**恶意代码分析与防治技术实验报告**

**Lab12**

**学号：6016252 姓名：马世骐 专业：**

**一、实验环境**

Windows7(VMWARE虚拟机)，Windows11本机

**二、实验工具**

STRINGS, IDAPro，PEVIEW，EXEInfo，YARA，IDA Python，Netcat，Wireshark，Procmon,WinDbg

**三、实验内容**

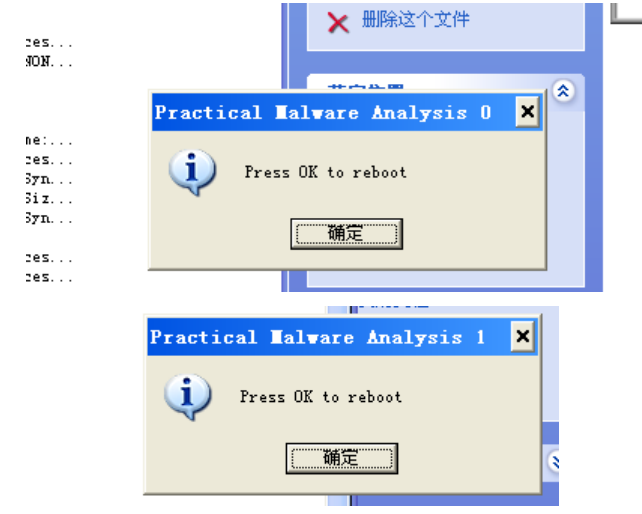
**LAB12-1**

**问题1**

|  |
| --- |
| 在你运行恶意代码可执行文件时，会发生什么？ |

**答：每分钟会弹出一个窗口。**

分析过程：



同时我们可以看见这个提示框上面的数字在每次提示的时候都会增加

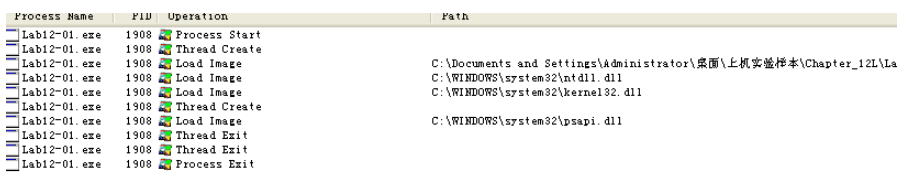
**问题2**

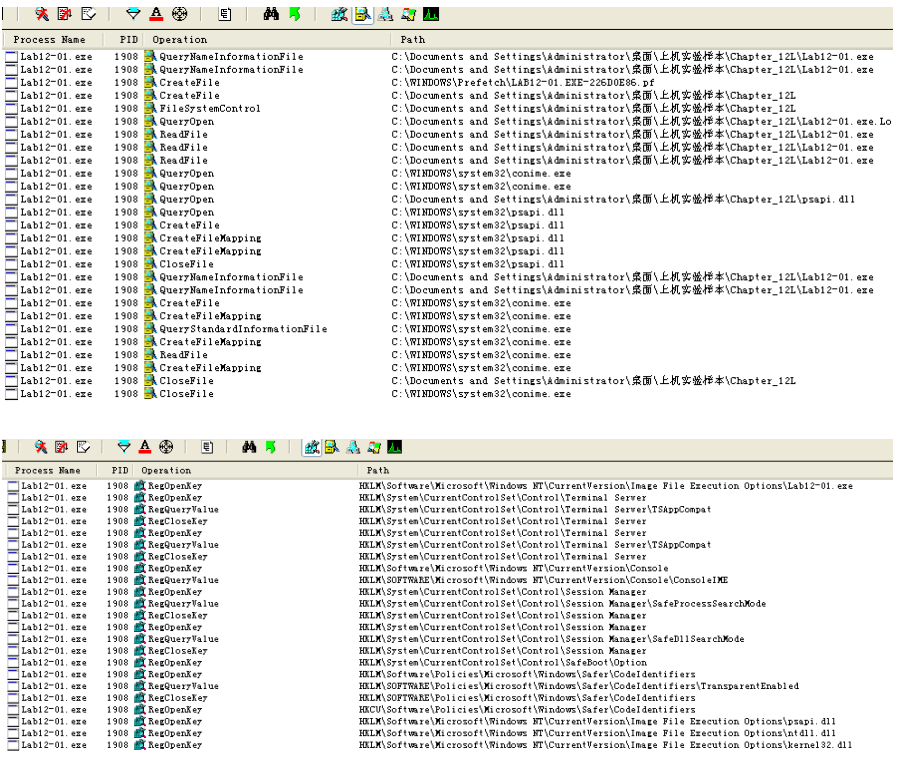
|  |
| --- |
| 哪个进程会被注入？ |

**答： explorer.exe。**

分析过程：

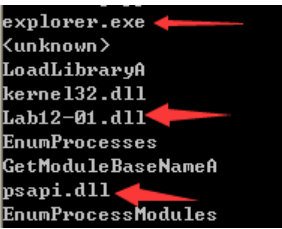
使用Procmon查看行为

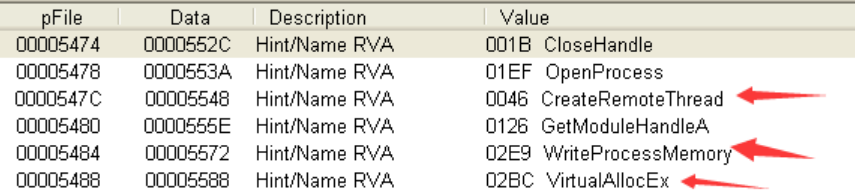




在procmon中没有看见什么有用的信息

对这两个文件进行基础静态分析。Lab12-01.exe有如下字符串和导入函数，从CreateRemoteThread、 WriteProcessMemory和VitualAllocEx中推测出该恶意代码可能有注入行为，被注入的可能是字符串中看到 的explorer.exe。





Lab12-01.dll有如下字符串和导入函数，MessageBoxA说明可能有关于窗口的操作



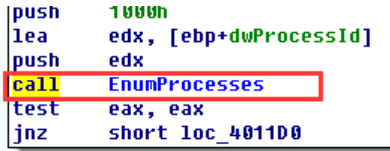
接下来使用IDA进行分析



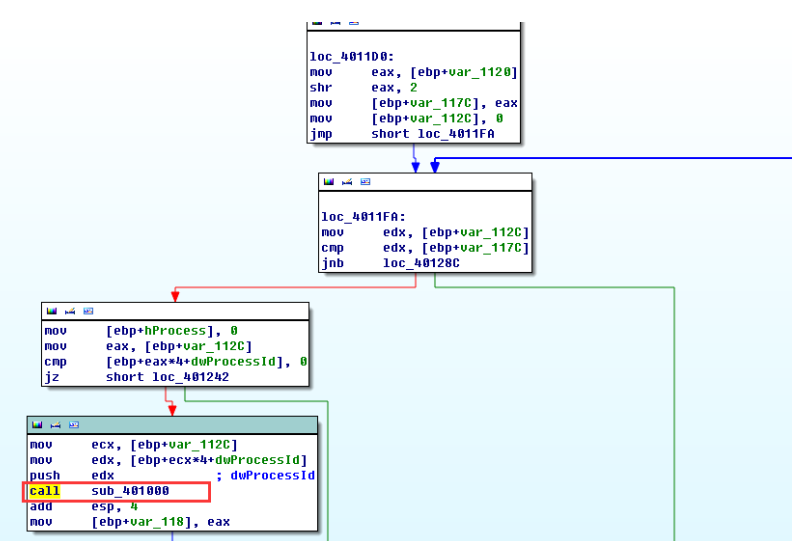
在main函数中我们可以看见连续三次导入了 psapi.dll 中的函数，根据导入的函数名可以想到这几个 函数的功能是进行进程的枚举。为了方便后面的观察，我们将这三个地址上的函数进行重命名。 重命名结束后为：



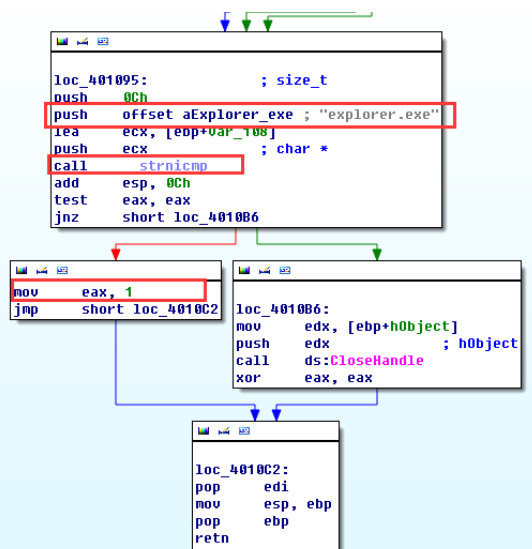
然后可以看见在下面先调用了刚刚导入的枚举函数



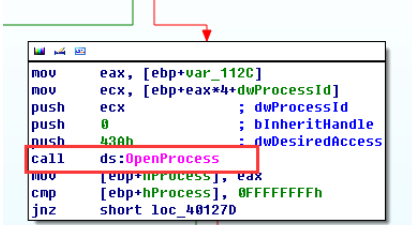
在调用完枚举函数以后我们可以看见一个循环



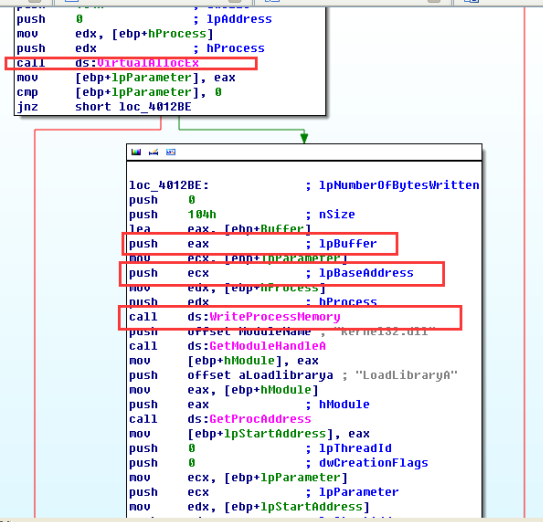
这个循环的作用就是对刚刚枚举的进程的PID进行分析，可以看见这里调用了一个sub\_40100函数，进入查看



可以看见这个函数和 explorer.exe 这个字符串进行比较，如果是，就返回1。换言之，这个函数的功能就是寻找explorer.exe这个进程。



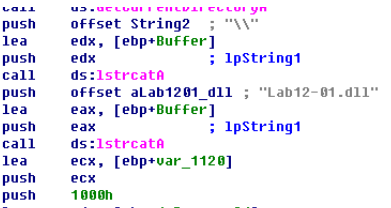
在找到这个进程以后，会调用 OpenProcess 函数



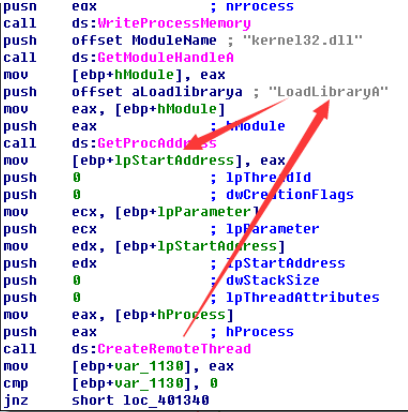
然后恶意代码在进程中分配了一块内存空间，并向其中写入了ecx的内容，根据上面的mov操作不难看 出，这里写入的是栈上的内容。为了更加直观，我们使用OD来查看这里的参数具体是什么值



发现ECX里放的是kernel32字样，结合之前存在的操作



可以确定这里就是存放的指向Lab12-02.dll的字符串，也就是说，这里将这个dll文件写入到了 explorer.exe中，也就是explorer.exe这个进程被注入。



之后我们可以看见这个恶意代码创建了一个线程，并且线程的起始地址是LoadLibraryA，之后 LoadLibraryA的参数是之前的Lab12-01.dll，也就是说，这里会强制将恶意文件加载到内存中去

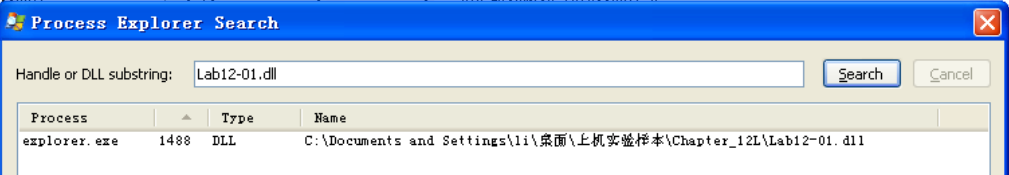
**问题3**

|  |
| --- |
| 你如何能够让恶意代码停止弹出窗口？ |

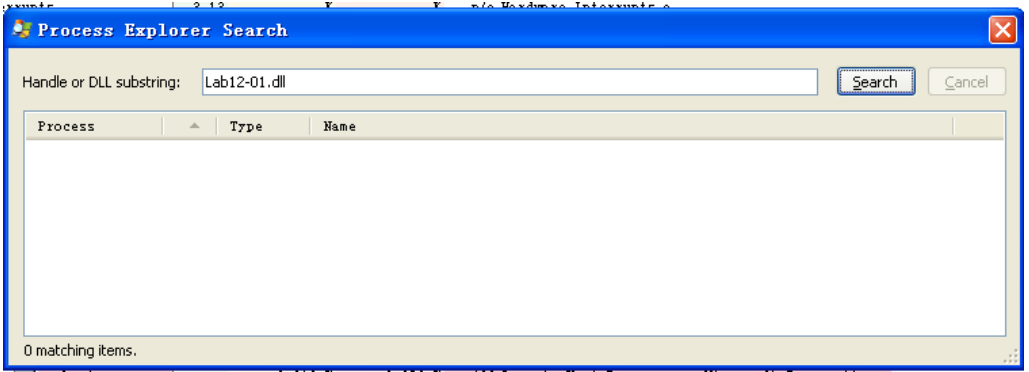
**答： 通过之前Procmon中监视的行为我们没有发现他对注册表有什么操作，也没有执行Hook等操作，那么 也就是说这只是一次性的注入，只需要能关闭这个进程就可以停止弹出。所以我们可以通过关闭 explorer.exe这个进程然后重新手动运行这个进程的方式停止恶意代码的行为；或者是直接重启电脑。**

分析过程：

在Process Explorer中查找Lab12-01.dll，可以看到它确实被注入到了explorer.exe中。为了停止这些弹 窗，我们可以先Kill Process，然后再用File->Run重新运行它即可。



可以看到此时不再有窗口弹出，也查找不到Lab12-01.dll了。



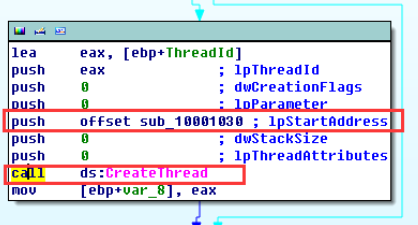
**问题4**

|  |
| --- |
| 这个恶意代码样本是如何工作的？ |

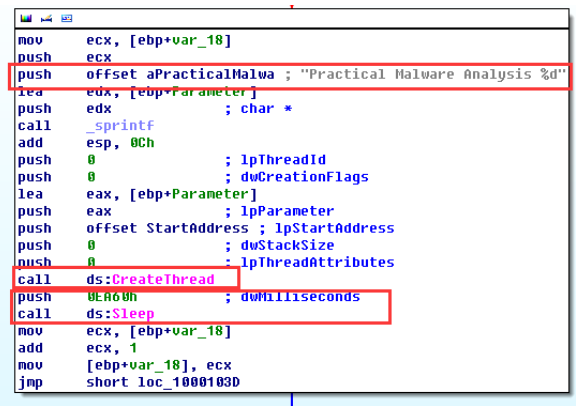
**答： 这个恶意代码执行DLL注入，强制explorer.exe加载Lab12-01.dll，该恶意dll文件每分钟会弹出一个消息框，并通过一个计数器来显示已经过去了多少分钟。**

分析过程：

经过分析我们可以知道这个exe程序只是实现注入，真正的功能是在dll文件中，所以接下来我们分析一下这个dll文件



可以看见dllMain中上来就是使用创建一个线程，并且线程中运行的函数是sub\_10001030。



进来以后发现这个函数就是一个死循环，循环体中就是创建线程，然后睡眠一分钟。结合上面的字符串和之前运行时发现的恶意样本的行为我们不难推断出，这个行为就是创建线程并显示相关内容。

**LAB12-2**

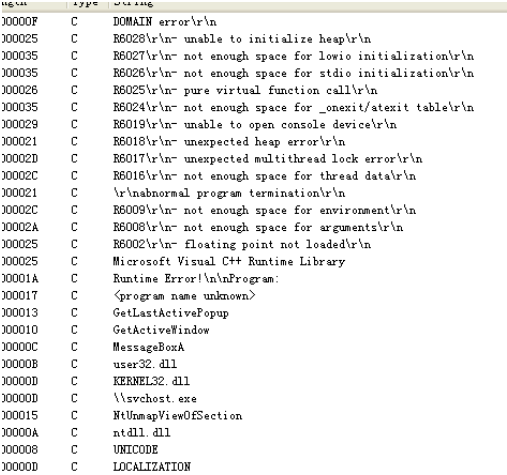
**问题1**

|  |
| --- |
| 这个程序的目的是什么 |

**答：这个程序的目的就是将系统目录下的svchost.exe进行进程替换，从而秘密的执行另外一个带有恶意行为的代码。**

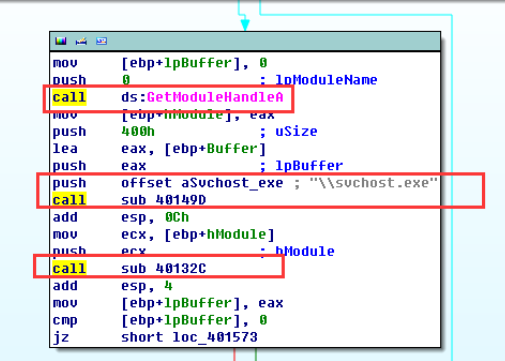
分析过程：

首先使用IDA查看一下程序，在分析之前，我们先查看一下有没有什么比较值得注意的字符串

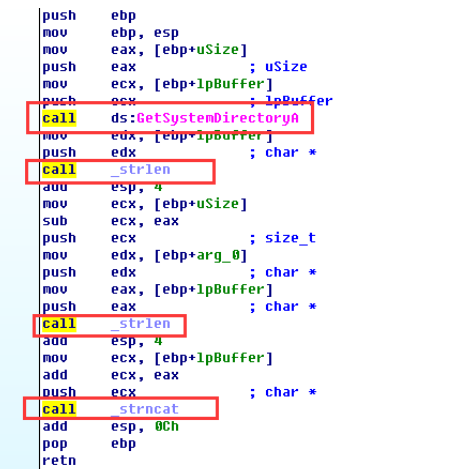


可以看见有一个 svchost.exe ，这个程序似曾相识，之前有的样本也创建了这个文件。其他就没有什么能引起注意的字符串了。

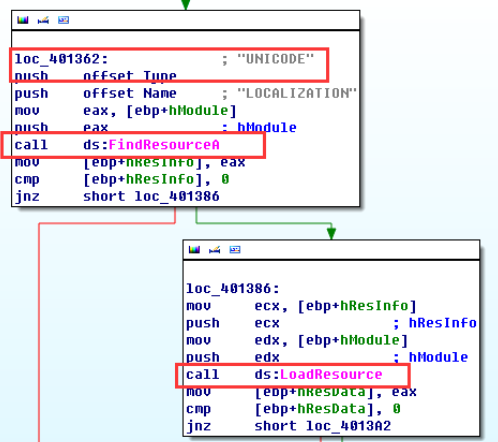
之后我们来分析一下相关的代码

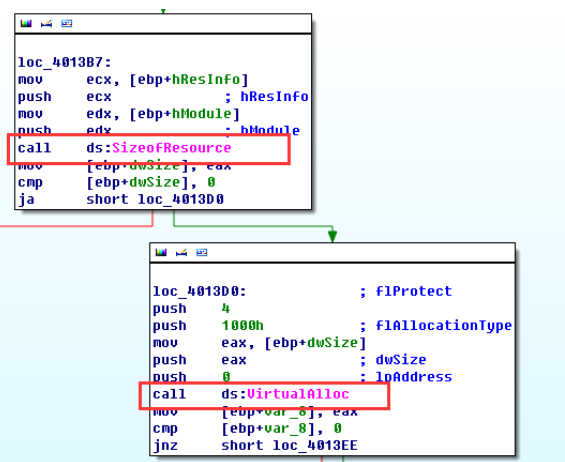


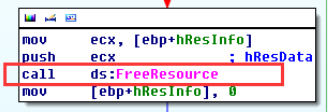
上来我们可以看见这个恶意样本先获得了句柄，然后对我们之前关注的exe进行了一定操作，之后又调用了另一个函数。我们先查看一下他对这个exe做了什么操作。



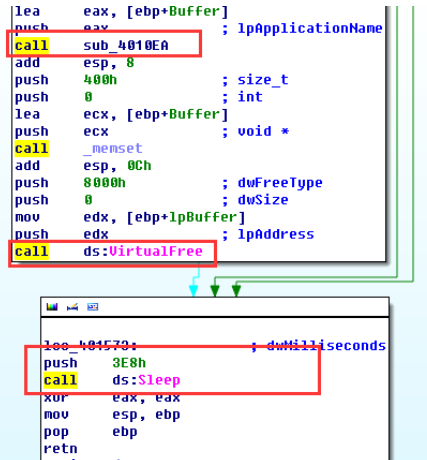
进来以后发现他是获取了系统目录，然后进行了字符串的拼接操作，也就是拼接出了这个程序的绝对路径。 进入到下面的调用的函数



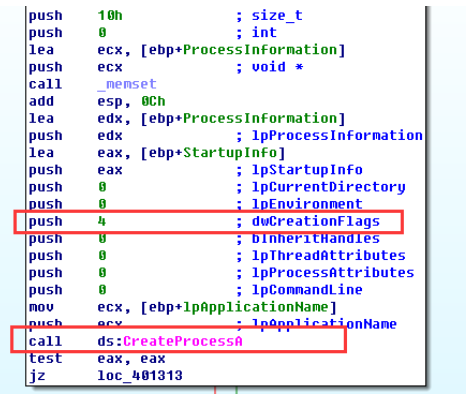




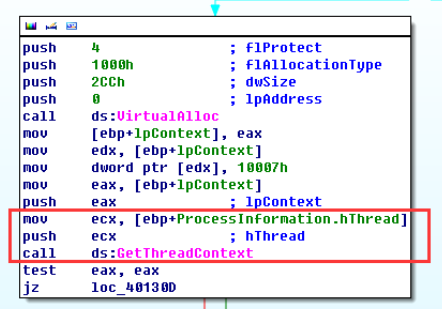
我们发现了一系列的操作：寻找资源、加载资源、计算大小、分配空间、释放资源，很明显这里就是将资源节里的东西加载到内存中去，并将其释放。



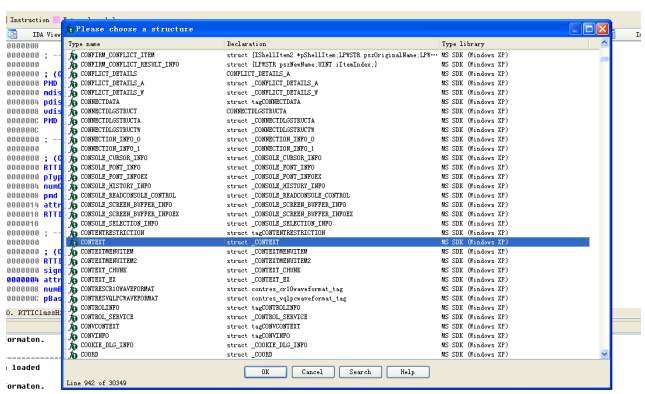
在释放完资源后调用了一个函数，这个程序睡眠一段时间就结束了。进入sub\_4010EA函数，可以看见



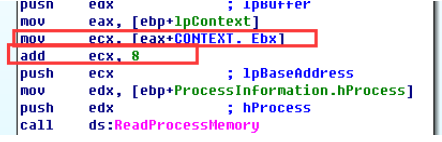
这个函数先创建了一个进程，并且上面的一个参数是4，表示这里进程会被创建，但是暂时不会执行。只有当后面进行调用的时候才会被启动。



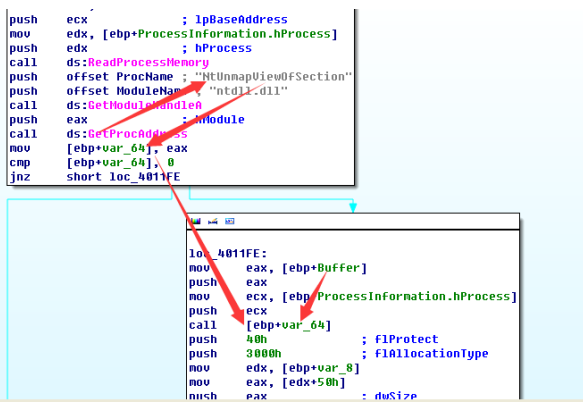
之后这个恶意代码获取了 ProcessInformation 这个线程的上下文执行环境，经过分析以后我们可以发 现，这里访问的其实就是刚刚被创建后挂起的进程。 为了能更加直观清晰的知道这个样本对上下文环境做了什么，我们在IDA中添加一个Context结构体进行观察



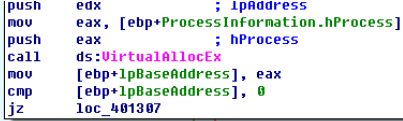
之后进行转换



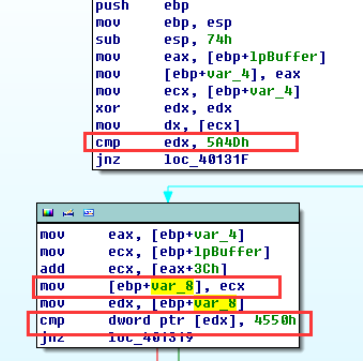
我们可以发现这里获取的是Ebx寄存器。经过查询以后我们可以知道，这个ebx的8字节偏移地址是 ImageBaseAddress，也就是被加载的可执行文件的起始部分的指针。 接下来我们可以看到一个调用的链



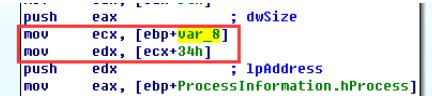
GetProcAddress用来获取上面的NtUnmapViewOfSection的地址，然后将其保存在[ebp+var\_64]中， 之后Buffer中的内容作为参数被这个函数调用。这个函数的功能就是将创建的进程中的内存清空，之后就能对这块内存进行一个注入。



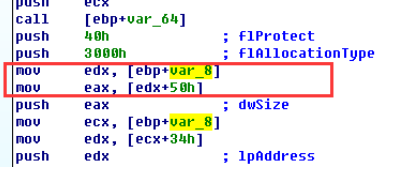
并且之后我们也可以看见他分配了一块空间。 在之前的代码中有一段是这样的：



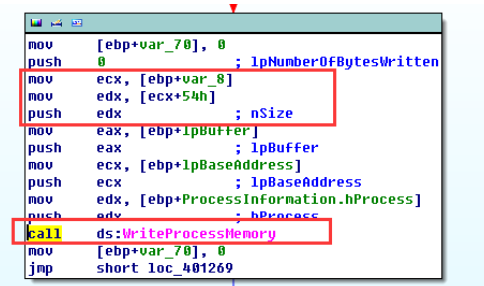
这里比较了开头的几个字节，去查看是否是PE文件格式。如果是就会将其赋到var\_8的位置上。



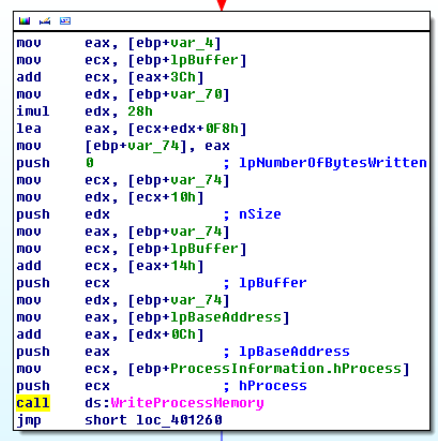
之后可以看见进行了一个地址的偏移，偏移量是34h，经过查询PE文件头的格式我们可以知道这里就是ImageBase



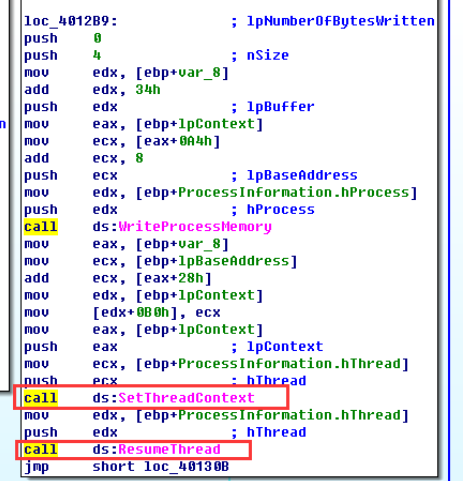
这里是获得了内存中映像的总尺寸。 再结合刚刚说的分配空间，就不难想到这里就是在映像基址分配对应大小的空间了。



之后样本向内存空间中写入了偏移地址为54h的内容，经过查询这里就是写入了PE文件头部的总大小。 那么之前两步的操作我们就不难联想到这之后的行为就是将一个PE文件写入到内存当中去



之后的循环就是不断的遍历PE文件的节表进行写入。



加载之后这里使用了一个SetThreadContext函数，这个函数可以设置eax寄存器，并且把他加载到被挂 起的进程内存空间中的入口点位置。最后调用 ResumeThread 函数，将这个挂起的线程调用。结合之前 说的将进程的空间进行了清空操作，然后将这个PE文件进行写入，那么也就是进行了进程替换的操作。 再想到之前我们分析时看见有一个字符串拼接操作，然后获取到了系统目录下的svchost.exe的绝对路 径，那么这里就是对这个进程进行替换了，并且替换的是恶意样本资源节中的内容。 所以，这个程序的目的就是将系统目录下的svchost.exe进行进程替换，从而秘密的执行另外一个带有恶意行为的代码。

**问题2**

|  |
| --- |
| 启动器恶意代码是如何隐蔽执行的？ |

**答： 经过之前的分析可以得知，使用进程替换，将svchost.exe替换为Lab12-02.exe，来隐蔽执行本程序。**

**问题3**

|  |
| --- |
| 恶意代码的负载存储在哪里？ |

**答： 根据上面的分析我们知道，该恶意代码的负载存储在这个程序类型是UNICODE，名字是LOCALIZATION的资源节中。**

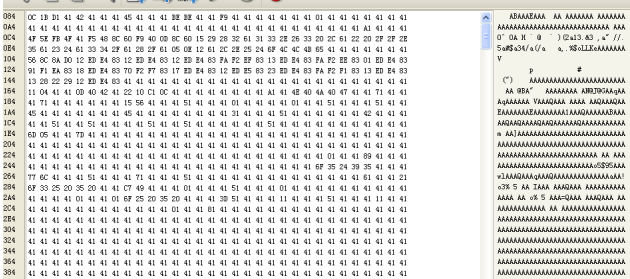
**问题4**

|  |
| --- |
| 恶意负载是如何被保护的？ |

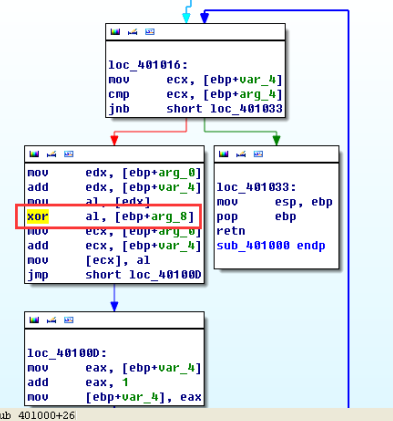
**答：上述恶意代码的负载是经过异或编码的。用于解码的函数是 sub\_40132C，它会调用sub\_401000将加密的负载 与0x41进行异或操作，达到解密的目的。**

分析过程：

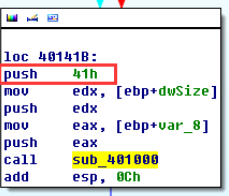
通过之前的分析可以知道恶意样本是将资源节中的内容释放并加载到内存中去，所以我们这里需要看一 下资源节中的内容。 使用resource\_hacker工具进行查看



通过之前的分析这里应该是一个4D 5A开头的一个PE文件结构，但是文件的开通并不是。同时我们可以 发现这个资源中有非常多的41，而正常来说一个PE文件中大部分的内容应该是00。那么我们也就不难想 到这个资源节应该是被进行了加密操作来保护，同时这个加密操作应该只是简单的异或操作，并且所使用的key应当就是41。 为了验证我们的猜想，去IDA中查看一下释放和加载的相关内容。



我们找到了他在最后调用了一个函数，这个函数的内部就是一直循环进行异或操作，也就验证了我们之前的猜想，这个恶意代码就是使用异或进行加密，从而保护自身的payload。 并且注意到这个函数在调用之前的参数就是刚刚我们猜到的41h，那么猜想成立



问题5

|  |
| --- |
| 字符串列表是如何被保护的？ |

**答： 如上，通过调用sub\_401000处的函数，对字符串列表进行异或编码，从而加密字符串。和之前的payload一样，都是使用41h作为密钥进行异或加密的**

**LAB12-3**

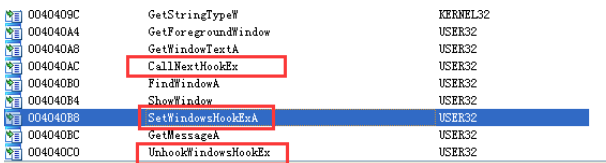
**问题1**

|  |
| --- |
| 这个恶意负载的目的是什么？ |

**答：这是一个击键记录器。**

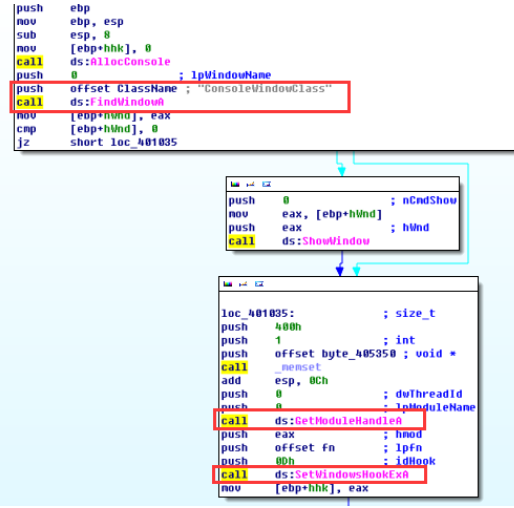
分析过程：

首先使用IDA查看一下这个程序的导入表

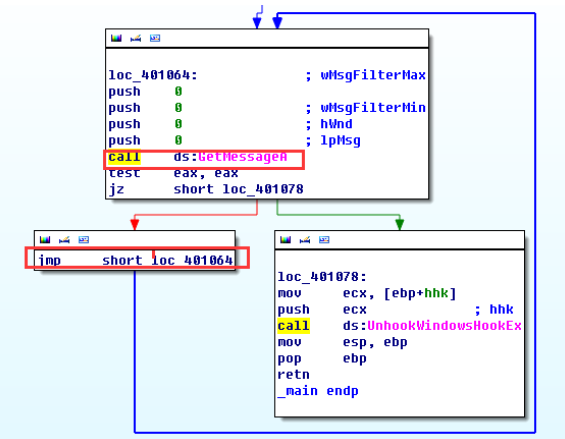


在导入表中有几个引起我们注意的函数就是设置hook

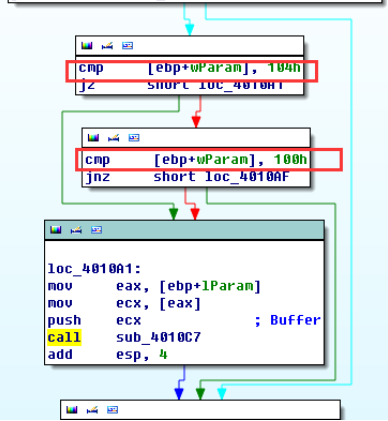
在main函数中我们首先可以看见



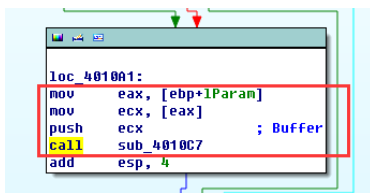
这个函数 同时我们可以看见在设置hook之前传递的参数是0Dh，经过MSDN上的查询我们可以发现，这个hook的功能就是监视键盘的输入，而hook被设定为了offset fn中的内容



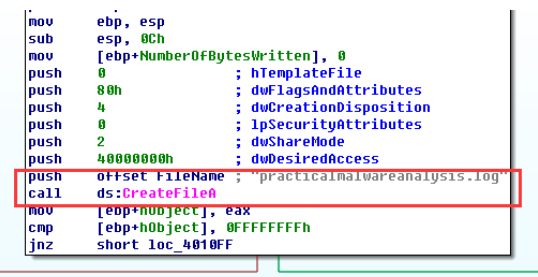
然后进入一个无限循环，将刚刚获取到的消息发送到hook绑定的函数中 进入到fn内部



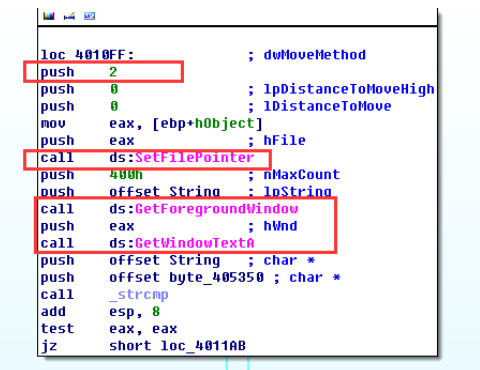
我们可以看见有两个地方进行了比较，经过查询我们可以发现这个地方其实就是在判断键盘按键的类型，之后我们可以看见：



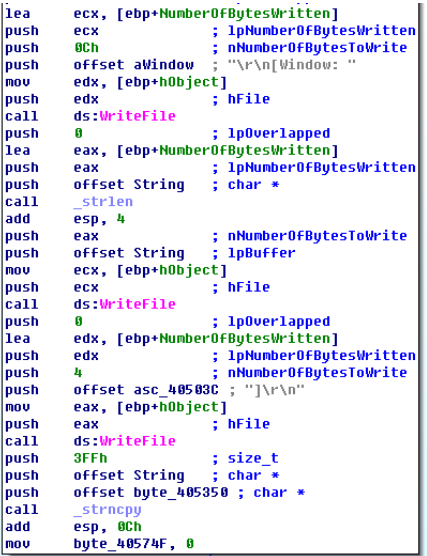
这里将刚刚获取的内容作为参数传递给了sub\_4010C7，接下来我们查看一下这个函数的内容



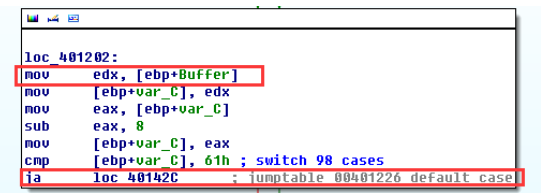
我们可以看见他打开了一个名为 practicalmalwareanalysis.log 的文件



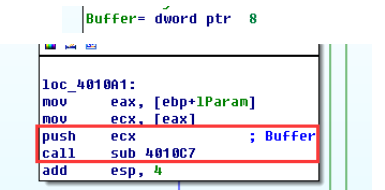
接下来他将文件的指针设置到了文件的末尾，也就是说接下来的内容会从文件的末尾进行输入。之后获取了按键的窗口和窗口的标题，也就获取了按键的来源。



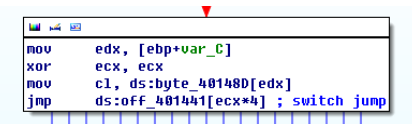
之后的就是向日志文件中写入刚刚记录的内容



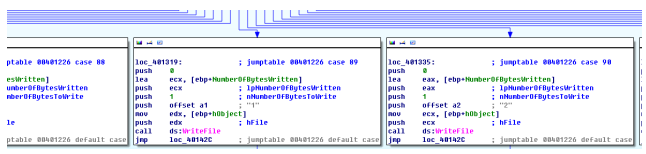
之后对Buffer中的内容进行读取，然后跳转到loc\_40142C的位置。 我们先来查看一下这个buffer中是什么内容



由此我们可以知道这里保存的是虚拟的按键码



之后利用这个按键码跳转到不同的分支



经过观察可以发现这里就是在文件中记录对应的按键内容 综上，这次的样本就是记录键盘的输入内容，保存到 practicalmalwareanalysis.log 中

**问题2**

|  |
| --- |
| 恶意负载是如何注入自身的？ |

**答：使用Hook注入，监控电脑的击键行为。**

**问题3**

|  |
| --- |
| 这个程序还注入了哪些文件？ |

**答：它还创建了日志文件practicalmalwareanalysis.log，将捕获到的击键行为记录在这个文件中。**

**LAB12-4**

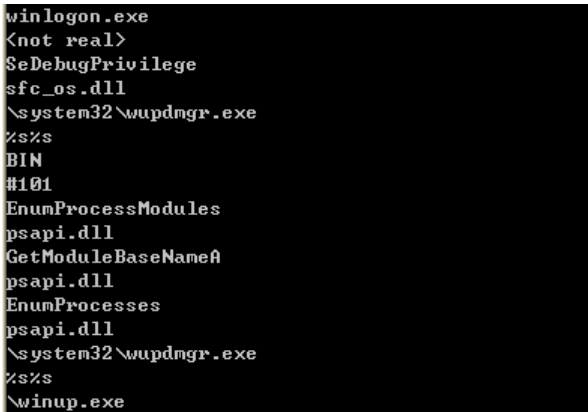
**问题1**

|  |
| --- |
| 位置 0x401000 的代码完成了什么功能？ |

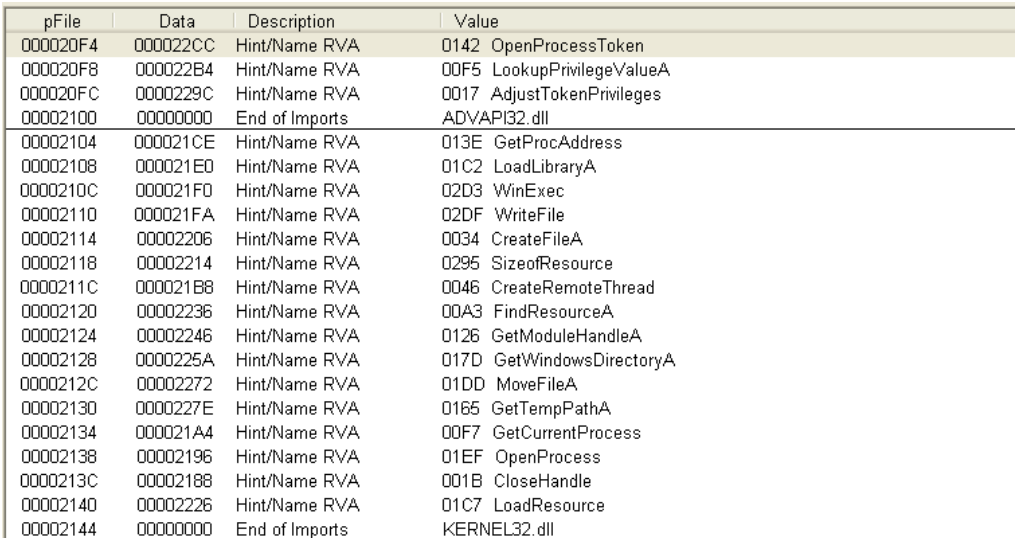
**答：用来判断给定的PID对应的是否是进程winlogon.exe。**

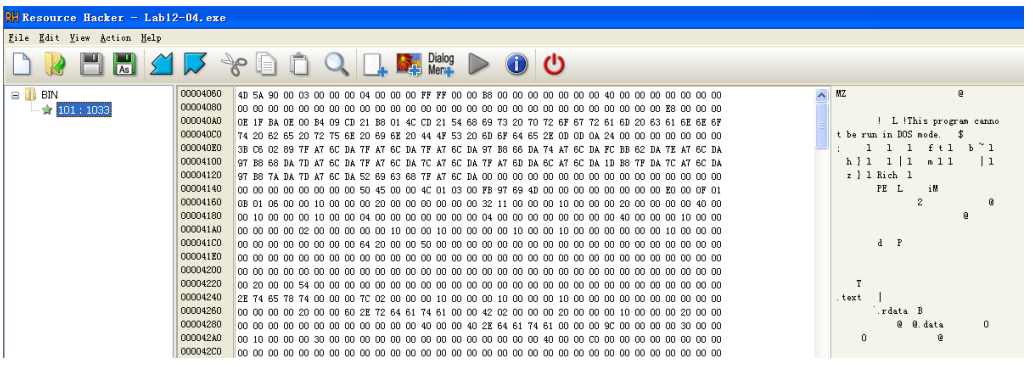
分析过程：

首先进行基础静态分析，Lab12-04.exe的字符串中包含了好多.exe文件和.dll文件的名字，如下：

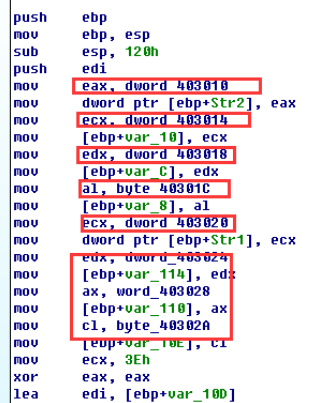


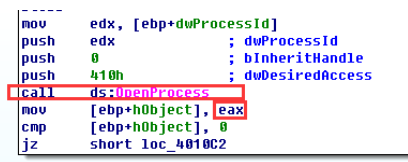
有以下导入函数，包括用来括创建线程的CreateRemoteThread、一些对资源进行操作的函数LoadResource和 FindResourceA，还有一些与特权级相关的函数LookupPrivilegeValueA和AdjustTokenPrivileges。用 Resource Hacker查看该程序，在资源节中发现了一个名为BIN的程序。

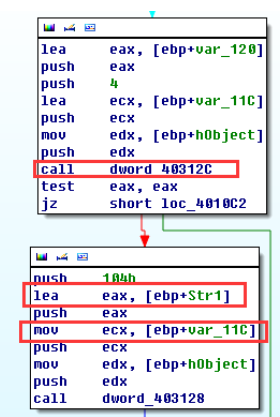




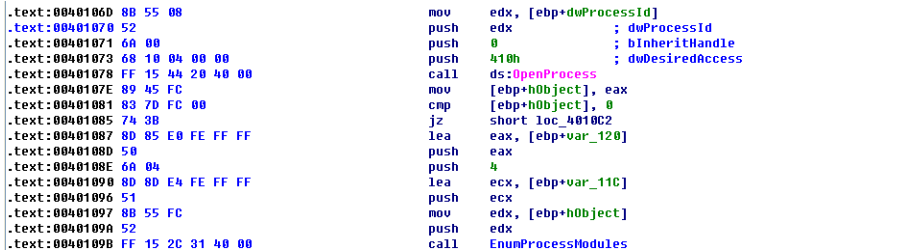
下面进行基础动态分析。双击运行该程序，发现它打开了一个网页，从Procmon中我们看到该恶意代码创建了 文件winup.exe，并且覆盖了wupdmgr.exe的Windows更新二进制文件。比较恶意代码释放的wupdmgr.exe和 在上面资源节中提取的BIN文件，发现它们是相同的。且使用netcat监测80端口，我们可以发现恶意代码试图 从www.practicalmalwareanalysis.com中获取updater.exe。 下面用IDA打开Lab12-04.exe，从main函数开始分析。开始的几行调用LoadLibraryA和GetProcAddress从 psapi.dll中解析Windows进程枚举的函数，并将得到的指针保存到三个全局变量中。为了增加代码的可读 性，我们对这三个全局变量重命名。







然后获取进程名，并保存在str1中 之后将其和str2进行比较 如果相同就会返回1，如果不相同就返回句柄。 综上，这个函数的功能就是遍历进程列表寻找 winlogon.exe 这个进程



首先调用了OpenProcess，参数是dwProcessId，用来获 得这次循环的进程的句柄，并将返回的句柄存入eax寄存器，作为参数调用函数myEnumProcessModules，获 取加载到这个进程所有模块的句柄数组，之后调用GetModuleBaseNameA获取进程模块的名字，存储在Str1 中。 之后将Str1和Str2进行比较，Str2中存的是字符串winlogon.exe，如果相同则返回1，不同则返回0。因此， 函数sub\_401000是在寻找进程winlogon.exe的PID。

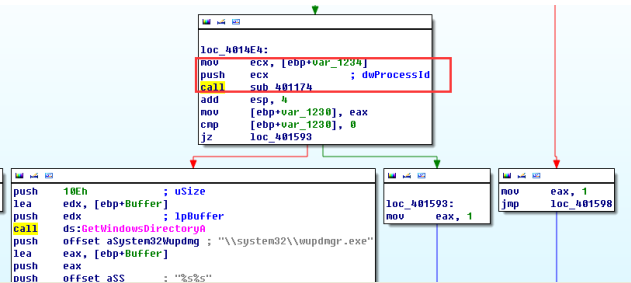
**问题2**

|  |
| --- |
| 代码注入了哪个进程？ |

**答：winlogon.exe。**

分析过程：

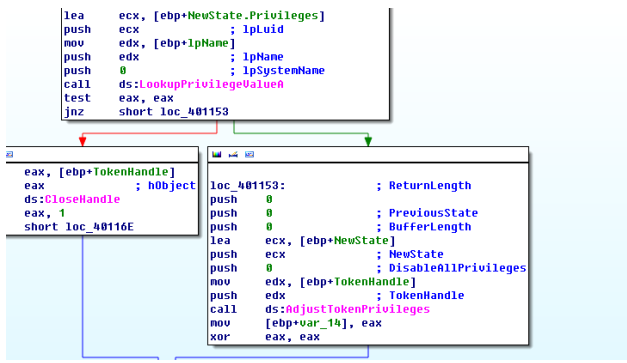
接下来我们进入到main中进行分析



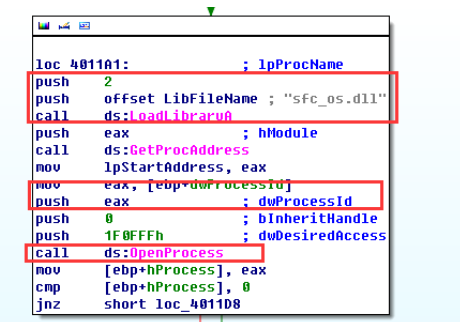
可以看见刚刚获得的句柄被作为参数传递给了sub\_401174，接下来查看一下这个函数做了什么



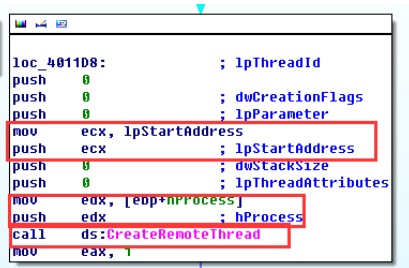
可以看见这个函数先调用了一个其他的函数，根据参数的提示可以看出来应该是和权限有关的函数。



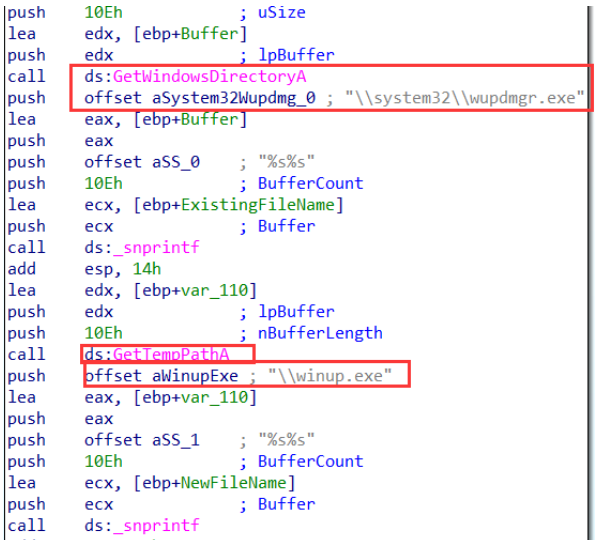
进来以后发现确实是在提升这个进程的权限。



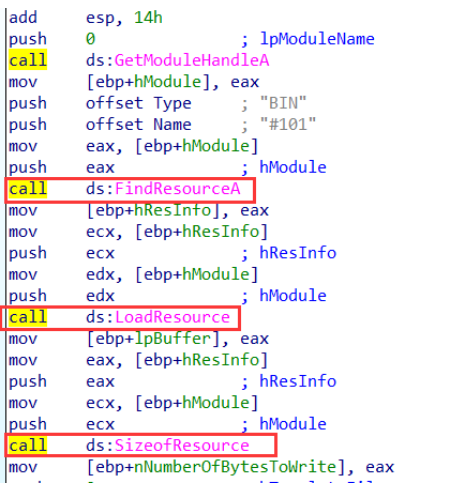
之后我们可以看见程序装载了sfc\_os.dll中偏移量为2的函数，然后打开了刚刚我们分析得到的 winlogon.exe 进程，并将句柄保存在了hProcess中。



之后创建了一个线程，可以看见hProcess就是刚刚保存的句柄。而这个lpStartAddress就是刚刚从dll文 件中加载的偏移为2的函数的指针。

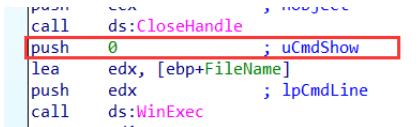


之后我们可以看见这个样本获取了系统路径，并将当其和 \\system32\\wupdmgr.exe 进行了拼接，形 成了完整的一个程序的路径。





然后我们可以看见这个程序加载了资源，并进行了写文件操作，最后运行了这个文件。经过之前的分析 我们可以知道这里写入的程序就是刚刚分析得到的 \\system32\\wupdmgr.exe 。正常来说，windows 的保护机制是会阻止对系统文件进行写入的，但是之前我们也分析过他调用了sfc\_os.dll中的函数来禁用 了windows的保护机制，所以这里是能成功写入的。 同时我们注意到，在启动这个修改过后的程序时，他设置了参数uCmdShow



这里就是表示不会显示任何窗口，以此来达到隐蔽执行的目的。 综上，本次的样本注入的进程为：wupdmgr.exe

**问题3**

|  |
| --- |
| 使用LoadLibraryA装载了哪个DLL程序？ |

**答：加载的是sfc\_os.dll，用来禁用Windows的文件保护机制。**

分析过程：

sfc\_os.dll是用于保护Windows的系统文件，它包含一系列运行在winlogon.exe中的线程，上面用到的序号2 函数是一个未公开的导出函数，为SfcTerminatewatcherThread。只有在winlogon.exe中才能成功地运行这 个函数，强制运行这个函数会导致在下次系统重启之前禁用Windows文件保护机制。

**问题4**

|  |
| --- |
| 传递给CreateRemoteThread调用的第4个参数是什么？ |

**答：传递给CreateRemoteThread的第4个参数lpStartAddress是sfc\_os.dll中进程2的指针**

分析过程：



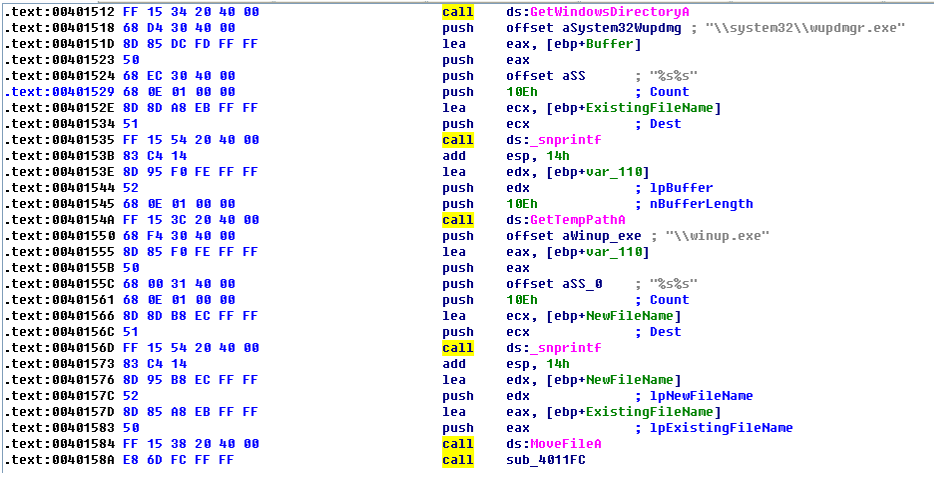
**问题5**

|  |
| --- |
| 二进制主程序释放出了哪个恶意代码？ |

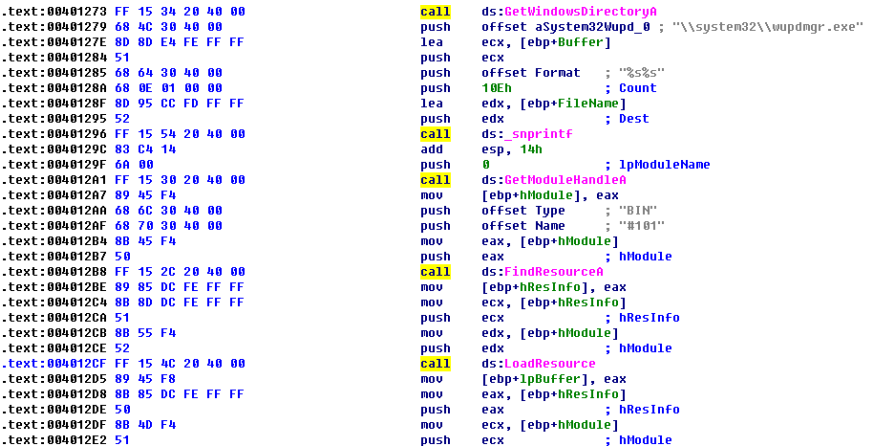
**答：从从资源节中释放出二进制文件BIN，将它复制到%TEMP%目录下，然后用这个二进制文件覆盖Windows的更新 程序wupdmgr.exe。**

分析过程：

回到主函数中，调用GetwindowsDirectoryA返回当前Windows目录的指针，将这个字符串和 \system32\wupdmgr.exe一起作为参数传递给\_snprintf调用，构建出字符串 C:Windows\system32\wupdmgr.exe并存储到ExistingFileName中。wupdmgr.exe是Windows用来自动升级的函数。



之后又用同样的方法构造出字符串C:\Documents and Settings\username\Local\Temp\winup.exe并存在 NewFileName中。调用MoveFileA将把wupdmgr.exe移到winup.exe中。 接下来调用了sub\_4011FC，进入这个函数进行分析。它调用了GetModuleHandleA获得当前进程的模块句柄， 然后进行了一系列对资源节的操作，首先调用了参数为#101和BIN的FindResourceA，用来提取它的资源节到 硬盘上。之后调用了LoadResource、SizeofResource、CreateFileA和WriteFile从资源节BIN中提取文件，并将提取到的文件写入到C:Windows\System32\wupdmgr.exe中，创建了一个新的Windows更新处理程序。



最后调用了winExec启动新的wupdmgr.exe，其参数uCmdShow=0，即SW\_HIDE，因此会隐藏这个程序的窗口。

**问题6**

|  |
| --- |
| 释放出恶意代码的目的是什么？ |

**答： 恶意代码向 winlogon.exe 注入一个远程线程，调用sfc\_os.dll中的导出函数 SfcTerminatewatcherThread，用来在下次启动之前禁用Windows的文件保护机制。通过释放出的这个二进 制而文件可以更新自己的恶意代码，并且使得用户还可以调用原始的wupdmgr.exe，特洛伊木马化 wupdmgr.exe。**

分析过程：

使用resource\_hacker将资源节的文件提取出来之后，使用IDA查看



可以看见这个样本先执行了系统路径下的winup.exe程序，这个程序的功能是进行windows的更新。之 后拼接了一个字符串，并将这个字符串保存在了Dest中。同时需要注意到的是，这个字符串的最后是 wupdmgrd.exe，而windows里自带一个程序为：wupdmgr.exe，两者只差了一个d，可以起到混淆的 作用。之后将这个位置信息作为参数之一传递给了URLDownloadToFileA



**IDA和YARA**

本次实验没有要求，故不再编写。

**四、实验心得**

**本次实验，我熟悉了对恶意代码分析工具有了更深入的理解，加深了我对恶意代码的理解和相关知识的掌握。在实验过程中，通过亲手分析恶意代码，我收获了很多，过程是非常快乐的。最后，我对本门课程的实验开始得心应手，做实验的速度越来越快了。**