

**《恶意代码分析与防治技术》课程实验报告**

**实验九**

****

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**

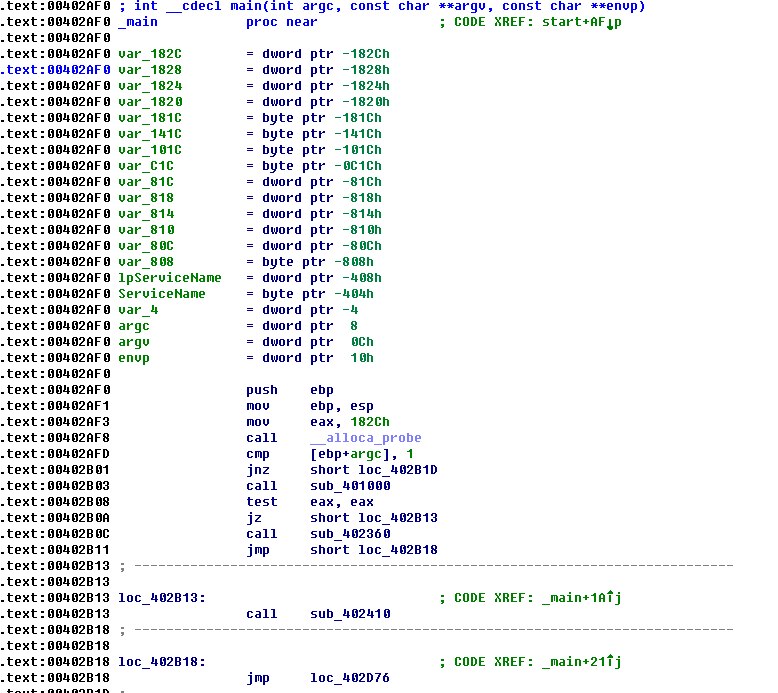
本次实验的主要目的是分析和理解恶意代码的内部工作原理，包括它的命令行选项、密码验证、网络特征以及恶意操作，以帮助研究人员和安全分析师更好地理解和对抗此类威胁。

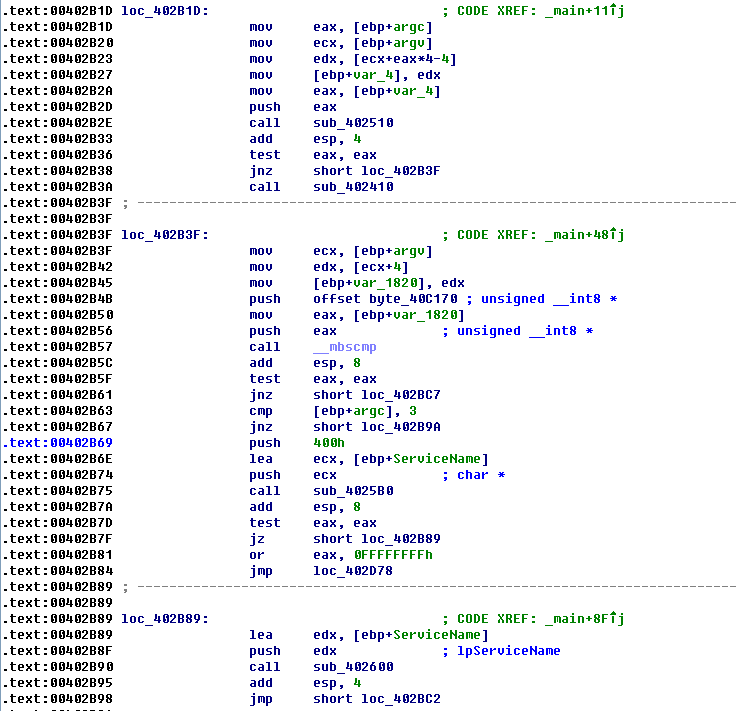
1. **实验原理**

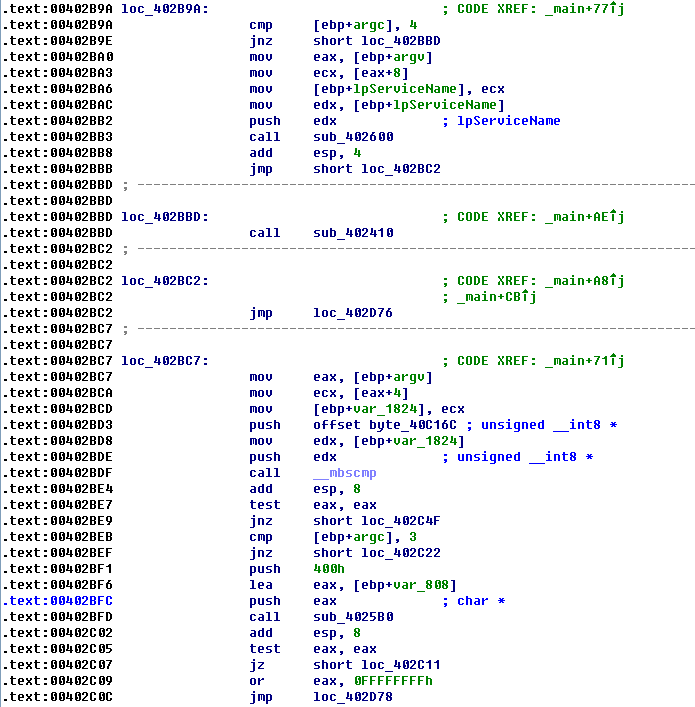
实验中使用了逆向工程工具IDA Pro和OllyDbg，以及汇编代码分析技术来深入了解恶意代码的行为和功能。

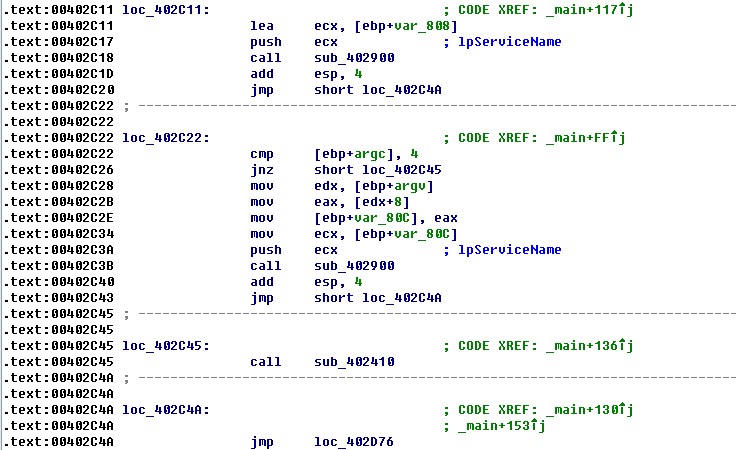
1. **实验过程**
2. **Lab9-1**

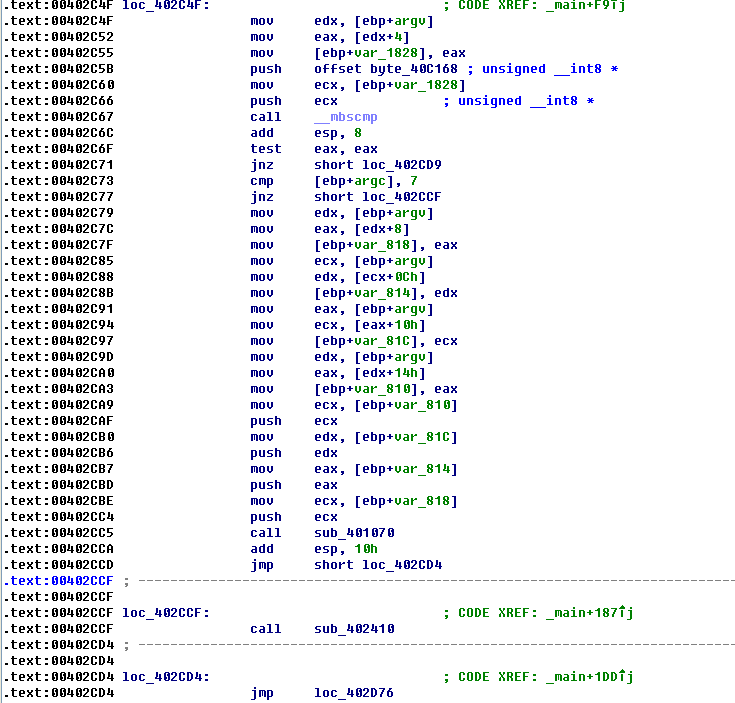
使用ida pro查看main函数

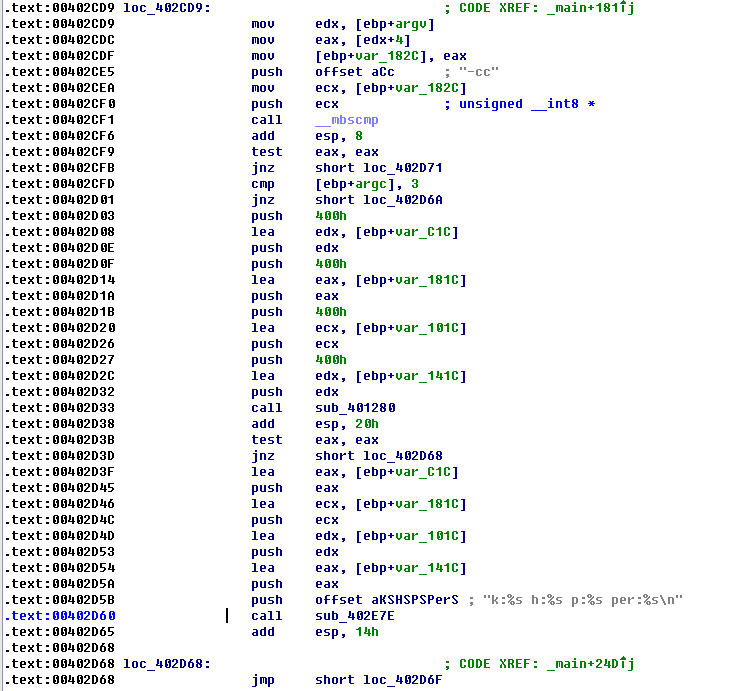


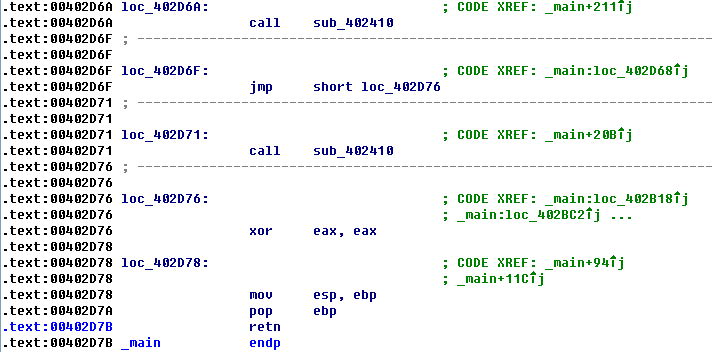




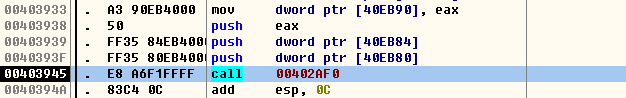




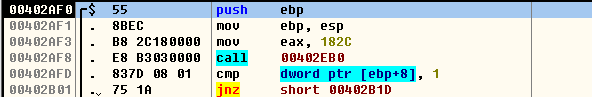


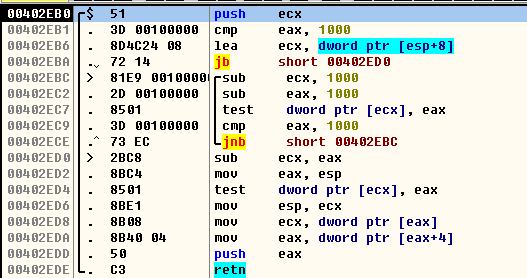


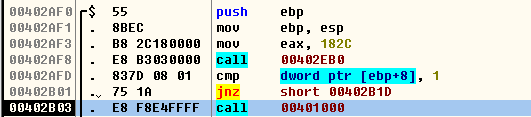
使用Ollydbg,F8键step-over,直至到达地址0x00403945,此处是对main函数的调用。



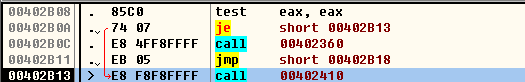
使用F7键step-into到main函数调用，恶意代码在地址0x00402AFD查看命令行参数的数量是否等于1。因为没有指定任何参数，所以检查成功，并且在地址0x00401000处恢复运行。



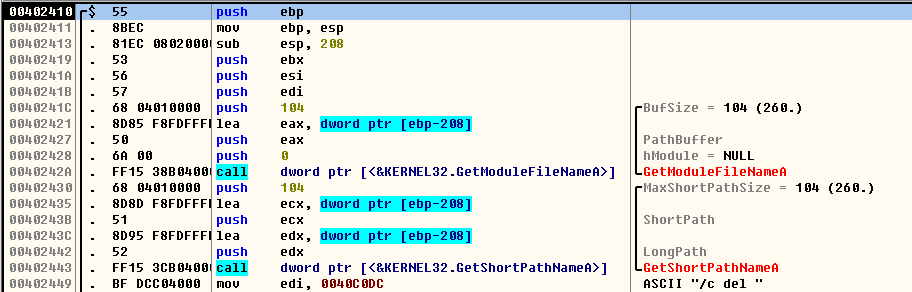


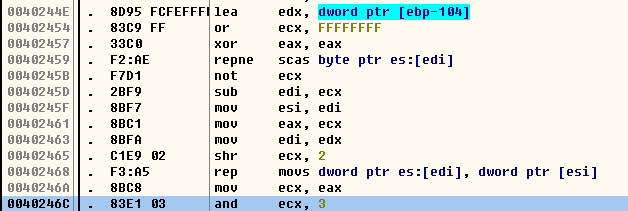


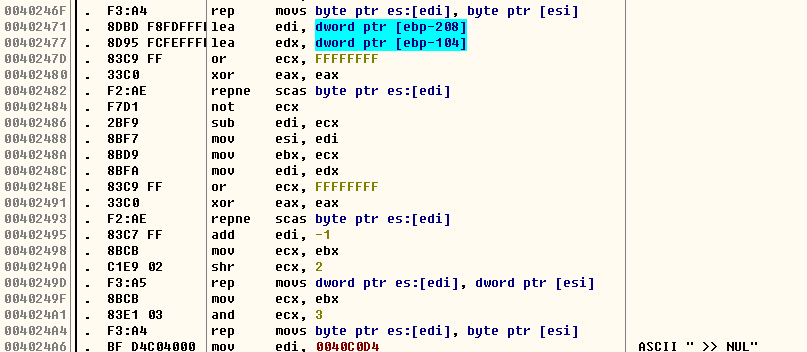
接下来，它试图打开注册表项HKLM\SOFTWARE\Microsoft \XPS,然而，由于注册表项不存在，函数返回0,所以调用进入0x00402410处的函数。



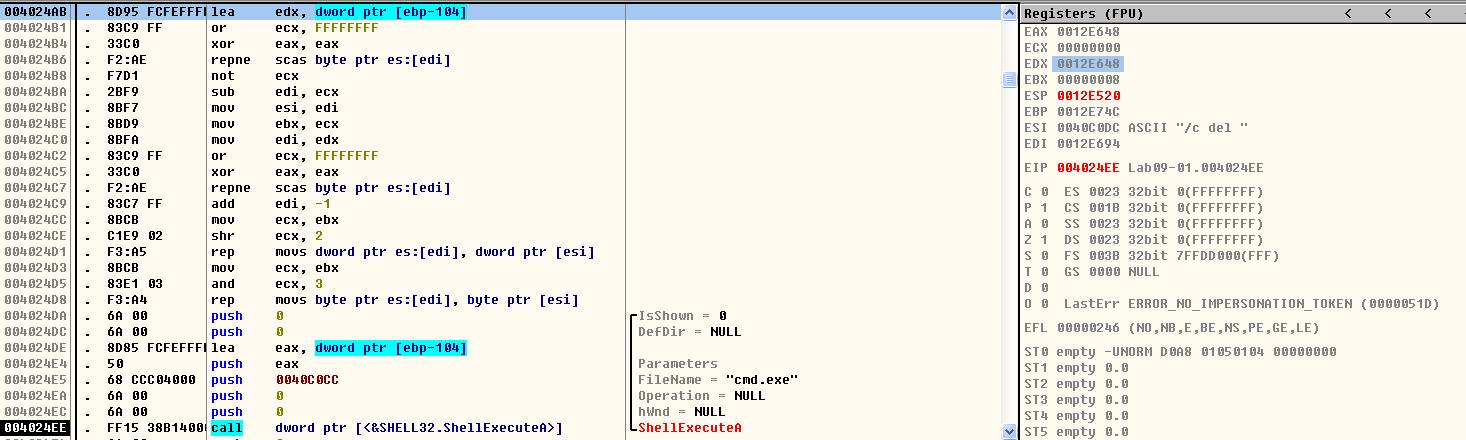
用GetModuleFilenameA获取当前可执行文件的路径，并构造出一个ASCII字符串：/c del path-to-executable >> NUL。



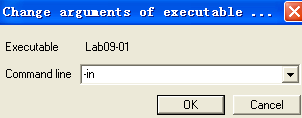




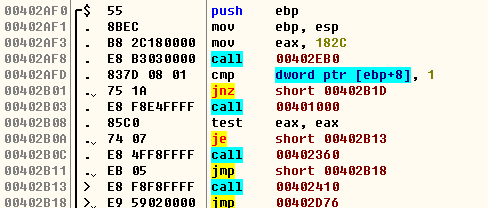
EDX寄存器值是0x0012E648,恶意代码通过在ShellExecuteA调用中结合构造的字符串和cmd.exe来尝试着从硬盘上删除自己。因为在OllyDbg打开了文件，所以Windows系统不允许删除这个文件。

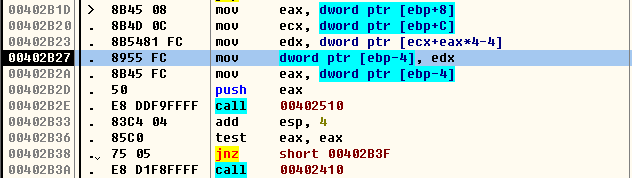


从字符串列表中选择任意一项，例如-in,并且使用它作为命令行参数，来测试恶意代码是否做了一些有意义的事情。为了做到这一点，选择Debug→Arguments,然后在OllyDbg参数对话框中添加参数-in。

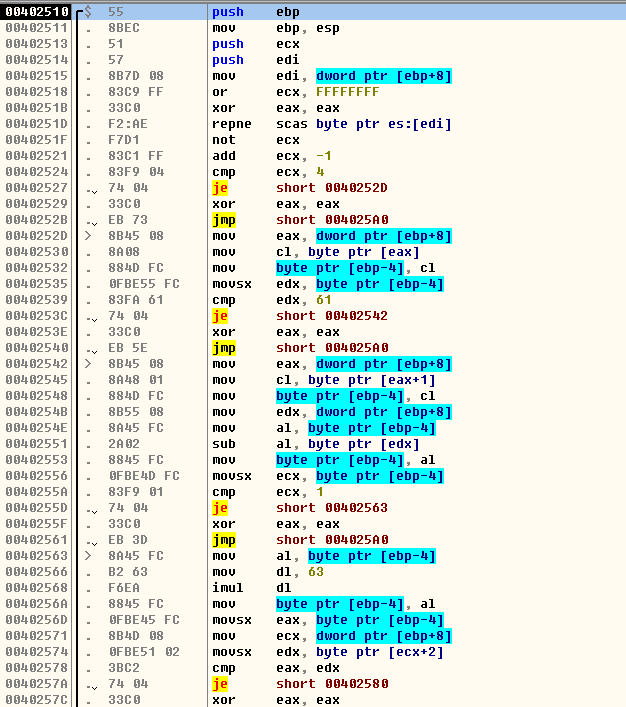


在命令参数接受检查之后，程序在0x00402B01地址处执行了跳转。argc变量，也就是传给程序字符串参数的个数，在栈帧高8字节处0x00402AFD,因为它是main函数的第一个参数。



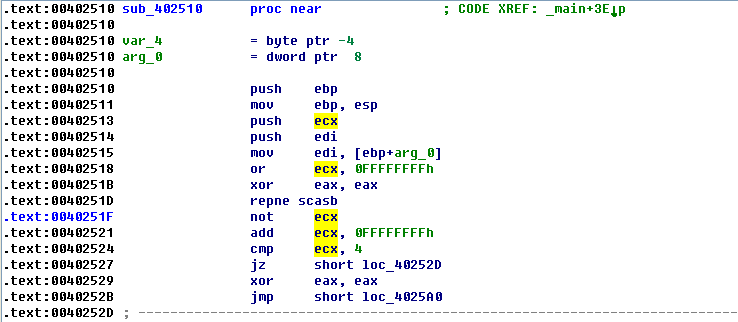


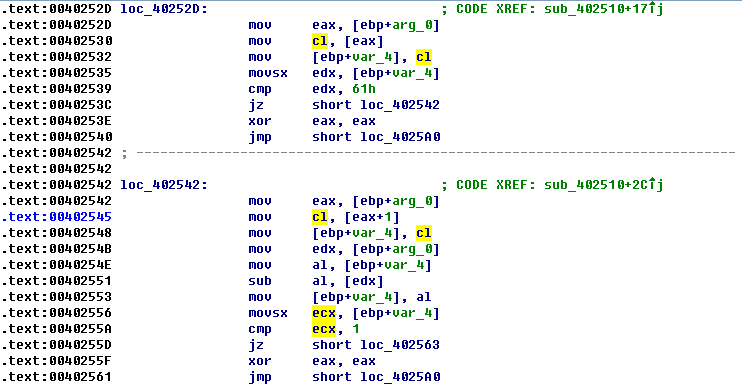
在地址0x00402B2E,最后一个命令行参数被传入到以地址0x00402510开始的函数。它是最后一个参数，因为标准C程序的main函数只带有两个参数：argc是参数的个数，而argv是指向命令行参数的一个数组指针。EAX包含argc,ECX包含argv。0x00402B1D和0x00402B20处的指令执行指针算法来选择命令行参数数组中的最后一个元素。这个指针最后在EAX中，并且在函数调用之前被压入到栈顶。

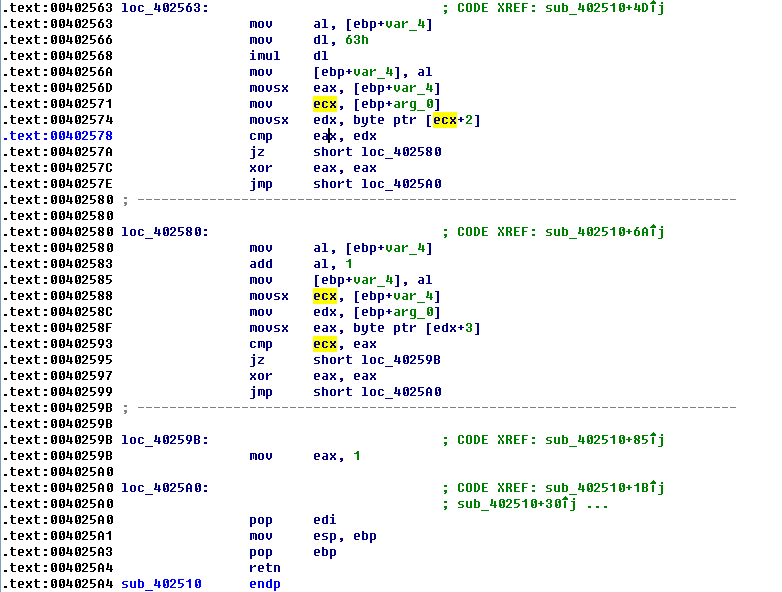


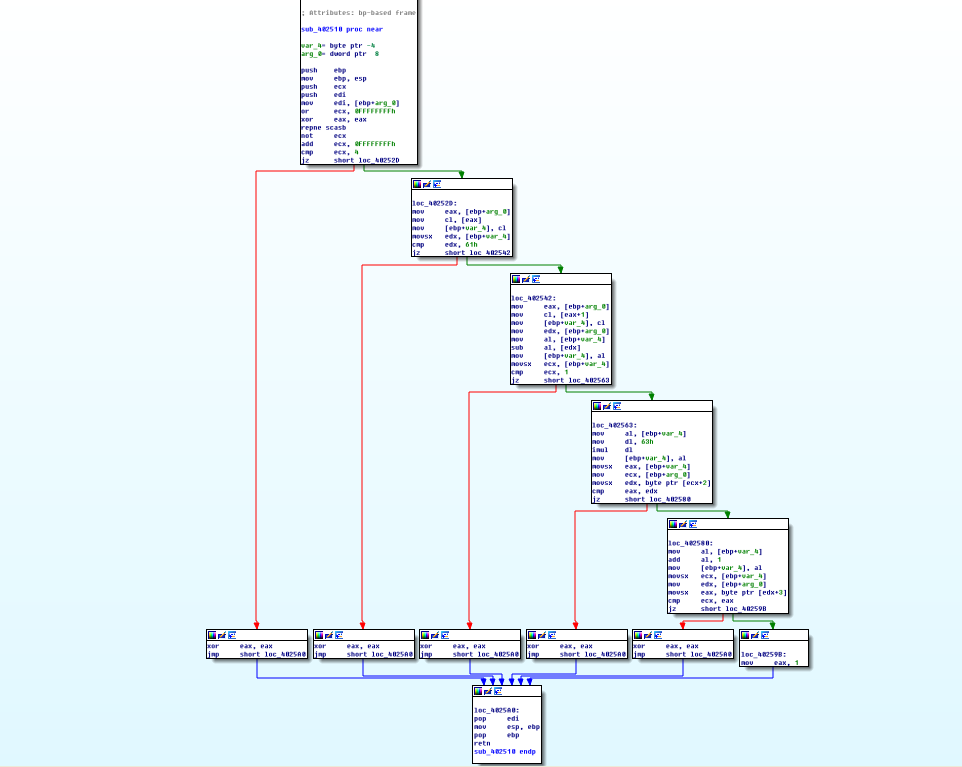
看到它使用算数操作ADD、SUB、MUL以及XOR字节大小操作数。这个例程(函数)看上去像使用一个令人费解、硬编码的算法进行输入的完整性检查。输入最有可能是某些类型密码或者代码。

对sub\_402510函数进行分析









密码是abcd。

观察到一个内联(inline)函数从地址0x0040251B～0x00402521调用了strlen。如果参数检查失败，EAX将被归零，并且在地址0x004025A0处cleanup函数恢复执行。只有正确的参数才能保证返回值1,但是我们可以使得不管参数如何，它在任何情况下都返回1。为了做到这一点，插入如下指令修补。

MOV EAX,0x1

RET

使用OllyDbg中的Assemble选项，对这些指令进行汇编，并得到6字节的指令序列：B801000000 C3。由于CALL指令准备堆栈，而RET指令负责清理栈，可以覆盖密码检查函数的开头指令，地址0x00402510处。要编辑指令，右键单击想修改的开始地址，选择Binary→Edit。

要取消Keep size复选框的勾选，在HEX+06域中输入十六进制的汇编值，然后单击OK按钮。OllyDbg会自动汇编，并且在相应的位置上显示出新的指令。

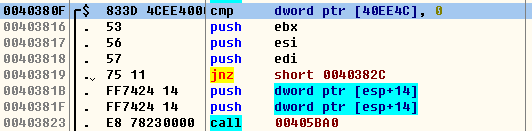




右键单击反汇编窗口，选择Copy to executable→All modifications来保存修改。接受所有对话框之后，将新版本保存为Lab09-01-patched.exe。

使用命令参数-in重新调试它。这次，恶意代码在地址0x00402510处成功通过了检查，并跳转到地址0x00402B3F。6条指令以后，指向第一个命令行参数的指针被压入栈中，紧邻着指向另外ASCII字符串(-in)的一个指针。

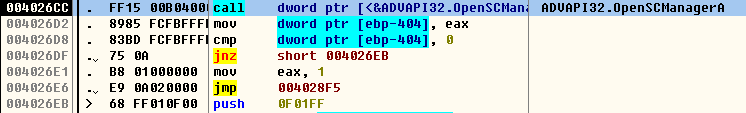
地址0x0040380F处的函数是\_\_mbscmp,字符串匹配函数，被IDA Pro FLIRT特征数据库识别出来了。恶意代码使用\_\_mbscmp检查列表支持的命令行参数选项，从而决定恶意代码的行为。



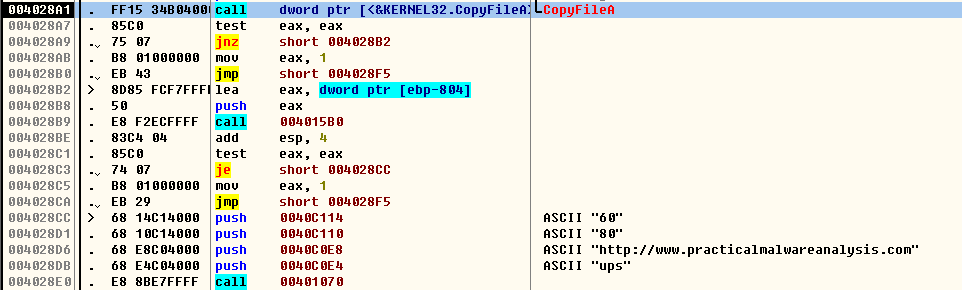
接下来，恶意代码检查提供的两个命令行参数，因为只提供了一个参数(-in),检查会失败，恶意代码再次试图删除自己。通过提供一个额外的命令参数，来通过这项检查。

在-in参数后面添加一个垃圾的命令参数，然后重启调试进程。恶意代码接受所有的命令行参数，并且执行它预期的行为。

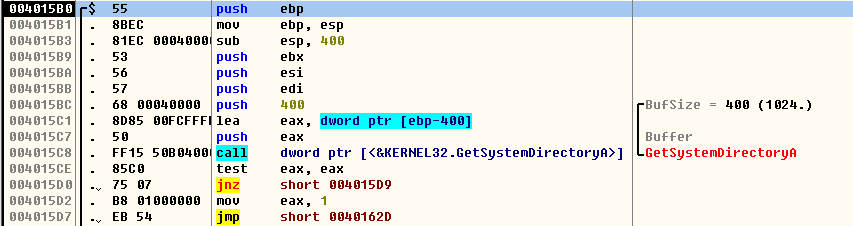
继续调试恶意代码，看到恶意代码使用一个与恶意代码可执行文件相同的basename,在地址0x004026CC处打开一个服务管理器。basename是去除目录路径和文件扩展名信息之后的文件名。如果服务不存在，则恶意代码以管理器服务basename作为名字，创建一个自启动的服务，并且设置二进制路径为%SYSTEMROOT%\system32\<filename>。



CreateServiceA被调用时堆栈调用的状态，包括：ASCII字符串的名字、描述以及路径。在地址0x004028A1处，恶意代码将自己复制%SYSTEMROOT%\system32\目录下。

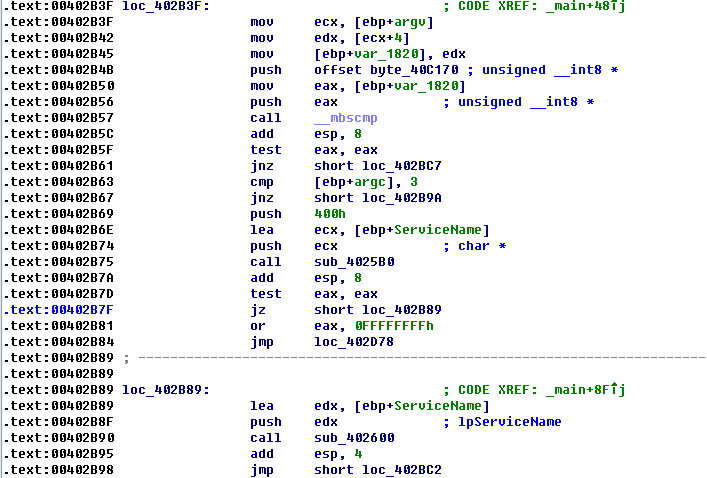


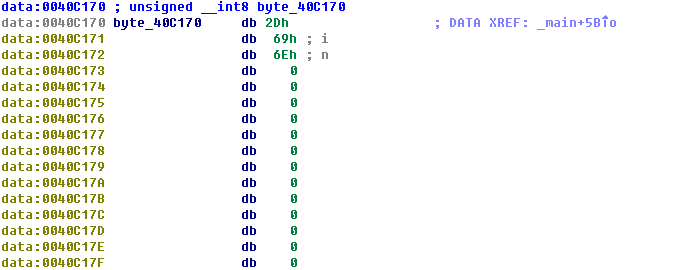
地址0x004015B0函数改变了复制文件的修改、访问和最后变化时间戳，来与那些kernel32.dll等系统文件保持一致。修改时间戳来和其他文件保持一致的技术称为timestomping。

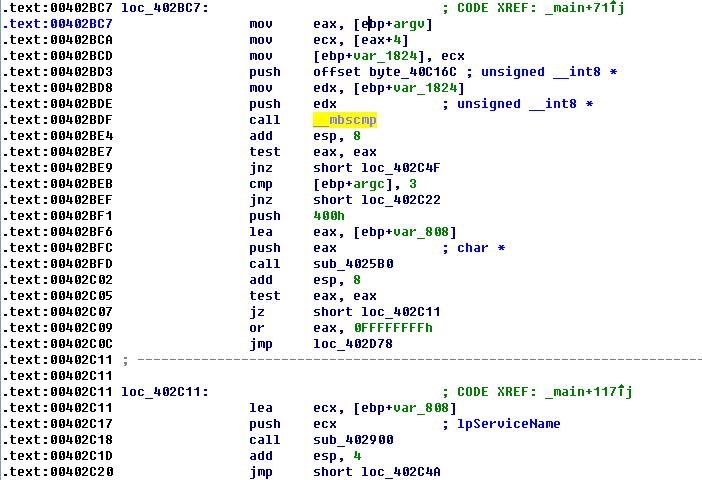


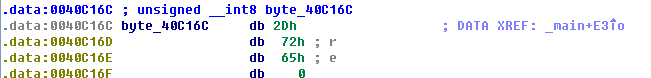
最后，恶意代码创建一个注册表项HKLM\SOFTWARE\Microsoft \XPS。Microsoft之后的空格使得它成为一个独特的主机感染标识。它在地址0x4011BE处，用EDX寄存器指向的缓冲区内容，来填充注册表下名为Configuration的键值。为了找出缓冲区的内容，我们在地址0x4011BE处设置断点，并且运行恶意代码(按F9键)。在寄存器面板窗口中，右键单击EDX寄存器的内容，选择Followin Dump。十六进制的转储视图显示了4个以NULL结束的字符串，随后是多个0，字符串包含值ups、http://www.practicalmalwareanalysis.com、80和60。这看起来像与具有网络能力恶意代码的相关配置数据。

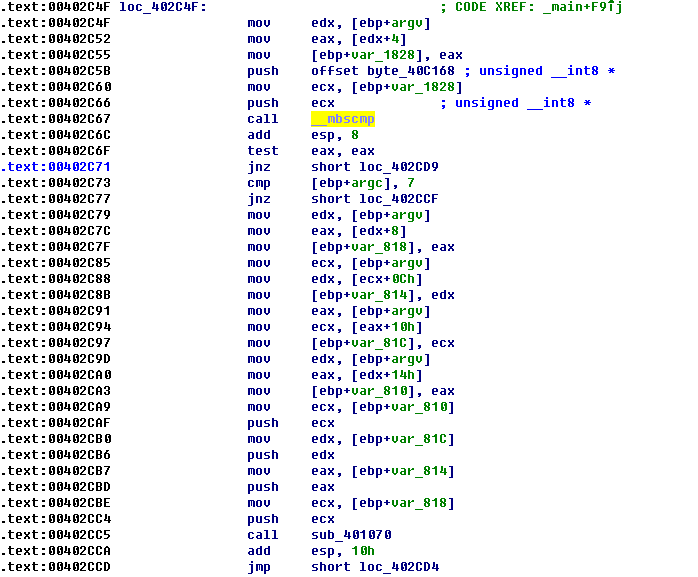
使用Ida pro在main函数中查找\_\_mbscmp调用来识别出支持的命令行选项。

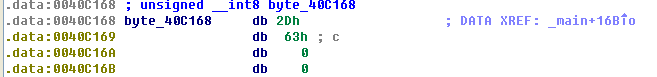


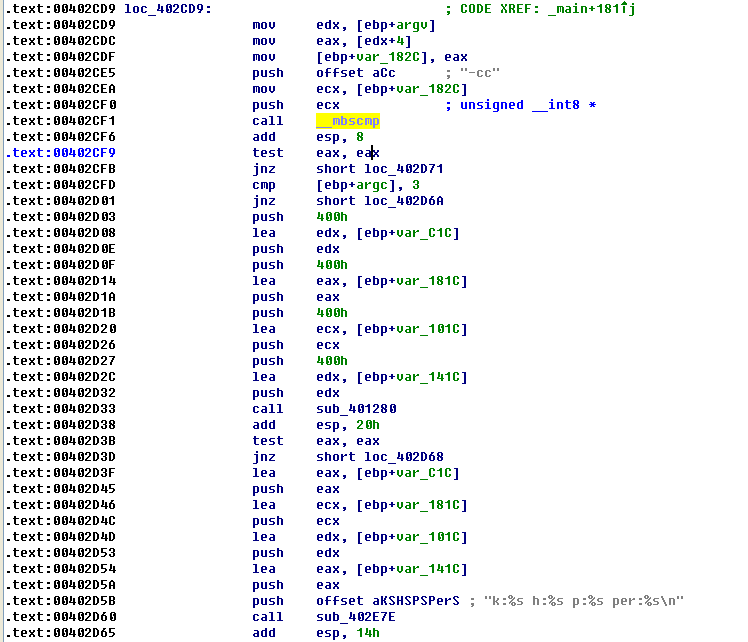


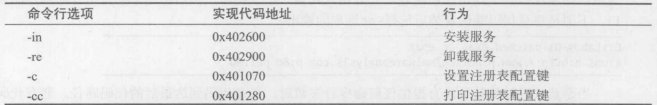




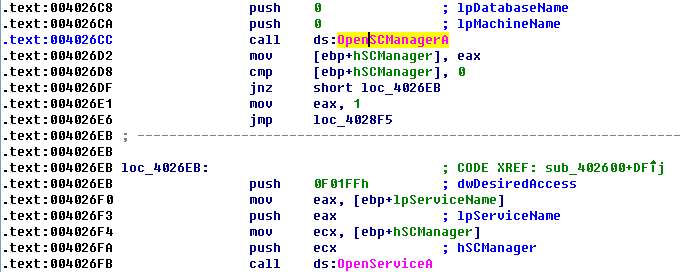


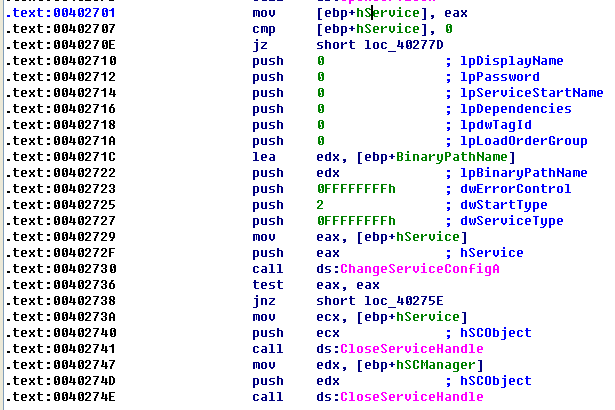


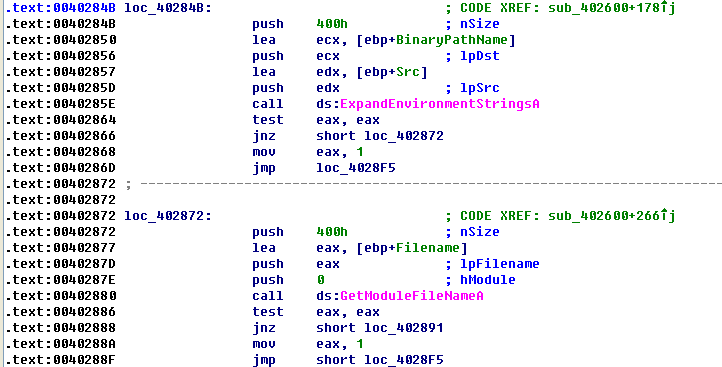




Sub\_402600检查系统根目录，构建服务的二进制路径，调用OpenSCManagerA函数，获取服务控制管理器的句柄，将句柄保存在hSCManager中。调用OpenServiceA函数，以获取服务的句柄，将句柄保存在hService中。如果服务不存在，它将使用CreateServiceA函数来创建一个新的服务，如果成功，将关闭服务句柄。



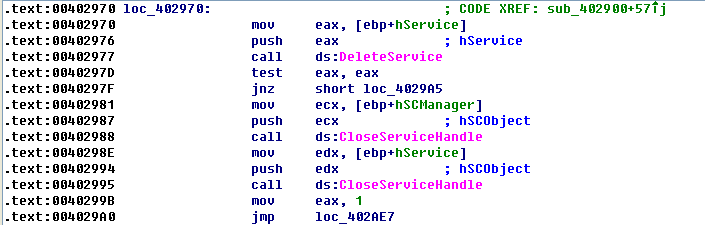


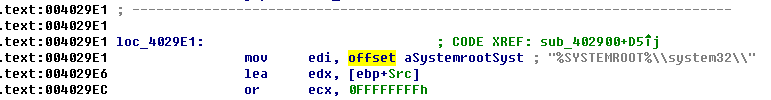




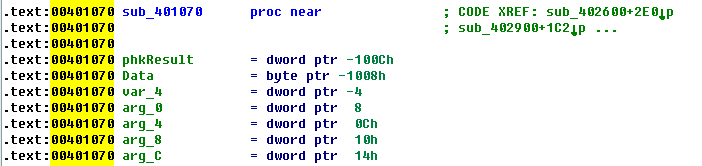
Sub\_402900使用OpenSCManagerA函数尝试打开服务控制管理器(SCM)，以获取对SCM的句柄,使用OpenServiceA函数尝试打开指定名称的服务，以获取对服务的句柄。定位安装恶意代码,然后使用DeleteService函数删除服务(0x00402977)。如果未能打开SCM或服务，或者删除服务失败，函数会继续执行，尝试删除服务的执行文件。它使用DeleteFileA函数来删除文件，并返回相应的结果。最后，它使用ExpandEnvironmentStringsA函数来展开一些环境变量，删除位于%SYSTEMROOT%\system32下的恶意代码备份，并且删除注册表中的配置值。

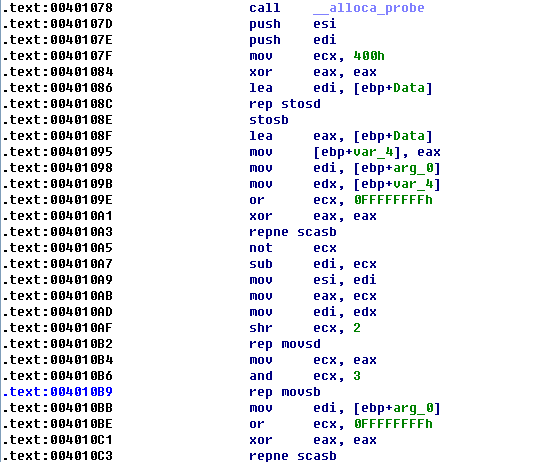


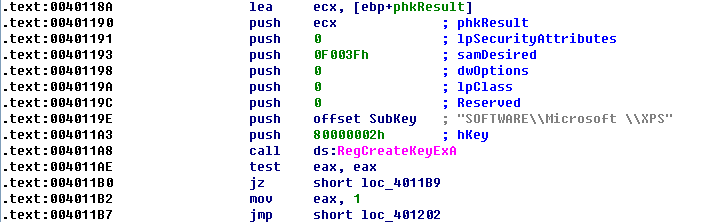


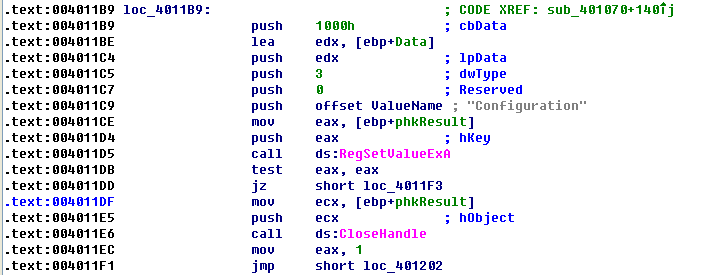


Sub\_401070在提供-c选项时才运行。这个函数带有4个参数，这个函数将它们连接在一起。使用rep stosd操作将一块内存区域（[ebp+Data]）初始化为0，这个区域的大小为0x400字节。复制传入的参数，这些参数包括四个用于指定注册表项的字符串，以及一个表示要创建的注册表项的句柄。使用字符串比较操作找到每个字符串的长度，并将它们存储在ecx中。然后，使用rep movsd和rep movsb操作，将这些字符串从一个内存位置复制到另一个内存位置。这些字符串的最终复制和拼接结果存储在[ebp+var\_4]中。调用RegCreateKeyExA函数，尝试创建一个指定名称的注册表项。函数将由此参数的缓冲区写入Windows注册表项HKLM\SOFTWARE\Microsoft \XPS下的Configuration键值中。提供-c选项的话，恶意代码允许用户更新Windows注册表中恶意代码的配置。



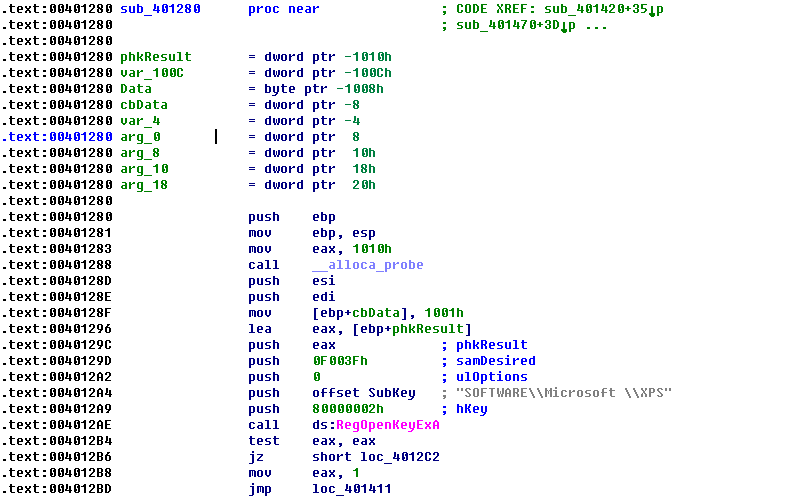


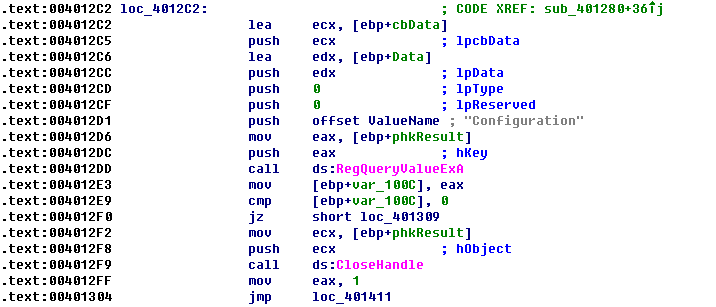




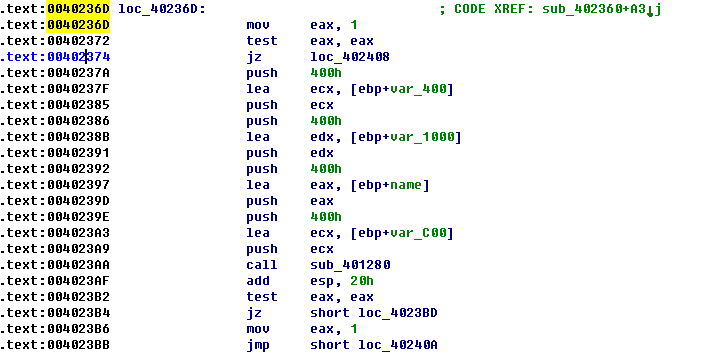
Sub\_401280在提供-cc选项时才运行。调用了RegOpenKeyExA函数，尝试打开指定名称的注册表项。检查RegOpenKeyExA的返回值，如果操作成功，它会将结果句柄存储在phkResult中，调用RegQueryValueExA函数，尝试查询注册表项中指定值的数据。检查RegQueryValueExA的返回值，如果操作成功，它会将查询到的数据的大小存储在lpcbData中。如果操作成功，函数会对查询到的数据进行一系列的处理，包括复制和拼接。它会将查询到的数据存储在Data缓冲区中。如果为恶意代码提供了-cc选项，则从注册表中读出当前配置，并且格式化为一个字符串，然后将这个字符串打印到控制台窗口。默认安装后选择-cc选项将会输出如下：

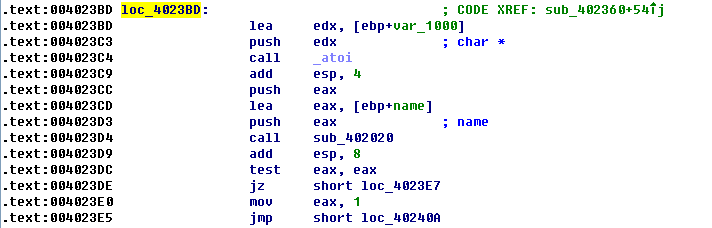
k:ups h:http://www.practicalmalwareanalysis.com p:80 per:60

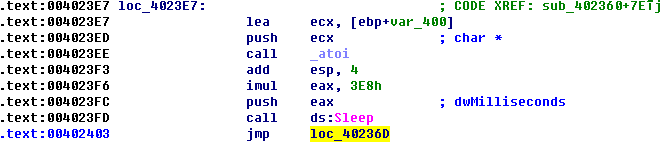


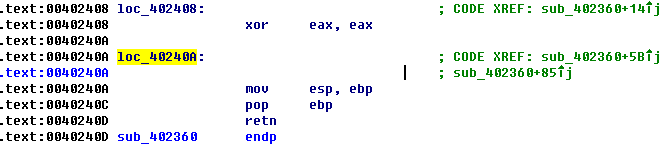


当恶意代码被安装且没有提供任何命令行参数时，恶意代码到达最后的代码路径。恶意代码通过确定注册表项是否被创建，来执行安装检查(0x00401000)。恶意代码执行的默认行为在地址0x00402360开始的函数中被发现。0x00402403处跳回到0x0040236D,0x040236D表示一个循环，且3个退出条件(地址0x004023B6、0x004023E0和0x00402408)直接导致程序终止。恶意代码获取当前的配置，调用一个函数，睡眠60秒，然后一直重复这个动作。



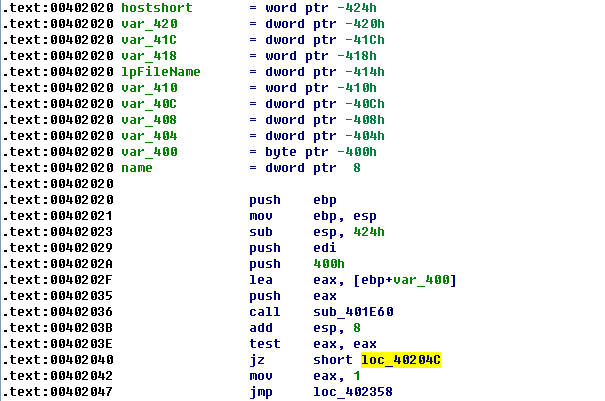


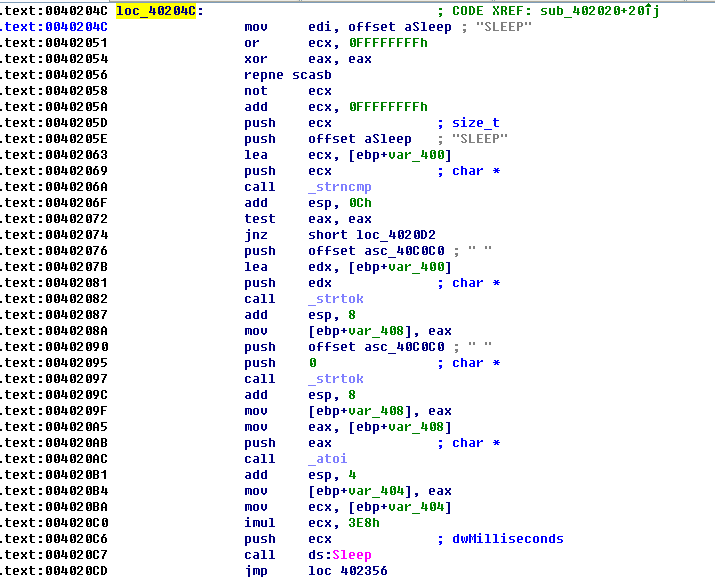




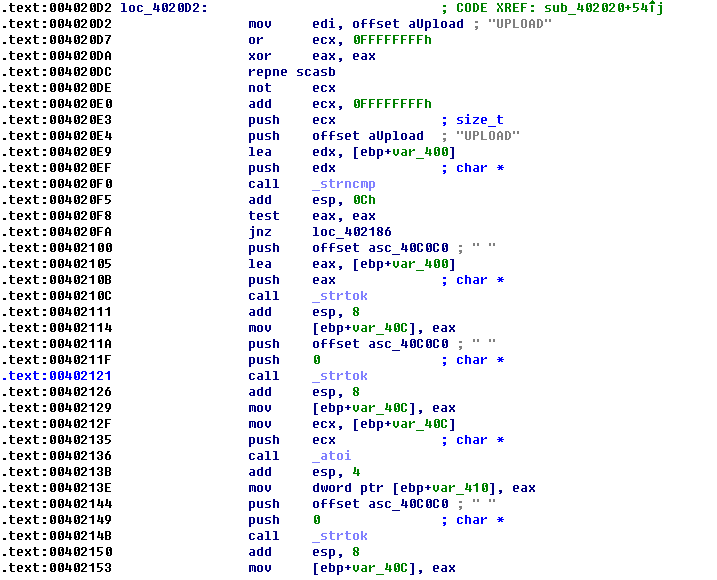
函数sub\_402020调用sub\_41E60后，将返回的字符串开头位置与支持的值列表进行比较：SLEEP、UPLOAD、DOWNLOAD、CMD和NOTHING。如果恶意代码碰到这些字符串之一，则它调用一个函数来响应这个请求。

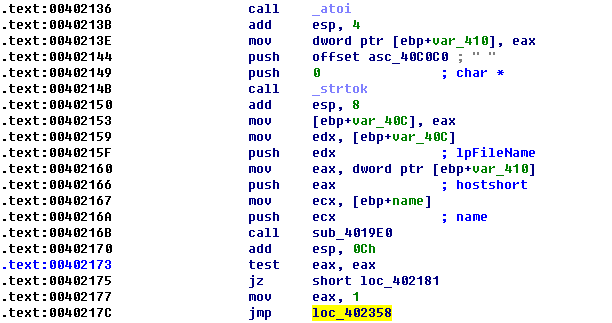
SLEEP:





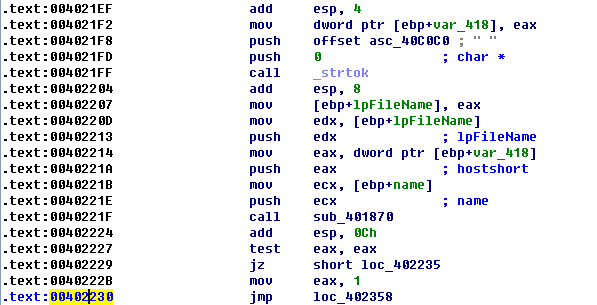
UPLOAD:



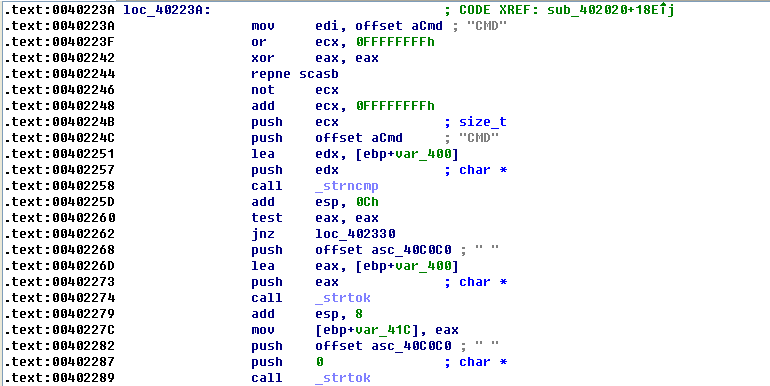


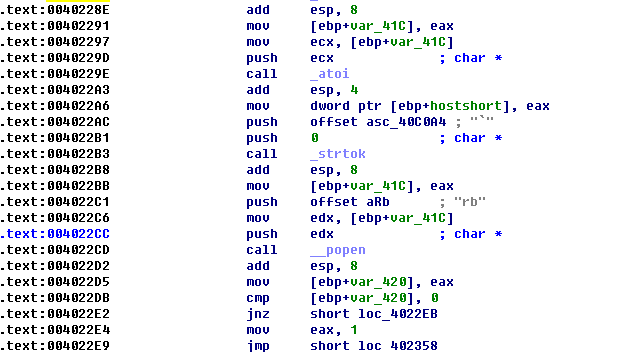
DOWNLOAD:



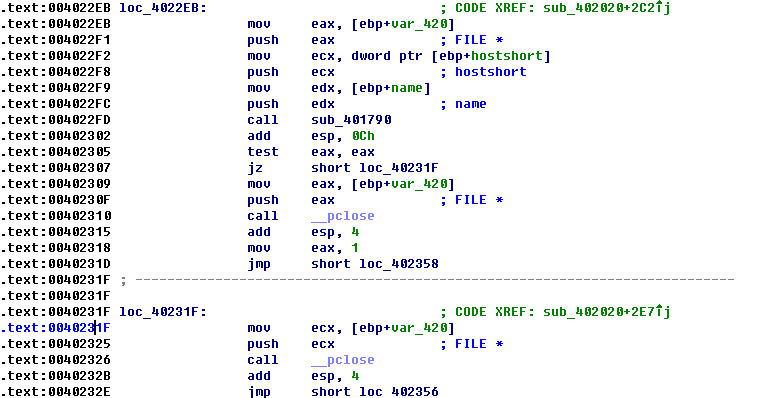


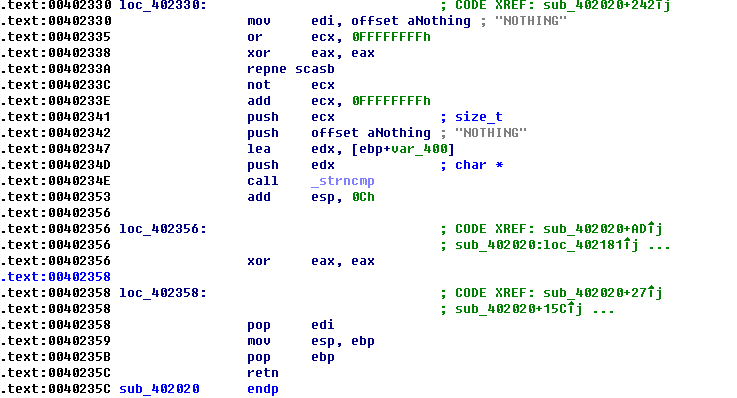
CMD:





NOTHING:

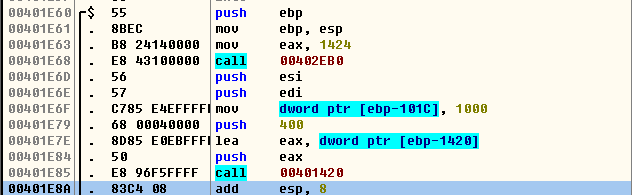




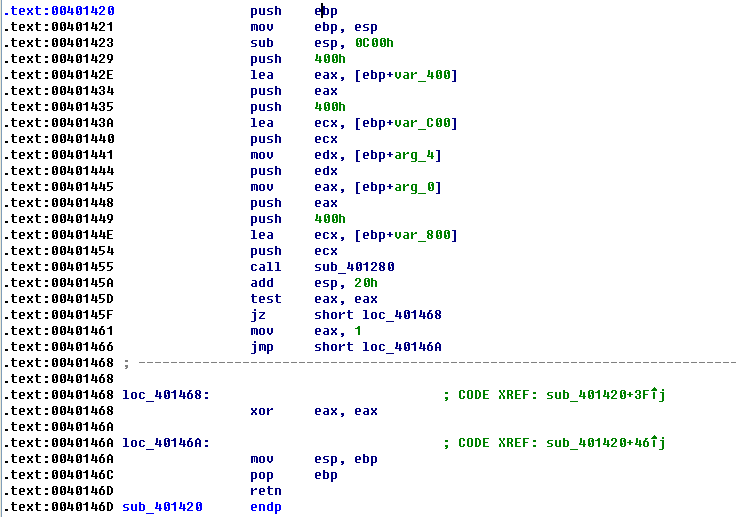
使用OllyDbg调试这个代码路径。做此操作之前，使用-cc运行恶意代码，并且保证恶意代码已经成功安装，如果恶意代码被安装，它会打印出当前的配置，如果没有，它会试图删除自己。

在Ollydbg中删除所有保存的命令行参数，在0x00401E60处设置断点。

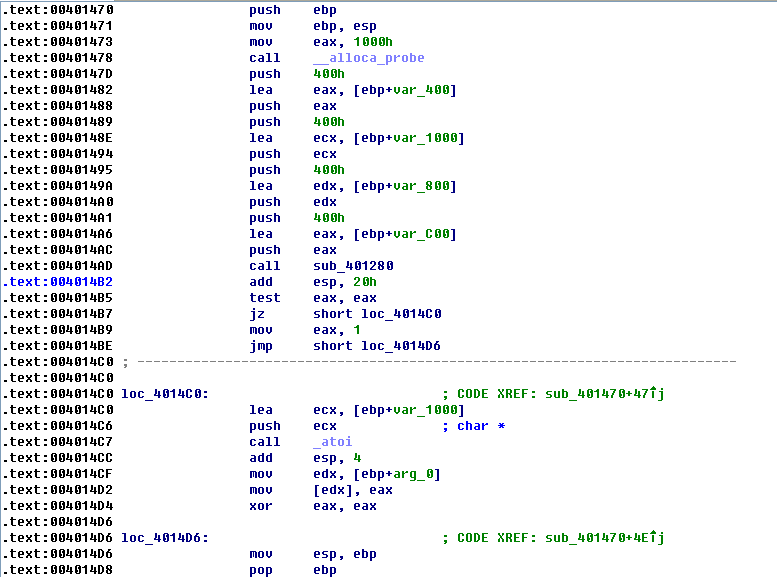
在地址0x401E85处的调用和它随后的指令(0x401E8A)等位置设置断点。第一个断点处，两个参数被压入到堆栈。在栈顶，看到了地址0x12BAAC,其次是整数0x400。



运行恶意代码(按F9键)到第二个断点。在以地址0x00401420开始的函数中，恶意代码将ASCII编码字符串http://www.practicalmalwareanalysis.com写入到缓冲区。推测这个函数从Windows注册表中获取某个特定的配置值，这个值在安装过程中被初始化，然后存入到缓冲区中。

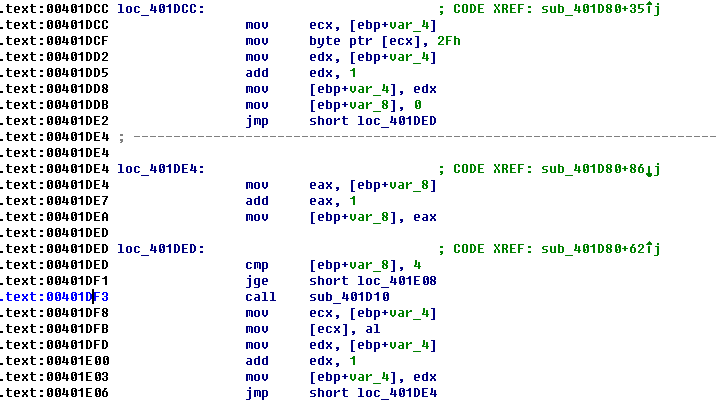


0x00401470开始的函数返回数字80(0x50),与0x401420开始的函数类似。这个字符串包含与位于http://www.practicalmalwareanalysis.com/服务器相关的端口号。

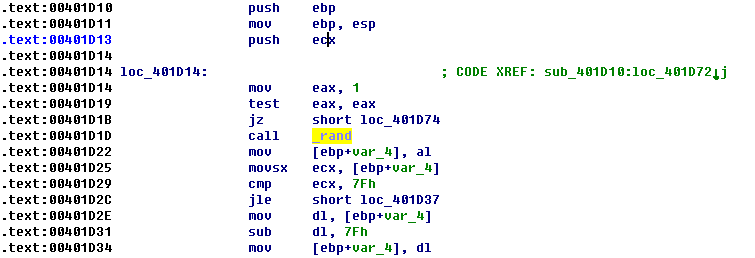


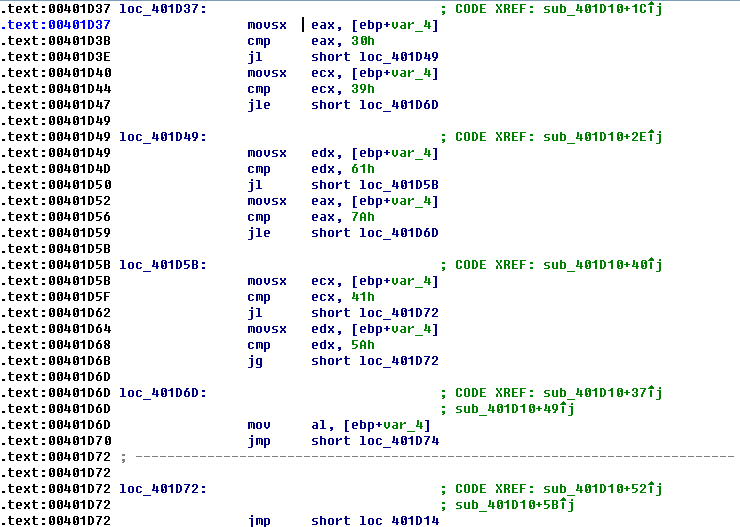
地址0x00401D80开始的函数返回一个包含随机字符的ASCII字符串。调试这个函数多次后，将会出现一个包含斜线(/，2Fh)和点(.，2Eh)字符的模式，返回的字符串可能对应一个类URL格式。当在独立的测试环境中分析恶意代码时，它会在下一个函数(开始地址是0x00401D80)中一直失败。











回到IDA Pro的反汇编视图中，恶意代码构造一个HTTP/1.0GET请求，并且连接一个远程系统。因为它是一个向外的合法HTTP请求，这种连接不会被防火墙拦截。恶意代码分析虚拟机禁用了网络，向外的连接永远不会成功，运行恶意代码就会失败。

跟踪反汇编列表，恶意代码尝试着连接注册表键值配置中记录的域名和端口。进一步分析反汇编代码显示，恶意代码在服务器返回的文档中搜索特殊的字符串(反引号、单引号、反引号、单引号、反引号),并且用它们来表示命令控制协议。

1. **如何让这个恶意代码安装自身?**

通过提供密码和-in选项，可以让程序安装自己。另外，可以通过修补二进制代码，来让程序跳过密码验证。

1. **这个恶意代码的命令行选项是什么?它要求的密码是什么?**

程序的命令行选项包括4个命令选项和一个密码。密码是字符串abcd,除了执行默认行为以外，所有的动作都需要输入密码。

-in选项通知恶意代码安装自己；-re选项通知恶意代码删除自己；-c选项通知恶意代码更新配置，包括恶意代码的标识IP地址；-cc选项通知恶意代码在控制台上打印当前的配置。默认情况下，一旦安装，这个恶意代码将成为一个后门程序。

1. **如何利用OllyDbg永久修补这个恶意代码，使其不需要指定的命令行密码?**

可以通过更改地址0x402510处函数开头的几个字节，让函数总是返回true,来修补这个二进制文件。完成这种行为的汇编指令是MOV EAX,0x1;RETN;,它对应的字节序列是B801000000 C3。

1. **这个恶意代码基于系统的特征是什么?**

恶意代码创建了注册表项：HKLM\Software\Microsoft \XPS\Configuration。恶意代码也创建了名为XYZ的管理服务。其中XYZ是安装时提供的一个参数，或者是恶意代码可执行文件的名字。最后，当恶意代码将自己复制到Windows系统目录时，它可能更改文件名为相匹配的服务名。

1. **这个恶意代码通过网络命令执行了哪些不同操作?**

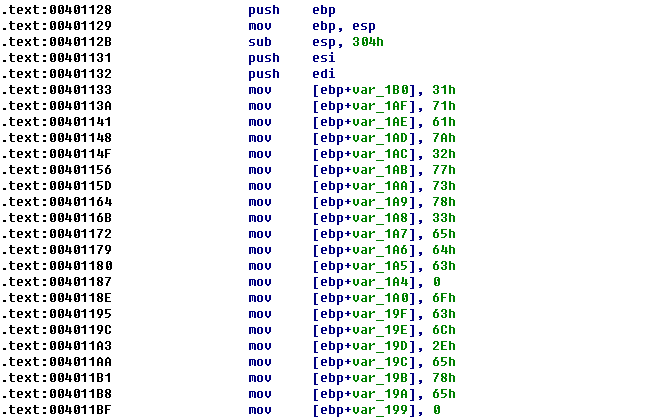
通过网络恶意代码可以被指示来运行下面5个命令之一：SLEEP、UPLOAD、DOWNLOAD、CMD或NOTHING。SLEEP指示恶意代码在给定的时间内不执行任何动作。UPLOAD命令则是让恶意代码从网络上读取一个文件，并且将这个文件写入到本地系统的指定路径下。DOWNLOAD命令指示恶意代码通过网络发送一个本地文件的内容到远程主机。CMD命令让恶意代码在本地系统中运行一个shell命令。NOTHING命令是一个空操作命令，它让恶意代码什么也不做。

1. **这个恶意代码是否有网络特征?**

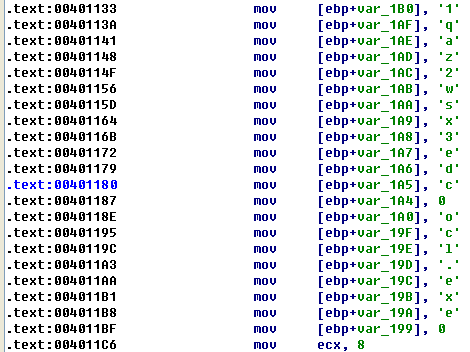
默认情况下，恶意代码向http://www.practicalmalwareanalysis.com/发出标识信号，然而这是可配置的，标识信号是对格式为xxxx/xxxx.xxx的资源的HTTP/1.0GET请求，其中x是一个随机的ASCII字符。恶意代码在它的请求中并不提供任何HTTP头部。

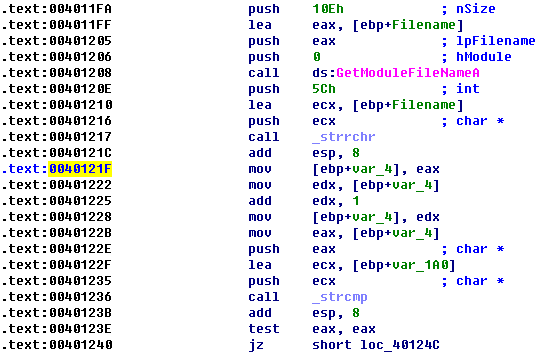
1. **Lab9-2**

查看main函数



这段代码在混淆字符串，通过一次将一个字符移入到堆栈，随后跟一个NULL终止符0x00401187和0x004011BF的方法来创建两个ASCII字符串。在0x4011C6处停止运行，右键单击EBP,选择Follow in Dump。通过滚动到第一个字符串[EBP-1B0],我们可以看到字符串1qaz2wsx3edc被创建。第二个字符串在[EBP-1A0]被创建并且被命名为ocl.exe。

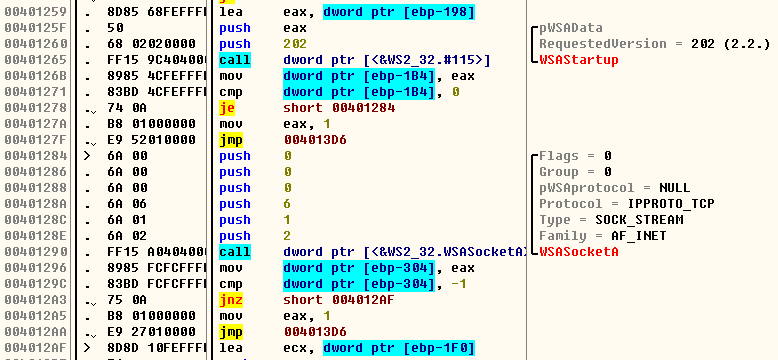




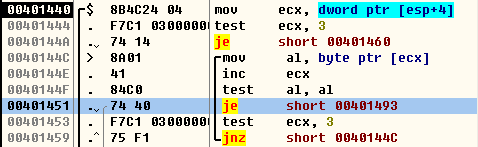
在0x401217处的调用上设置断点，看到两个参数被压入到堆栈。第一个是斜线，第二个GetModuleFileNameA调用的返回值(当前可执行文件的文件名)。恶意代码向后搜索斜线，试图获取当前正在运行的可执行文件的文件名。如果step-over\_strrchr调用，看到EAX指向字符串\Lab09-02.exe。

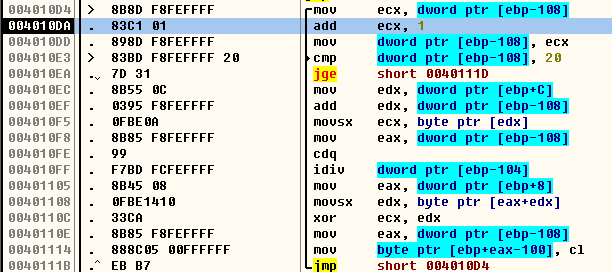
在0x401236处\_strcmp调用上设置一个断点来确定哪个字符串被用来比较。一旦断点命中，看到两个字符串被发送到\_strcmp。第一个是GetModuleFileNameA调用返回值的指针(考虑到斜线，在0x401225处加1),另外一个是oc1.exe(先前解密的字符串)。如果字符串匹配，EAX应该包含0。test eax,eax将会置位零标志位，然后跳到地址0x40124C处执行。如果条件为假，则程序退出。为了正确运行，恶意代码应该被命名为ocl.exe。

看到调用了WSAStartup和WSASocket导入函数，恶意代码将发生一些网络行为。



在0x401089处设置断点，该函数传入两个参数，一个是堆栈缓冲区（加密的字符串），一个是混淆字符串1qaz2wsx3edc。单步到0x401440处的调用，这个调用将密钥字符串传递给了strlen函数。它返回0xC并且将它移入[EBP-104]。接下来，[EBP-108]被初始化为0。OllyDbg已经注意到进程中一个循环，[EBP-108]是一个计数器，它在0x4010DA处被递加并且在0x4010E3处与0x20比较。随着循环的继续运行，密钥字符串通过一个idiv和mov的指令序列。



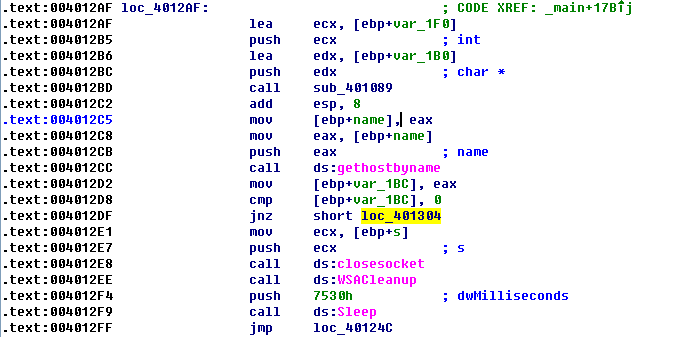


为了防止加密字符串长度大于密钥的长度，恶意代码使用模运算来让它在字符串上循环。在0x0040110C看到一个XOR指令。

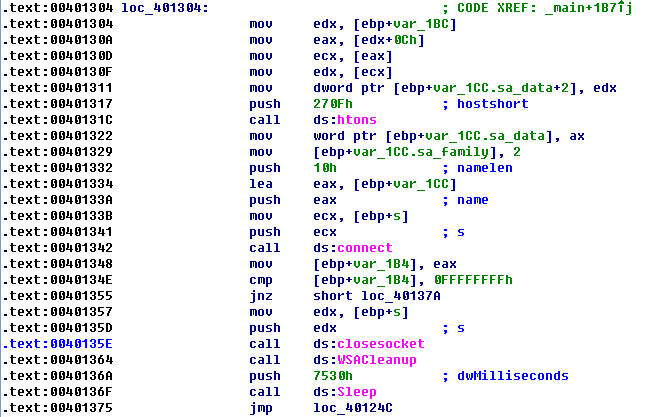
在0x4010F5上设置一个断点，EDX寄存器指向的值被移入到ECX,这也告诉了我们在函数后面执行异或操作的值。当我们在EDX寄存器上单击Follow in Dump,看到它是一个指向这个函数第一个参数的指针(加密的字符串)。我们在0x0040110C设置一个断点来查看循环中第一个迭代与什么值异或。EDX包含0x31(密钥字符串的第一个字节),并且再次看到ECX包含0x46。

多点几次play后，看到字符串www.prac。这可能是恶意代码试图通信的域名的开始字符串。继续运行，直到var\_108([EBP-108]，计数器变量)等于0x20。一旦执行了jge short 0x40111D指令后，最终放EAX寄存器的是字符串www.practicalmalwareanalysis.com(恰好它的长度等于0x20),然后函数返回main函数。这个函数使用字符串1qaz2wsx3edc的多字节异或循环来解密字符串www.practicalmalwareanalysis.com

主函数中EAX传给了调用gethostbyname。这个函数会返回一个IP地址并且填充sockaddr in结构。

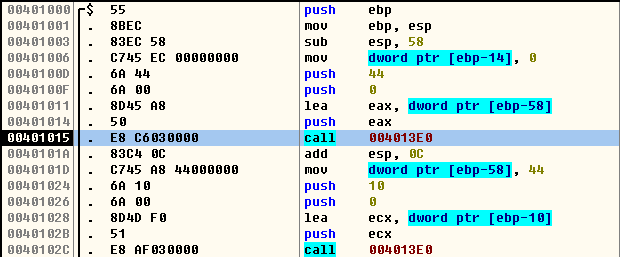


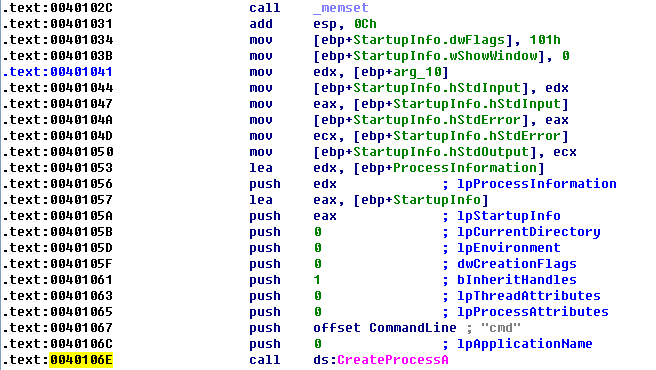
接下来，看到带有参数0x270f(十进制的9999)的ntohs调用。这个参数与0x2一起移入到sockaddr\_in结构，0x2表示sockaddr\_in结构中的AF\_INET(Internet套接字代码)。下一个调用将连接www.practicalmalwareanalysis.com上TCP端口9999和恶意代码。如果连接成功，恶意代码将会运行到0x0040137A。如果失败，恶意代码将会sleep 30秒，回到main函数的开始并且一直重复这个过程。可以使用NetCat和ApateDNS来将恶意代码连接到我们控制的一个IP上。



step-into到0x4013A9处的函数调用(step-into 0x401000),可以看到两个到0x4013E0的函数调用。这是OllyDbg不能识别系统调用memset的另外一个例子，然而IDA Pro能识别这个函数。接下来，在0x40106E处看到一个CreateProcessA调用，调用之前，一些结构被填充。







这是一个反向shell,传递给CreateProcessA的STARTUPINFO结构被修改。调用CreateProcessA后，它以隐藏窗口的方式运行cmd.exe,所以它对被攻击的用户不可见。在调用CreateProcessA之前，一个套接字被创建并且一个与远程服务器的连接被建立。这个套接字与cmd.exe标准的流绑定(stdin、stdout和stderr)。

修改STARTUPINFO结构后，传递参数给函数CreateProcessA。我们看到因为cmd.exe在0x00401067作为一个参数被传递，CreateProcessA将要运行cmd.exe。这个结构的成员wShowWindow在0x0040103B处被设置为SW\_HIDE,当cmd.exe启动时会隐藏窗口。在0x00401044、0x0040104A以及0x401050处，看到STARTUPINFO结构中的标准流被设置为一个套接字。这直接绑定套接字和cmd.exe的标准流，所以cmd.exe被启动后，所有经过套接字的数据都将发送到cmd.exe,并且cmd.exe产生的所有输出都将通过套接字发出。

1. **在二进制文件中，你看到的静态字符串是什么?**

静态出现在二进制文件中唯一有意义的字符串是导入函数和字符串cmd。

1. **当你运行这个二进制文件时，会发生什么?**

它没有做太多事情就终止了。

1. **怎样让恶意代码的攻击负载(payload)获得运行?**

运行它之前将它命名为ocl.exe。

1. **在地址0x00401133处发生了什么?**

一个字符串在栈上被创建，攻击者利用它来混淆静态分析技术和字符串工具中的字符串。

1. **传递给子例程(函数)0x00401089的参数是什么?**

传入到子例程0x401089的参数是字符串1qaz2wsx3edc和一个数据缓冲区。

1. **恶意代码使用的域名是什么?**

恶意代码使用的域名是practicalmalwareanalysis.com。

1. **恶意代码使用什么编码函数来混淆域名?**

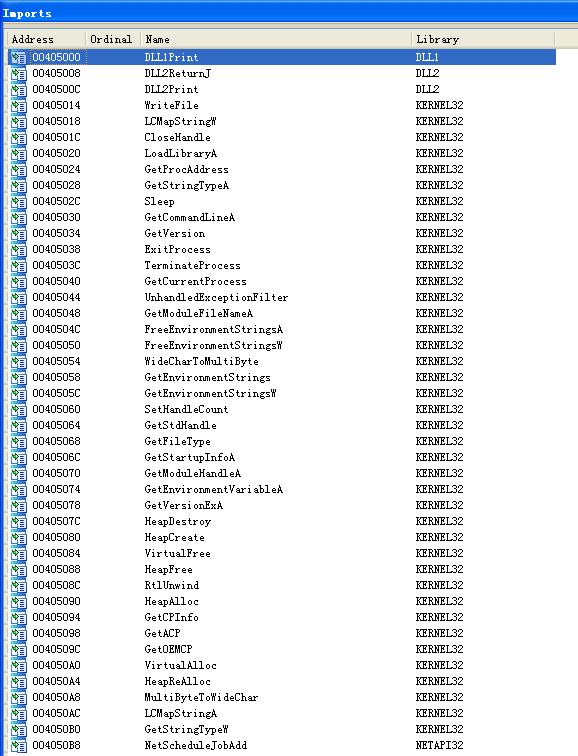
恶意代码用字符串1qaz2wsx3edc异或加密的DNS名来解密域名。

1. **恶意代码在0x0040106E处调用CreateProcessA函数的意义是什么?**

恶意代码设置stdout、stderr和stdin的句柄到socket(被用在CreateProcessA的STARTUPINFO结构中)。由于用cmd作为CreateProcessA的参数调用CreateProcessA,因此通过绑定一个套接字与命令shell来创建逆向shell。

1. **Lab9-3**

查看导入表

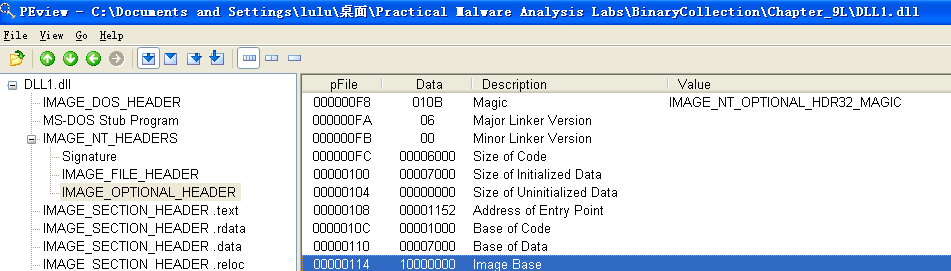


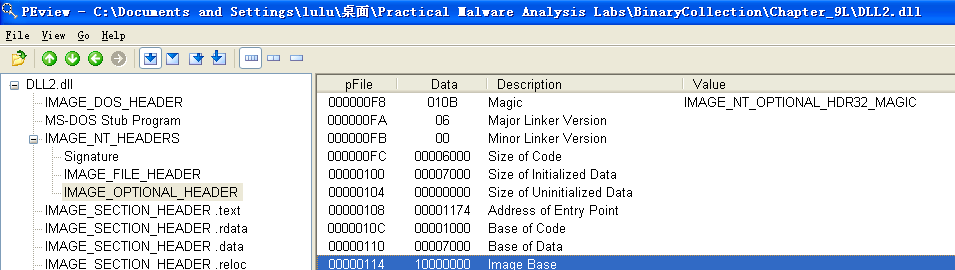
导入表包括：kernel32.dll、NetAPI32.dll、DLL1.dll和DLL2.dll。查找LoadLibrary调用并且检查在调用它之前哪个字符串被压入到了堆栈。看到LoadLibrary的两个交叉引用：分别压入user32.dll和DLL3.dll,所以这些DLL可能在运行过程中被动态地加载。

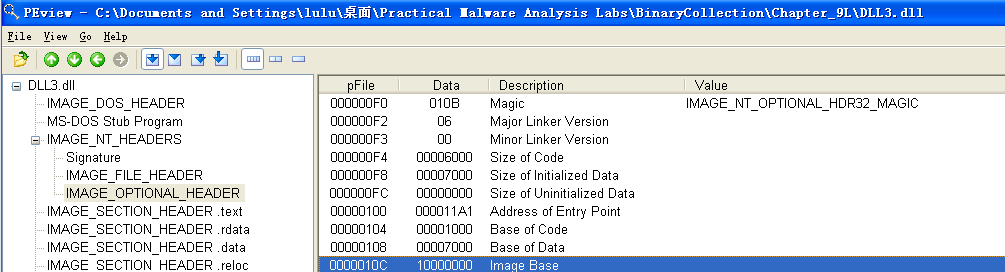




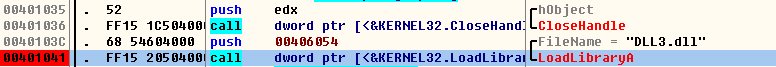
通过PEview检查DLL请求的基准地址。将DLL1.dIl、DLL2.dll、DLL3.dll加载到PEview，单击IMAGE\_OPTIONAL\_HEADER,查看映像基地址的值都是0x10000000。



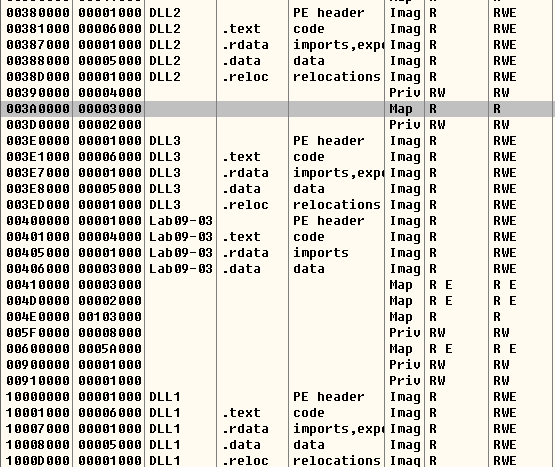




将Lab09-03.exe加载到OllyDbg中，在0x00401041设置一个断点，然后单击play。如果命中断点，F8 LoadLibrary调用。此时，三个DLL都被载入到Lab09-03.exe。

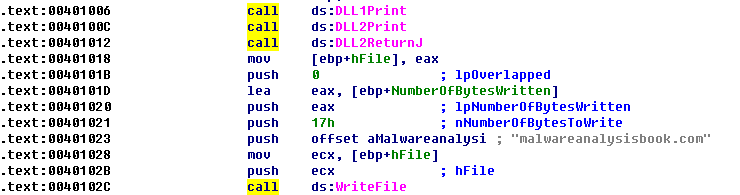


选择View→Memory弹出内存映射窗口。

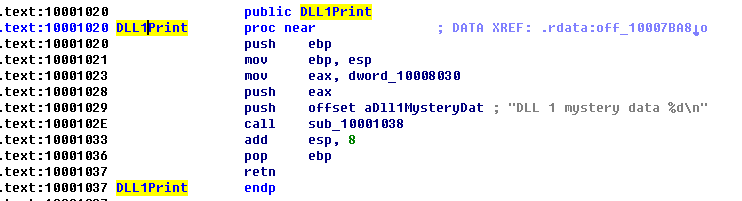


DLL1.dll被加载到0x10000000,DLL2.dll被加载到0x00380000,DLL3.dll被加载到0x003E0000。

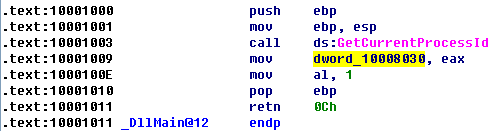
查看DLL1.dll和DLL2.dll导出的调用



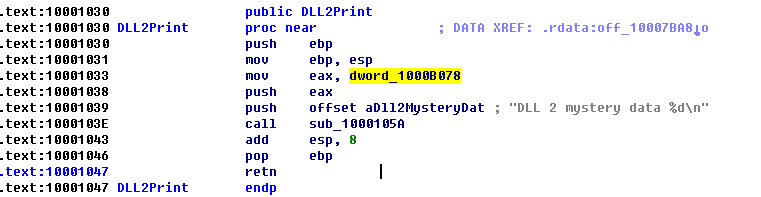
查看DLL1.dll，看到函数DLL1Print输出“DLL 1 mystery data”。



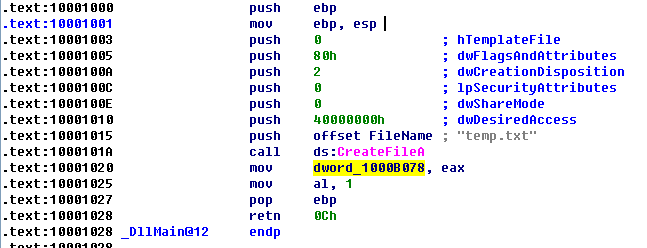
查看dword\_10008030的交叉引用，看到只有GetCurrentProcessId函数返回值移入该全局变量的时候，才会被DllMain访问。



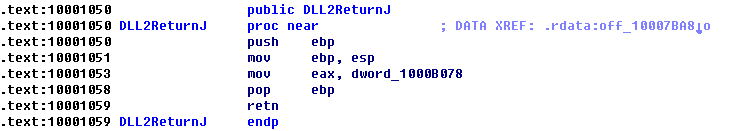
查看DLL2.dll，看到函数DLL2Print输出“DLL 2 mystery data”。



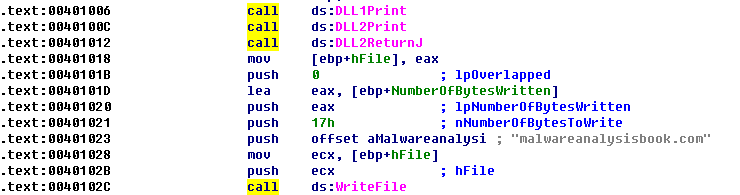
查看dword\_1000B078的交叉引用，看到只有CreateFileA函数返回值移入该全局变量的时候，才会被DllMain访问。CreateFileA打开temp.txt文件的一个文件句柄，如果文件不存在，这个函数会创建它。DLL2Print输出temp.txt的句柄值。



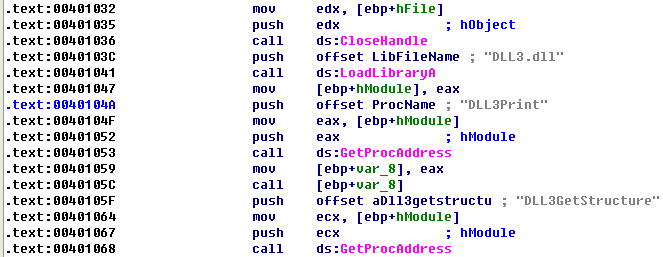
查看导出函数DLL2ReturnJ,发现它返回DLL2Print输出的句柄。



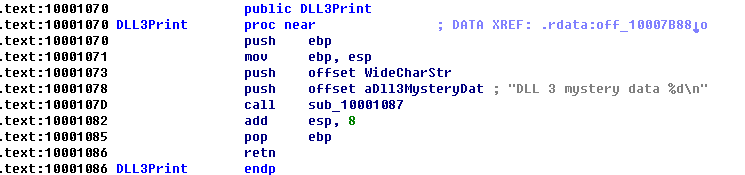
Lab10-03.exe的0x00401018处句柄被移入到hObject，在0x0040102B处被传给了WriteFile,malwareananalysisbook.com被写入到句柄标识的文件。



Lab09-03.exe中WriteFile之后，DLL3.dll被LoadLibrary加载，随后使用GetProcAddress动态解析DLL3Print和DLL3GetStructure。ida查看Lab09-03.dll和Lab09-03.exe。



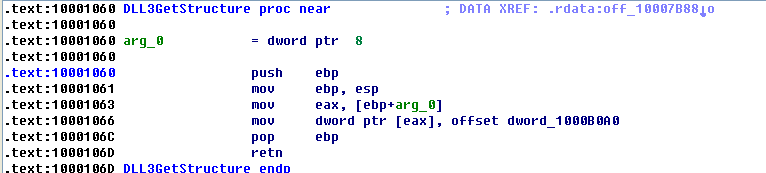
首先调用了DLL3Print,输出“DLL3 mystery data”。随后是全局变量，出现在0x1000B0C0处。



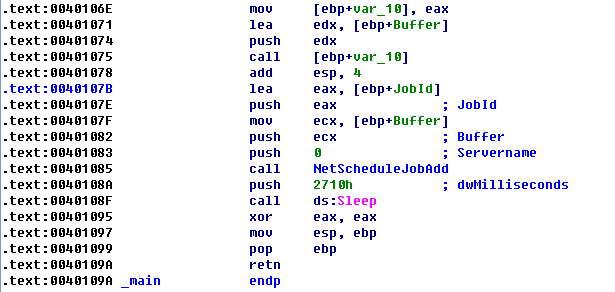
检查全局变量的交叉引用，它在D11Main中被初始化为www.malwareanalysisbook.com,所以字符串的内存地址会再次被输出。



DLL3GetStructure返回全局变量dword\_1000BOA0的一个指针，D1lMain似乎用数据和字符串初始化这个位置的某些结构体。

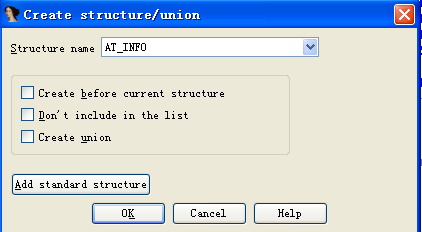


查看Lab09-03.exe如何使用数据描绘结构体的内容。

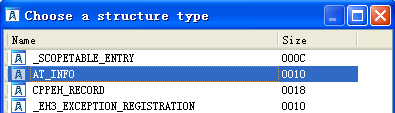


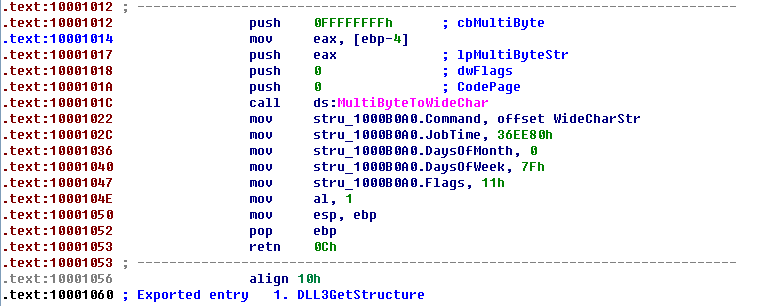
Buffer是一个指向AT\_INFO结构体的指针，随后它被传入到NetScheduleJobAdd。

在ida中打开Lab09-03.dll，在结构体窗口中按INSERT键并且添加标准的结构体AT\_INFO。

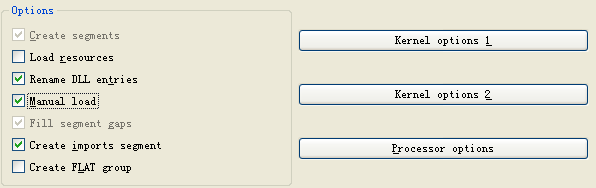


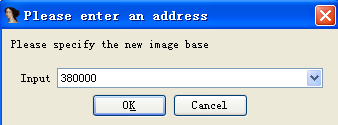
到达dword\_1000BOA0在内存中的位置。选择Edit→Struct Var,单击AT\_INFO。看到计划任务被设置为一周中的任意一天1:00AM ping malwareanalysisbook.com。





当DLL加载时，可以勾选手动加载框来将DLL2.dll载入到一个不同的位置。在Please specify the new image base域，输入380000。与OllyDbg加载DLL时一样，IDA Pro将对所有的偏移量做适当调整。







1. **Lab09-03.exe导入了哪些DLL?**

它的导入表包括：kernel32.dll、NetAPI32.dll、DLL1.dll和DLL2.dll。恶意代码动态加载的为user32.dll和DLL3.dll。

1. **DLL1.dll、DLL2.dll、DLL3.dll要求的基地址是多少?**

这三个DLL都要求相同的基准地址：0x10000000。

1. **当使用OllyDbg调试Lab09-03.exe时，为DLL1.dll、DLL2.dll、DLL3.dll分配的基地址是什么?**

DLL1.dll被加载到0x10000000,DLL2.dll被加载到0x00380000,DLL3.dll被加载到0x003E0000。

1. **当Lab09-03.exe调用DLL1.dll中的一个导入函数时，这个导入函数都做了些什么?**

DLL1Print被调用，它打印出"DLL 1 mystery data",随后是一个全局变量的内容。

1. **当Lab09-03.exe调用WriteFile函数时，它写入的文件名是什么?**

DLL2ReturnJ返回的文件名temp.txt被传入到WriteFile调用。

1. **当Lab09-03.exe使用NetScheduleJobAdd创建一个job时，从哪里获取第二个参数的数据?**

Lab09-03.exe从DLL3GetStructure中获取NetScheduleJobAdd调用的缓冲区，它动态地解析获得第二个参数的数据。

1. **在运行或调试Lab09-03.exe时，你会看到Lab09-03.exe打印出三块神秘数据。DLL1的神秘数据，DLL2的神秘数据，DLL3的神秘数据分别是什么?**

神秘数据1是当前进程的标识。神秘数据2是打开temp.txt文件的句柄，神秘数据3是字符串ping www.malwareanalysisbook.com在内存中的位置。

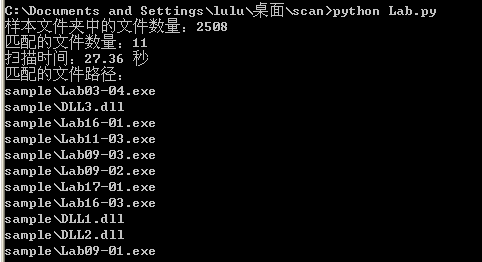
1. **如何将DLL2.dll加载到IDA Pro中，使得它与OllyDbg使用的加载地址匹配?**

当使用IDA Pro加载DLL时选择手动加载，当提示时，输入新的映像基准地址。本例中，基准地址是0x00380000。

1. **Yara**

根据特征字符串，编写Yara规则如下：

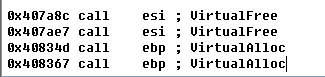
|  |
| --- |
| rule RuleforLab090101 |
| { |
| meta: |
| description = " Lab09\_01.exe" |
| strings: |
| $s1="http://www.practicalmalwareanalysis.com" fullword ascii |
| $s2="SOFTWARE\\Microsoft \\XPS" fullword ascii |
| $s3="Configuration" fullword ascii |
| $s4="SLEEP" fullword ascii |
| $s5="UPLOAD" fullword ascii |
| $s6="DOWNLOAD" fullword ascii |
| $s7="CMD" fullword ascii |
| $s8="NOTHING" fullword ascii |
| $s9="JanFebMarAprMayJunJulAugSepOctNovDec" fullword ascii |
| $s10="SunMonTueWedThuFriSat" fullword ascii |
| $s11="HTTP/1.0" fullword ascii |
| $s12="%SYSTEMROOT%\\system32\\" fullword ascii |
| $s13="runtime error" fullword ascii |
| $s14="TLOSS error" fullword ascii |
| $s15="SING error" fullword ascii |
| $s17="command.com" fullword ascii |
| $s18="COMSPEC" fullword ascii |
| $s19=" (8PX" fullword ascii |
| $s20="700WP" fullword ascii |
| $s21="ppxxxx"fullword ascii |
| $s22="(null)" fullword ascii |
| $s23=".com" fullword ascii |
| $s24=".bat" fullword ascii |
| $s25="-in" fullword ascii |
| $s26="-re" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 70KB and |
| all of them |
| } |
|  |
| rule RuleforLab090102 |
| { |
| meta: |
| description = " DLL1.dll" |
| strings: |
| $s1="JanFebMarAprMayJunJulAugSepOctNovDec" fullword ascii |
| $s2="SunMonTueWedThuFriSat" fullword ascii |
| $s3="DLL 1 mystery data"fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 60KB and |
| all of them |
| } |
|  |
| rule RuleforLab090201 |
| { |
| meta: |
| description = " Lab09\_02.exe" |
| strings: |
| $s1="runtime error "fullword ascii |
| $s2="TLOSS error" fullword ascii |
| $s3="SING error" fullword ascii |
| $s4="DOMAIN error"fullword ascii |
| $s6="GetLastActivePopup" fullword ascii |
| $s7="GetActiveWindow" fullword ascii |
| $s8="cmd" fullword ascii |
| $s9="- floating point not loaded" fullword ascii |
| $s10="- not enough space for stdio initialization" fullword ascii |
| $s11="- not enough space for \_onexit/atexit table"fullword ascii |
| $s12="abnormal program termination" fullword ascii |
| $s13="R6028" fullword ascii |
| $s14="Runtime Error!" fullword ascii |
| $s15="Program: "fullword ascii |
| $s16="- unexpected multithread lock error" fullword ascii |
| $s17="- pure virtual function call" fullword ascii |
| $s18="- unable to open console device" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 70KB and |
| all of them |
| } |
|  |
| rule RuleforLab090202 |
| { |
| meta: |
| description = " DLL2.dll" |
| strings: |
| $s1="JanFebMarAprMayJunJulAugSepOctNovDec" fullword ascii |
| $s2="SunMonTueWedThuFriSat" fullword ascii |
| $s3="DLL2Print" fullword ascii |
| $s4="DLL2ReturnJ" fullword ascii |
| $s5="temp.txt" fullword ascii |
| $s6="DLL 2 mystery data" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 60KB and |
| all of them |
| } |
|  |
| rule RuleforLab090301 |
| { |
| meta: |
| description = " Lab09\_03.exe" |
| strings: |
| $s1="DLL3GetStructure" fullword ascii |
| $s2="DLL3Print" fullword ascii |
| $s3="malwareanalysisbook.com" fullword ascii |
| $s4="DLL3.dll" fullword ascii |
| $s5="DLL1.dll" fullword ascii |
| $s6="DLL2.dll" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 70KB and |
| all of them |
| } |
|  |
| rule RuleforLab090302 |
| { |
| meta: |
| description = " DLL3.dll" |
| strings: |
| $s1="JanFebMarAprMayJunJulAugSepOctNovDec" fullword ascii |
| $s2="SunMonTueWedThuFriSat" fullword ascii |
| $s3="DLL3Print" fullword ascii |
| $s4="DLL3GetStructure" fullword ascii |
| $s5="DLL3.dll" fullword ascii |
| $s6="DLL 3 mystery data" fullword ascii |
| condition: |
| uint16(0) == 0x5a4d and |
| uint32(uint32(0x3c))==0x00004550 and filesize < 70KB and |
| all of them |
| } |



1. **IDA Python**

遍历所有函数，排除库函数或简单跳转函数，当反汇编的助记符为call或者jmp且操作数为寄存器类型时，输出该行反汇编指令

|  |
| --- |
| import idc |
| import idautils |
| for func in idautils.Functions(): |
| flags=idc.GetFunctionFlags(func) |
| if flags & FUNC\_LIB or flags & FUNC\_THUNK: |
| continue |
| dism\_addr = list(idautils.FuncItems(func)) |
| for line in dism\_addr: |
| m=idc.GetMnem(line) |
| if m=='call' or m=='jmp': |
| op=idc.GetOpType(line,0) |
| if op==o\_reg: |
| print '0x%x %s' % (line,idc.GetDisasm(line)) |



1. **实验结论及心得体会**

逆向工程分析是一种强大的方法，可以帮助安全专业人员深入理解和对抗恶意软件和威胁。通过逆向工程，可以揭示恶意代码的内部工作原理，识别其特征，理解其攻击方式，以及发现安全漏洞。这种技术对于网络安全研究和威胁分析非常重要。在进行逆向工程分析时，要谨慎处理样本，以确保不会对系统造成损害，并且遵守法律和道德准则。