

**《恶意代码分析与防治技术》课程实验报告**

**实验七**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

班 级 信息安全1班

1. **实验目的**

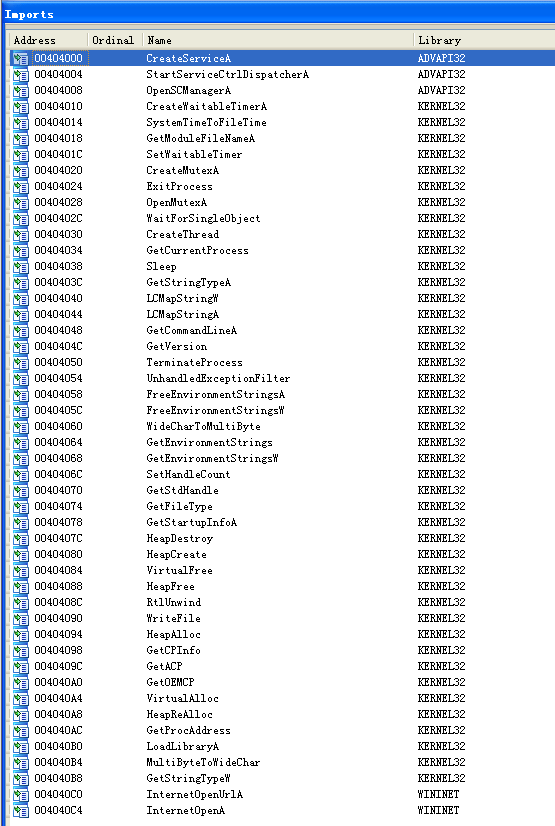
本次实验的主要目的是分析和理解一系列恶意代码的实验样本，包括Lab07-01.exe、Lab07-02.exe、Lab07-03.exe、Lab07-03.exe，以及编写相应的Yara规则来检测这些恶意代码的存在。通过分析这些实验样本，可以了解恶意代码的行为、结构和特征，以及如何使用Yara规则来检测类似的恶意代码。

1. **实验原理**

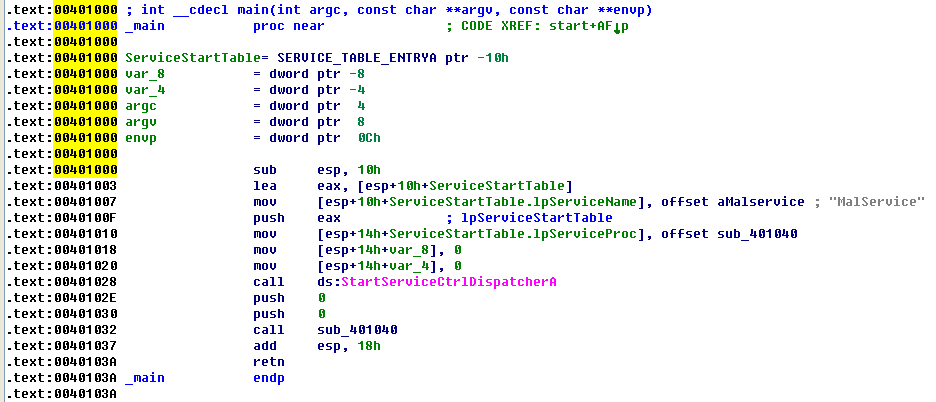
实验基于静态分析和反汇编技术，通过使用IDAPro等工具来分析恶意代码的汇编指令和逻辑结构。

1. **实验过程**
2. **Lab7-1**

Ida pro打开Lab07-01.exe文件，查看导入表，看到了OpenSCManagerA、StartServiceCtrlDispatcherA和CreateServiceA函数，提示了这个文件创建了一个服务。InternetopenA和InternetopenUrl的调用告诉这个程序可能连接到一个URL并下载内容。

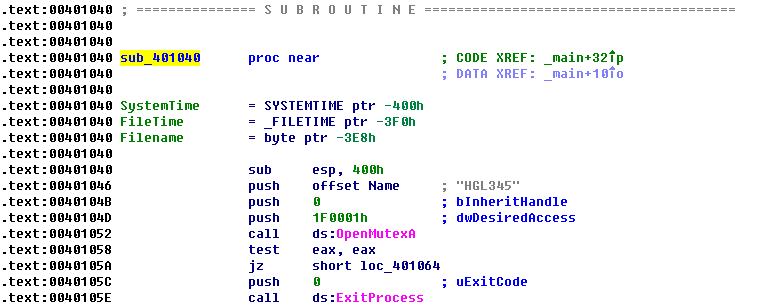


查看主函数



StartServiceCtrlDispatcherA这个函数被程序用来实现一个服务，并且立即被调用，该函数指定了服务控制管理器会调用的服务控制函数。指定了sub\_401040在调用StartServiceCtrlDispatchA之后被调用。

查看sub\_401040



首先对栈进行参数的压栈操作，从右边的注释中可以看见在栈中压入了"MalService"这个字符串。调用OpenMutexA，这个调用试图去获取一个命名为HGL345的互斥量句柄。如果这个调用成功，程序就会退出，否则，将跳转到 loc\_401064 处。

查看loc\_401064，



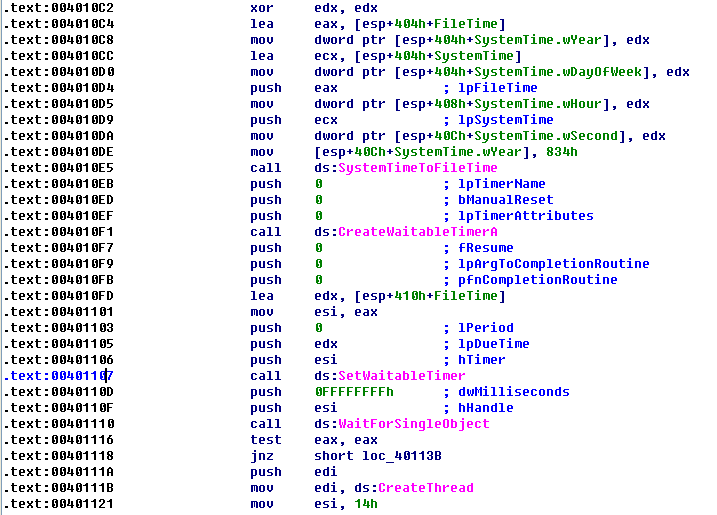
创建一个名为HGL345的互斥量。这两处对互斥量组合调用，被设计来保证这个可执行程序任意给定时刻只有一份实例在系统上运行。如果有一个实例已经在运行了，则对OpenMutexA的第一次调用成功，并且这个程序就会退出。如果打开失败则会创建这个互斥变量，之后调用OpenSCManagerA打开服务控制管理器并获取当前进程的全路径名。这个互斥量用来保证系统只打开了一个服务，而不会进行多次创建。

接下来调用OpenSCManagerA,打开一个服务控制管理器的句柄，以便这个程序可以添加或修改服务。调用 GetCurrentProcess 获取当前进程。

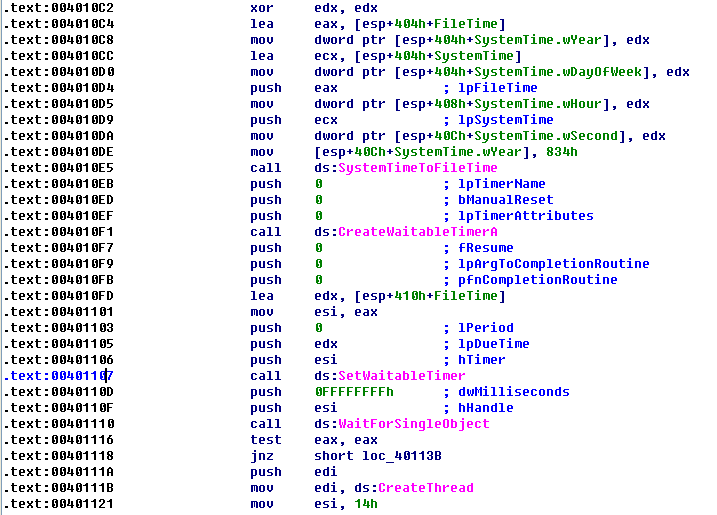
接下来调用GetModuleFileNameA函数，返回当前正在运行的可执行程序或一个被加载DLL的全路径名。第一个参数是要获取路径名的模块句柄，或者将它设置为NULL来获取这个可执行体的全路径名。

CreateServiceA关键的参数是0x004010A8处BinaryPathName,0x004010AB处dwStartType以及0x004010AD处的dwServiceType。这个可执行程序的二进制路径与用GetModuleFileName调用得到的当前正在运行可执行程序路径是一样的。GetModuleFileNameA通过动态地获得这些信息，它可以安装这个服务，而不用考虑哪个可执行程序被调用，或是它被保存在哪里。

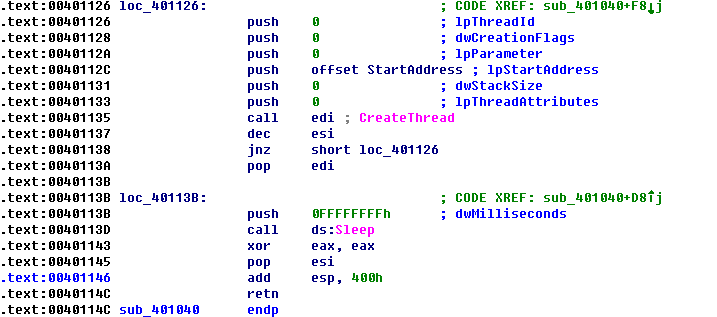
dwStartType可能的值是SERVICE\_BOOT\_START(Ox00)、SERVICE\_SYSTEM\_START(0x01)、SERVICE\_AUTO\_START(Ox02)、SERVICE\_DEMAND\_START(0x03),以及SERVICE\_DISABLED(0x04)。这个恶意代码传入0x02,它对应SERVICE\_AUTO\_START,暗示这个服务在系统启动时自动运行。



IDA Pro标记了一个是SYSTEMTIME的结构体，对秒、分、小时、天的不同域用来指示时间。本例中，所有值首先被设置为0,然后表示年的值在0x004010DE处被设置为0x0834,也就是10进制的2100。这个时间代表2100年1月1日的0：00。然后，程序调用SystemTimeToFileTime在不同时间格式之间转换。

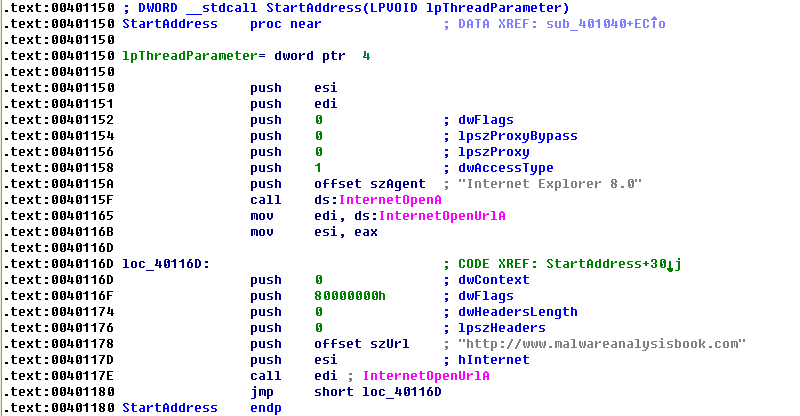


接下来，程序调用CreateWaitableTimer、SetWaitableTimer,以及WaitForSingleObject。传给SetWaitableTimer的lpDueTime参数是SystemTimeToFileTime返回的FileTime。这段代码随后使用WaitForSingleObject进入等待，直到2100年1月1日。ESI被设置为计数器0x14(十进制20)，代码接着循环20次。循环的末尾，ESI被递减，并且当它到达0时，循环退出。



调用CreateThread 函数，lpStartAddress参数告诉我们哪个函数被当作这个线程的起始地址使用——本例中标记为StartAddress。

查看StartAddress



调用InternetopenA来初始化一个到网络的连接，接着在一个循环中调用InternetopenUrlA,并且一直下载该网址的主页。循环末尾的jmp指令(0处)是一个无条件跳转，这意味着这段代码永远不会终止；调用InternetopenUrlA,并且一直下载www.malwareanalysisbook.com的主页。CreateThread被调用了20次，有20个线程一直调用InternetopenUrlA。

显然，这个恶意代码的目的是将自己在多个机器上安装成一个服务，进而启动一个DDoS攻击。如果所有被感染的机器在同一时间(2100年1月1日)连接到服务器，它们可能使服务器过载并无法访问该站点。

1. **当计算机重启后，这个程序如何确保它继续运行(达到持久化驻留)?**

这个程序创建服务MalService,来保证每次在系统启动后运行。

1. **为什么这个程序会使用一个互斥量?**

这个程序使用一个互斥量，来保证在同一时间这个程序只有一份实例在运行。

1. **可以用来检测这个程序的基于主机特征是什么?**

可以搜索一个名为HGL345的互斥量，以及MalService服务。

1. **检测这个恶意代码的基于网络特征是什么?**

这个恶意代码使用用户代理Internet Explorer 8.0,并和www.malwareanalysisbook.com通信。

1. **这个程序的目的是什么?**

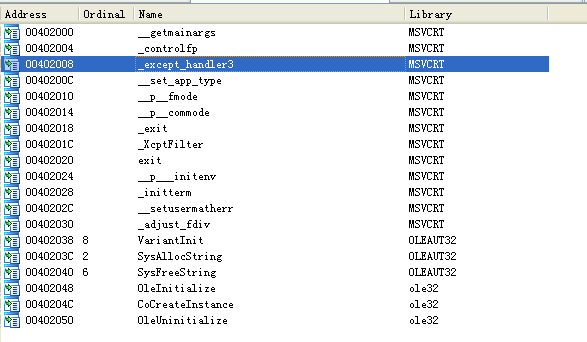
这个程序等待直到2100年1月1日的0:00，发送许多请求到http://www.malwareanalysisbookcom/,大概是为了对这个网站进行一次分布式拒绝服务(DDoS)攻击。

1. **这个程序什么时候完成执行?**

这个程序永远不会完成。它在一个定时器上等待直到2100年，到时候创建20个线程，每一个运行一个无限循环。

1. **Lab7-2**

查看导入表，后6个函数是与com相关的。



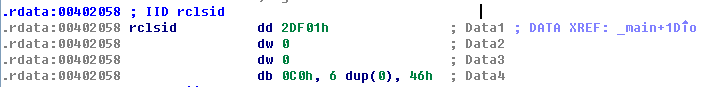
当运行这个程序时，打开一个Internet Explorer，并弹出一个访问的网址是http://www.malwareanalysisbook.com/ad.html的广告。没有发现任何这个程序修改系统或安装它自己以便在计算机重启后运行的证据。

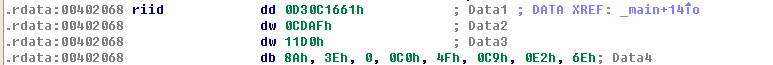
查看main函数



该恶意代码首先初始化COM,并调用OleInitialize和CoCreateInstance获得一个COM对象。返回的COM对象被保存在栈上的ppv变量中，参数 riid 和 rclsid分别表示接口标识符（IID）和类标识符（CLSID）。

单击rclsid显示0002DF01-0000-0000-C000-000000000046，单击riid显示D30C1661-CDAF-11D0-8A3E-00C04FC9E26E。





这个IID是IWebBrowser2，CLSID对应Internet Explorer。



VariantInit 的功能是释放空间、初始化变量； SysAllocString 是用来给分配内存，并返回BSTR；SysFreeString 是用来释放刚刚分配的内存的。

分配内存时给之前string分析出来的url分配内存，而在释放之前调用了一个 dword ptr [edx+2Ch] 。

EAX指向COM对象的位置，然后被解引用。EDX指向这个COM对象的基址。这个对象中偏移0x2C处的一个函数被调用，IWebBrowser2接口的偏移0x2C是Navigate函数。

使用IDA Pro中的Structures窗口，创建一个结构体，并标记这个偏移。当Navigate函数被调用，Internet Explorer将导航网址

http://www.malwareanalysisbook com/ad.html。

调用Navigate函数之后，会执行一些清理函数，然后程序终止。这个程序不会持久化地安装它自己，并且也不修改系统。它简单地显示一个一次性的广告。

1. **这个程序如何完成持久化驻留?**

这个程序没有完成持久化驻留。它运行一次然后退出。

1. **这个程序的目的是什么?**

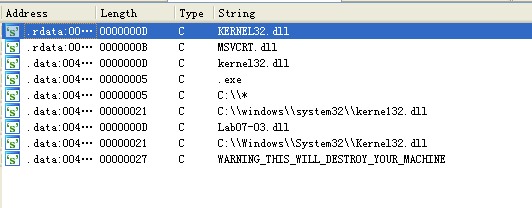
这个程序给用户显示一个广告网页。

1. **这个程序什么时候完成执行?**

这个程序在显示这个广告后完成执行。

1. **Lab7-3**

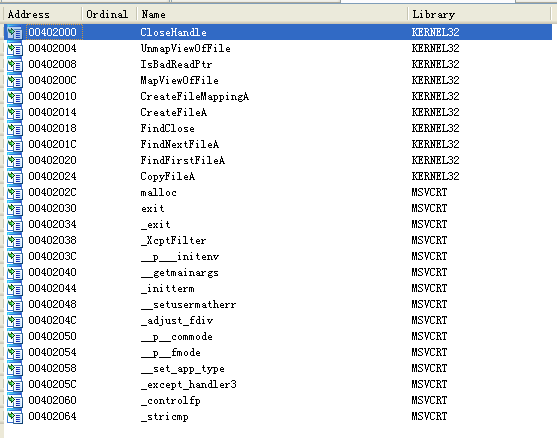
查看字符串



字符串kerne132.dll很明显被设计来看起来与kernel32.dll相似，用1替换了l。

字符串Lab07-03.dll告诉我们这个.exe可能以某种方式在这个实验中访问这个DLL。字符串WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE是本书专门用来修改这个恶意代码的人工制品。

查看导入表

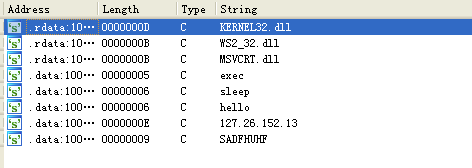


CreateFileA、CreateFileMappingA,以及MapViewOfFile,告诉这个程序可能创建并打开一个文件，然后将它映射到内存中。FindFirstFileA和FindNextFileA函数告诉这个程序可能搜索目录，查找文件；CopyFileA说明程序会复制它找到的文件。

但并没有导入Lab07-03.dll(或使用任何这个DLL中的函数)、LoadLibrary,或者GetProcAddress，暗示可能没有在运行时加载那个DLL。

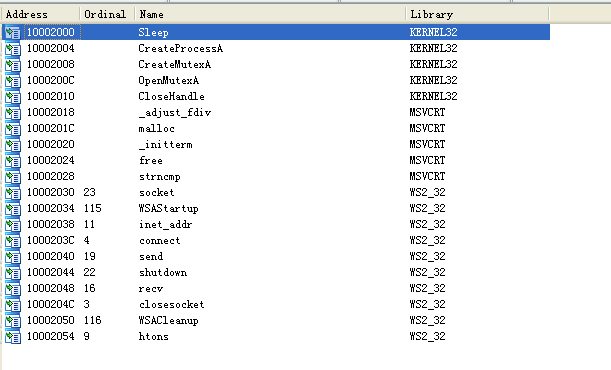
Ida pro打开Lab07-03.dll文件

查看Lab07-03.dll的字符串



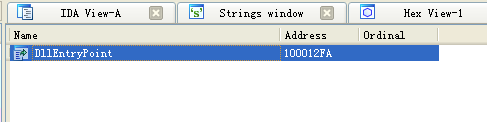
可以看到字符串hello、sleep、exec等，还看到一个ip地址，127.26.152.13。“exec”说明这个程序有可能是个后门程序，“127.26.152.13”这个 ip 很有可能就是个后门木马。

查看Lab07-03.dll的导入表



看到从ws2\_32.dll中的导入表中包含了要通过网络发送和接收数据所需要的所有函数。CreateProcess函数告诉我们这个程序可能创建另外一个进程。CreateMutexA 和 OpenMutexA 说明这个 dll 文件会创建和打开一个互斥变量，还会调用 Sleep 函数来休眠。

查看Lab07-03.dll的导出表



发现它不能被另外一个程序导入，尽管一个程序还是可以调用LoadLibrary来载入没有导出的DLL。

当运行这个可执行程序时，快速退出，而没有什么引人注目的活动。

查看Lab07-03.dll的main函数



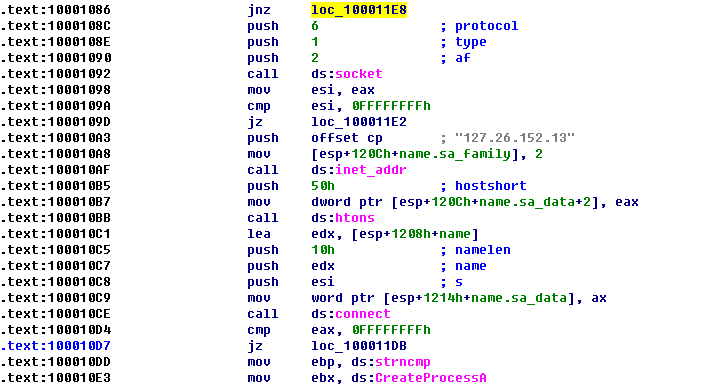


这个样本首先分配了一个非常大的栈空间（11F8h，也就是4600d），调用通过库函数\_alloca\_probe,来在空间中分配栈。fdwReason 的值赋值给了 eax，并和 1 比较大小，如果不等于 1 就跳转到 loc\_100011E8 执行，程序结束；由此分析，这个代码是希望 fdwReason=1 的，进而才能继续执行下面的代码。



将这个 byte\_10026054 赋值给 al，而 byte\_10026054 是 0，也就是将 al 置0。将 al 存入[esp+1208h+Str2]，将 eax 置为 0，压入其他参数，调用函数 OpenMutexA 打开了一个名为“SADFHUHF”的互斥量，检查返回值，如果 eax=0，即调用成功则不跳转，继续执行；否则将跳转到 loc\_100011E8 处，程序结束。接着又调用 CreateMutexA 来创建一个名为“SADFHUHF”互斥量。

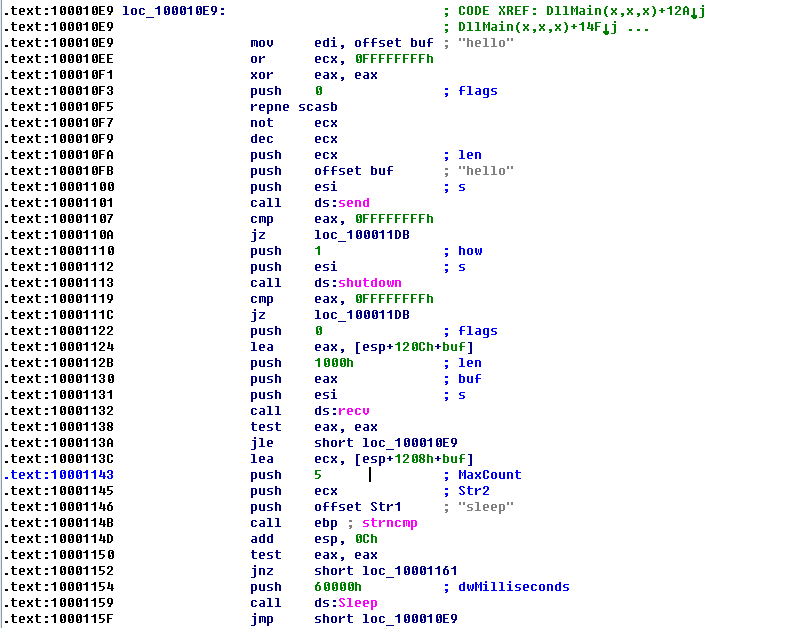
对OpenMutexA和CreateMutexA的函数调用，保证同一时间只有这个恶意代码的一个实例在运行。



调用 WSAStartup 函数初始化套接字，其参数 202h 说明推荐调用者使用的最高版本号套接字版本是 2.2，lpWSAData 是指向 WSADATA 数据结构的指针，用于接收 Windows Sockets实现的详细信息，即本机系统实际使用的版本号。调用成功后，程序会调用 socket 函数创建一个流式套接字，然后将返回值赋值给 esi，并与 0FFFFFFFFh 进行比较，也就是十进制的-1，如果返回值大于-1，程序继续执行，否则将跳转到 loc\_100011E2处，做一些清理工作后退出程序。

执行 connect 函数，参数s的值是 esi，是刚刚 WSAStartup 初始化之后保存在 esi 栈中的套接字；name 的值是edx，指向的经过 inet\_addr 和 htons 调用后的“127.26.152.13”，即从主机序转换成了网络序；namelen 的值是 10h也就是十进制的16；端口参数是0x50,也就是端口80,这个端口通常被Web流量所使用。与 0FFFFFFFFh 进行比较，也就是十进制的-1，如果返回值大于-1，程序继续执行，否则将跳转到 loc\_100011DB处。

ebp 存储 strncmp 函数的位置，就是指向 strncmp 函数的一个值；ebx是指向 CreateProcessA 的指针。



buf参数保存了将要通过网络发送的数据，指向buf的指针代表字符串hello,并做了相应的标记。这似乎是一个受害机器发送的问候，来使服务器知道它已经准备好执行一个命令了。

调用 send 这个函数，将 buf 里面的“hello”发送出去，调用 shutdown 函数，关闭 socket 连接。

调用recv函数,接收一个数据并存入 buf 中。该函数的返回值是接收的字节数，如果没有接受到数据（eax=0）或者报错时（eax<0）时，JLE 就会跳转到 send 开始的地方重复执行，否则将调用 strncmp。接收缓冲区在 0x1000开始，大小是 5 个字节，告诉我们这个要被执行的命令是我们接收缓冲区中保存的任意 5 字节的东西。第一个访问buf的指令是lea指令，将获得一个指向那个位置的指针。对recv的调用将连入的网络流量保存到栈上。



strncmp函数将检查前count个字符是不是字符串sleep,此处count=5是字符串sleep的长度。在调用这个函数后，它立刻检查返回值是不是0,如果是，它调用Sleep函数来睡眠60秒。如果远程服务器发送sleep命令，这个程序将调用Sleep函数。

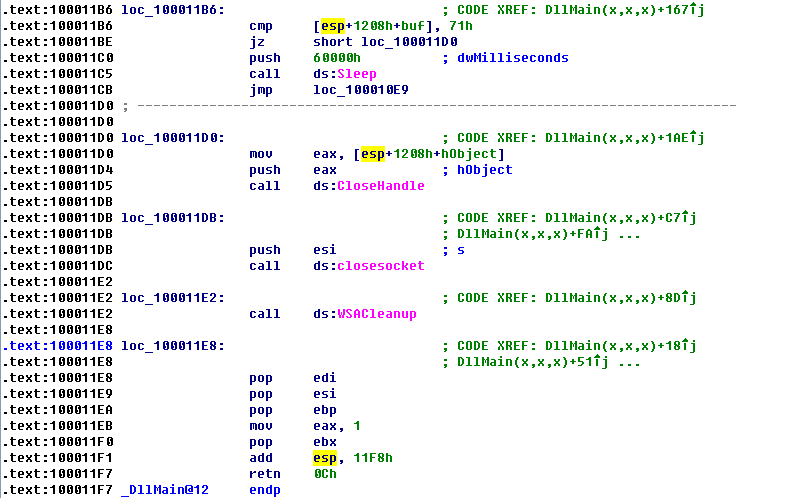


如果返回值不是0，检查这个缓冲区是否是以exec开始的。如果是，strncmp函数将返回0,并且这段代码将顺序执行到jnz指令，并调用CreateProcessA函数。

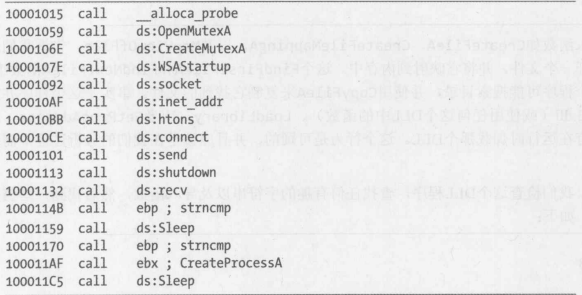
CreateProcessA函数有很多参数，其中CommandLine参数告诉我们要被创建的进程。CommandLine对应在0x10001010处的值0x0FFB。



如果不是 exec 的话，会将 Str2（即 buf）的大小与 71h 进行比较，判断缓冲区是否大于该值，如果相等则跳转loc\_100011DB，结束程序；否则，会调用 sleep 休眠大约 6 分钟，再跳转到 send 处循环执行。

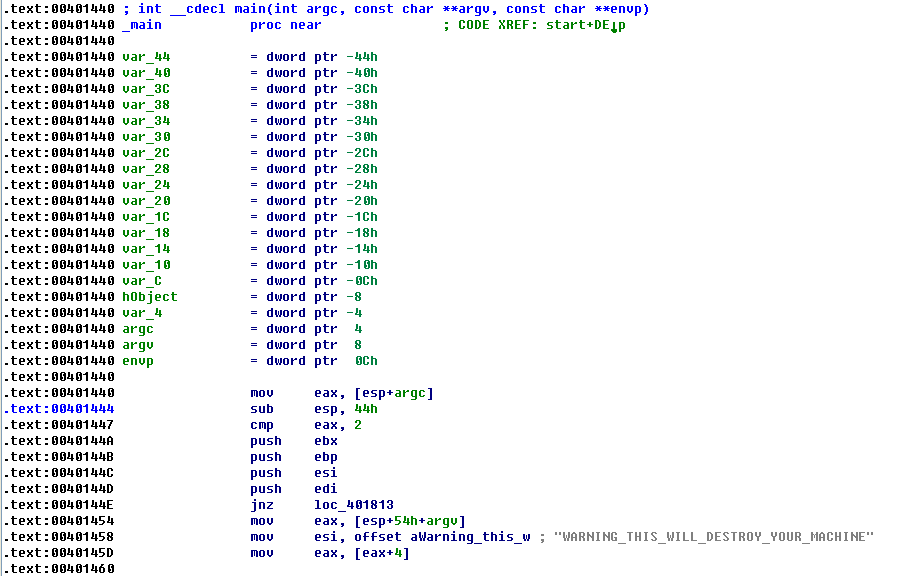


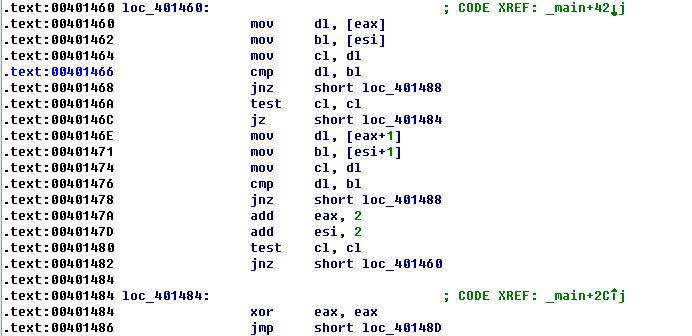
只查看call指令，



接收缓冲区在0x10000开始，并且这个值是用lea指令来设置的，这告诉我们这个数据本身是保存在栈上的，并且不仅仅是一个指向数据的指针。同样，0x0FFB是我们的接收缓冲区5个字节的事实告诉我们这个要被执行的命令是我们接收缓冲区中保存的任意5字节的东西。在这个案例中，这意味着从远程服务器接收到的数据将会是exec Ful1PathOfProgramToRun。当这个恶意代码从远程服务器接收到这个exec FullPathOfProgramToRun命令行字符串时，它会用FullPathOfProgramToRun来调用CreateProcessA。

查看Lab07-01.exe的main





第一个比较检查参数个数是否是2。如果参数个数不是2,代码在0x0040144E处跳转到loc\_401813，使程序提前退出。(这是当我们试图执行动态分析时发生的事情，即程序会快速终止)

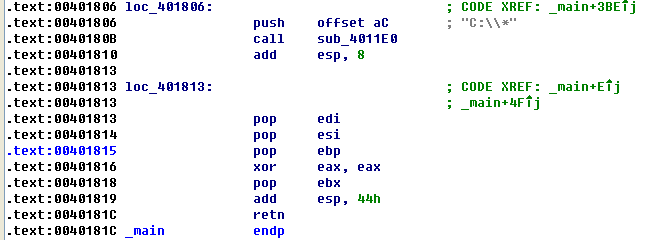
eax 指向了 argv 的开始地址，第三行这里又将 eax 这个指针向后移动了4 个 bit，即一个字节，eax 指向 argv[1]，也就是跳过了函数名，指向了具体的传入的参数。由于在之前将[esp+54h+argv]放入到了eax中，这里再加4之后取内容其实也就是取argv[1]到eax中，将WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE字符串移动到ESI寄存器。

在loc\_401460之间的循环，会比较保存在ESI和EAX中的值。如果它们不一样，这个程序跳转loc\_401488，并从这个函数返回，而不做任何其他事情。否则，会再比较该字符是否为 0，如果是的话，将跳出循环。

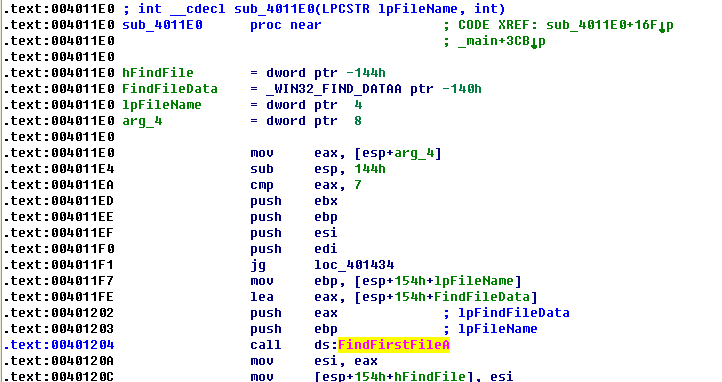
查看loc\_4017D4



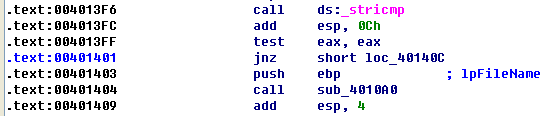
它在两个打开的文件上调用CloseHandle,然后调用CopyFile,这个函数复制Lab07-03.dll并把它放在C:\Windows\System32\kerne132.dll,这很明显的意图是看起来像kernel32.dll。猜测kerne132.dll会在与kernel32.dll相同的位置上运行。



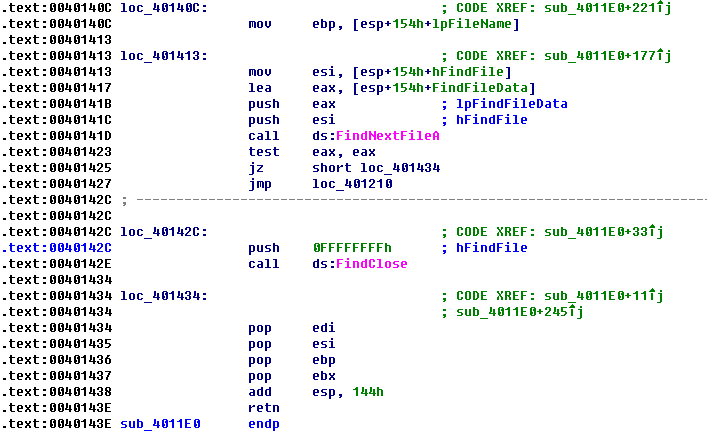
查看sub\_4011E0



这个函数的第一个参数被标记为lpFilename，被用来作为传递给接受一个文件名作为参数的Windows API函数。这个函数做的一件首要的事情是在C:\\\*上调用FindFirstFile,来搜索C:\\驱动器。

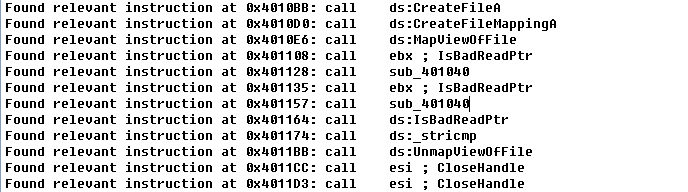


这个字符串比较用一个字符串和.exe检查，然后调用在sub\_4010A0函数，来查看它们是否匹配。

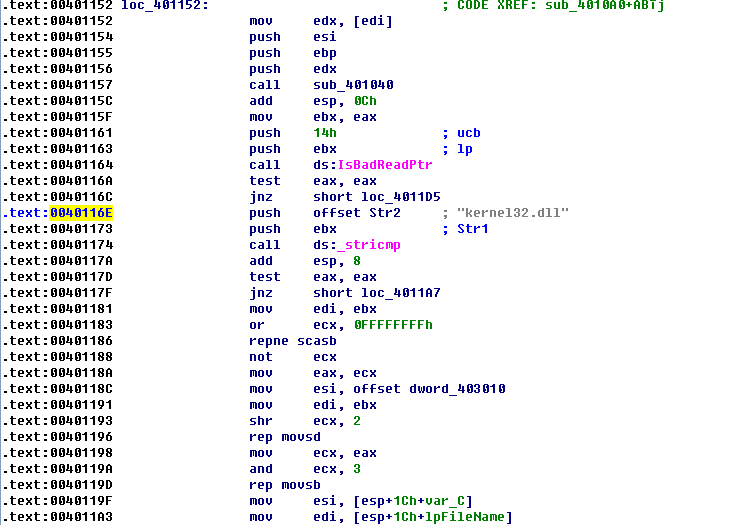


有一个对FindNextFileA函数的调用，然后一个jump调用，暗示这个功能在一个循环中被执行。在这个函数的末尾，FindClose被调用，然后这个函数以一些异常处理代码终止。

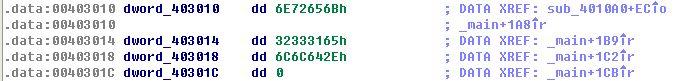
查看函数sub\_4010A0的函数调用



首先调用CreateFile、CreateFileMapping,以及MapViewOfFile来映射整个文件到内存中，整个文件被映射到内存空间，并且这个程序可以读写这个文件，而不需要任何附加的函数调用。



检查一个字符串值是不是kernel32.d11。这个程序调用repne scasb和rep movsd,这在功能上和strlen以及memcpy函数是等价的。EDI寄存器被rep movsd指令使用，被来自0x00401191处EBX的值加载，ebx被0x00401173处传给stricmp的值加载。如果这个函数找到字符串kernel32.d11,这段代码用某些东西替换它。转到rep movsd指令，并查看到源头在偏移dword\_403010。



将光标放在dword\_403010的同一行，并按A键，它会将这个数据转换为字符串kerne132.dll。

这个可执行文件遍历整个文件系统来查找以.exe结尾的文件，在.exe文件中找到字符串kernel32.dll的位置，并使用kerne132.dll替换它。Lab07-03.dll被复制到C:\Windowsl\System32目录中并被命名为kerne132.dll。这个恶意代码修改可执行文件让它们访问kerne132.dll,而不是kernel32.dll。这意味着kerne132.dll会替代kernel32.dll被修改过的可执行文件所加载。

1. **这个程序如何完成持久化驻留，来确保在计算机被重启后它能继续运行?**

这个程序通过写一个DLL到C:\Windows\System32,并修改系统上每一个导入它的.exe文件，来达到持久化驻留。

1. **这个恶意代码的两个明显的基于主机特征是什么?**

这个程序通过硬编码来使用文件名kerne132.dll,这是一个很好的检测特征(注意数字1的使用而不是字母l)。这个程序使用一个硬编码命名为SADFHUHF的互斥量。

1. **这个程序的目的是什么?**

这个程序的目的是创建一个很难删除的后门，来连接到一个远程主机。这个后门有两个命令：一个用来执行命令，一个用来睡眠。

1. **一旦这个恶意代码被安装，你如何移除它?**

这个程序很难被删除，是因为它感染系统上的每一个.exe文件。可能在这个例子中，最好方法是从一个备份恢复系统。

如果从备份恢复比较困难，可以留下这个恶意的kernel32.dll文件并修改它，来删除恶意的内容。还可以复制kernel32.dll,并将它命名为kernel32.dll,或者写一个程序来取消所有对PE文件的修改。

1. **Yara**

**Yara:**

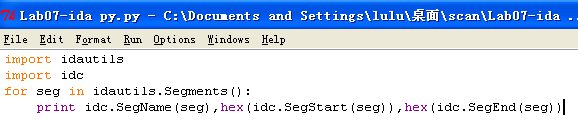
|  |
| --- |
| rule RuleforLab07\_01 |
| { |
| meta: |
| description = " Lab07\_01.exe" |
| strings: |
| $s1 = "http://www.malwareanalysisbook.com" fullword ascii |
| $s2 = "MalService" fullword ascii |
| $s3 = "Internet Explorer 8.0" fullword ascii |
| $s4 = "HGL345" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 70KB and |
| all of them |
| } |
|  |
| rule RuleforLab07\_02 |
| { |
| meta: |
| description = "Lab07-02.exe" |
| strings: |
| $s1 = "http://www.malwareanalysisbook.com/ad.html" fullword wide |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 50KB and |
| 1 of them |
| } |
| rule RuleforLab07\_03dll |
| { |
| meta: |
| description = "Lab07-03.dll" |
| strings: |
| $s1 = "SADFHUHF" fullword ascii |
| $s2 = "127.26.152.13" fullword ascii |
| $s3 = "141G1[1l1" fullword ascii |
| $s4 = "1Y2a2g2r2" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 500KB and |
| all of them |
| rule RuleforLab07\_03exe |
| { |
| meta: |
| description = "Lab07-03.exe" |
| strings: |
| $x1 = "C:\\windows\\system32\\kerne132.dll" fullword ascii |
| $x2 = "C:\\Windows\\System32\\Kernel32.dll" fullword ascii |
| $s3 = "kerne132.dll" fullword ascii |
| $s4 = "Lab07-03.dll" fullword ascii |
| $s5 = "WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 50KB and |
| 1 of ($x\*) and all of them |
| } |

**Python:**

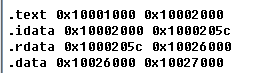
|  |
| --- |
| # -\*- coding: utf-8 -\*- |
| import os |
| import yara |
| import time |
|  |
| # 定义回调函数以获取匹配的文件名 |
| def my\_callback(data, file\_path): |
| print("匹配的文件：", data['rule'],"文件路径：", file\_path) |
|  |
| # 定义YARA规则 |
| rules = yara.compile("Lab07.yara") |
| # 扫描的样本文件夹路径 |
| sample\_folder = "sample" |
|  |
| # 初始化变量 |
| matched\_files = 0 |
| start\_time = time.time() |
| matched\_file\_paths = set() # 用集合来跟踪已匹配的文件路径 |
|  |
| # 遍历样本文件夹中的文件 |
| for root, dirs, files in os.walk(sample\_folder): |
| for file\_name in files: |
| file\_path = os.path.join(root, file\_name) |
| try: |
| # 如果文件路径已经匹配过，跳过 |
| if file\_path in matched\_file\_paths: |
| continue |
|  |
| # 使用YARA规则扫描文件 |
| matches = rules.match(file\_path) |
|  |
| # 如果有匹配的规则，记录匹配的文件数量 |
| if matches: |
| matched\_files += 1 |
| matched\_file\_paths.add(file\_path) |
| except Exception as e: |
| pass |
|  |
| # 计算扫描时间 |
| end\_time = time.time() |
| scan\_time = end\_time - start\_time |
|  |
| # 输出结果 |
| print("样本文件夹中的文件数量：{}".format(len(files))) |
| print("匹配的文件数量：{}".format(matched\_files)) |
| print("扫描时间：{:.2f} 秒".format(scan\_time)) |
| # 输出匹配的文件路径 |
| print("匹配的文件路径：") |
| for file\_path in matched\_file\_paths: |
| print(file\_path) |



1. **IDA Python**



打印所有段名称、段起始地址、段结束地址。



1. **实验结论及心得体会**
2. **实验结论**

Lab07-01: 这个程序创建服务MalService,来保证每次在系统启动后运行。使用用户代理Internet Explorer 8.0,并和www.malwareanalysisbook.com通信。等待直到2100年1月1日的0:00，发送许多请求到http://www.malwareanalysisbookcom/,大概是为了对这个网站进行一次分布式拒绝服务(DDoS)攻击。这个程序永远不会完成。它在一个定时器上等待直到2100年，到时候创建20个线程，每一个运行一个无限循环。

Lab07-02:

这个程序没有完成持久化驻留,它运行一次然后退出。给用户显示一个广告网页后完成执行。

Lab07-03:通过写一个DLL到C:\Windows\System32,并修改系统上每一个导入它的.exe文件，来达到持久化驻留。通过硬编码来使用文件名kerne132.dll,使用一个硬编码命名为SADFHUHF的互斥量。这个程序的目的是创建一个很难删除的后门，来连接到一个远程主机。这个后门有两个命令：一个用来执行命令，一个用来睡眠。但这个程序很难被删除，因为它感染系统上的每一个.exe文件。

1. **心得体会**

静态动态分析的重要性：实验中使用IDAPro等工具进行静态和动态分析是深入了解恶意代码的关键。通过反汇编和代码分析，可以揭示恶意代码的内部结构、功能和行为。

恶意代码多样性：实验中的不同样本展示了恶意代码的多样性。它们采用不同的策略和行为，包括检查网络连接、下载网页、修改文件系统和注册表等。了解这些多样性有助于更好地理解恶意代码的复杂性。

Yara规则的应用：编写Yara规则是一种强大的手段，用于检测恶意代码的存在。通过识别特定的特征字符串和行为模式，可以有效地检测和分类恶意代码样本。