



DOKUMENTACE 2. PROJEKTU IPK

varianta ZETA: Sniffer Paketů

Ondřej Mach (xmacho12)

Contents

1 Úvod	2
2 Implementace	2

1 Úvod

Tento projekt se zabývá problematikou zachytávání paketů v počítačové síti. Cílem tohoto projektu je implementace programu, který dokáže zachyčovat pakety na vybraném rozhraní. Tento program dále filtruje pakety podle protokolu a vypisuje čitelný výstup.

2 Implementace

2.1 Obecné údaje

Program byl implementován v jazyce C++ s využitím knihovny `libpcap`. Tato knihovna poskytuje high-level rozhraní zachycování paketů. Její implementace zpřístupní i pakety, které nejsou určeny pro daný počítač. `Libpcap` je kompatibilní se systémy UNIXového typu i s Windows, tento projekt je však implementován s ohledem pouze na UNIX. Vývoj probíhal na Linuxu, překlad byl otestován i na FreeBSD.

2.2 Chování programu

Program je spouštěn z příkazové řádky a svůj výstup posílá na `stdin`. V případě jakékoli chyby (typicky nedostatečná oprávnění) vypíše chybovou hlášku na `stderr` a ukončí se. Každý zachycený paket se vypíše jako zpracované hlavičky a poté RAW data s přepisem do ASCII po straně.

Na linkové vrstvě (L2) program podporuje pouze Ethernet hlavičky. Z nich jsou na výstup vypsány MAC adresy zdroje a cíle.

Na síťové vrstvě (L3) je již rozlišováno mezi IPv4, IPv6, ARP a ostatními pakety. V případě, že je nalezena IPv4 nebo IPv6 hlavička, je na výstup vypsána IP adresa zdroje a cíle.

Na transportní vrstvě (L4) jsou podporovány protokoly TCP a UDP. Pokud je jeden z nich rozpoznán, na výstup je zapsán zdrojový a cílový port.

2.3 Struktura programu

Celá struktura je přizpůsobena knihovně `libpcap`. Po spuštění jsou nejprve parsovány argumenty pomocí funkce `getopt_long` (GNU rozšíření `getopt`). Dále se program pokusí otevřít dané síťové rozhraní, případně vypsat všechna dostupná rozhraní.

V další fázi běhu program zkompile zadáný filtr funkcí `pcap_compile` a aplikuje ho na dané rozhraní.

Nakonec je spuštěna funkce `pcap_loop`, která odchytlí zadáný počet paketů a pro každý zavolá `packet_callback`. `packet_callback` je nejdůležitější funkce v programu, která provádí samotnou analýzu paketů a jejich výpis. Zde jsou používány i pomocné funkce, které vypisují data ve správných formátech. Mezi ně patří třeba `formatTimestamp`, `formatMAC` nebo `formatIPv6`.

3 Testování

3.1 Nastavení

Pro test v praxi byl vybrán protokol Telnet. Dříve byl velmi hojně používán pro vzdálenou správu, dokud nebyl nahrazen SSH. Telnet je také nechalně proslulý svým nešifrovaným provozem, proto je ideálním kandidátem k zachytávání paketů. Test byl uskutečněn mezi hostitelským operačním systémem a hostem běžícím ve virtuálním stroji. Na hostovi byl nainstalován a spuštěn telnet server. Na hostiteli byl spuštěn wireshark a ipk-sniffer pro sledování provozu. Nakonec se hostitel připojil přes telnet k serveru běžícímu na hostovi.

3.2 Spuštění

```
sudo ./ipk-sniffer -i virbr0 -n 100 -p 23
```

ipk-sniffer je potřeba spouštět jako root, protože běžný uživatel nemá dostatečná oprávnění pro zachycování paketů. Argument `-i` značí rozhraní, `-n` značí počet paketů. Argument `-p` značí port - v tomto případě je použit port 23, což je standardní port pro Telnet.

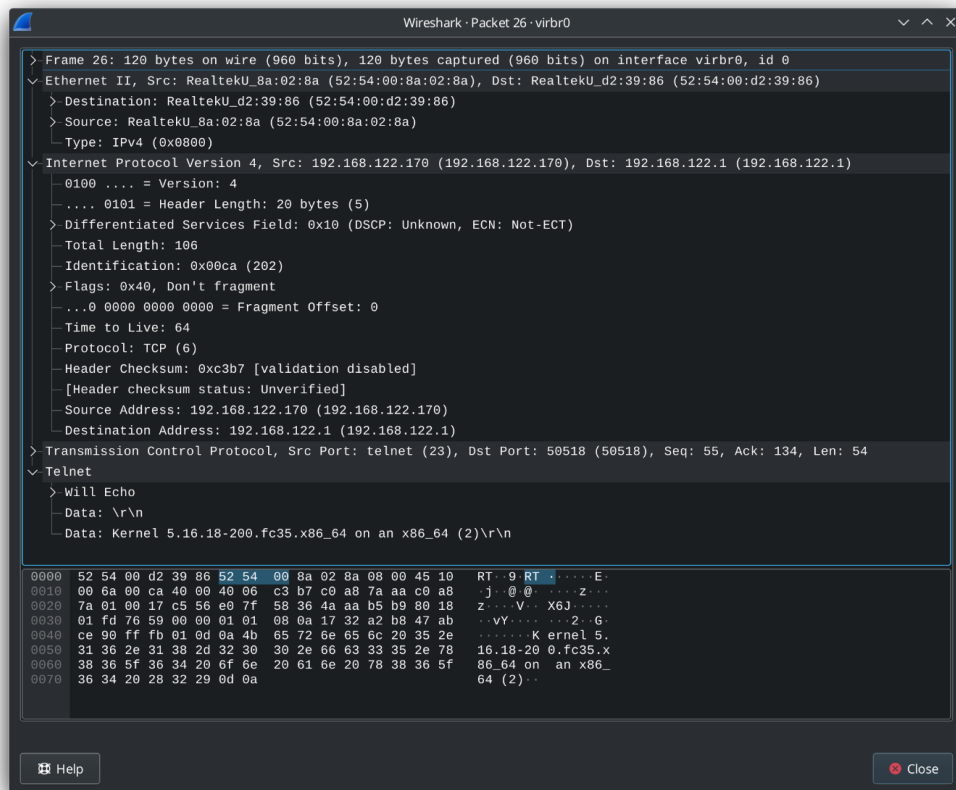
Po zahájení spojení se na wiresharku i v našem snifferu začaly okamžitě ukazovat pakety. Bohužel telnet není snadno čitelný a srozumitelný v ascii a uživatelský vstup chodí po jednom písmenu. Byl vybrán jeden paket, ve kterém telnet server vypisuje zprávu před přihlášením. Níže je umístěn screenshot tohoto paketu v ipk-snifferu v porovnání s Wiresharkem.

```

timestamp: 2022-04-17T12:28:10.913828+02:00
src MAC: 52-54-0-8a-2-8a
dst MAC: 52-54-0-d2-39-86
frame length: 120 bytes
src IP: 192.168.122.170
dst IP: 192.168.122.1
src port: 23
dst port: 50518

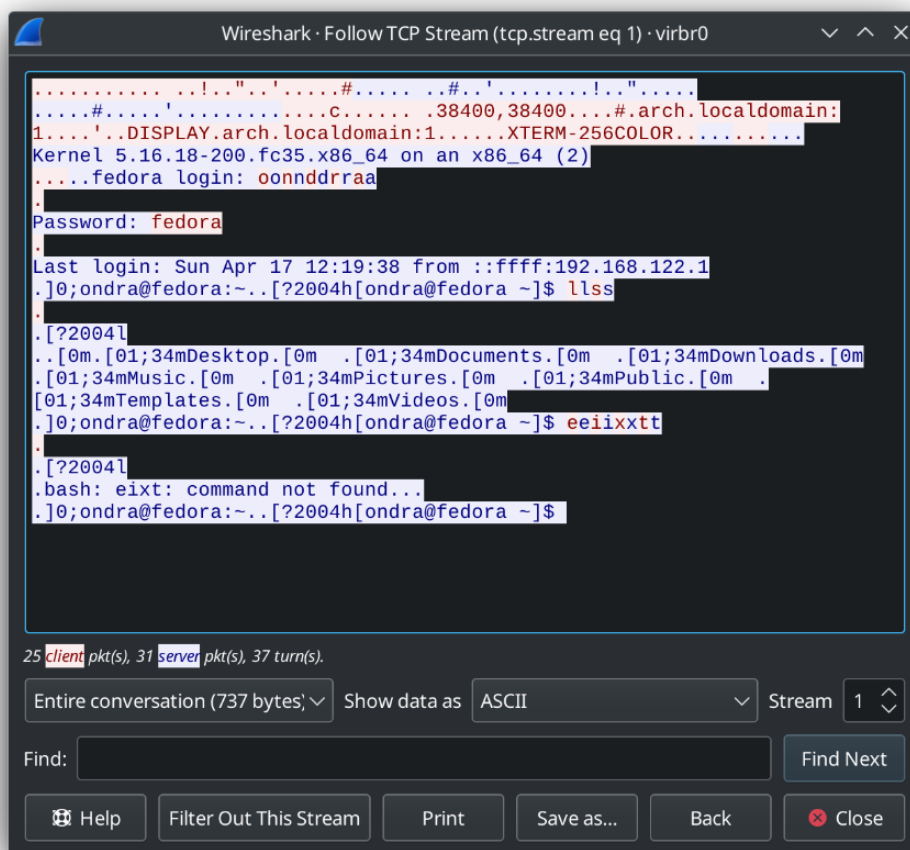
0x0000: 52 54 00 d2 39 86 52 54 00 8a 02 8a 08 00 45 10 RT..9.RT.....E.
0x0010: 00 6a 00 ca 40 00 40 06 c3 b7 c0 a8 7a aa c0 a8 .j..@. ....z...
0x0020: 7a 01 00 17 c5 56 e0 7f 58 36 4a aa b5 b9 80 18 z....V..X6J....
0x0030: 01 fd 76 59 00 00 01 01 08 0a 17 32 a2 b8 47 ab ..vY.....2..G.
0x0040: ce 90 ff fb 01 0d 0a 4b 65 72 6e 65 6c 20 35 2e .....Kernel 5.
0x0050: 31 36 2e 31 38 2d 32 30 30 2e 66 63 33 35 2e 78 16.18-200.fc35.x
0x0060: 38 36 5f 36 34 20 6f 6e 20 61 6e 20 78 38 36 5f 86_64 on an x86_
0x0070: 36 34 20 28 32 29 0d 0a                               64 (2)..

```



Z výpisů je zřetelné, že se oba programy shodují na významu paketů a interpretaci headerů.

Přestože v našem programu není příliš čitelné jaká data byla vyměněna, protokol není šifrovaný. Proto je ve Wiresharku funkce follow TCP stream, která sleduje data v paketech a poté je vypíše na jednu obrazovku (screenshot).



The image shows a Wireshark window titled "Wireshark · Follow TCP Stream (tcp.stream eq 1) · virbr0". The main pane displays a terminal session with the following text:

```
.....!..".'.#.....#..'.!..".....  
.....#.....'.c..... .38400,38400....#.arch.localdomain:  
1.....'.DISPLAY.arch.localdomain:1.....XTERM-256COLOR.....  
Kernel 5.16.18-200.fc35.x86_64 on an x86_64 (2)  
.....fedora login: oonndrraa  
Password: fedora  
Last login: Sun Apr 17 12:19:38 from ::ffff:192.168.122.1  
.]0;ondra@fedora:~.[?2004h[ondra@fedora ~]$ llss  
.[?2004l  
..[0m.[01;34mDesktop.[0m .[01;34mDocuments.[0m .[01;34mDownloads.[0m  
.[01;34mMusic.[0m .[01;34mPictures.[0m .[01;34mPublic.[0m .  
.[01;34mTemplates.[0m .[01;34mVideos.[0m  
.]0;ondra@fedora:~.[?2004h[ondra@fedora ~]$ eeiiixtt  
.[?2004l  
.bash: eixt: command not found..  
.]0;ondra@fedora:~.[?2004h[ondra@fedora ~]$
```

Below the terminal output, the status bar shows "25 client pkt(s), 31 server pkt(s), 37 turn(s)". The "Show data as" dropdown is set to "ASCII". The "Stream" dropdown is set to "1". The "Find:" field is empty. At the bottom, there are buttons for "Help", "Filter Out This Stream", "Print", "Save as...", "Back", and "Close".