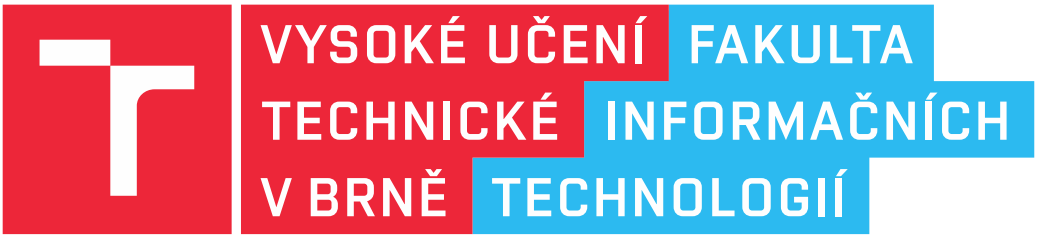


Projektová dokumentácia

Implementácia sniffera packetov

IPK, projekt č.2

23. apríla 2022 Tomáš Ondrušek (xondru18)

Obsah

Úvod

Cieľom projektu bolo implementovať program, ktorý dokáže odchytávať sieťovú komunikáciu na danom rozhraní, pomocou filtra získať len zhodné pakety a následne ich vypísať v požadovanom formáte. Projekt bol naimplementovaný v jazyku C++ hlavne pre jeho rozšírenie STD oproti jazyku C, ktoré umožňuje jednoduchšiu prácu so 'stringami'. Program umožňuje odchytávať a filtrovať pakety na rôznych portoch a s rôznymi protokolmi. Medzi podporované protokoly patria UDP, TCP, ARP, ICMP.

Teória

Pri študovaní problematiky boli použité ako zdroje informácií tcpdump[1] a stackoverflow[2].

Sniffer

Sniffer paketov – tiež známy ako analyzátor paketov, analyzátor protokolov alebo sieťový analyzátor – je softvér používaný na monitorovanie sieťovej prevádzky. Sniffery fungujú tak, že skúmajú toky dátových paketov, ktoré prechádzajú medzi počítačmi v sieti, ako aj medzi počítačmi v sieti a väčším internetom. Tieto pakety sú určené a adresované konkrétnym počítačom, ale používanie paketového sniffera v „transparentnom režime" umožňuje IT profesionálom, koncovým používateľom alebo útočníkom preskúmať akýkoľvek paket bez ohľadu na miesto určenia.

Paket

Paket označuje v informatike blok dát prenášaný v počítačových sieťach založených na prepojovaní paketov. Paket sa skladá z riadiacich dát (metadát) a z užívateľských dát (užitočné zaťaženie, anglicky payload). Riadiace dáta poskytujú sieti potrebné dáta na doručenie paketu, napríklad adresu zdroja a ciela, kódy pre detekciu chýb, kontrolné súčty a informácie o poradí. Obvykle sa riadiace dáta nachádzajú v hlavičkách paketov a na ich konci, pričom užívateľské dáta sú medzi nimi.

Rôzne komunikačné protokoly používajú rôzne konvencie na rozlišovanie medzi riadiacimi prvkami a dátami. V binárnych synchrónnych prenosoch používajú pakety 8-bitové skupiny (bajty) a na vymedzenie jednotlivých prvkov sú použité špeciálne znaky. Ďalšie protokoly (napríklad Ethernet), definujú začiatok hlavičky a dátových prvkov ako pozíciu vzhľadom na začiatok paketu. Niektoré protokoly formátujú informácie na úrovni bitov namiesto použitia bajtovej úrovne [1].

1 https://www.tcpdump.org/manpages/pcap.3pcap.html
2 https://stackoverflow.com/questions/tagged/pcap

Implementácia

Implementačné detaily

Implementácia začína spracovaním argumentov, kde sa roztriedia argumenty a zároveň sa nastavia 'flagy' pre jednotlivé protokoly. Následne sa vytvorí filter za pomoci spomínaných flagov a voliteľného portu, ktorý bude použitý na odchytenie paketov. Tento port sa používa len pri UDP a TCP, keďže ARP ani ICMP nemajú protokoly. Tento filter zatiaľ v tvare 'stringu' je skompilovaný pomocou funkcie pcap_compile. Následne je filter aplikovaný pomocou pcap_setfilter a začnú sa odchytávať pakety pomocou funkcie pcap_loop. Keď už sú pakety odchytené je zavolaná funkcia, ktorá paket spracuje cez switch-case konštrukciu a podľa príslušného protokolu sú dáta v pakete naformátované a vytlačené na STDOUT. Hrubšia štruktúra programu bola inšpirovaná sniffex-om[3] a návodom na na prácu s pcap [2].

Knižnice

Na implementáciu sniffera boli použité nasledovné knižnice na správu paketov, hlavne knižnica pcap.h

- <iostream>
- <string>
- <string.h>
- <stdio.h>
- <unistd.h>
- <ctype.h>
- <cmath>
- <sstream>
- <pcap.h>
- <net/ethernet.h>
- <netinet/ip.h>
- <netinet/ip_icmp.h>
- <netinet/in.h>
- <netinet/tcp.h>
- <netinet/udp.h>
- <netinet/if_ether.h>
- <netinet/ether.h>
- <netinet/ip6.h>
- <signal.h>

Netinet knižnice boli použité na prácu s hlavičkami paketov a prácu s nimi.
Knižnica signal.h bola použitá na odchytenie signálu SIGINT a následnému korektnému uvoľneniu.

3 https://www.tcpdump.org/other/sniffex.c

Dôležité funkcie

Funkcie ktoré boli naimplementované na spracovávanie a prácu s dátami paketu a ktoré zaisťujú celkovú funkčnosť programu.

void print_error_and_exit(string string)

Funkcia vytlačí chybovú hlášku na STDERR a ukončí program s chybou.

void print_interfaces(pcap_if_t *interface_list)

Funkcia vytlačí všetky dostupné rozhrania a uvoľní zoznam pomocou pcap_freealldevs(). Rozhrania sú uložené v dátovom type zoznam cez ktorý sa iteruje pomocou ->next a dáta sa vytlačia na STDOUT.

pcap_if_t *get_interfaces()

Funkcia nájde všetky rozhrania pomocou pcap_findalldevs() a vráti zoznam.

void check_existing_interface(string interface, pcap_if_t *interface_list)

Funkcia kontroluje, či je dané rozhranie v premennej interface platné a či je v zozname rozhraní. Ak nie je, program sa ukončí s chybou.

void parse_args(int argc, char *argv[])

Funkcia skontroluje, či sú nastavené nejaké argumenty, ak nie, vypíše všetky dostupné rozhrania. Ak sú nastavené argumenty, skontroluje -i alebo –interface. Ak nie je zadané, vytlačia sa všetky dostupné rozhrania. Ak je dané rozhranie, uloží ho. Potom funkcia skontroluje -h alebo --help argument a ak ho nájde, vypíše nápovedu. Potom funkcia skontroluje argumenty ako {-p port} {[--tcp|-t] [--udp|-u] [--arp] [--icmp] [--igmp]} {-n num} a nastaví booleovské príznaky alebo zodpovedajúce argumenty. Funkcia kontroluje správne argumenty 'num' a 'port'. Ak nie je nastavený žiadny protokol, príznaky pre všetky protokoly sú nastavené na hodnotu TRUE.

void add_to_filter(string str)

Funkcia pripojí or a 'str' do premennej filter. Ak je premenná filter prázdna, funkcia nepripojí prvé „or“.

void create_filter()

Funkcia vytvorí filter, ktorý sa potom použije vo funkcii pcap_compile() na vytvorenie filtra používaného na filtrovanie paketov. Funkcia sa ovláda pomocou booleovských príznakov protokolov. Ak je príznak nastavený na hodnotu true, zavolá sa funkcia add_to_filter() s príslušným protokolom, ktorý sa má pridať k filtru. Funkcia spracováva aj okrajové prípady, ako je parameter -p num set bez možnosti udp alebo tcp, takže sa automaticky pridajú na filtrovanie. Funkcia pracuje s globálnou premennou filter.

void print_packet_data(const void *addr, int len)

Funkcia vytlačí paketové dáta vo formáte 'hexdump' a zmení všetky netlačiteľné znaky na bodky [3] [4].

void print_ipv6(const struct in6_addr *addr)

Funkcia vytlačí IPv6 adresu v správnom formáte [5].

void print_formatted_time(const struct pcap_pkthdr *header)

Funkcia preberá dve hodnoty z hlavičky paketu - tv_sec a tv_usec a transformuje ich do formátu času RC3339 [6].

void print_mac_address(const struct ether_header *ether_header)

Funkcia vytlačí zdrojové a cieľové MAC adresy z ethernetovej hlavičky.

void print_ipv4(struct ip *ip)

Funkcia vytlačí IPv4 zo štruktúry IP.

void print_mac_address(const struct ether_header *ether_header)

Funkcia vytlačí zdrojové a cieľové MAC adresy z ethernetovej hlavičky [7].

void sniffer(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)

Funkcia nájde paket a vytlačí jeho obsah. Prvé správne hodnoty sú priradené k štruktúram ako ip, iphdr, ip6_hdr, ether_header, udphdr, tcphdr. Potom sa pomocou prepínača nájde a spracuje správny protokol.

Podporované protokoly sú UDP, TCP, ARP, ICMP, ICMPv6.

Nájdený protokol sa potom vytlačí vytlačením času zachytenia paketu, zdrojovej a cieľovej MAC adresy, dĺžky rámca, zdrojovej a cieľovej IP adresy a následne naformátovaných údajov [8] .

void catch_sigint(int signal)

Funkcia spracuje SIGINT, uvoľní všetky pridelené zdroje a skončí s chybou.

Testovanie

Popis testovania

Testovanie slúžilo na overenie, či sa požadované dáta vypisujú v správnom formáte. Pri implementácií bol použitý princíp TDD (Test Driven Development), pri ktorom sa kontrolovala správnosť za pomoci nástroja wireshark[4]. Pre generáciu paketov bol použitý príkaz nping[5].

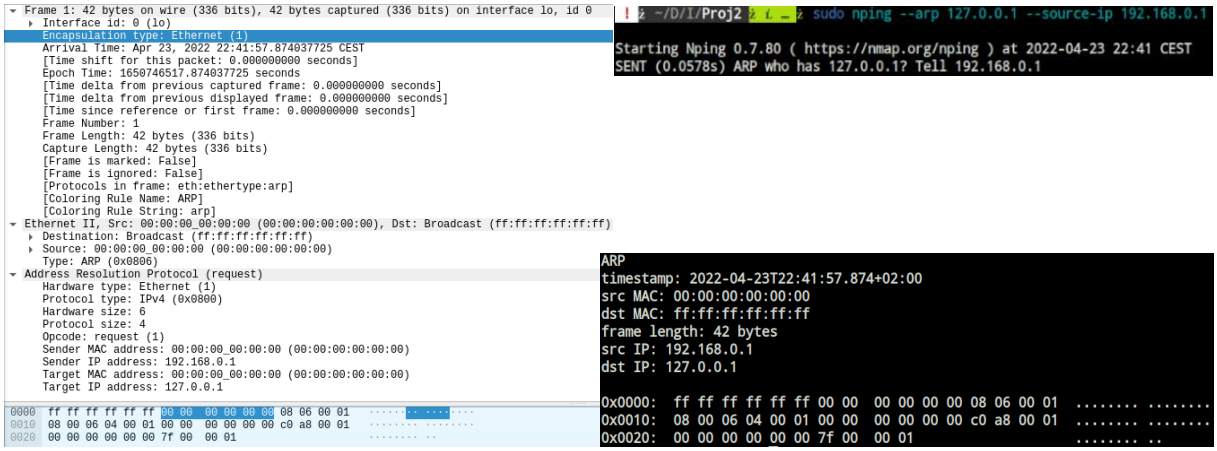
Priebeh testovania

Pri testovaní bol použitý vyššie spomínaný nástroj nping, ktorý dokázal generovať ARP, UDP, TCP a ICMP. Nástroj bohužiaľ nevedel generovať ICMPv6 pakety ani UDP alebo TCP pakety IPv6. ICMPv6 pakety a UDPv6 pakety bolo možné odchytiť z štandardného rozhrania, ale TCPv6 pakety nebolo možné nájsť a odchytiť a teda ani otestovať.

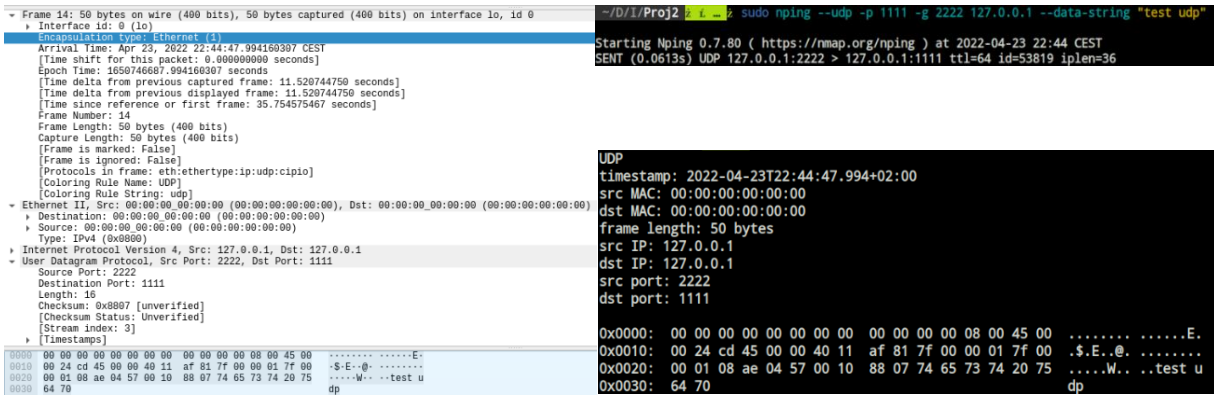
Report z testovania

Ako report z testovania sú priložené screenshoty z nástroja Wireshark, nástroja pre generovanie paketov a implementovaného sniffera pre všetky možné protokoly. [6]

ARP

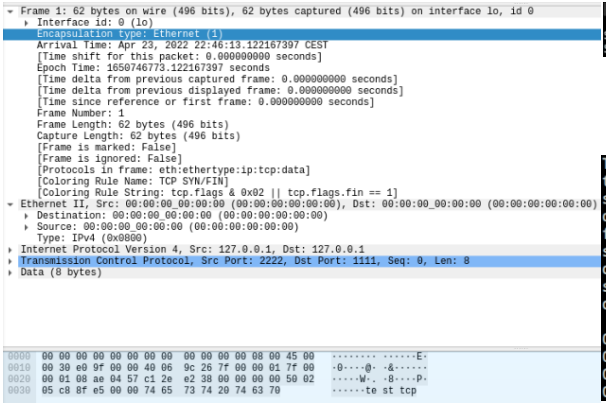


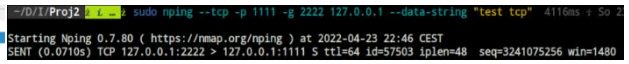
UDP

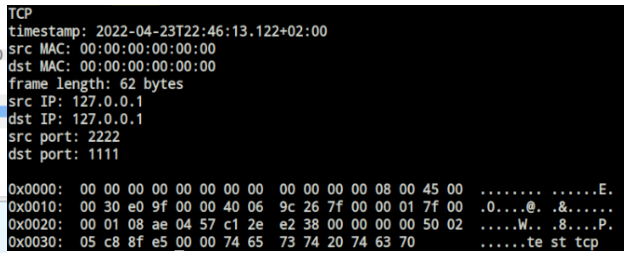


4 Nástroj Wireshark z webovej lokality https://www.wireshark.org/
5 Nástroj nping z webovej lokality https://nmap.org/nping/#:~:text=Nping%20is%20an%20open%20source,full%20control%20over%20protocol%20headers.
6 Názov protokolu pri sniffer-e(pravá strana testovacieho reportu pre jednotlivý protokol) je len z informačných dôvodov a vo finálnej verzii sa nenachádza.

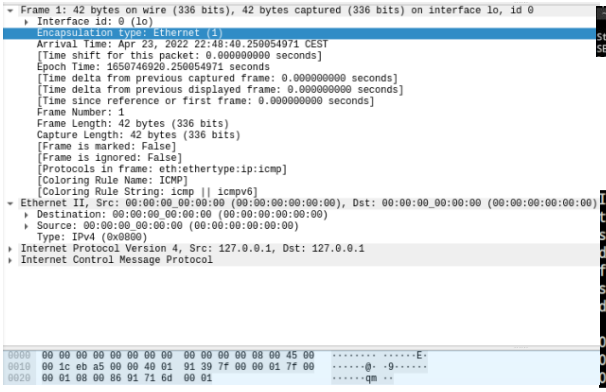
TCP

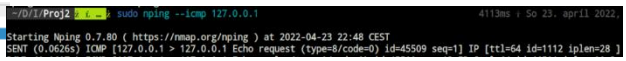


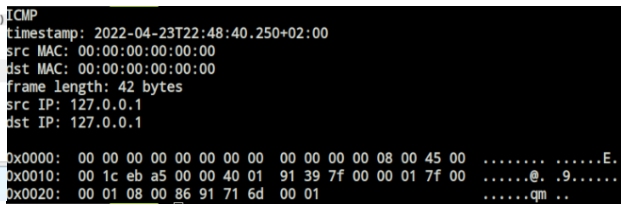




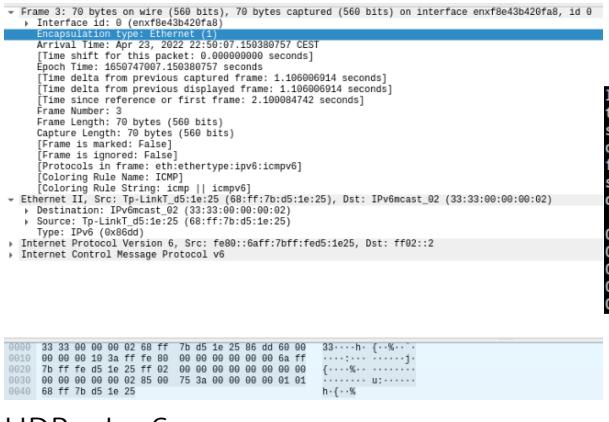
ICMP

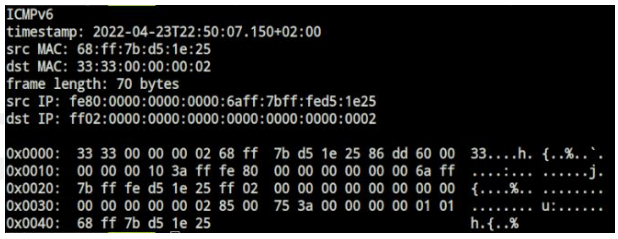




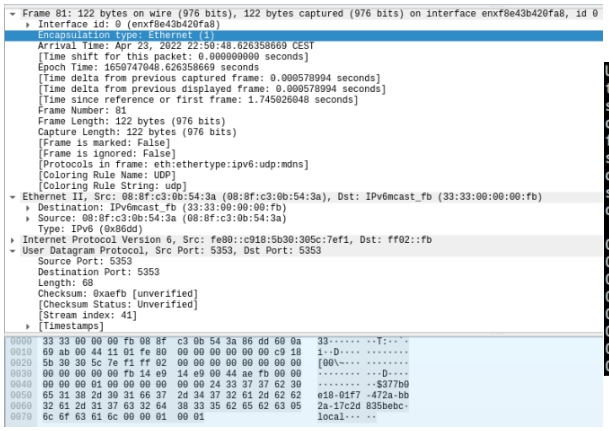


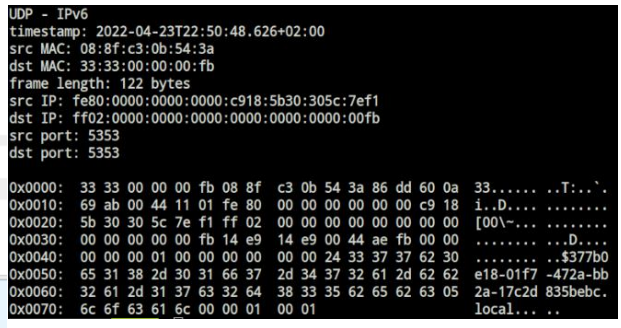
ICMPv6





UDP – Ipv6





Bibliografia

[1] „Wikipedia,“ 8 8 2021. [Online]. Available: https://cs.wikipedia.org/wiki/Paket. [Cit. 23 4 2022].

[2] S. Moon, „binarytides,“ 31 7 2020. [Online]. Available: https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/. [Cit. 23 4 2022].

[3] „programcreek,“ [Online]. Available: https://www.programcreek.com/cpp/?code=mq1n%2FNoMercy%2FNoMercy-master%2FSource%2FClient%2FNM_Engine%2FINetworkScanner.cpp. [Cit. 23 4 2022].

[4] paxdiablo, „stackoverflow,“ 15 10 2011. [Online]. Available: https://stackoverflow.com/questions/7775991/how-to-get-hexdump-of-a-structure-data. [Cit. 23 4 2022].

[5] P. Buddy a nategoose, „errorsfixing,“ 28 10 2021. [Online]. Available: https://errorsfixing.com/expand-an-ipv6-address-so-i-can-print-it-to-stdout/. [Cit. 23 4 2022].

[6] alk, „stackoverflow,“ 15 9 2017. [Online]. Available: https://stackoverflow.com/questions/46757406/how-to-print-timestamp-of-a-packet-read-from-pcap-file. [Cit. 23 4 2022].

[7] user257111, „stackoverflow,“ 24 12 2010. [Online]. Available: https://stackoverflow.com/questions/4526576/how-do-i-capture-mac-address-of-access-points-and-hosts-connected-to-it. [Cit. 23 4 2022].

[8] T. T. Group, „tcpdump,“ 5 7 2005. [Online]. Available: https://www.tcpdump.org/other/sniffex.c. [Cit. 23 4 2022].