Projekt 2 z predmetu IPK

Variant ZETA: Sniffer paketov

Samuel Budai

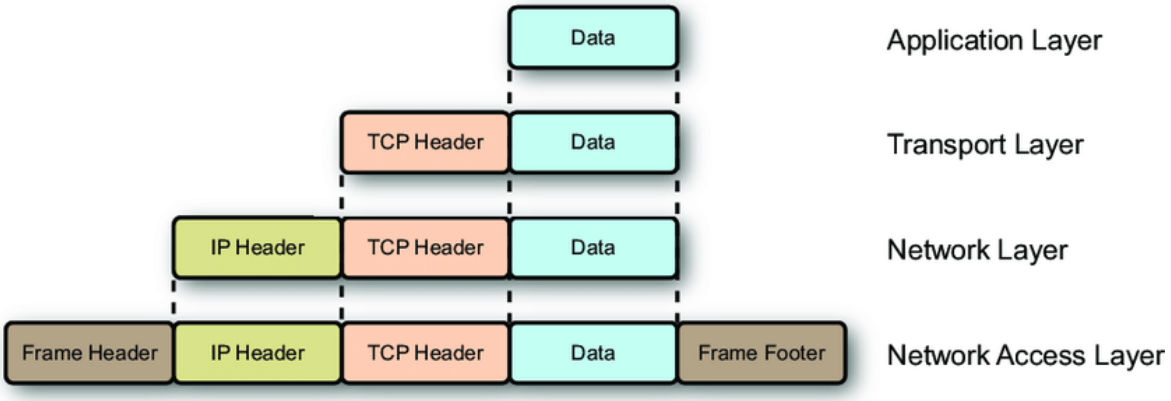
xbudai01

22.4.2022

Obsah

1. Úvod

Úlohou tohto projektu bolo navrhnúť a implementovať sieťový analyzátor v C/C++/C# schopný na určitom sieťovom rozhraní zachytávať a filtrovať prichádzajúce pakety. V prvom rade bolo potrebné naštudovať formát protokolu ako aj formáty hlavičiek protokolov ktoré sieťový analyzátor spracováva, aby následne mohol vypísať konkrétne údaje pre daný paket [1].



Obrázok 1 Názorná štruktúra paketového zapuzdrenia [2]

Ako môžeme vidieť z Obrázka 1 na začiatku paketu sa nachádza tzv. hlavička eternetového rámca (MAC hlavička). Táto hlavička má štandardne veľkosť 14 bytov. Obsahuje adresu prijímateľa, adresu odosielateľa obe s veľkosťou 6 bytov a informácie o type prevádzky (EtherType) s veľkosťou 2 byte-i [1].

Za hlavičkou eternetového rámca nasleduje sieťová vrstva, teda konkrétne hlavička protokolu IP. Ktorá môže byť buď vo verzii 4 (IPv4) alebo vo verzii 6 (IPv6). Protokol IPv4 je štvrtou verziou protokolu IP a jedným zo základných protokolov pre komunikáciu v súčasnom Internete. Adresovanie tohto protokolu je založené na 32-bitových adresách a hlavička tohto protokolu má nasledujúci formát:

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
⋮	⋮																																
56	448																																

Obrázok 2: Formát hlavičky IPv4 [3]

Údaje ktoré nás pre naše potreby zaujímajú z Obrázka 2, sú: nultý byte hlavičky v ktorom sa na prvých 4 bitoch nachádza verzia protokolu a na zvyšných 4 bitoch veľkosť hlavičky, nakoľko veľkosť IPv4 hlavičky sa môže meniť. Ďalej deviaty byte v ktorom sa nachádza typ transportného protokolu

teda či sa jedná napr. o TCP/UDP komunikáciu. Nakoniec nás zaujímajú ešte štyri byte-i s ofsetom 12 bytov a 4 byte-i s ofsetom 16 bytov kde sa nachádzajú IP adresy odosielateľa a prijímateľa [3].

Druhou verziou spomínaného protokolu IP je verzia 6 teda IPv6 označuje nastupujúci protokol pre komunikáciu. Postupne nahradzuje protokol IPv4. Jeho hlavnou výhodou je rozšírenie adresného priestoru a lepšia schopnosť prenášať dáta vysokou rýchlosťou. Čo treba podotknúť je fixná veľkosť hlavičky tohto formátu a to 40 bytov. Z hlavičky tohto protokolu nás bude zaujímať protokol ktorý má v tejto hlavičke označenie „Next header“ a pristúpime k nemu s ofsetom 6bytov. A rovnako ako pri IPv4 nás bude zaujímať adresa prijímateľa a odosielateľa ktorá ale v tomto prípade má veľkosť 16 bytov teda k nim musíme pristupovať s ofsetom 8 a 24 [4].

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Traffic class								Flow label																			
4	32	Payload length																Next header								Hop limit							
8	64	Source address																															
12	96																																
16	128																																
20	160																																
24	192	Destination address																															
28	224																																
32	256																																
36	288																																

Obrázok 3: Fixný formát hlavičky IPv6 [4]

V prípade ak získaný transportný protokol spadá do kategórie TCP/UDP vieme z neho extrahovať ďalšie užitočné informácie pre náš sieťový analyzátor ako sú porty prijímateľa a odosielateľa ktoré sa v týchto protokoloch nachádzajú na prvých dvoch bytoch [5][6].Protokoly ICMP a ARP nebolo nutné viac rozoberať, nakoľko ich zo zadania náš sieťový analyzátor nijak viac neanalyzuje.

2. Implementácia

Prvým dôležitým krokom ktorý náš sieťový analyzátor musí vykonať pri spustení je spracovanie vstupných argumentov na ktoré sme v implementácii použili knižnicu *getopt.h* [7]**.** Po spracovaní a kontrole správnosti argumentov, prichádza v prípade ak bol pre skript zadaný parameter *-i* kontrola správnosti a existencie daného rozhrania pomocou funkcie *check_interface*. Ak parameter zadaný nebol, vypíšu sa dostupné rozhrania pre daný systém. Vykoná sa tak pomocou funkcie založenej na podobnom princípe a to funkciou *print_all_interfaces* [8]. Ďalším krokom je príprava zariadenia, vytvorenie a nastavenie filtra podľa zadaných argumentov a následné otvorenie spojenia pre zachytávanie paketov z premávky pomocou funkcie *pcap_loop*. Implementácia tejto časti bola vykonaná pomocou knižnice *pcap.h* [9][10][11].

Následne je v rámci kódu implementovaná tzv. “callback” funkcia ktorá spracováva zachytené pakety, extrahuje z nich dôležité informácie a následne tieto informácie tlačí na štandardný výstup. Pre potreby „callback“ funkcie a jednoduchšiu prácu s eternetovým rámcom sme prevzali štruktúru reprezentujúcu hlavičku tohto rámca [10]. Pomocou tejto štruktúry a vlastností jednotlivých hlavičiek

protokolov ktoré sme spomínali v úvode, sa z daného paketu získajú potrebné informácie ako čas zachytenia paketu ktorý sa následne spracuje pomocou pomocnej funkcie *timeval_to_string* ktorá prekonvertuje časový údaj vo formáte timeval do podoby ktorá sa dá ľahko vytlačiť [12]. Ďalej sa v správnom formáte extrahuje MAC adresa [13]. Veľkosť paketu, a v prípade TCP/UDP transportných protokolov aj IP adresa a porty prijímateľa a odosielateľa. Na záver "callback" funkcie sa pomocou tzv. „hexdump-u" v programe reprezentovaného funkciou *print_packet_data* vypíše okrem týchto vlastností aj celkový obsah paketu [14].

3. Testovanie

Testovanie prebiehalo v troch rôznych fázach. Prvým druhom testovania bolo zachytávanie premávky na porte 443, s ktorým sme testovali funkčnosť otvorenia spojenia ako aj zachytávanie paketov. Po usúdení, že všetko vyzerá v poriadku sme jednotlivé funkcionality ručne otestovali pomocou linuxové nástroja Netcat vďaka ktorému sme vedeli otestovať funkciu protokolov IPv4, IPv6 ale aj TCP, UDP, ICMP a ICMP6. Poslednou fázou testovania bola kontrola správnosti dát na ktorú sme využili nástroj s licenciou open source a to Wireshark. Naším sieťovým analyzátorom sme zachytávali simultánne s wiresharkom pakety na živej premávke a následne sme vykonali ručnú kontrolu správnosti dát, teda či sa zachytené dáta zhodujú.

4. Zdroje

[1] Ethernet frame. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2021. [cit. 24.04.2022]. Dostupné z: https://en.wikipedia.org/wiki/Ethernet_frame

[2] Packet encapsulation. TCP/IP architecture encapsulates the data from... | Download Scientific Diagram. *ResearchGate | Find and share research* [online]. Copyright © 2008 [cit. 24.04.2022]. Dostupné z: https://www.researchgate.net/figure/Packet-encapsulation-TCP-IP-architecture-encapsulates-the-data-from-the-upper-layer-by_fig4_49288737

[3] IPv4. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022. [cit. 24.04.2022]. Dostupné z: https://en.wikipedia.org/wiki/IPv4

[4] IPv6 packet. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022. [cit. 24.04.2022]. Dostupné z: https://en.wikipedia.org/wiki/IPv6_packet

[5] Transmission Control Protocol. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 24.04.2022]. Dostupné z: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[6] User Datagram Protocol. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022. [cit. 24.04.2022]. Dostupné z: https://en.wikipedia.org/wiki/User_Datagram_Protocol

[7] Getopt (The GNU C Library). *The GNU Operating System and the Free Software Movement* [online]. Dostupné z: https://www.gnu.org/software/libc/manual/html_node/Getopt.html

[8] C - pcap_findalldevs to display all interfaces is stuck in an infinite loop - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. 2020. Dostupné z: https://stackoverflow.com/questions/61370713/c-pcap-findalldevs-to-display-all-interfaces-is-stuck-in-an-infinite-loop

[9] Using libpcap in C. DevDungeon, *Virtual Hackerspace* [online]. 2015. Dostupné z: https://www.devdungeon.com/content/using-libpcap-c

[10] Carstens T. *Programming with pcap* [online]. 2002. Dostupné z: https://www.tcpdump.org/pcap.html

[11] pcap_compile(3): compile filter expression - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/3/pcap_compile

[12] time - I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers*. 2018. [online]. Dostupné z: https://stackoverflow.com/questions/48771851/im-trying-to-build-an-rfc3339-timestamp-in-c-how-do-i-get-the-timezone-offset

[13] c++ - Is these a way to set the output of printf to a string? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers. 2013.* [online]. Dostupné z: https://stackoverflow.com/questions/19382198/is-these-a-way-to-set-the-output-of-printf-to-a-string

[14] Let's Build a Hexdump Utility in C. *Darren Mulholland* [online]. Dostupné z: http://www.dmulholl.com/lets-build/a-hexdump-utility.html