

**计算机网络实验报告**

**实验十一**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2210977

姓 名 陈恩宝

班 级 信安班

1. **实验目的**

1. 复习教材和课件内第 14 章的内容。

2. 完成 Lab14 的实验内容。

1. **实验原理**

**1.IDA Pro**

IDA Pro（Interactive DisAssembler Professional）是一款由Hex-Rays开发的静态反汇编和调试工具，广泛应用于逆向工程、恶意软件分析和安全研究。它因其强大的功能、灵活的插件支持和用户友好的界面而受到许多安全研究人员和逆向工程师的青睐。

主要功能

静态分析：

IDA Pro可以对二进制文件进行静态反汇编，将机器代码转换为人类可读的汇编语言。

支持多种文件格式，包括PE（Windows可执行文件）、ELF（Linux可执行文件）、Mach-O（macOS可执行文件）等。

动态分析：

IDA Pro支持调试功能，可以与多种调试器集成，进行动态分析。

可以在运行时监视程序的行为，捕捉函数调用、变量变化等信息。

图形化界面：

提供友好的用户界面，支持图形化展示程序的控制流图（Control Flow Graph，CFG）和函数调用图，帮助用户更好地理解程序结构。

自动化分析：

通过自定义脚本（使用IDC或Python）来实现自动化任务，比如自动标记函数、识别字符串、分析数据结构等。

IDA Pro提供了丰富的API，用户可以创建插件以扩展其功能。

多种架构支持：

支持多种处理器架构，包括x86、x64、ARM、MIPS、PowerPC等，适用于不同类型的二进制文件分析。

反编译器：

IDA Pro的Hex-Rays Decompiler插件可以将汇编代码反编译为C语言，简化代码阅读和理解。

代码注释和标记：

用户可以在反汇编结果中添加注释、标记和自定义数据结构，便于后续分析和团队协作。

**2.IDA Python**

IDA Python是IDA Pro的一个功能强大的扩展，允许用户使用Python脚本来自动化分析任务、定制IDA Pro的行为和增强其功能。由于Python的简洁和强大，IDA Python为逆向工程师和安全研究人员提供了一个灵活的环境，以实现复杂的分析和处理。

主要特点

自动化任务：

IDA Python允许用户编写脚本来自动化常见的分析任务，如函数标记、数据结构识别和字符串提取等。通过自动化，用户可以节省大量手动工作时间。

与IDA API集成：

IDA Python能够访问IDA Pro的API，允许用户控制IDA的各个方面，比如获取程序信息、修改反汇编结果、添加注释等。用户可以利用API执行复杂的分析逻辑。

增强分析能力：

通过编写自定义脚本，用户可以实现自己的分析工具，例如，构建特定的漏洞检测器或实现某种特定的算法，从而增强IDA Pro的分析能力。

图形界面与可视化：

IDA Python可以与IDA Pro的图形界面进行交互，创建自定义的图形界面元素，如对话框和控件，以便用户输入和展示信息。

灵活的脚本运行：

用户可以在IDA Pro的交互式命令窗口中直接输入和运行Python脚本，也可以将脚本保存在文件中进行批量处理。

**3.恶意代码的网络特征**

恶意代码通过网络传播、执行、控制和与其他恶意活动通信时表现出来的特定行为或模式。这些特征通常用于检测、分析和防御恶意代码。以下是一些常见的恶意代码的网络特征：

1. 通信协议特征

恶意代码往往通过不同的通信协议与攻击者或其他受感染的设备进行通信。常见的通信协议特征包括：

HTTP/HTTPS：许多恶意软件使用 HTTP 或 HTTPS 协议进行 C2（命令和控制）通信，因为它们能通过常规 Web 流量隐藏。

DNS：恶意软件可能通过 DNS 查询与 C2 服务器通信，甚至利用 DNS 进行数据 exfiltration（数据外泄）。

FTP/SFTP：恶意软件可能使用 FTP 或 SFTP 协议上传或下载恶意文件或数据。

ICMP：恶意代码可能使用 ICMP 协议（如 Ping）进行简易的 C2 通信或数据传输，常用于绕过防火墙和检测机制。

2. 流量模式特征

恶意代码的流量模式往往与正常的网络流量有所不同。常见的流量模式特征包括：

流量异常：恶意代码可能产生异常的网络流量模式，比如频繁的小数据包传输、大量的数据流入流出，或者在不寻常的时间段内的活动。

持续的连接尝试：许多恶意软件会定期尝试与 C2 服务器保持连接。

数据泄露行为：恶意软件可能会进行大量的数据外泄，或者尝试与远程主机上传敏感信息。

3. 端口和IP特征

异常端口：恶意软件可能会尝试连接到非标准的端口，或在不常见的端口上建立连接。

动态 IP 或 IP 池：许多恶意软件会使用动态 IP 地址（例如通过 VPN、代理或僵尸网络），使得其网络特征变得更加难以追踪和识别。

与黑名单 IP 的通信：恶意软件通常会与已知的恶意 IP 地址进行通信，这些地址通常会被黑名单监控系统检测到。

4. C2（命令和控制）通信特征

加密通信：现代恶意软件通常通过加密通道与 C2 服务器进行通信，例如使用 TLS/SSL 或自定义加密协议。这使得检测更加困难。

域名生成算法（DGA）：恶意软件可能会使用 DGA 生成大量的随机域名用于 C2 通信，以避免 DNS 封锁。

间歇性通信：恶意软件与 C2 服务器之间的通信通常是间歇性的，可能每隔一段时间或在特定条件下才会发生。

Beaconing（信标）：恶意软件可能会定期向 C2 服务器发送信标请求，报告其状态或等待进一步指令。

5. 行为和异常检测

恶意软件的网络行为常常与正常用户行为差异较大：

数据包大小异常：恶意软件生成的数据包大小可能与正常通信的数据包大小有显著不同，可能过小或过大。

重复或异常的网络请求：恶意代码可能发出大量重复的请求，或在网络中进行多次连接尝试，导致异常流量。

不寻常的地理位置：恶意软件可能与位于不同地理位置的控制服务器进行通信，或者突然改变通信的目标位置。

6. 基于时序的特征

定时执行：一些恶意代码会在特定的时间点或周期内执行网络操作。它们可能会每小时或每天定期与 C2 服务器通信。

时延特征：恶意软件可能会利用网络延迟特征来避开检测。例如，恶意代码可能有意增加响应时间，以减少检测系统的注意。

7. 反侦察特征

恶意代码为了避免被检测，可能会具有一些反侦察机制：

代理和混淆：恶意软件可能使用代理、Tor 网络或虚拟专用网络（VPN）来掩盖其真实的网络活动。

IP 地址轮换：一些恶意代码会频繁更换通信服务器的 IP 地址，以避免被封锁或检测。

自我隐匿：恶意代码可能会检测防火墙、IDS/IPS（入侵检测系统）或沙箱环境，如果发现自己在监控环境中运行，会停止或变得不活跃。

8. 垃圾流量和蜜罐测试

垃圾流量：一些恶意代码故意生成大量垃圾流量，试图掩盖其真实的网络活动。

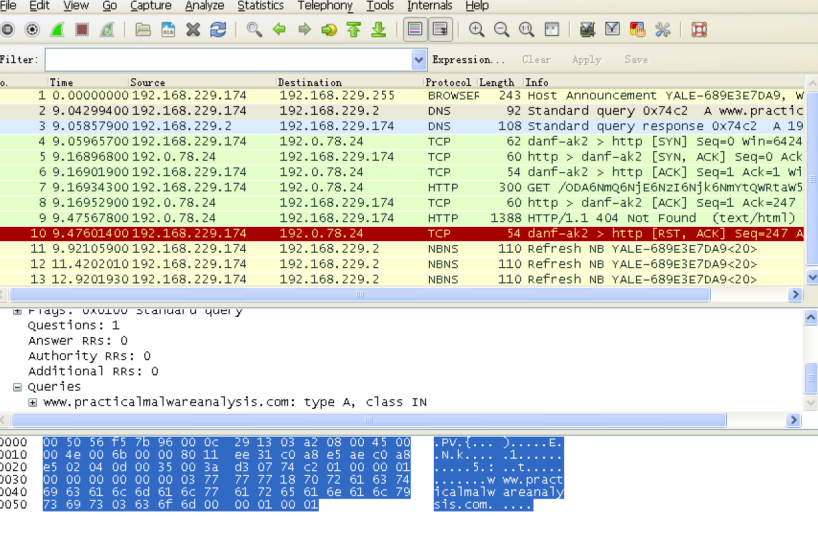
蜜罐测试：恶意软件可能会进行一些简单的测试，尝试与蜜罐进行通信来检查是否存在防御机制。如果与蜜罐连接成功，它可能会继续执行恶意活动。

**三．实验过程**

**Lab14-01**

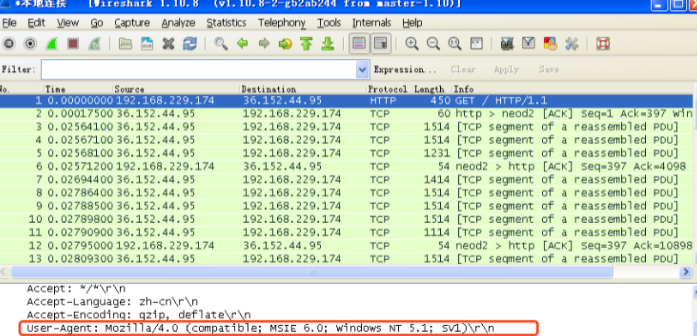
**实验环境：**实验设备环境：windows xp  
实验工具：IDAPro,wireshark，processmonitor，processexplore

**实验流程：**首先我们采用wireshark抓包监控



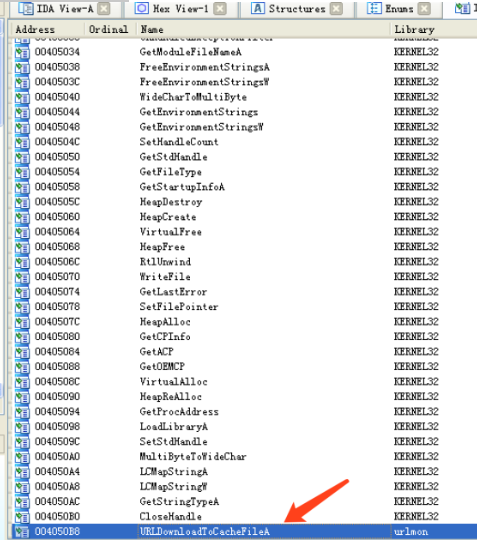
可以看到过滤出http之后，注意到有个get请求，访问的是www.practicalmalwareanalysis.com，但是get后面的字符串似乎是随机的字符串

然后继续在我们使用这台电脑自带的浏览器访问网页，从下图可以看到



可以看到Mozilla/4.0，这和我们在wireshark中看到的user-agent是一样的，说明正常的浏览器行为和我们运行程序产生的流量中看到的user-agent是相同的，说明恶意代码很可能使用了COM API接口

接着通过IDA分析  
在imports窗口看到



这段代码的功能是将文件下载到缓存中。由此可以推测，程序使用了 COM API 接口，而通过 COM API 接口发出的 HTTP 请求，大部分内容都来自 Windows 系统内部。这也解释了为什么在不同的主机上，抓取到的 User-Agent 字符串会有所不同，而每个 User-Agent 与在相同主机上使用浏览器手动访问时抓取到的值是相同的。

**Q1.恶意代码使用了哪些网络库?它们的优势是什么?  
A1.该程序中包含了 URLDownloadToCacheFile 函数，该函数利用了 COM 接口。当恶意代码通过 COM 接口发起 HTTP 请求时，请求中的大部分内容来自 Windows 系统内部。这使得仅凭网络特征进行针对性的检测变得非常困难，因为请求的结构和内容与常规的浏览器请求存在显著差异，难以通过常规手段识别。**

这个函数似乎是唯一使用的网络函数，所以可以对其查看交叉引用逐层跟入。

此时，程序执行到了 sub\_4011a3 函数。该函数不仅调用了 URLDownloadToCacheFileA，还调用了 CreateProcessA，推测是为了为下载的文件创建一个新的进程。为便于理解，我们将 sub\_4011a3 重命名为 downloadRun。接下来，我们将继续分析 URLDownloadToCacheFileA 附近的代码。



在调用处上方看到了一个字符串，跟入如下所示

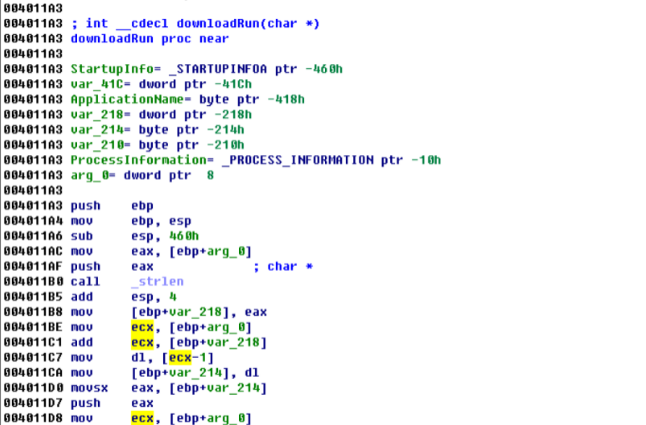


就是我们看到的get请求的那个形式

其作为sprintf函数的输入，而其输出作为

URLDownloadToCacheFileA函数调用的参数

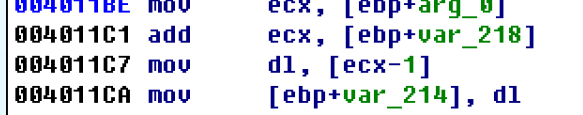
在格式化的字符串中我们看到URI的中间部分是被%s定义，而png文件内容是被%c定义，我们向前回溯，看看参数的输入点



在 004011db 处，push 的内容是 %s 对应的字符串，而 ecx 寄存器的值来自 arg\_0，并且 arg\_0 是 downloadRun 函数的唯一参数。

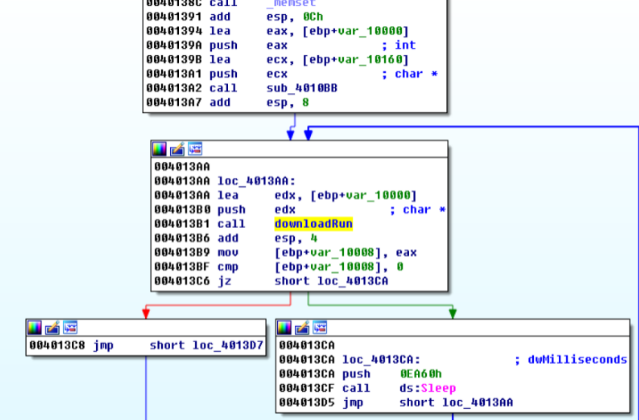
在 004011d7 处，push 的内容是 %c 对应的字符。

在 004011b0 处，使用 strlen 函数获取 arg\_0 字符串的长度，并将结果保存在 var\_218 中。

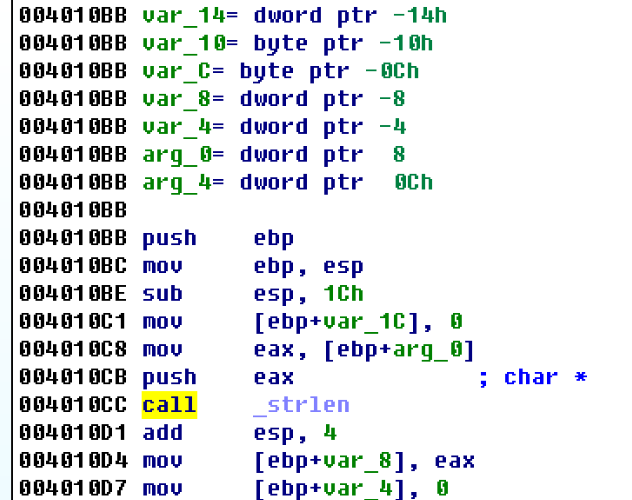


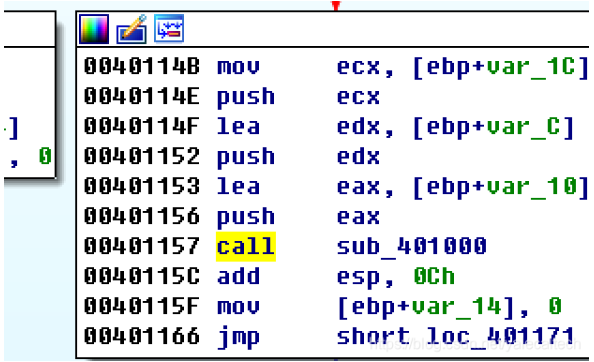
接下来，%s 字符串的最后一个字符被复制到局部变量 var\_214，该变量用作 %c 的参数。这也解释了在 Wireshark 中看到的现象：文件名 %c 与字符串 %s 的最后一个字符相同。

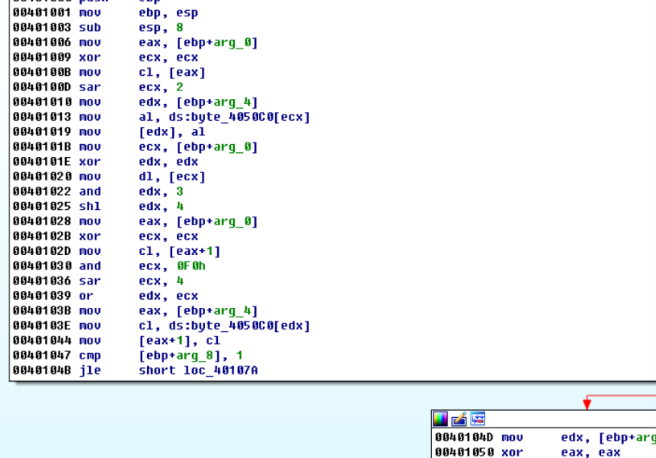
接下来，我们需要分析该函数的参数。为了进一步理解，我们查看了函数的交叉引用，进入了 main 函数。

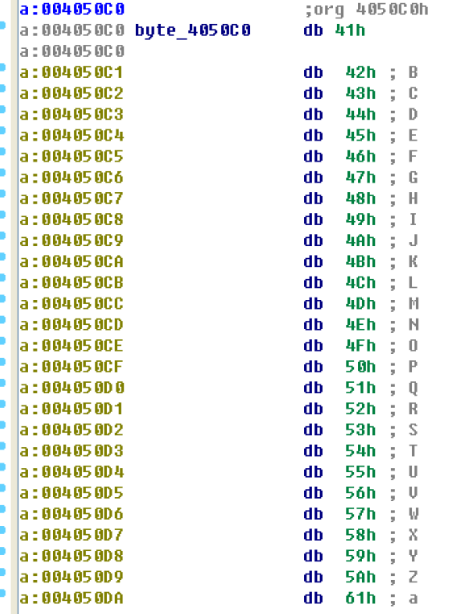


可以看到被调用的函数是处在一个循环结构里,里面还有对sleep的调用，而downloadRun的参数是来自sub\_4010bb  
跟入sub\_4010bb，分析，逐步跟入

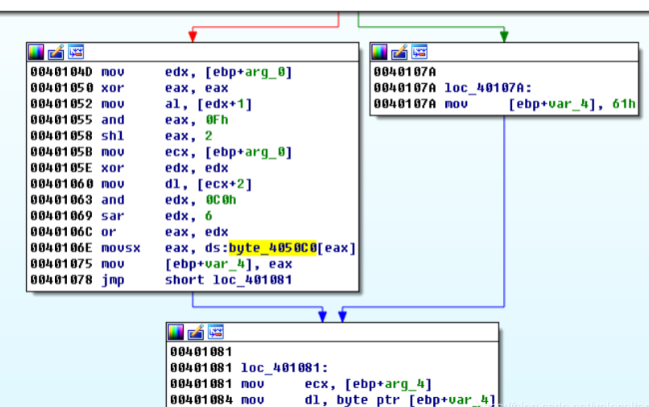




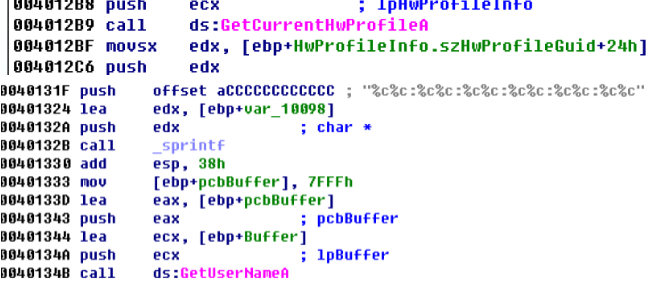




这其实是标准base64编码字符串，回到原函数继续往下



我们可以看到该控制流往左是选择编码字符，往右是选择填充字符  
右边的61h其实是‘a’，而不是’=‘  
说明sub\_401000并不是一个标准的base64编码函数。因为标准的base64是用‘=‘来填充的  
回到main函数  
在其前面有





在 main 函数中，我们可以看到调用了 GetCurrentHwProfileA、GetUserNameA 和 sprintf 函数。具体分析如下：

GetCurrentHwProfileA 函数：这个函数返回当前硬件配置文件的 GUID。函数返回的 GUID 中的前 6 个字节被提取，并以 MAC 地址的形式打印出来。这个 MAC 地址成为字符串 %s 中的第一个部分。

GetUserNameA 函数：此函数获取当前用户的用户名。用户名成为 %s 字符串中的第二个部分。

sprintf 函数：sprintf 用于格式化字符串，将硬件配置文件的前 6 个字节（以 MAC 地址形式）和用户名组合在一起，形成最终的字符串。这两个字符串被拼接为 %s%s。

通过这种方式，最终的字符串由硬件的 MAC 地址和当前用户的用户名组成。

**Q2.用于构建网络信令的信息来源，什么样的条件会引起信令的改变?**

**A2.信息来源于主机的 GUID 和 用户名 的一部分。具体来说：**

**GUID：每台主机的 GUID 在操作系统中是唯一的，因此可以用于标识每台设备。在信令中，使用了 GUID 的前 6 个字节，这些字节通常具有较高的唯一性，因为它们与硬件和系统密切相关。**

**用户名：用户名则会根据登录的用户而变化，因此它是一个相对动态的标识符，依据当前系统登录的用户而改变。**

**这种组合方式利用了 GUID 的唯一性和 用户名 的动态性，从而在某些情境下能够生成一个既具有唯一性又具有某种辨识度的标识符。**

**Q3.为什么攻击者可能对嵌入在网络信令中的信息感兴趣?**

**A3.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。**

我们尝试使用base64工具进行解码wireshark捕获到的get的字符串

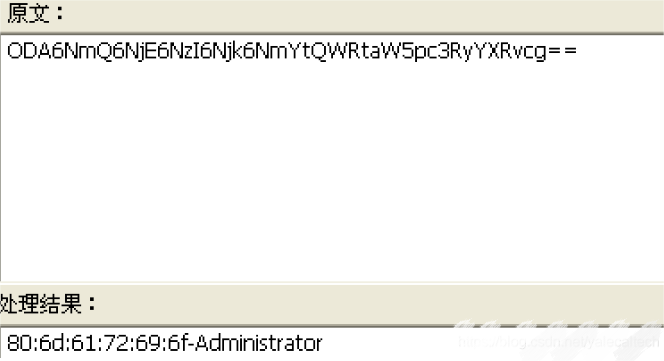


可以看到，分析得到的格式与我们从 IDA 中的逆向分析结果是一致的。在这里，有两点需要特别注意：

解码时的错误解析：在编码时，恶意代码并没有使用 a 填充的 Base64，而是使用了 = 填充的标准 Base64。所以在解码时，= 填充符会被正确识别，而不是将末尾的 aa 当作可替换的填充字符。这会导致解码时，aa 被错误地解释为标准字符。

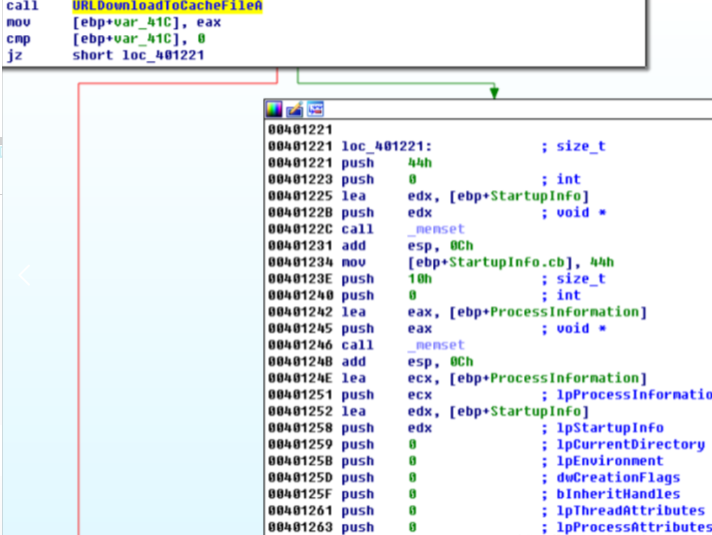
编码时的标准 Base64 填充：如果是标准的 Base64 编码，最后的 aa 实际上应该被解码为 ==，这是因为标准 Base64 编码要求在最后填充两个等号字符 == 来保证数据长度是 4 的倍数。

因此，如果使用标准 Base64 编码，解码后的结果应该是这样的：



**Q4.恶意代码是否使用了标准的Base64编码?如果不是，编码哪里不一样?  
A4.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。**

知道字符串的来源后，我们看看收到某些内容后会发生什么  
可以看到URLDownloadToCacheFileA成功后会走右边



这里调用了CreateProcessA



它会将URLDownloadToCacheFileA返回的路径名作为参数，一旦恶意代码下载了一个文件，就执行该文件然后退出

**Q5.恶意代码的主要目的是什么?  
A5.通过该恶意代码下载并运行其他代码。**

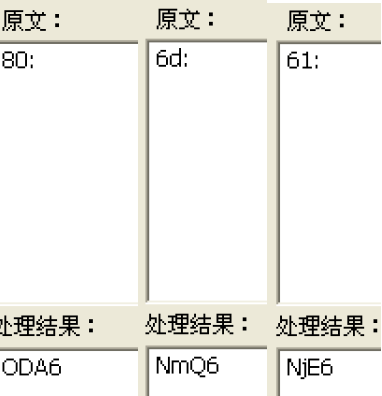
我们返回分析解码后的结果

可以看到，这里有两个关键点需要注意：

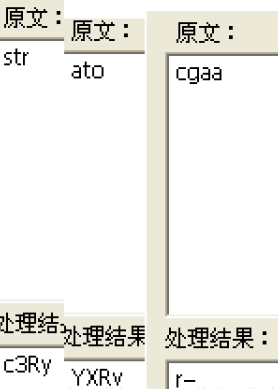
填充字符 : 和硬件配置文件字节与用户名之间的 - ：这些符号在恶意代码的编码过程中起到了分隔和填充的作用。: 作为填充符号可能是在将数据编码为 Base64 之前，作为占位符或分隔符进行使用。而 - 则是在硬件配置文件字节与用户名之间分隔的符号，这样的格式帮助恶意代码构建了特定的编码结构。

Base64 编码：在数据被发送到网络之前，恶意代码将上述内容进行了 Base64 编码。Base64 编码通常会将数据转化为 ASCII 字符串，从而确保可以安全地通过网络传输。

为了更好地分析这个编码结构，我们可以尝试将数据按三字符一组进行分割，看看会有什么样的管理方式或数据模式。通过这种方式，我们能够更清楚地理解编码和解码过程中的关键步骤及其背后的逻辑。







我们可以看到一些规律：

Base64 编码时的规则：由于原始字符串中的冒号是三字符组中的第三个字符，因此编码后的每个四字符组的第四个字符的比特都来自于原始字符串中的第三个字符。从上面的分析中可以看出，冒号之后每四个字符组的最后一个字符为 6。由于破折号 - 的存在，第六个四字符组以 t 结尾。

已知的信息：我们已经知道 URI 至少有 24 个字符，并且明确了 6 和 t 的具体位置。同时，下载的文件名是单个字符，而这个字符就是路径的最后一个字符。

因此，我们可以编写两个正则表达式来帮助我们匹配字符串和下载的文件。

第一个正则表达式

用于匹配包含字符 6 和 t 结尾的 Base64 编码四字符组的模式：

[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,}

这个正则表达式匹配一系列 Base64 编码的字符组，其中每个组的最后一个字符为 6，并且包含一个以 t 结尾的组。

第二个正则表达式

用于匹配下载的文件，其中文件名是一个字符后跟 .png：

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

这里，\1 表示匹配第一个捕获组的字符，这个字符就是 / 前面的 Base64 编码字符串中的第一个字符。通过这种方式，我们能够匹配单字符文件名后跟 .png 的格式。

这样，我们就可以通过正则表达式匹配到所需的文件名和 Base64 编码的数据。

**Q6.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?**

**A6.在恶意代码的通信过程中，可以作为检测目标的元素包括以下几种：**

**域名：恶意代码可能通过特定的域名与远程服务器通信，因此域名是一个重要的检测目标。通过监控与域名相关的活动，可以发现潜在的恶意通信。**

**冒号（:）：冒号在恶意代码的 Base64 编码中具有特殊作用，它可能是某些特定标记或者格式分隔符。在解码过程中，冒号的存在可能影响解码结果，从而成为检测恶意代码的线索。**

**Base64 解码后的破折号（-）：恶意代码在编码过程中，可能使用破折号作为某种分隔符或者标记。Base64 解码后，破折号的出现可能与恶意通信的格式相关，因此也是一个值得注意的检测元素。**

**URI 的 Base64 编码的最后一个字符：在恶意代码的通信中，URI 的 Base64 编码的最后一个字符可能用于标识下载的文件类型或文件名。例如，Base64 编码的 URI 最后的字符可能作为文件名的第一个字符，且该文件名通常是 .png 文件。这个特征可以用来识别恶意文件的下载活动。**

**通过关注这些特定的元素（如域名、冒号、破折号以及 Base64 编码的最后一个字符），可以有效地识别和检测恶意代码的通信模式，并及时采取相应的防御措施。**

**Q7.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?**

**A7.如果防御者没有意识到操作系统在决定这些元素中的作用，他们可能会错误地将 URI 中的元素作为检测目标。通常情况下，Base64 编码的字符串以 a 结尾，这会导致文件名显示为 a.png。然而，当用户名长度是 3 的倍数时，Base64 编码后的字符串的最后一个字符以及文件名将依赖于编码用户名的最后一个字符。在这种情况下，文件名变得不可预测。**

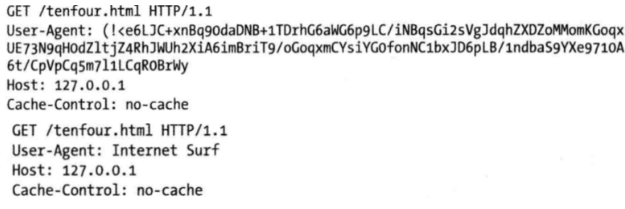
**Q8.哪些特征集可能检测到这个恶意代码(以及新的变种) ?**

**A8.推荐的特征集请参考上述详细分析过程。**

**Lab14-02**

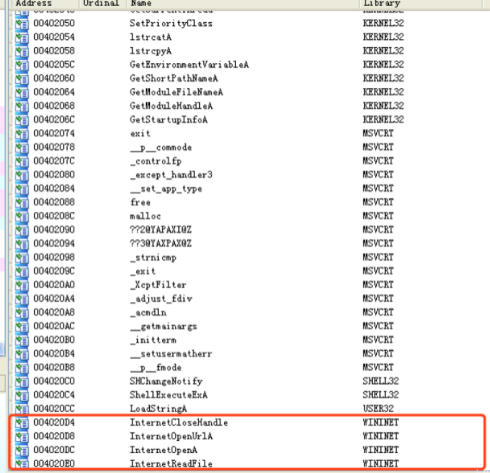
**实验环境：**实验设备环境：windows xp  
实验工具：IDAPro,wireshark，processmonitor，processexplore

首先我们采用wireshark抓包监控



我们注意到 User-Agent 的值比较异常，且如果程序继续运行，其值会保持不变。然而，在不同的主机上运行时，这个值却会发生变化。这表明，User-Agent 中编码的信息依赖于特定主机的信息。

接下来，我们可以使用 IDA 进行更深入的分析，查看程序是如何生成这个 User-Agent 字符串的，以及它如何利用主机信息来动态构建这个值。

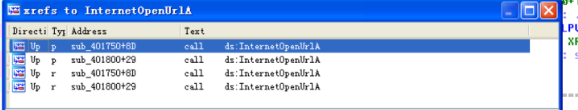


可以看到，从WinINet库中导入了函数InternetOpenA、InternetOpenUrlA，InternetReadFile，和InternetCloseHandle。

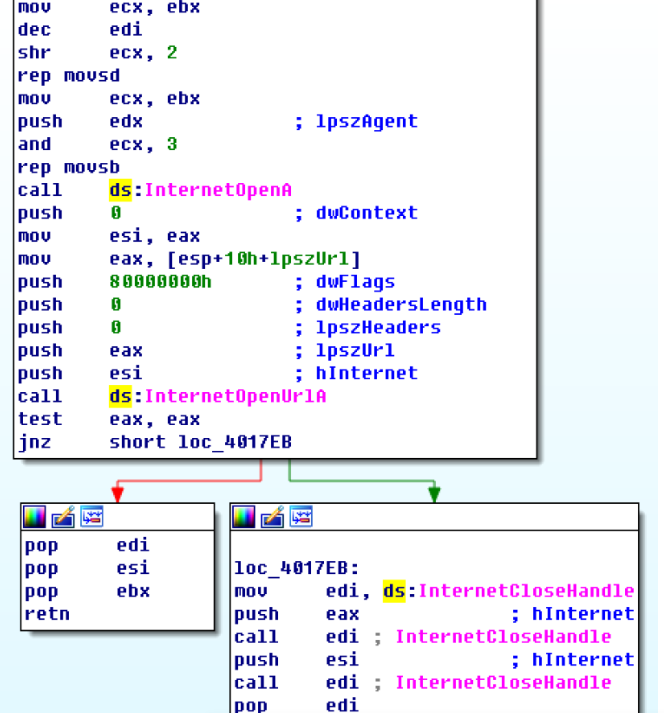
**Q1.这个恶意代码使用哪些网络库?使用这些库的好处和坏处是什么?**

**A1.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cookie和缓存，可以由操作系统提供。**

然后我们查看InterOpenUrlA交叉引用

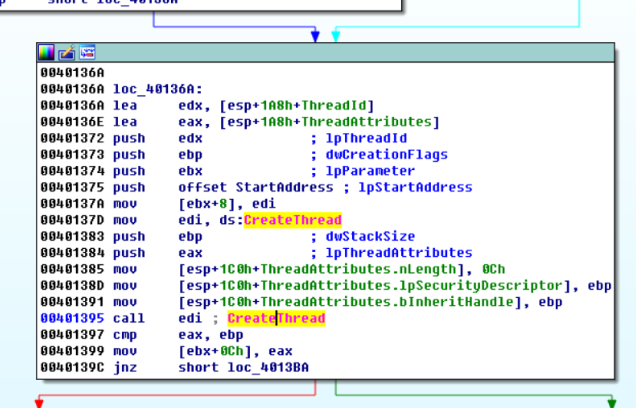


先跟入sub\_401750

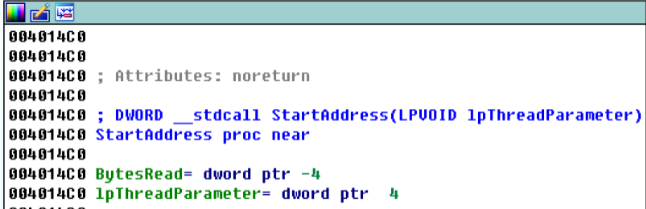


我们可以看到，deFlags 的值为 0x8000000，经过查询后发现，这个参数值对应的标志是 INTERNET\_FLAG\_PRELOAD。当这个标志被置位时，它会导致信令中出现 Cache-Control: no-cache 字符串，从而指定浏览器或客户端不使用缓存。

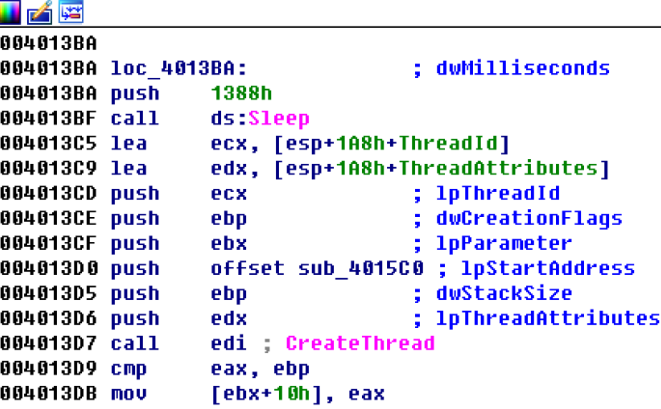
我们跟入winmain



可以看到00401395处调用CreateThread，其lpStartAddresss参数被标记为StartAddress，它就是新线程的开始地址

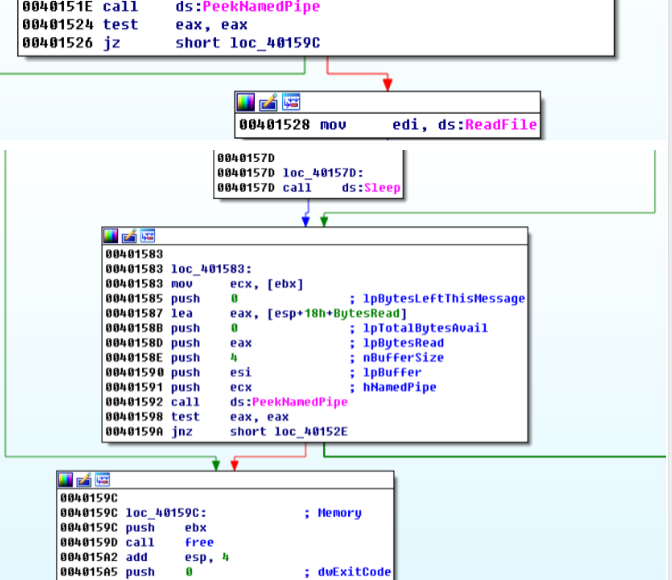


可以看到其地址位于0x004014c0  
下图是第二处CreateThread函数调用

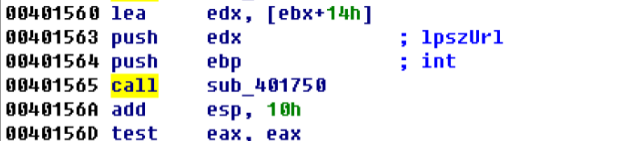


其lpStartAddress参数被标记为sub\_4015c0,这也是一个新线程开始的地址  
为了便于后续分析，将sub\_4015c0重命名为s\_thread2\_start

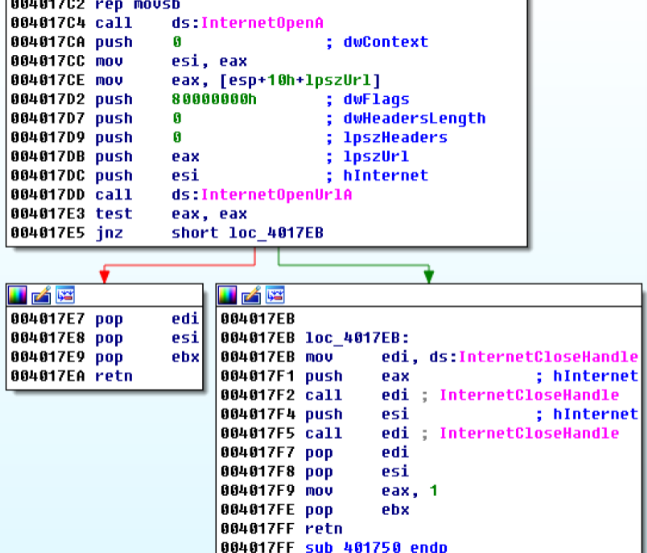
我们继续分析StartAddress



可以看到它调用了malloc,PeerkNamePiep,ReadFile，ExitThread,Sleep和一些内部函数，以及下图的函数

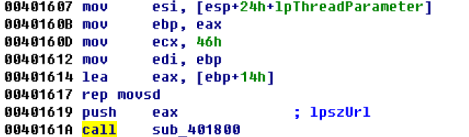


它调用了sub\_401750,跟入

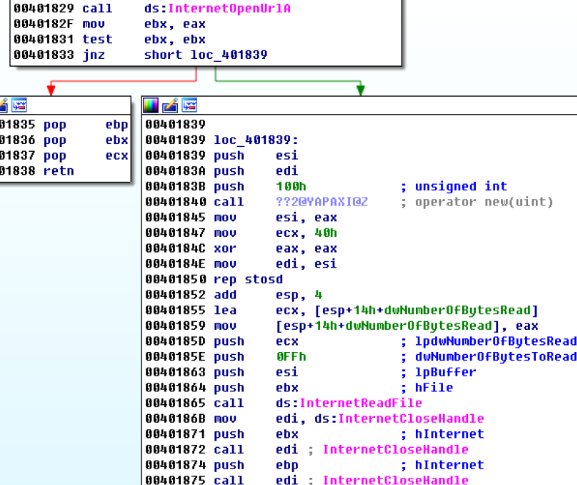


可以看到它调用了导入表中看到的网络相关函数。所以给sub\_4017050重命名为s\_Internet1。

再来看看s\_thread\_start  
可以看到malloc,WriteFile,ExitThread,Sleep调用，以及下图的函数



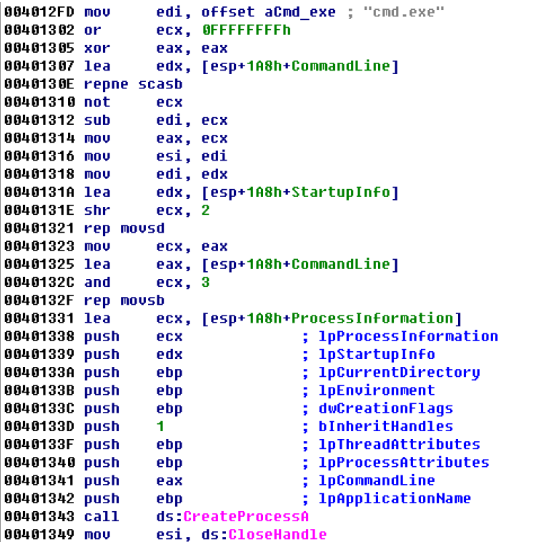
它调用了sub\_401800



可以看到这个函数在结构上和之前那个函数类似，所以我们将sub\_401800重命名为s\_internet2

这两个都有调用PeedNamedPipe（用于检测命名管道的新输入）。为了确定这两个线程读取或者写入的内容，我们需要继续分析线程的来源，也就是winmain函数。

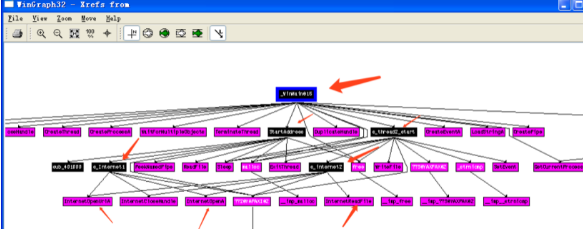




可以看到，该函数调用了：CreatePipeA,GetCurrentProcess,

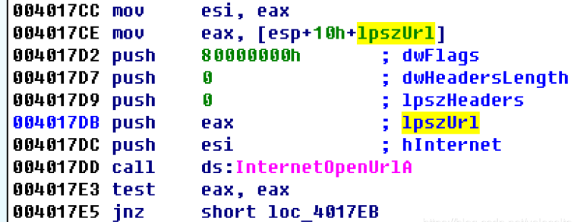
DuplicateHandle以及CreateProcessA在调用CreateThread启动线程之前。其中CreateProcessA函数查询资料后可知是创建一个新的cmd.exe进程，其他函数设置新进程，使stdin,stdout与提供的命令进程的句柄进行绑定

查看winmain的交叉引用

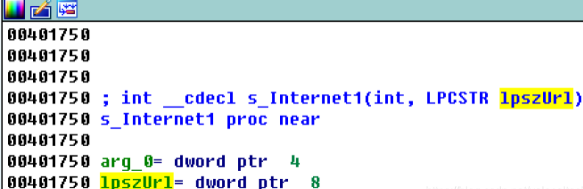


箭头标出的部分是关键。这种结构属于一种常见的反向命令 shell 模式。攻击者启动一个新的命令 shell 作为线程，并启动一个单独的线程来读取命令 shell 的输入并写入输出。StartAddress 使用 PeekNamedPipe 来检查命令 shell 的输入，如果有数据存在，则通过 ReadFile 进行读取。读取到的数据随后通过 s\_Internet1 发送到远程位置。与此同时，另一个线程 s\_thread2\_start 使用 s\_internet2 连接到远程位置。如果命令 shell 上有新的输入，另一个线程会将这些输入写入命令 shell 的输入管道。

那么我们来看看传递给s\_inetrnet1中Internet函数的参数，看看这些参数的来源。  
首先是InternetOpenUrlA函数



向上回溯

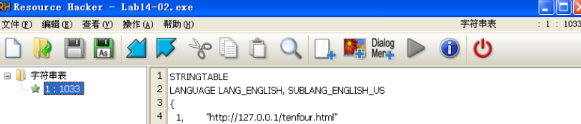


可以看到是它是s\_internet1的第二个参数

回到其上层函数是StartAddress，而其参数是lpParameter，往上回溯，发现它来自LoadString

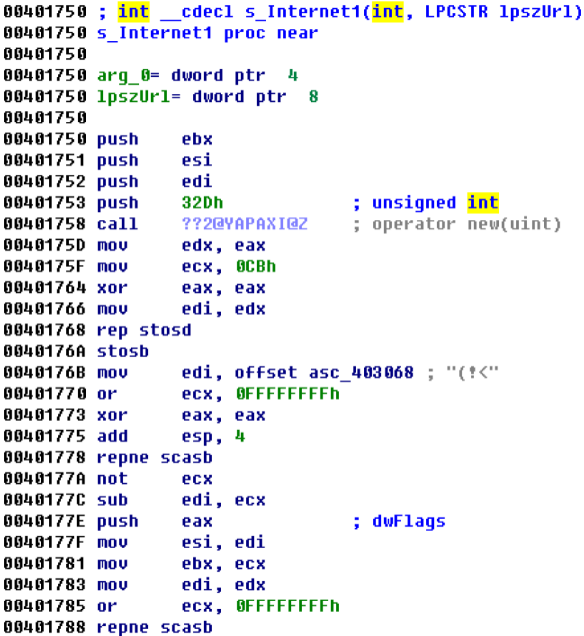


其功能是从资源节里读取字符串  
所以我们可以使用Resource Hacker分析该文件的资源内容

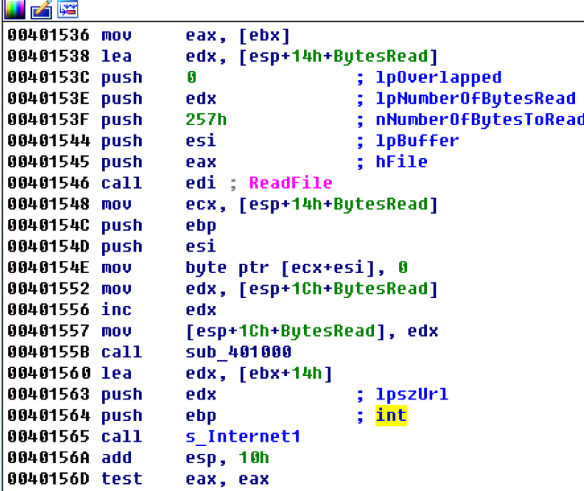


**Q2.恶意代码信令中URL的信息源是什么?这个信息源提供了哪些优势?  
A2.由上述分析可知信息源是PE文件中的字符串资源节包含一个用于命令和控制的URL。它可以在不重新编译恶意代码的情况下，可以让攻击者使用资源节来部署多个后门程序到多个命令与控制服务器位置。**

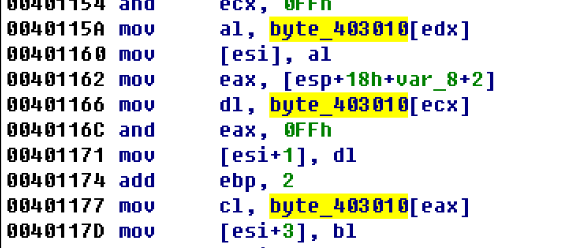
将我们抓到的信令和这个用作信令的url相比较，可以看到发送出的信令的部分内容就是来自于资源节  
s\_Inetrnet一个参数是url，还有个参数是user-agent，回到IDA中s\_internet1处继续分析



我们从上图可以看到在0040176b看到了(!<，这和我们在信令中看到的user-agent字符串开头是一样的，可是user-agent其他的内容来自哪里呢？  
我们回到StartAddress调用s\_internet1附近



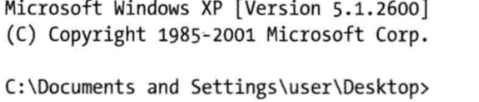
在其之前，调用了sub\_401000,这个函数会接收两个输入参数，并输出user-agent字符串中的主要内容。



这是一个自定义的base64的变种，它使用的base64字符串保存在byte\_403010，内容如下所示



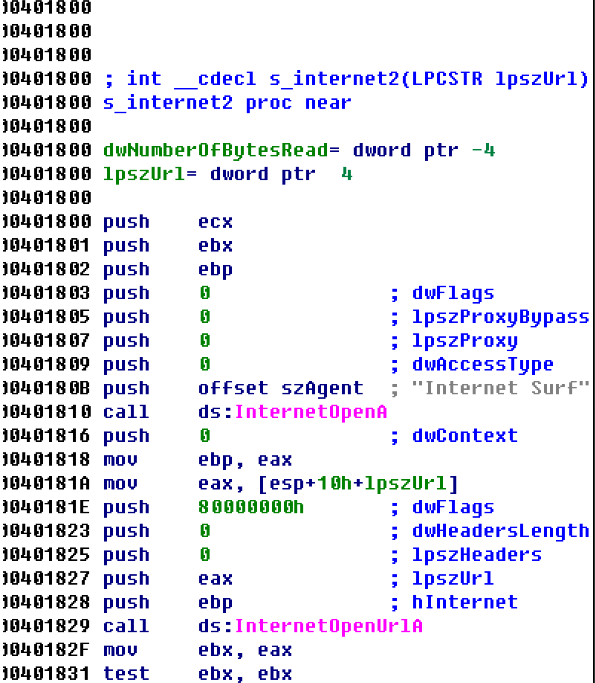
然后解码



**Q6.恶意代码的编码方案是标准的吗?**

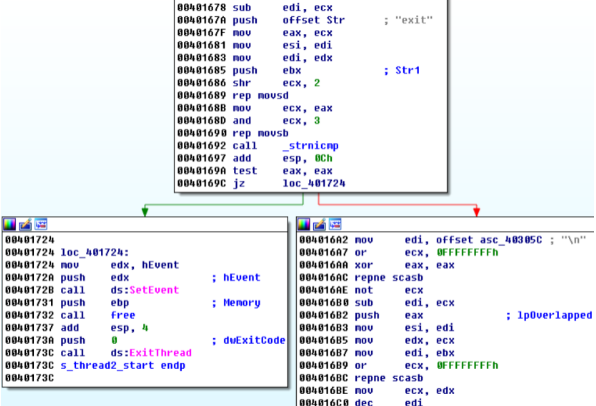
**A6.不是标准的，编码方案类似Base64， 但是使用自定义的字母**

再来跟入s\_internet2



它接收的url和s\_internet1是一样的，但与s\_internet1不同的是user-agent，除此之外还静态定义了Internet Surf

再看看s\_thread2start，这个函数是用于向命令shell传递输入



如0040167a,00401692所示，strnicmp是用于测试传入的网络内容。如果传入的是exit，则会往左然后退出。说明这函数还有根据输入内容自动终止的功能。

**Q3.恶意代码利用了HTTP协议的哪个方面，来完成它的目的?**

**A3.攻击者滥用了 HTTP 的 User-Agent 域，通常该域应包含应用程序的信息。恶意代码创建了一个线程来对该域中的传出信息进行编码，同时又创建了另一个线程，使用静态域来表示该域作为通道的“接收”端。**

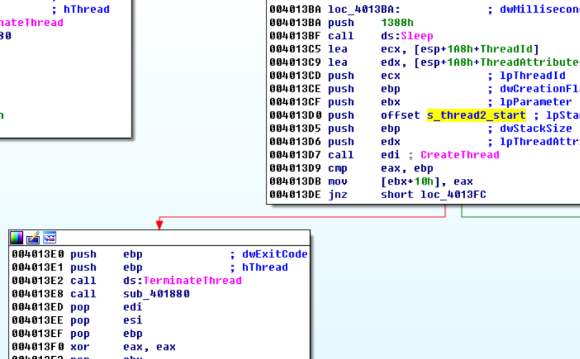
**Q4.在恶意代码的初始信令中传输的是哪种信息?**

**A4.初始的信令是一个编码后的shell命令行提示。**

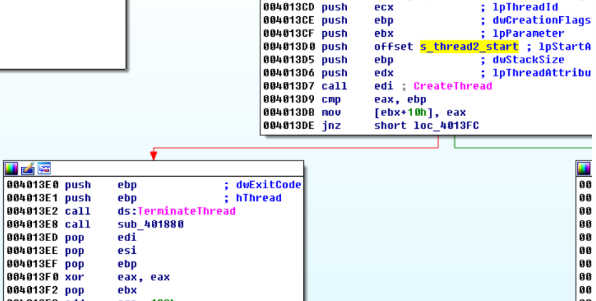
回到winmain  
我们可以看到，最后是有三个可能的函数结局，第一个



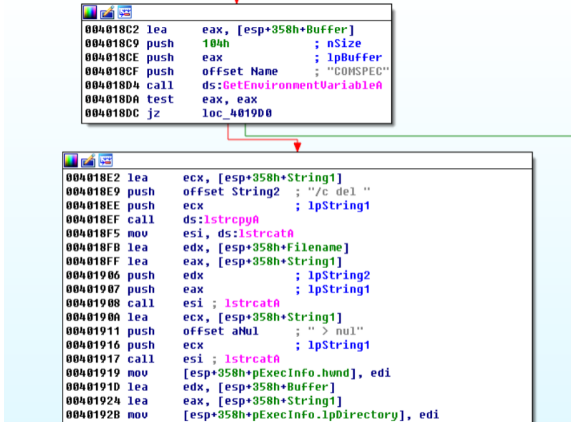
第一个进程没有成功创建则会终止  
第二处：



第二个线程没有成功创建则终止  
第三处：



当s\_thread2\_start退出后，则左边的流程，会调用sub\_401880,其作用是一旦恶意代码退出，则会从磁盘上删除恶意代码



跟入之后，看到：0x401880实现了自我删除的ComSpec方法

**Q5.这个恶意代码通信信道的设计存在什么缺点?**

**A5.尽管攻击者对传出的信息进行了编码，但并未对传入的命令进行编码。此外，由于服务器需要通过 User-Agent 域中的静态元素来区分通信通道的两端，这使得服务器的依赖关系变得非常明显。因此，User-Agent 域可以作为特征生成的一个重要目标元素。**

**Q7.通信是如何被终止的?**

**A7.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。**

**Q8.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?**

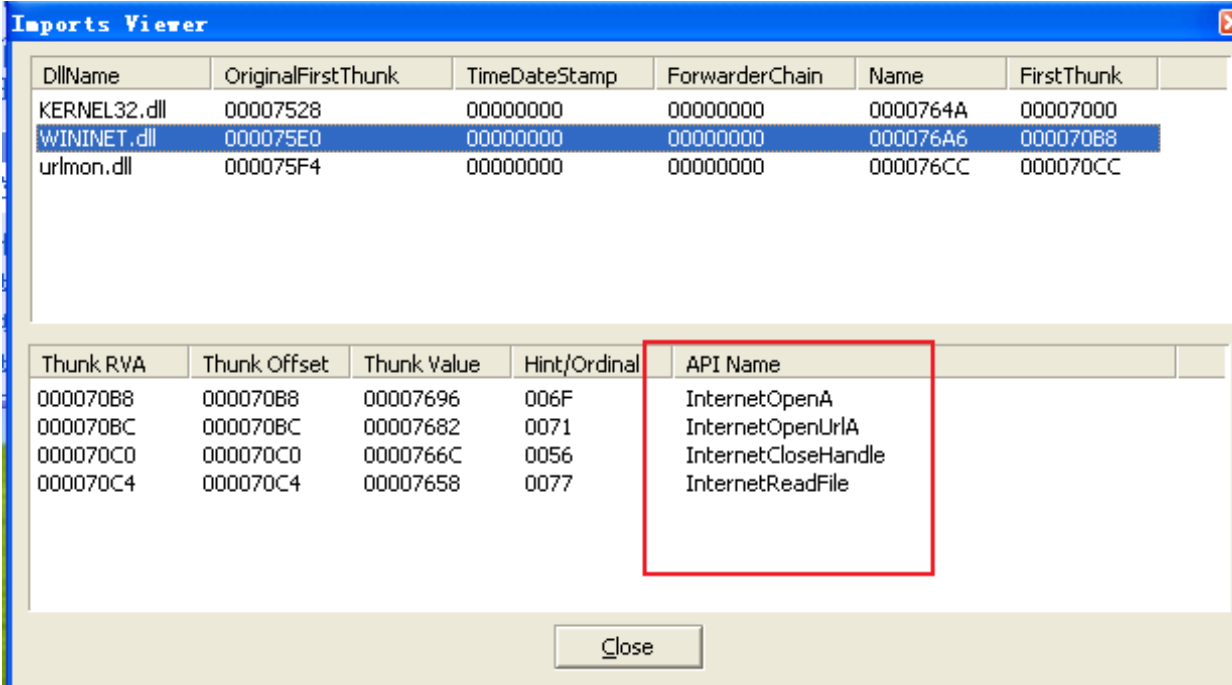
**A8.这个恶意代码是一个小巧且简单的后门程序，其唯一目的是为远程攻击者提供一个 shell 命令接口。然而，通过监测常见的出站 shell 命令活动的网络特征，并无法有效地发现它。基于它试图删除自己的行为，我们推测这段恶意代码可能是攻击者工具包中的一个一次性组件。**

**Lab14-03**

**实验环境：**实验设备环境：windows xp  
实验工具：IDAPro,wireshark，processmonitor，processexplore

静态分析：

查壳——无壳



分析输入表

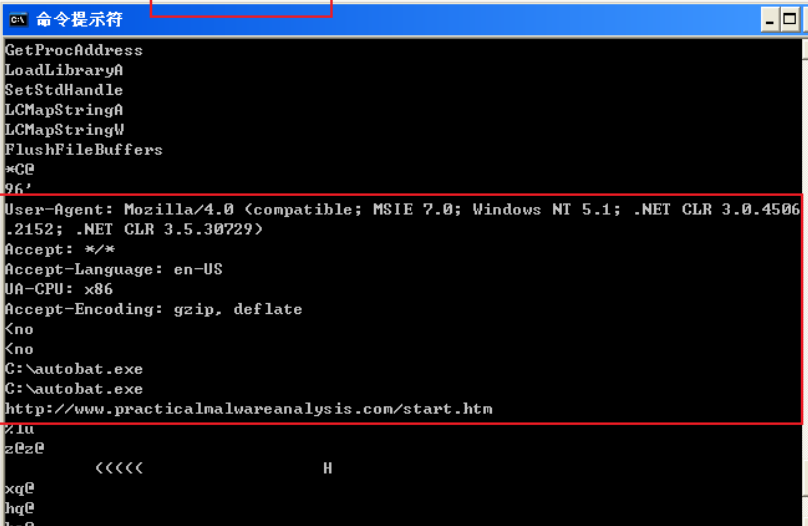
KERNEL32.DLL：读写，创建文件，创建文件，加载库文件等

WININET.DLL：URL访问，网络文件读取等

urlmon.dll：URLDownloadtoCacheFileA函数下载文件到缓存

与此同时，还能看到具有着 Internet 开头的 OpenUrl，ReadFile 等一系列函数。用于打开网络 URL 等行为。

String分析



出现base64相似的编码字符串，猜测可能使用base64进行数据加密

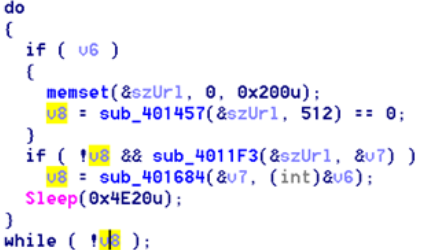
出现HTTP协议相关字段，猜测会进行网络请求：User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729) ：是 HTTP 请求头部的 UserAgent 字段。即产生了浏览器请求

出现exe程序，可能会执行：autobat.exe：自动化的批处理任务。

出现网络资源，可能会请求网络资源： http://www.practicalmalwareanalysis.com/start.htm

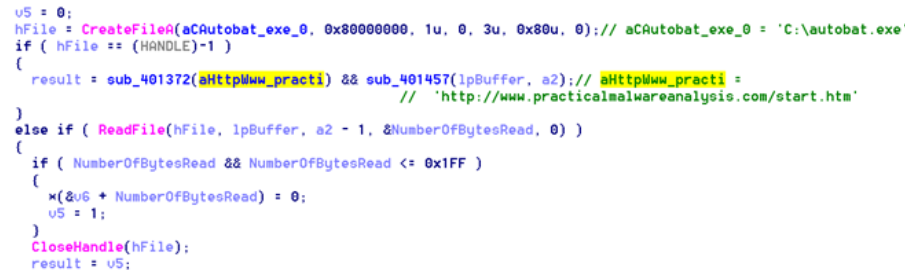
IDA分析

Main函数的逻辑比较简答，主要是一个do while循环：



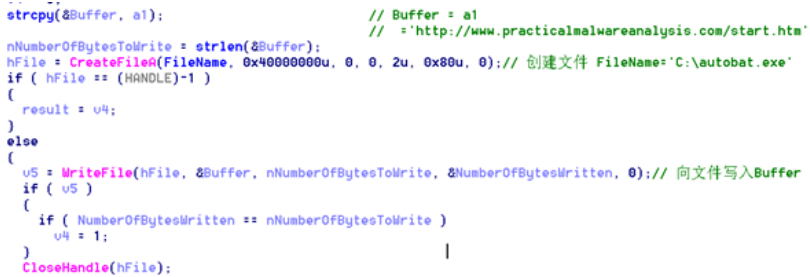
对循环体进行分析，首先调用了函数sub\_401457，该函数由两个参数，一个是字符串szUrl，另一个是数字；跟进该函数进行分析

(1) 该函数内部首先调用CreateFileA函数打开文件’C:\autobat.exe’ ，然后调用sub\_401372

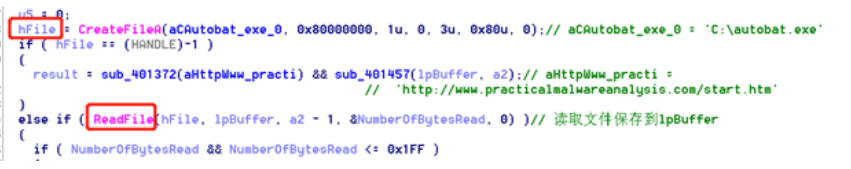


(2) 进入函数sub\_401372，该函数的参数是字符串’http://www.practicalmalwareanalysis.com/start.htm’ ，一个网络资源；进入该函数发现该函数将创建文件’C:\autobat.exe’，并且将字符串

’http://www.practicalmalwareanalysis.com/start.htm’ 写入到文件中。

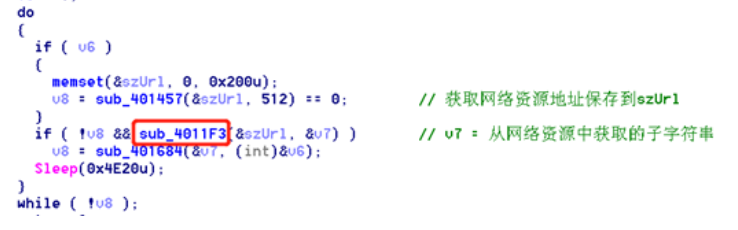


(3) 回到函数sub\_401457，继续往下，调用函数ReadFile将文件中的数据读取到缓冲区lpBuffer中；此处的文件正是前面创建的’C:\autobat.exe’，它内部存储的数据正是网络资源字符串’http://www.practicalmalwareanalysis.com/start.htm’。



因此函数sub\_401457的功能就是在本地创建文件保存网络资源地址

回到循环体，继续往下，调用函数sub\_4011F3对网络资源进行操作，进入该函数，

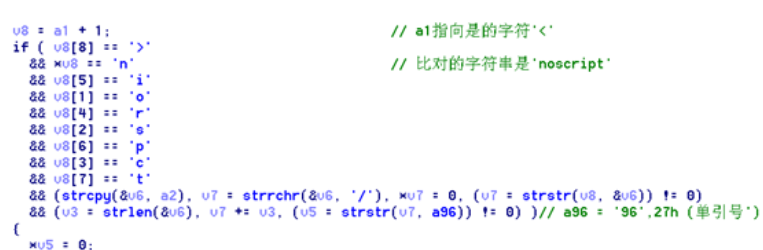


(1) 该函数首先初始化HTTP字段元素，然后根据传入的URL进行网络资源访问

(2) 从网络资源读取数据后保存到Buffer中，然后匹配‘<no’字符串

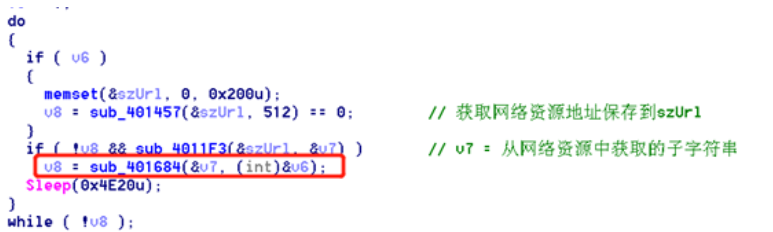


(3) 然后调用sub\_401000进行进一步匹配

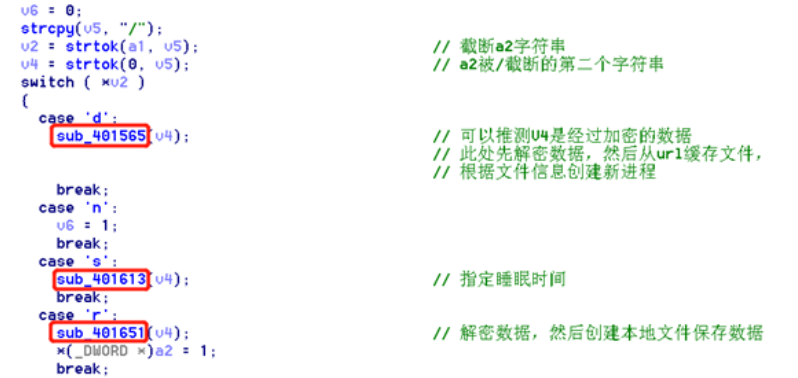


(4) 返回从网络中获取的满足匹配条件的字符串

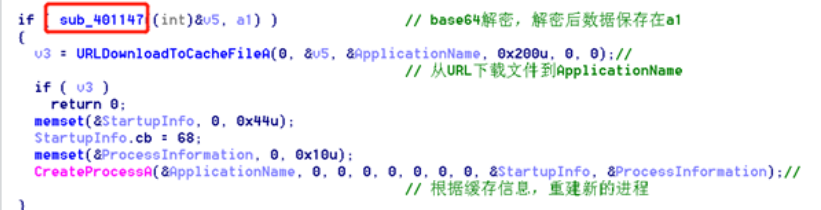
继续往下，将获取的字符串传入函数sub\_401684



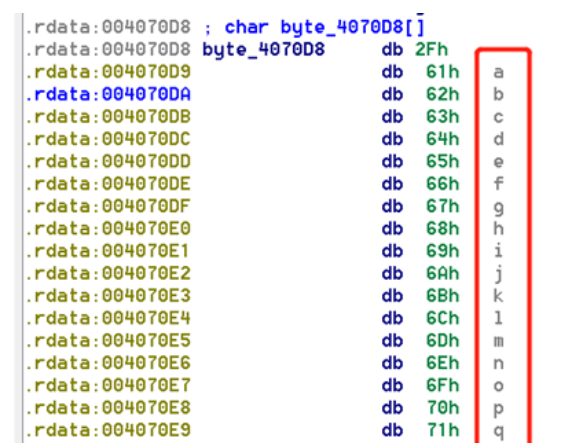
(1) 进入函数sub\_401684进行分析，发现该函数先将字符串进行截断，然后进行一个switch跳转，跳转的条件就是字符串的首字母，那么可疑猜测这就是根据C&C发送的指令执行不同的操作；每个命令的方式都已经分析如下。



(2) 需要注意的是，其中的加密函数部分都调用函数sub\_401147



(3) 该函数与传统的base64并不相同，它采用了自定义的编码字符串



到此，整个恶意程序的分析就完成了

现在对恶意程序的功能进行总结

（1） 创建本地配置文件

（2） 根据配置文件发送网络请求，其中的HTTP部分元素已经被硬编码

（3） 将网络请求获取的数据进行缓存，并且进行字符串检查

（4） 对返回的数据进行解密，不同的指令执行不同的操作，其中包括睡眠sleep，重定向redirect（修改本地配置文件），下载并执行其他恶意文件download，不执行任何操作nop。

**Q1: 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?**

**A1：硬编码的头部包括 Accept、Accept-Language、UA-CPU、Accept-Encoding 和 UserAgent。但是它犯了错误, 多添加了一个额外的 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。**

**Q2: 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?**

**A2：只有在配置文件不可用时，域名和 URL 路径才会被硬编码。这个硬编码的 URL 应该与所有配置文件一起构成特征。然而，将硬编码组件作为检测目标可能比结合硬编码组件和动态 URL链接进行检测效果更好。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。**

**Q3: 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?**

**A3：恶意代码通过 Web 页面上 noscript 标签中的特定组件获取命令，这类似于本章中提到的注释域的例子。使用这种技术，恶意代码可以向一个合法的网页发出指令，并接收合法内容，这使得防御者更加困难地区分恶意流量和合法流量。**

**Q4: 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?**

**A4：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 noscrpt 标签，此URI0 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) 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。恶意软件的设计者设定了一系列特定的字符串，用以探测其功能。该软件在寻找 noscript 标签时，首先搜索“<no”，然后通过独特的字符对比方法确认这一标签。它还重复使用了域名的缓存区来核查命令内容。此外，它对字符串“96”进行了三个字符的搜索，而唯一的单字符搜索是“/”。在匹配命令时，它只考虑首个字符，因此，攻击者可以通过 Web 响应中的“soft”或“seller”等词，实际上向恶意软件发送休眠命令。流量分析可能显示攻击者使用“soft”这个词向恶意软件发送命令，这可能会误导分析人员在其特征中使用整个词。攻击者可以使用“seller”或任何以“s”开头的词来发出命令，而无需修改恶意软件。**

**Q5: 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?**

**A5：sleep 命令是未编码的，数字表示休眠秒数。在两条命令中，参数使用了非标准简单编码，而非 Base64 编码。参数由偶数个数字组成（移除尾部的 96 后），每对数字表示数组/abcdefghijklmnopqrstuvwxyz0123456789:. 的索引。这些参数仅用于 URL 之间的通信，因此不需要大写字符。这种编码的优点是非标准性，需要逆向工程来理解其内容。其缺点在于它可能在字符串输出中被识别为可疑，因为 URL 总是以相同的方式开始，形成了一种一致性模式。**

**Q6: 这个恶意代码会接收哪些命令?**

**A6：恶意软件的命令包括 quit、download、sleep 和 redirect。quit 命令简单地退出程序。download 命令用于下载并执行可执行文件，允许攻击者指定下载 URL。redirect 命令修改恶意软件的配置文件，导致新的信号 URL。**

**Q7: 这个恶意代码的目的是什么?**

**A7: 该恶意软件本质上是一个下载器，其优点包括基于 Web 的控制，以及在被识别为恶意后易于调整。**

**Q8: 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?**

**A8：特定的恶意软件行为元素可能成为独立的检测目标，例如与静态定义的域名、路径和动态发现的 URL 相关的特征；与信号中的静态组件相关的特征；能够识别命令初始请求的特征；以及能够识别特定属性的命令和参数的特征。**

**Q9: 什么样的网络特征集应该被用于检测恶意代码?**

**A9:两个 Snort 检测规则。以下是提取的规则：**

**1. 第一个检测规则：**

**alert tcp $EXTERNAL\_NET $HTTP\_PORTS -> $HOME\_NET any (msg:"PM14.33 Downloador Redirect Command"; content:"/082022016370000"; pcre:"/\^/\[dr\][^\/]\*\/082022016370000\/"; sid:20001433; rev:1;)**

**这个规则用于检测从外部网络到内部网络的 TCP 连接，特别是通过 HTTP 端口。它搜索包含特定内容（"/082022016370000"）的流量，并且这个内容符合正则表达式定义的模式。**

**2. 第二个检测规则：**

**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;)**

**这个规则同样用于检测从外部网络到内部网络的 TCP 连接，通过 HTTP 端口。它搜索包含数字 "96" 的流量，并且这个数字符合正则表达式定义的模式。**

**Yara检测**

**1.Yara规则**

rule Lab14\_01\_exe

{

meta:

description = "Lab14\_01\_exe:Yara Rules"

date = "2024/12/22"

author = "xize"

strings:

$clue1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" wide ascii

$clue2 = "http://www.practicalmalwareanalysis.com/%s/%c.png" wide ascii

condition:

all of them //Lab14-01.exe

}

rule Lab14\_02\_exe

{

meta:

description = "Lab14\_02\_exe:Yara Rules"

date = "2024/12/22"

author = "xize"

strings:

$clue1 = "WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVmn0pqrstuvwxyz" wide ascii

$clue2 = "/c del" wide ascii nocase

$clue3 = "cmd.exe" wide ascii

condition:

all of them //Lab14-02.exe

}

rule Lab14\_03\_exe

{

meta:

description = "Lab14\_03\_exe:Yara Rules"

date = "2024/12/22"

author = "ErwinZhou"

strings:

$clue1 = "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)" wide ascii

$clue2 = "C:\\autobat.exe" wide ascii nocase

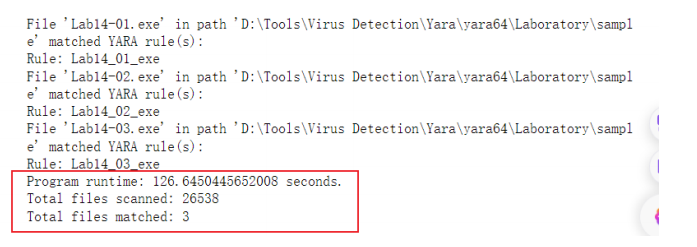
condition:

all of them //Lab14-03.exe

}

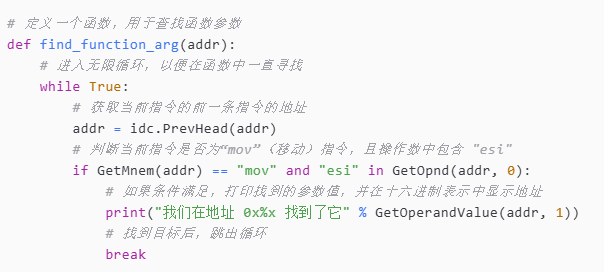
扫描脚本同前几次

**2.检测结果**



**IDAPython 辅助样本分析**

1.查找函数参数的Python脚本：



2.提取完整字符串的Python脚本：



3.反编译并打印函数的Python脚本：



**三、实验结论**

在本次实验中，我通过综合运用静态分析工具和动态分析方法，对Lab14的三个恶意代码样本进行了深入的分析，并成功回答了实验中提出的问题。重点分析了恶意代码通过注入实现隐蔽启动的多种方式，包括对操作系统级别的更改。总体来说，实验取得了圆满成功，不仅加深了我对病毒分析工具IDAPro的理解，还提升了我在动态分析和静态分析方面的能力。

**四、心得体会**

通过这次实验，我获得了宝贵的知识和实践经验，具体包括：

熟练掌握IDAPro：我进一步熟练掌握了IDAPro这一强大的病毒分析工具，它在动态分析中发挥了重要作用。

提升动态分析能力：通过使用OllyDBG等工具，我提高了自己在动态分析和反混淆解密方面的能力。

总结来说，这次实验不仅让我对病毒分析有了更全面的认识，还激发了我对安全领域的兴趣。我将继续努力学习，以提升我的病毒分析技能。