**恶意代码分析与防治技术实验报告**

**Lab12**

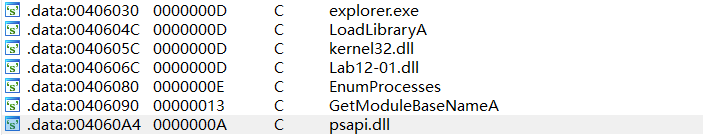
**学号：2013018 姓名：许健 专业：信息安全**

1. **实验内容**

完成课本Lab12的实验内容

1. **问题解答**
2. Lab12-1

**基础静态分析**



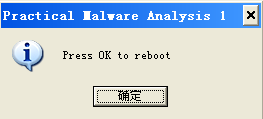
可能是某种形式的进程注入行为，首要目标是判断被注入的代码以及注入的目标进程。

导入函数：CreateRemoteThread、WriteProcessMemory、VirtualAllocEx

可疑字符串：explorer.exe、psapi.dll、Lab12-01.dll

**基础动态分析**

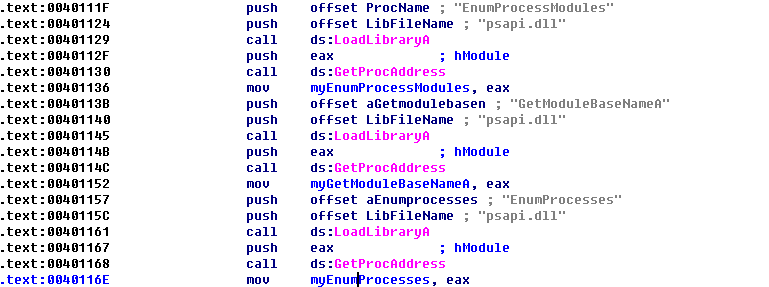
运行该程序，恶意代码每隔一段时间都会弹出窗口，使用Procmon和Process Explorer分析不出有用的信息。



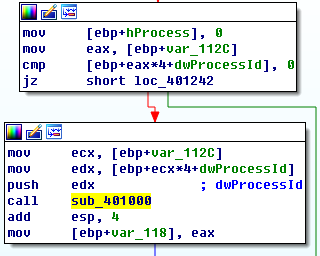
**使用IDAPro高级静态分析**

查看main函数，恶意代码在解析psapi.dll中Windows枚举进程的函数，使用了LoadLibraryA和GetProcAddress手动地解析3个函数。

这个恶意代码保存指向三个函数指针，我们重命名这些全局变量，更容易地标识这些在后续分析中调用的函数。

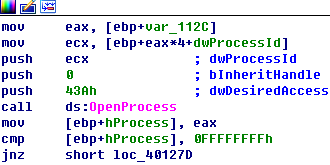


动态解析这些函数之后，这段代码调用了myEnumProcesses，它获取在系统中每一个进程对象的PID。返回一个由局部变量dwProcessId引用的PID数组，dwProcessId被用来在一个循环中迭代进程列表，并对每一个PID调用sub\_401000。



检查sub\_401000函数，可以看到这个动态解析的导入函数EnumProcessModules在PID被传给OpenProcess函数后被调用。接下来是对GetModuleBaseNameA的调用。

这个动态解析的函数GetModuleBaseNameA被用来将PID翻译成进程名。恶意代码将进程名与explorer.exe比较，说明恶意代码在内存中查找explorer.exe进程。一旦找到explorer.exe该函数将返回1，并且这个main函数将调用OpenProcess，来打开一个指向它的句柄。

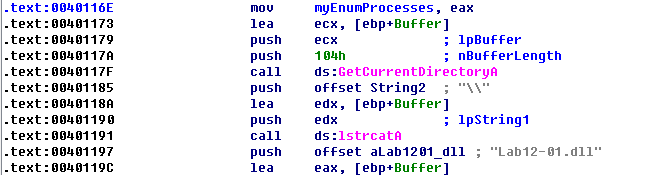


如果恶意代码成功获取这个进程的句柄，以下这段代码将被执行，这个句柄hProcess将被用来操纵这个进程。

调用VirtualAllocEx动态地在explorer.exe中分配内存；0x104字节被压入dwSize而被分配，如果VirtualAllocEx成功，一个指向被分配内存的指针将被移动到lpParameter中，并和进程句柄一起传给WriteProcessMemory，以便往explorer.exe写入数据，Buffer是要写入的内容。

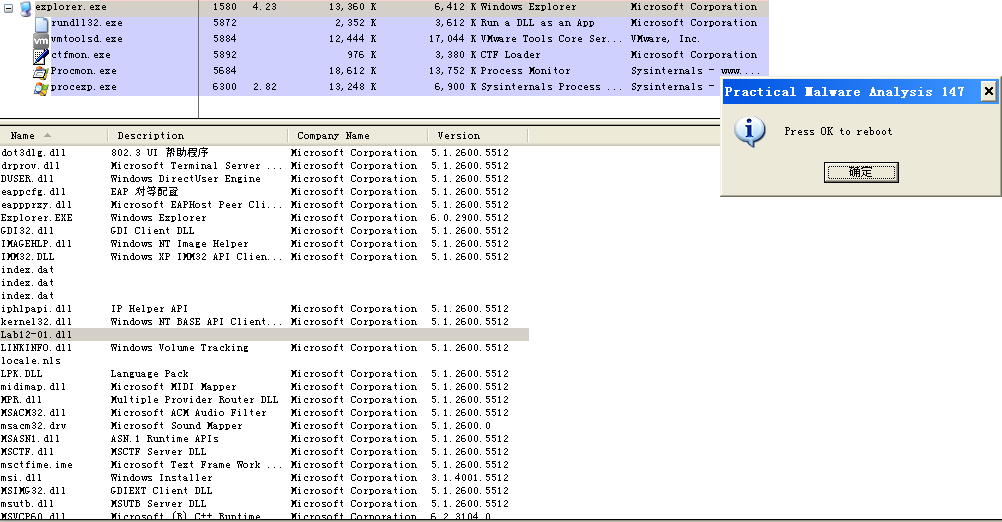


为了理解是什么内容被注入，我们查看Buffer被设置的地方。它被设置成当前目录，并追加了Lab12-02.dll的一个路径，因此恶意代码将Lab12-02.dll的路径写入到explorer.exe的进程中。如果这个恶意代码成功地将这个DLL的路径写入到explorer.exe中，创建远程线程的代码将被执行。

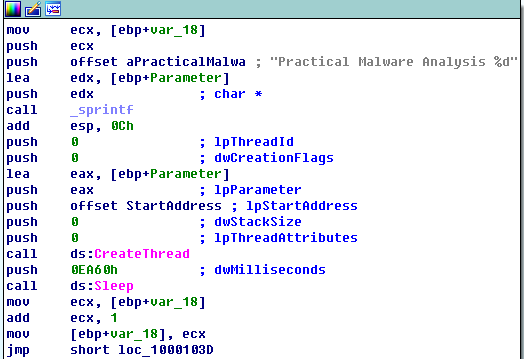




对GetModuleHandleA以及GetProcAddress的调用被用来获取LoadLibraryA的地址。恶意代码中的LoadLibraryA的地址与explorer.exe中的地址是相同的，存储在lpStartAddress，该地址被提供给CreateRemoteThread，从而强制explorer.exe调用LoadLibraryA。LoadLibraryA的参数通过CreateRemoteThread中的lpParameter传递，也就是包含Lab12-01.dll的字符串。



使用Process Explorer查看explorer.exe的进程，可以看到加载的Lab12-02.dll



查看Lab12-01.dll，该dll会一直创建线程，创建消息框并显示“Press OK to reboot”。

1. 在你运行恶意代码可执行文件时，会发生什么？

运行该程序，恶意代码每隔一段时间都会弹出窗口，大概一分钟。

1. 哪个进程会被注入？

explorer.exe

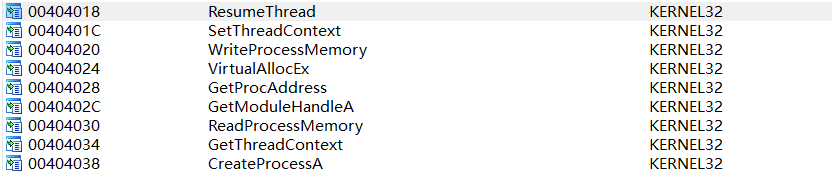
1. 你如何能够让恶意代码停止弹出窗口？

重启explorer.exe

1. 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，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，并通过一个计时器，显示已经过去多少分钟。

1. Lab12-2



恶意代码可能会创建新的进程，并修改进程中的线程执行上下文。

导入函数ReadProcessMemory和WriteProcessMemory告诉我们这个程序对进程内存空间进行了直接读写。



导入函数LockResource和SizeofResource告诉我们这个进程比较重要的数据结构可能保存在哪。

**使用IDA Pro高级静态分析**

查看位于0x0040115F处调用CreateProcessA函数的目的

Push 4指令被IDA Pro标记为参数dwCreationFlags，这是CREATE\_SUSPENDED标志，它允许进程被创建但并不启动。这个进程将不会执行，除非等这个主进程调用API ResumeThread函数，它才会被启动。

程序使用GetThreadContext访问线程上下文，GetThreadContext的hThread参数与传递给CreateProcessA的参数处于同一缓冲区，它告诉我们这个程序正在访问挂起线程的上下文。获取进程句柄十分重要，因为程序将使用这个进程句柄与挂起进程进行交互。



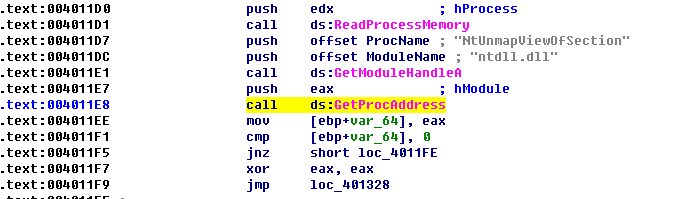
这个进程被用于ReadProcessMemory的调用，使用IDA Pro添加CONTEXT结构体，右键0x004011C3来解析这个结构体的偏移，偏移0xA4实际上通过[ebx+CONTEXT.\_Ebx]来引用这个进程的EBX寄存器。



这个新创建就被挂起的进程EBX寄存器总是包含一个指向进程环境块(PEB)的数据结构，程序以8字节递增结构体，并将这个值压到栈上，作为要读取内存的起始地址。

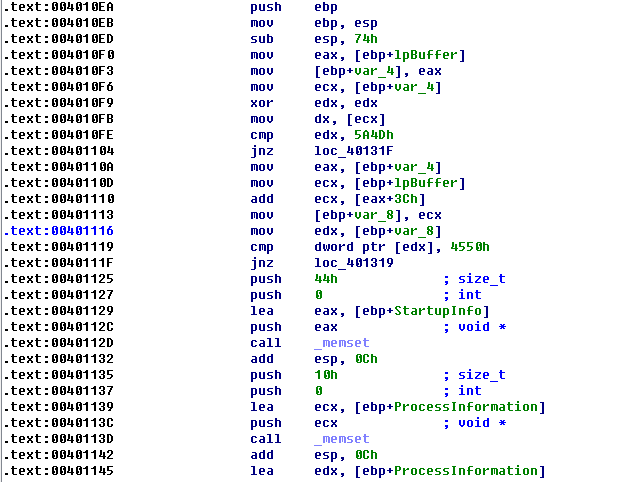
查看官方文档，PEB数据结构的8字节偏移处是一个指向ImageBaseAddress（被加载的可执行文件起始部分）的指针。将这个指针作为读取位置，并读取4个字节，被标记为Buffer的变量将包含被挂起的进程的ImageBase。

程序使用在0x004011E8处的GetProcAddress，手动解析导入函数UnMapViewOfSection，并且在0x004011FE处，ImageBaseAddress作为UnmapViewOfSection的一个参数传入。UnMapViewOfSection的调用从内存中移除这个被挂起进程，此时这个程序将不再执行。

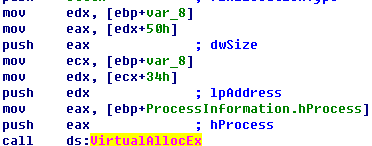


恶意代码调用VirtualAllocEx，在一个被挂起进程中为一个可执行文件分配内存，在这个函数的开头，程序检测在0x004010FE处的魔术值MZ和在0x401119处的魔术值PE。如果这个检查是有效的，我们知道var\_8是一个指针，指向加载到内存的PE头。

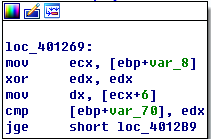




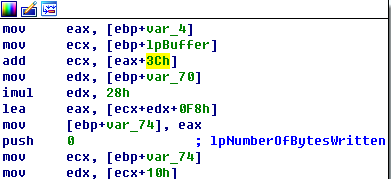
我们可以观察到其将var\_8的值存储在了ecx中，之后将ecx加上了34h，通过查找PE文件结构图发现从基地址加上34h以后就是映像基址的位置，之后将映像基址存入到了edx中，将其作为lpaddress参数传入VirtualAllocEx中：



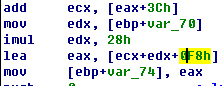
其他传入此函数的参数有一个是加上了50h，查看PE结构图发现基地址偏移50h以后就是内存中影像总尺寸。如果程序调用成功就会向内存中写入数据，将数据写入到被挂起的内存中间去，这个程序的作用就是移动一个PE文件到另一个内存地址空间去。



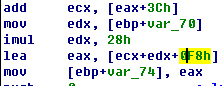
我们可以看到其在ecx的基础上加上了6偏移，放入了dx寄存器中，现在此寄存器中存储的就是区段数，所以这段循环的意义在于复制PE文件的可执行段到挂起进程里面。



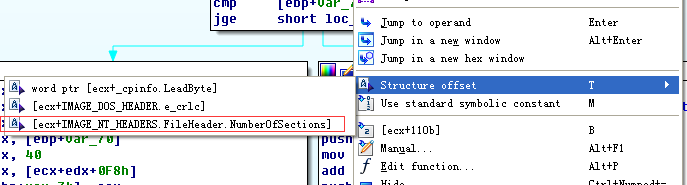
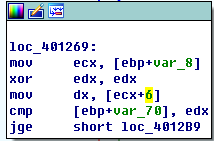
var\_4指向的是PE文件MZ的位置，偏移3C后指向PE标志位的偏移，此位置保存的就是PE文件的偏移位标识，可以根据这个获得PE文件头的位置，此时ecx中保存的就是PE文件头的位置：



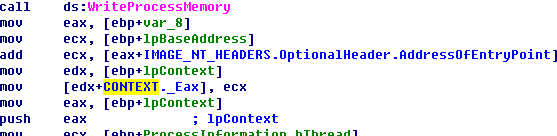
var\_4指向的是PE文件MZ的位置，偏移3C后指向PE标志位的偏移，此位置保存的就是PE文件的偏移位标识，可以根据这个获得PE文件头的位置，此时ecx中保存的就是PE文件头的位置：



由于edx中存储的是0值，也就是说eax中存储的是PE文件头偏移F8的地址也就是区段头的名称，长度为40，将上一部分的edx乘法指令的十六进制转换为十进制为40，也就是说edx会随着循环次数的增加意40的倍数增加，所以此模块的作用就是不断访问imageSectionHeader的数据结构。

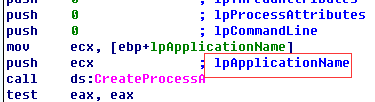


此函数可以修改eax寄存器中的数据，并可以eax中的值设置为可执行文件的加载入口点。将上方的十六进制偏移进行转化如下：

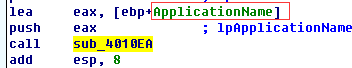


可以清晰的观察到其是将程序执行的入口点放置在了eax中。

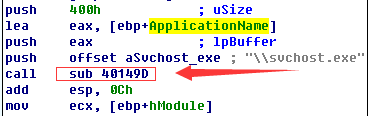
下面调用了ResumeThread函数，此函数调用成功以后就证明成功的将之前createThread函数创建的进程替换为了另一个进程A。现在我们需要确定进程A是什么。



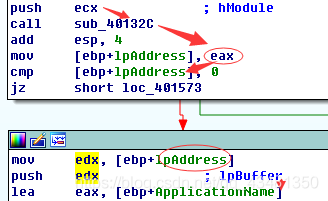
此时需要通过圈出的参数获知进程，现在我们回到这个函数最开始的地方，鼠标放置在内存地址上后ctrl+X查看其交叉引用，点击第二个main函数的位置



可以看到圈出来的部分就是决定进程名称的偏移，向上查找所有带有此名称的指令：

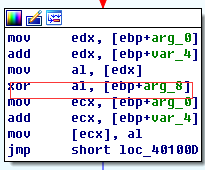
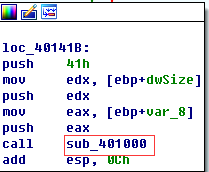


可以看到此处是将svchost.exe文件作为参数传递给了sub\_40149D函数，点击进入此函数，发现其调用了获取系统路径函数并且调用了strcat字符串连接函数，所以此函数的目的就是构造svchost.exe路径。也就是说有进程替换了svchost.exe。向下分析查看是哪个进程替换了此程序。



可以看到进程名称一路向上关联，其中最上面的ecx存储的是svchost.exe的进程句柄，下面我们进入sub\_40132C这个函数内部，发现其调用了很多有关于资源的API函数，那么就可以说明PE文件的资源节里面一定包含了重要的数据，这个函数的目的就是将这些重要的数据拷贝到svchost.exe程序里面。

下面可以采用Resource Hacker查看一下资源中的内容。将文件拖入到软件中以后，Action->Save Resource to a BIN File,将其保存为exe文件。



发现其有异或操作，异或的对象是arg\_8，此参数是此函数的此三个参数，回到上一层查看第三个参数是41h。下面我们进行解密，使用WinHex软件将从资源节中提取出的exe文件拖入，Edit->Modify Data->XOR 41，点击确定就可以解析出原有文本了。

1. 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。

1. 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。

1. 恶意代码的负载存储在哪里？、

这个恶意代码的有效载荷(payload)被保护在这个程序的资源节，这个资源节的类型是UNICODE，且名字是LOCALIZATION。

1. 恶意负载是如何被保护的？

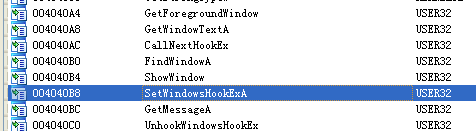
保存在这个程序资源节中的恶意有效载荷是经过XOR编码过的。这个解码例程可以在sub\_40132c处找到，而XOR字节在0x0040141B处可以找到。

1. 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。

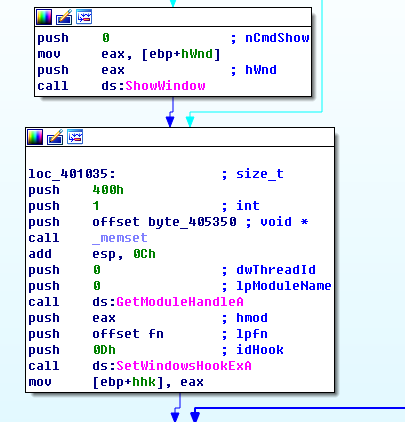
1. Lab12-3

查看导入函数



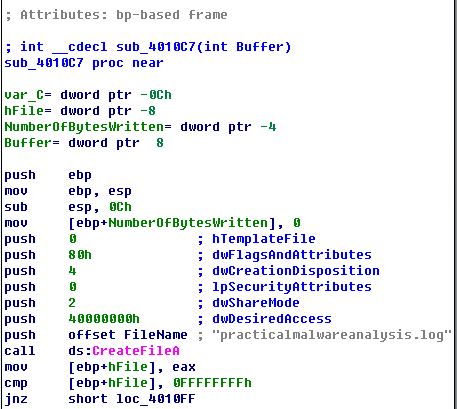
其调用了系统设置钩子函数，这个函数允许进程挂钩，也允许监控进程的API函数。

SetWindowsHookExA函数创建一个钩子，其中参数idHook是消息事件的类型 ，lpfn是hook的函数地址。0Dh-->WH\_KEYBOARD\_LL-->监控键盘的消息，fn-->Hook函数的地址，对击键的消息做了手脚。



根据函数调用序列可以看出应该是一个键盘记录器 ，并记录在了一个文件中

var\_C是这个函数的唯一一个参数，就是一个虚拟按键码。先进去byte\_40148D的数组，将索引到的值为cl。 再乘以4，再去索引。最后决定跳转的位置。



1. 这个恶意负载的目的是什么？

是一个击键记录器。

1. 恶意负载是如何注入自身的？

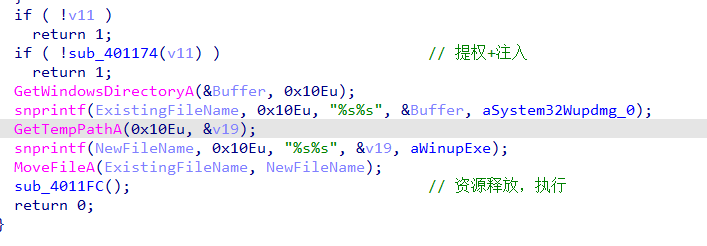
使用挂钩注入，偷取击键记录。

1. 这个程序还创建了哪些其他文件？

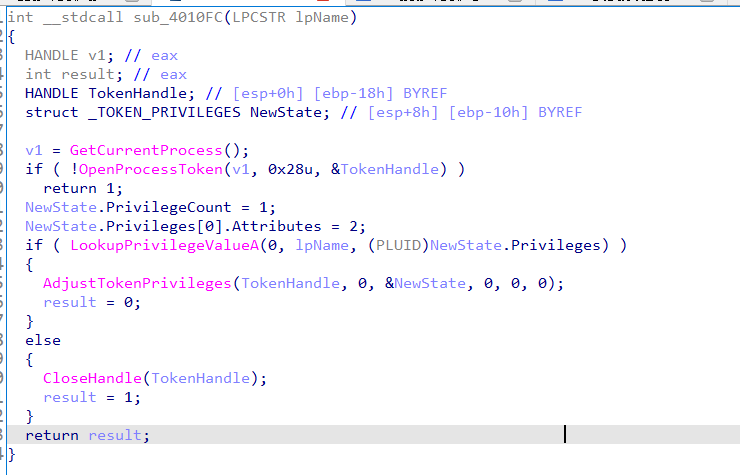
这个程序创建文件praticalmalwareanalysis.log，来保存击键记录。

1. Lab12-4

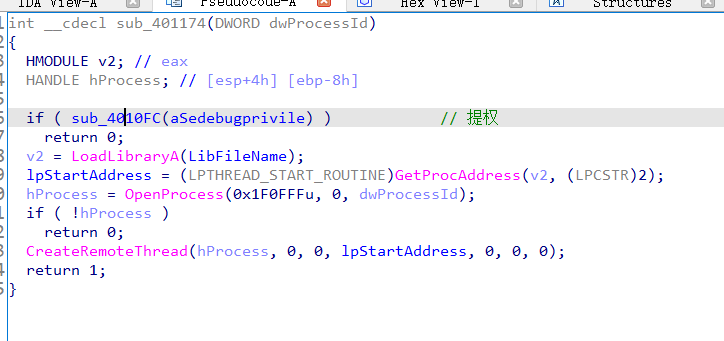
Lab12-4.exe前部分与Lab12-1是一致的，此处不再进行分析在401000中是winlogon.exe，也就是前面这部分是获取winlogon.exe的PID。



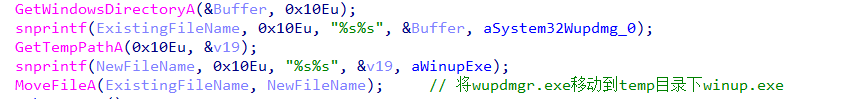
首先是进行了提权，判断是否有SeDebugPrivilege权限，如果没有的话会进行提权。



然后会进行远程线程注入，注入函数是sfc\_os.dll中的序号为2的函数

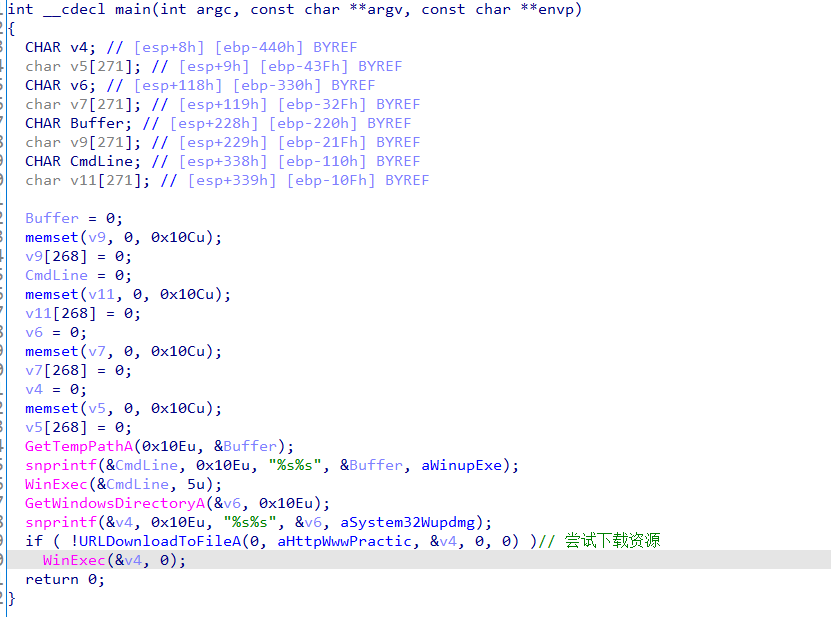


然后会移动文件



最后是一个资源释放，释放到C:\Windows\system32\wupdmgr.exe，并且隐藏窗口启动它。

把资源节中的程序dump下来分析，首先看到启动了winup.exe也就是原始的wupdmgr.exe，然后会访问http://www.practicalmalwareanalysis.com/updater.exe ，进行下载更新恶意代码。



1. 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。

1. 代码注入了哪个进程？

winlogon.exe是被注入的进程

1. 使用LoadLibraryA装载了哪个DLL程序？

DLL sfc\_os.dll用来禁止Windows的文件保护机制。

1. 传递给CreateRemoteThread调用的第4个参数是什么？

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。

1. 二进制主程序释放出了哪个恶意代码？

wupdmgr.exe，原始的wupdmgr.exe被放到了temp目录下，重命名为了winup.exe

1. 释放出恶意代码的目的是什么？

会从指定的域名中下载一些资源，来进行更新，而windows本身的wupdmgr.exe被转移到temp目录下，在持久化驻留的同时，也不影响原程序的运行。