VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK – Projekt č. 2
Zeta: Sniffer paketů

Adam Sedláček | xsedla1e

27.4.2020

Obsah

Úvod

Zadání úlohy bylo naprogramovat sniffer paketů na daném rozhraní. Byly zde i možnosti lehké filtrace v podobě UDP/TCP protokolu, a daného portu. Výpis v podobě hexdump.

Sniffer se hodí zejména na analyzování paketů, jak jejich obsahu, tak už jen to z jaké adresy se k nám dostávají informace, případně obdobně u portu, jestli komunikujeme například na “zabezpečeném” portu 443 (https), nebo klasickém 80 (http). Wireshark umožňuje daný stream paketů zachytit a později jej přehrát, pokud se jedná o službu co toto zajišťuje.

Tudíž sniffer se hodí např. pro poskytovatele internetu nebo nějaké streamované služby, případně se může hodit i při odhalování nějaké nekalého podstrkávání paketů nebo zjištění tajného odesílání dat, kdy o tom uživatel nemusí ani tušit (nejmenovaný operační systém).

Implementace

Pro implementaci jsem si zvolil C/C++, hlavně z důvodu jednoduché kompatibility mezi jazyky a knihovnými funkcemi. Využil jsem PCAP knihovnu, která velmi ulehčila implementaci. Vesměs celý projekt je zavolání asi 5 funkcí z této knihovny. Kde tato knihovna chytře řeší pomocí callback volání při odchytu paketu.

Využil jsem i některé předpřipravené struktury/funkce z jiných knihoven, jako je například *time.h* pro tisk aktuálního času, nebo *netinet.h* pro IP/UDP/TCP hlavičky, kde je přímo předpřipravená struktura a hezky se dá "napasovat" přímo a nemusí tedy člověk sám ručně psát tuto strukturu (ani bitové operace, které jsou pomocí bitfieldů). Což opět ušetřilo velmi práci a spousty nervů.

Argumenty programu

Zvolil jsem cestu ručního naprogramování, a to hlavně z důvodu, že uživatel klidně může zadat vícero parametrů a já vím přesně co jak bylo zadáno. Dovoluji tedy zavolat spustit daný program s dvojitými argumenty, vždy se bere ten poslední `./ipk-sniffer -i enp0s3 -n 10 -p 30 -p 80` zde se vezme port 80 nikoliv 30.

Veškeré argumenty jsou volitelné, kromě rozhraní. Pokud, ale uživatel spustí program pouze s názvem programu nebo s parametrem *-i*, dostane seznam všech dostupných rozhraní.

Pokud je zadáno cokoliv špatně, program se ukončí s návratovou hodnotou 1 a dá uživateli najevo co bylo špatně.

Parametry spuštění

- -i *rozhraní* tento parametr je povinný
- -p X číslo portu je volitelné, pokud není zadáno bere se libovolný port
- --tcp | -t pouze TCP pakety
- --udp | -u pouze UDP pakety
- -n počet kolik paketů se má zobrazit, výchozí je n = 1

Pokud se nezvolí `--tcp` nebo `--udp` berou se libovolný protokol (tedy udp, i tcp naráz).
Port je číselná hodnota od 0 do 65535.
Počet paketů se bere jedna a větší (omezení je zde do int32).

Příklady spuštění:

```
./ipk-sniffer
./ipk-sniffer -i
./ipk-sniffer -i eth0 -p 23 --tcp -n 2
./ipk-sniffer -i eth0 -u
./ipk-sniffer -i eth0 -n 100
./ipk-sniffer -i eth0 -p 22 --tcp --udp
./ipk-sniffer -i eth0 -p 21
./ipk-sniffer -i eth0
```

Výstup snifferu

Dovolil jsem si malinko pozměnit od ukázky, která byla poskytnuta. A to zarovnáním, aby výstup byl pokaždé konzistentní a “hezký”. Pokud se nepřenesla žádná data, je vypsáno “No data”. Čas je brán systémový a ne z paketu, tím se lehce eliminuje problém s časovými pásmy apod.

Bez dat

```
00:06:34 student-vm : 51208 > server-13-32-99-166.prg50.r.cloudfront.net : 443
No data.
```

S daty v podobě hexdump (z nepochopeného důvodu problém s formátováním, tudíž nahrazeno výstřižkem)

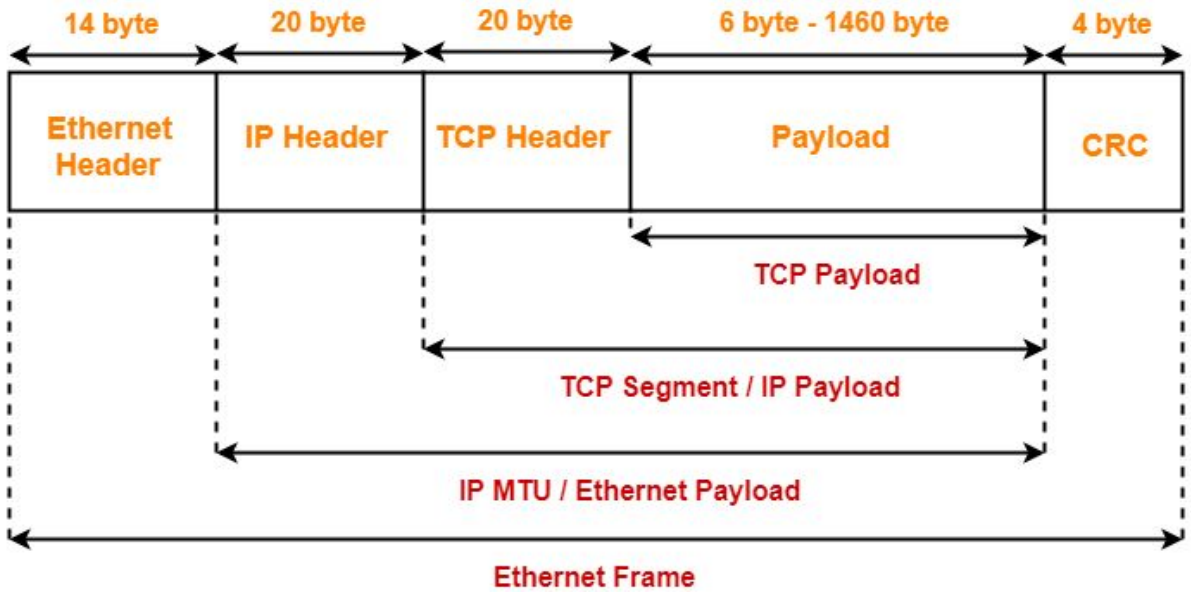
```
00:06:38 student-vm : 55291 > resolver1.opendns.com : 53

0x0000  61 4a 01 00 00 01 00 00 00 00 00 01 06 61 73 69  aJ...........asi
0x0010  6d 6f 76 06 76 6f 72 74 65 78 04 64 61 74 61 09  mov.vortex.data.
0x0020  6d 69 63 72 6f 73 6f 66 74 03 63 6f 6d 06 61 6b  microsoft.com.ak
0x0030  61 64 6e 73 03 6e 65 74 00 00 01 00 01 00 00 29  adns.net.......)
0x0038  02 00 00 00 00 00 00 00                          ........
```

PCAP

Využil jsem následující funkce, které API poskytuje:

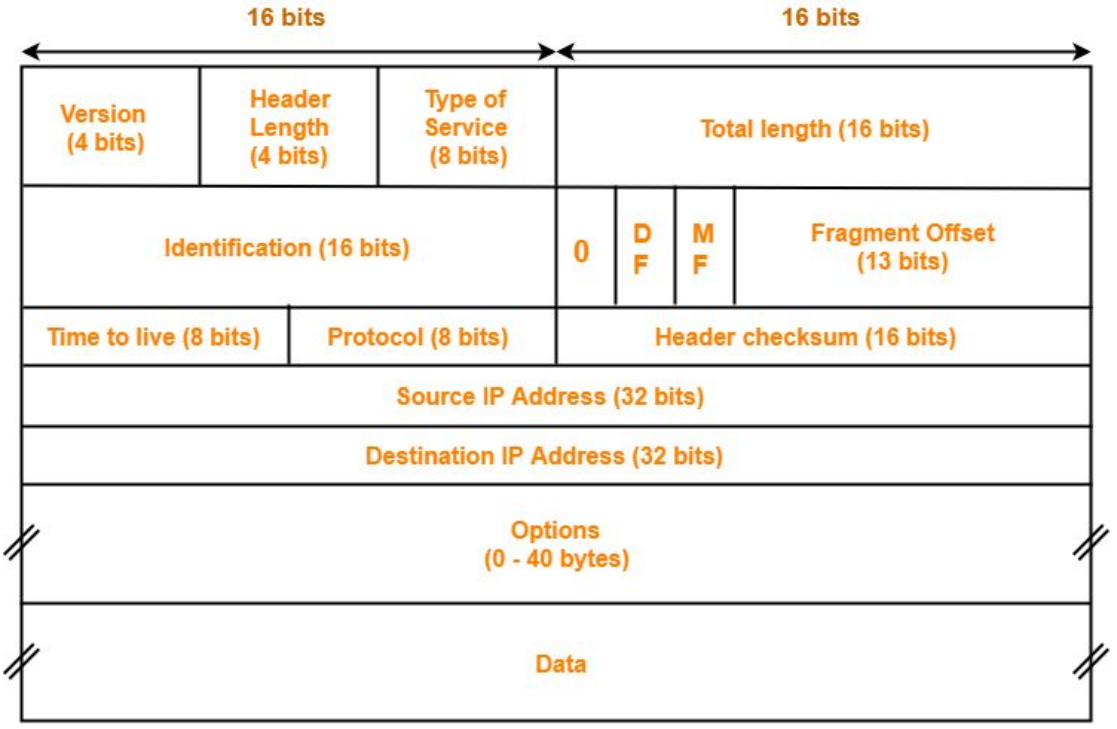
- pcap_lookupnet - zjištění IP adresy a masky pro dané rozhraní
- pcap_findalldevs - zjistí všechny dostupné rozhraní
- pcap_open_live - otevření odchytu paketů na síti a následnou práci s pakety
- pcap_compile - zkompiluje string do požadovaného formátu pro pcap filtr
- pcap_setfilter - nastaví předkompilovaný filtr
- pcap_loop - využití na odchycení N paketů a vždy když odchytí packet zavolá se callback funkce, kde se může řešit co s daným paketem se udělá.
- pcap_close - uzavře a uvolní struktury, které využívá pcap

Ethernet rámec



Ethernetová hlavička má vždy 14 bajtů. Jak je na obrázku výše, tak se jednotlivé vrstvy “balí do sebe”, to je hlavní pointa vícevrstvé architektury, tj. není potřeba zpětná kompatibilita, protože vždy daná hlavička je jako data jiného “segmentu”. Tudíž zjistíme zda se jedná o ethernetovou hlavičku, poté můžeme přistoupit na IP hlavičku, z té opět přistoupit na TCP/UDP hlavičku a pak až k samotným datům. V projektu nebyla potřeba, ale mohla by se tisknout například MAC adresa cíle/zdroje.

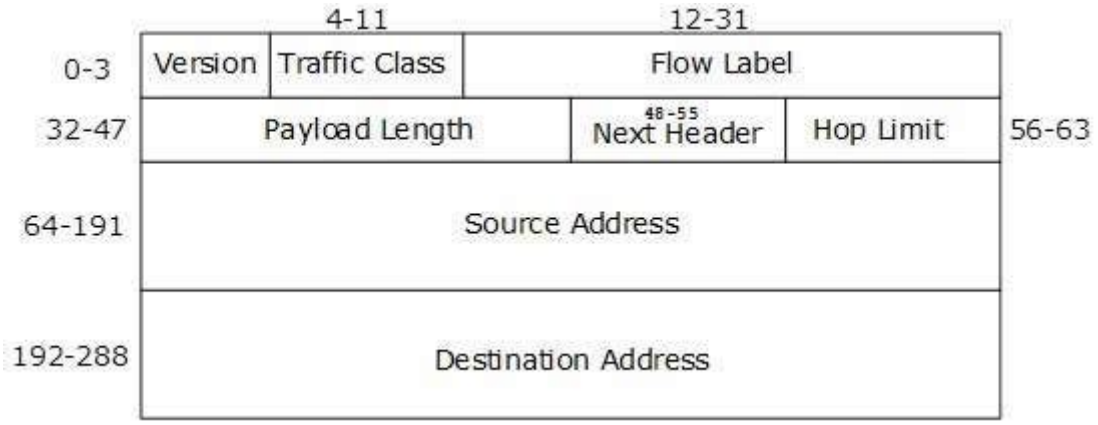
IP hlavička



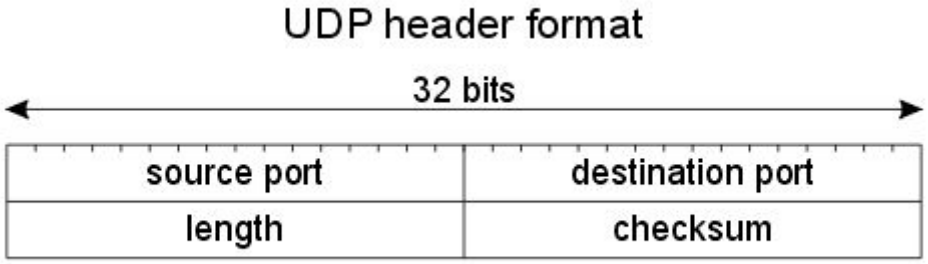
IPv4 Header

Pro jednoduchou práci s IP hlavičkou jsem si pomohl datovou strukturou z *netinet/ip.h,* která hezky namapovává od daného pointeru, takže už stačí pouze vybrat správný údaj. Hned jako první údaj je verze protokolu, protože IPv6 je rozdílné od IPv4, například v proměnlivé délce hlavičky (header length), který se musí počítat “ručně”, kde velikost může být od 20 do 60 bajtů, a to hlavně kvůli poli options. Pro rozeznání zda se jedná o UDP nebo TCP jsem využil protocol kolonku, kde 6 je pro TCP a 17 pro UDP (plný seznam se dá dohledat online například v RFC) .

IPv6 hlavička je fixní délku 40 bajtů. Od 4kové verze se liší v tom, že se značně zjednodušila. Ale dané informace stále má. Také chybí nepotřebný checksum, o který se stará jiná vrstva.

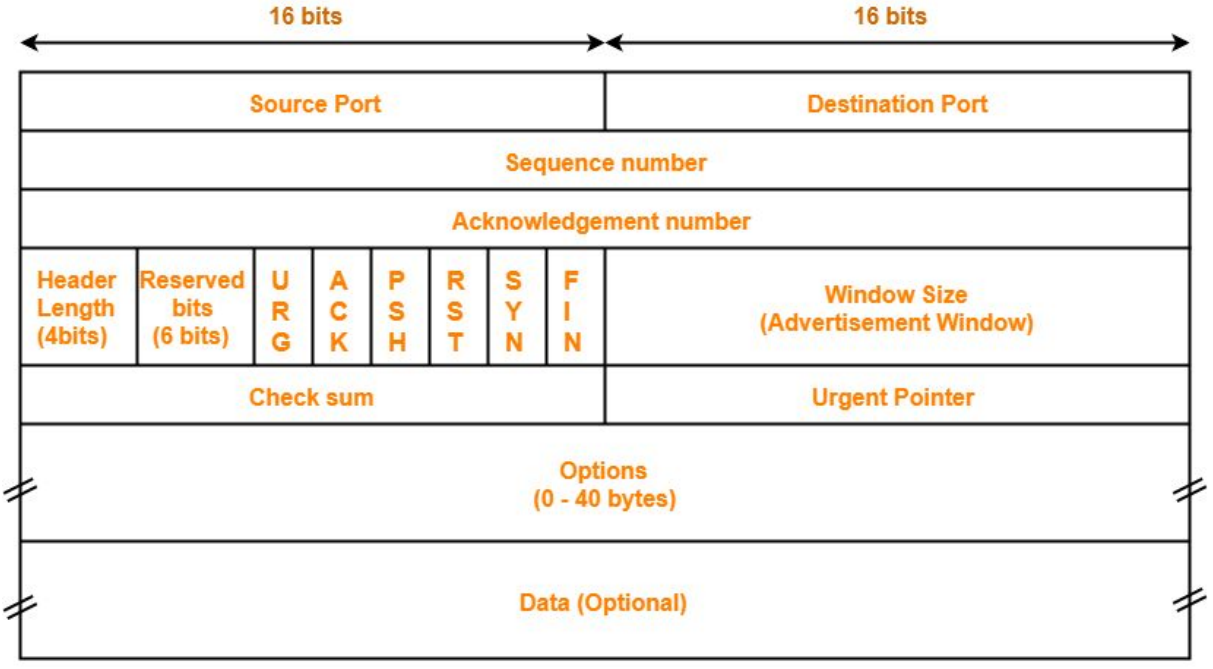


UDP hlavička



UDP hlavička má vždy fixní hodnotu, takže se s ní pracuje velmi příjemně. Opět jsem využil *netinet/udp.h,* takže práce byla ještě o to příjemnější. Zde jsem pouze vytáhl source port a destination port.

TCP hlavička



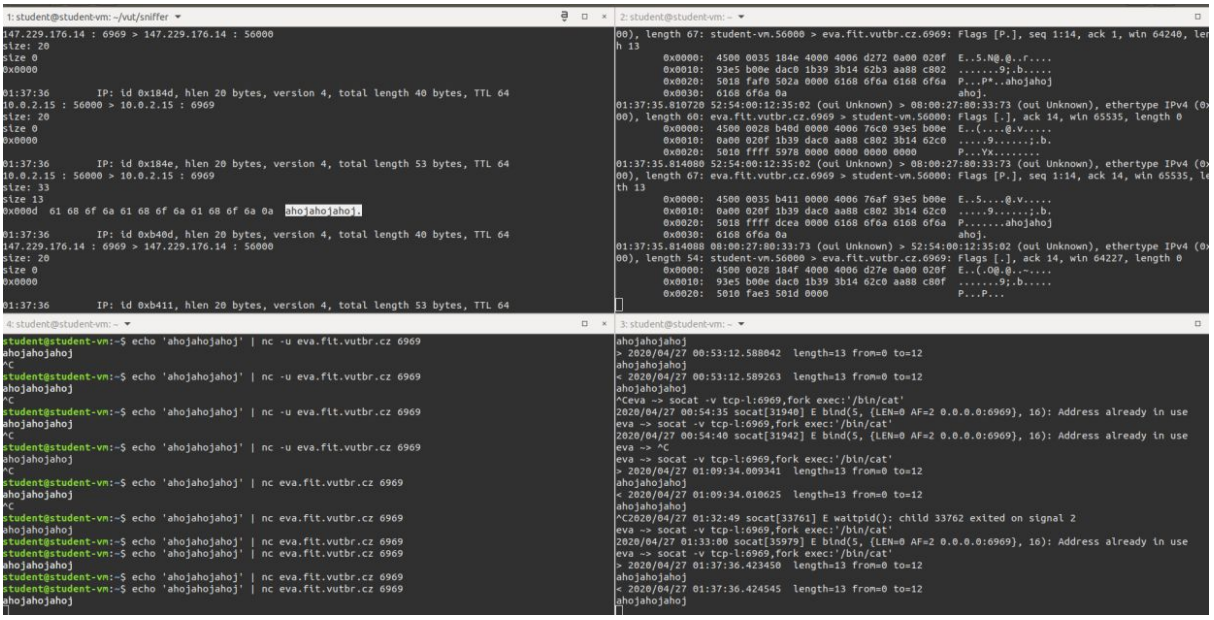
TCP Header

Opět jsem využil *netinet/tcp.h*, která práci opět zpříjemnila, zde je ale opět variabilní délka hlavičky a je potřeba dopočítat, kde se budou data nacházet. Tudíž celý header může mít od 20 do 60 bajtů. Potřebnou délku je třeba vymaskovat nebo využít bitfield, takto dostaneme pak jak zjistit pointer na data.

Testování

K testování jsem si otevřel vlastní port, kde jsem si poslal jednoduše čitelnou zprávu pomocí NetCat, a to “ahojahojahoj”, kde jsem poté sledoval jak se zachová TCPdump.

Využil jsem i Wiresharku, kde jsem sledoval provoz na síti pro 100 paketů a poté zkontroloval, zda to odpovídá s mojím snifferem.



Literatura | Odkazy

http://www.tcpdump.org/pcap3_man.html
https://tools.ietf.org/html/rfc791
https://tools.ietf.org/html/rfc793
https://tools.ietf.org/html/rfc768
http://man7.org/linux/man-pages/man3/getaddrinfo.3.html
http://man7.org/linux/man-pages/man3/getnameinfo.3.html
https://linux.die.net/man/3/inet_ntoa
https://linux.die.net/man/3/ntohs

KUROSE, James F. a Keith W. ROSS. *Computer networking: a top-down approach*. Seventh edition; Global edition. Essex: Pearson, 2017, 852 stran : ilustrace (některé barevné) ; 23 cm. ISBN 978-1-292-15359-9.