel

描述已自动生成

1. 恶意负载是如何自我注入的

首先SetWindowsHookExA调用值得我们注意，该函数将应用程序定义的挂钩过程安装到挂钩链中。你将安装挂钩过程来监视系统的某些类型的事件。这些事件与特定线程或与调用线程位于同一桌面中的所有线程相关联。查阅MSDN手册，我们可以得知，该函数的第一个参数idHook为0Dh代表WH\_KEYBOARD\_LL，表示可以监视键盘事件。第二个参数表示指向挂钩过程的指针。

应用程序

低可信度描述已自动生成

所以，该恶意程序通过hook注入来实现恶意程序注入的。

1. 这个程序创建了什么文件系统残留物

接下来分析Hook过程。首先判断按键类型，如果是系统键按下就将键码传给函数sub\_4010C。

图形用户界面, 应用程序

描述已自动生成

接下来分析sub\_4010C7函数。首先打开一个名为practicalmalwareanalysis.log文件。

文本

描述已自动生成

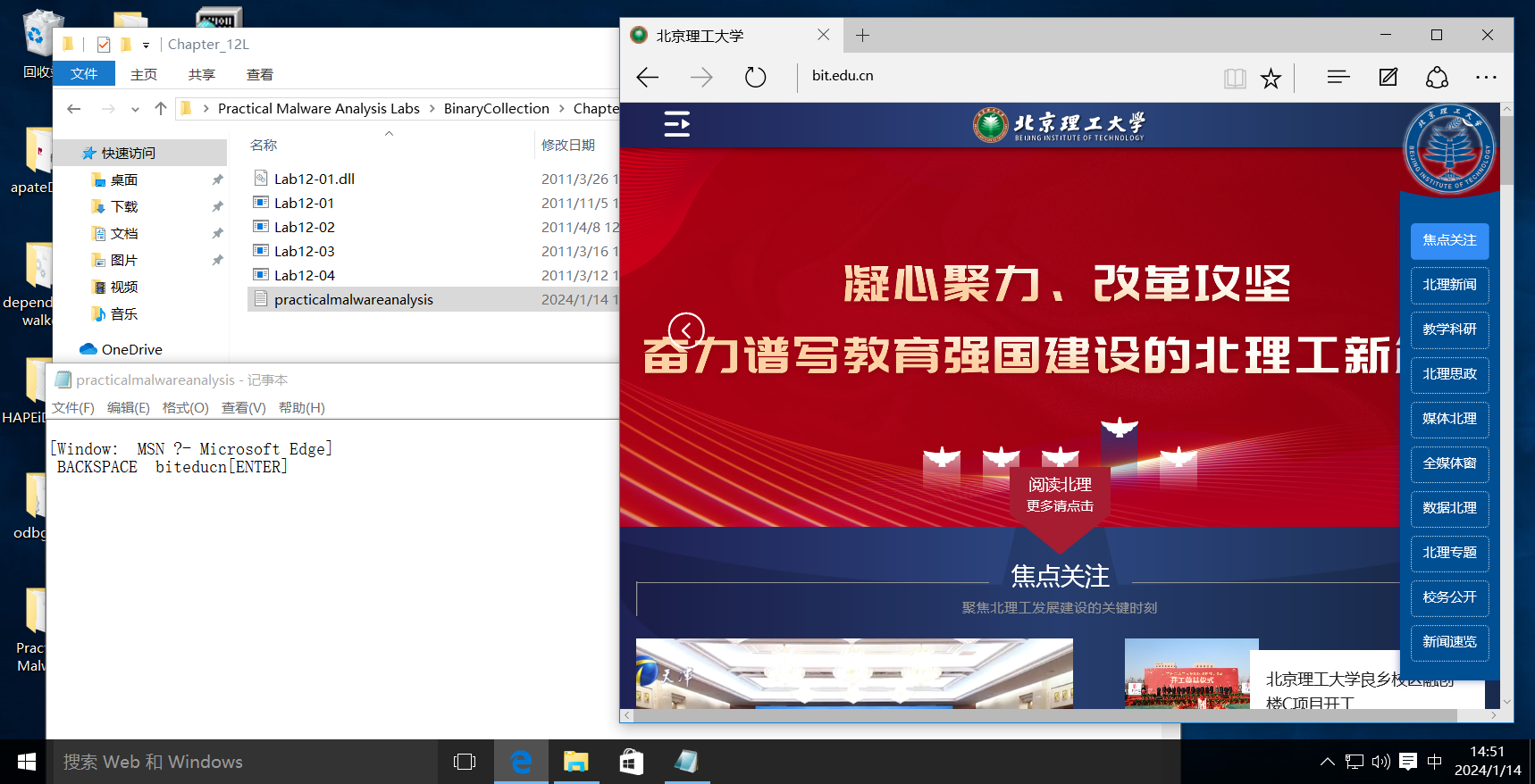
接着函数调用了GetForegoundWindow、GetWindowsText，其中GetForegroundWindow函数获取键盘输入的窗口，然后调用GetWindowsText来将窗口标题写入文件中。

图形用户界面, 文本, 应用程序

中度可信度描述已自动生成

之后还有一个跳转表来实现不同按键的不同处理。

实际运行该软件，结果和我们分析所一致。



综上所述，该恶意程序会创建一个practicalmalwareanalysis.log文件来记录按键信息。

# Lab 12-4

1. 0x401000处的代码完成了什么

使用IDA Pro打开文件，并找到调用sub\_401000函数的地方。像之前的实验一样，该恶意程序首先调用了EnumProcesses函数来获得PID数组。

图形用户界面, 应用程序

描述已自动生成

接下来分析sub\_401000函数。该函数中有两个局部变量String1、String2，其中String1被赋值为”<not real>”，String2被赋值为“winlogon.exe”。

文本

描述已自动生成

之后该函数调用了OpenProcess、EnumProcessModles函数来获取每个进程中模块信息。之后，对每个模块调用GetModuleBaseNameA来获取模块名。之后调用strcmp来比较模块名是否为String2即“winlogon.exe”。

图形用户界面, 应用程序

描述已自动生成

综上，sub\_401000处的代码的作用是判断传入的PID是否为winlogon.exe。

1. 哪个进程被注入了代码

接着上面的分析继续，如果进程为winlogon.exe，该恶意程序就调用sub\_401174函数。在该函数中又调用了CreateRemoteThread函数，从这里我们就可以知道，winlogon.exe进程被注入。

图形用户界面, 文本, 应用程序

描述已自动生成

1. 使用LoadLibraryA加载了什么DLL

在函数sub\_401174中，可以发现函数调用了LoadLibraryA函数，该函数的参数为“sfc\_os.dll”，所以加载了sfc\_os.dll库。

文本

描述已自动生成

1. 传递给CreateRemoteThread调用的第四个参数是什么

检查CreateRemoteThread调用附近的代码，可以发现第四个参数为lpStartAddress，它之前被赋值为sfc\_os.dll中的导出的第二个函数的地址。

图形用户界面, 应用程序

描述已自动生成



经过查阅相关资料，该函数的作用是禁用Windows文件保护直到下一次系统重启。

1. 主可执行文件生成了什么恶意软件

如果上面的线程注入成功，则会接着执行下面的代码。下面代码的主要功能是将wupdmgr.exe文件移动到用户临时文件夹下，并命名为winup.exe。



然后调用了sub\_4011FC函数。该函数中调用了FindResourceA、LoadResource、SizeofResource、CreateFileA、WriteFile。它们将资源节BIN中的数据写入到wupdmgr.exe中。

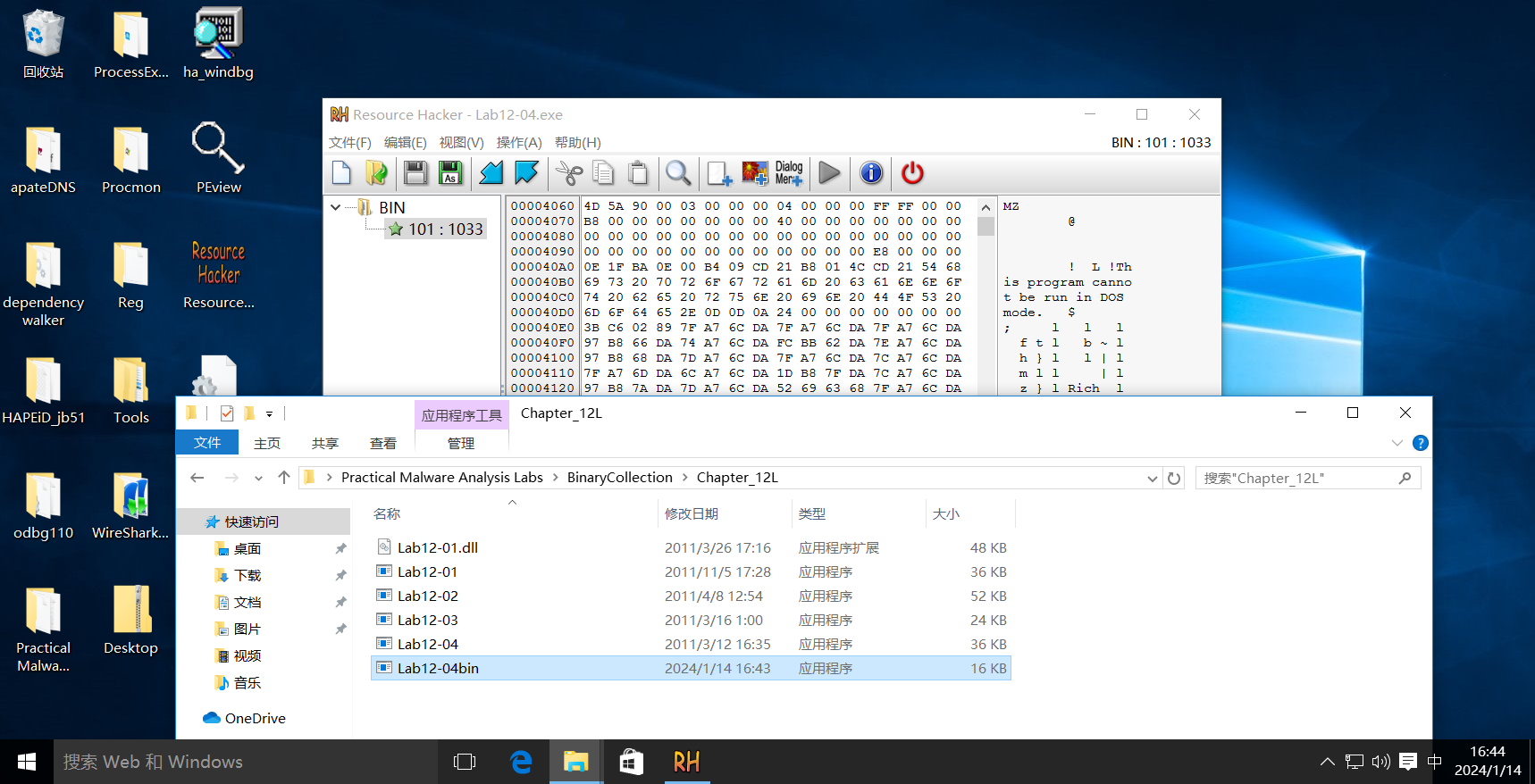
图片包含 图形用户界面

描述已自动生成

综上，该恶意软件将wupdmgr.exe移动到临时文件夹下并命名为winup.exe，之后抽取资源节部分并将其写入wupdmgr.exe。

1. 这个和产生的恶意软件的目的是什么

使用Resource Hacker来将恶意软件的资源节部分提取出来。然后使用IDA Pro打开来进行分析。



电脑萤幕的截图

描述已自动生成

可以看到该恶意软件首先调用WinExec函数来执行之前移动到临时文件夹的winup.exe。这可以使得用户的更新请求得以正常执行。

文本

中度可信度描述已自动生成

接着，恶意软件拼接出字符串“C:\Windows\system32\wupdmgrd.exe”存在局部变量中，注意这里多了一个字母d。

图形用户界面, 文本, 应用程序

描述已自动生成

然后，恶意负载调用了URLDownloadToFileA，该函数从 <http://www.practicalmalwareanalysis.com/updater.exe>下载文件保存到C:\Windows\system32\wupdmgrd.exe，然后恶意软件会运行该文件。

图形用户界面, 文本, 应用程序

描述已自动生成

综上，该恶Lab12-04.exe恶意软件的作用是在winlogon.exe中进行远程线程注入来执行sfc\_os.dll中的2号函数来禁用Windows的文件保护，然后从资源节中抽取出二进制文件来代替wupdmgr.exe。被替代的wupdmgr.exe会从网络上下载恶意软件然后执行。

# 实验总结

本次实验，让我进一步巩固了之前学到的知识，尤其是IDA Pro的使用更加熟悉。同时，本次实验也让我熟悉了常见的一些恶意软件隐藏执行的方式，其中进程注入让我打破了以往各个进程的虚拟地址空间是绝对独立的想法，实际上用户也可以通过系统调用来修改另一个进程的地址空间的内容，从而达到注入执行的目的。