Matej Delinčák, ID: 102895

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Analyzátor sieťovej komunikácie

2020/21, ZS

Matej Delinčák

Matej Delinčák, ID: 102895

Znenie zadania

1) **Výpis všetkých rámcov v hexadecimálnom tvare** postupne tak, ako boli zaznamenané v súbore. Pre každý rámec uveďte:
   a) Poradové číslo rámca v analyzovanom súbore.
   b) Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu.
   c) Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 - Raw).
   d) Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.

   Vo výpise jednotlivé bajty rámca **usporiadajte po 16 alebo 32 v jednom riadku**. Pre prehľadnosť výpisu je vhodné použiť neproporcionálny (monospace) font.

2) Pre rámce typu **Ethernet II a IEEE 802.3 vypíšte vnorený protokol**. Študent musí vedieť vysvetliť, aké informácie sú uvedené v jednotlivých rámcoch Ethernet II, t.j. vnáranie protokolov ako aj ozrejmiť dĺžky týchto rámcov.

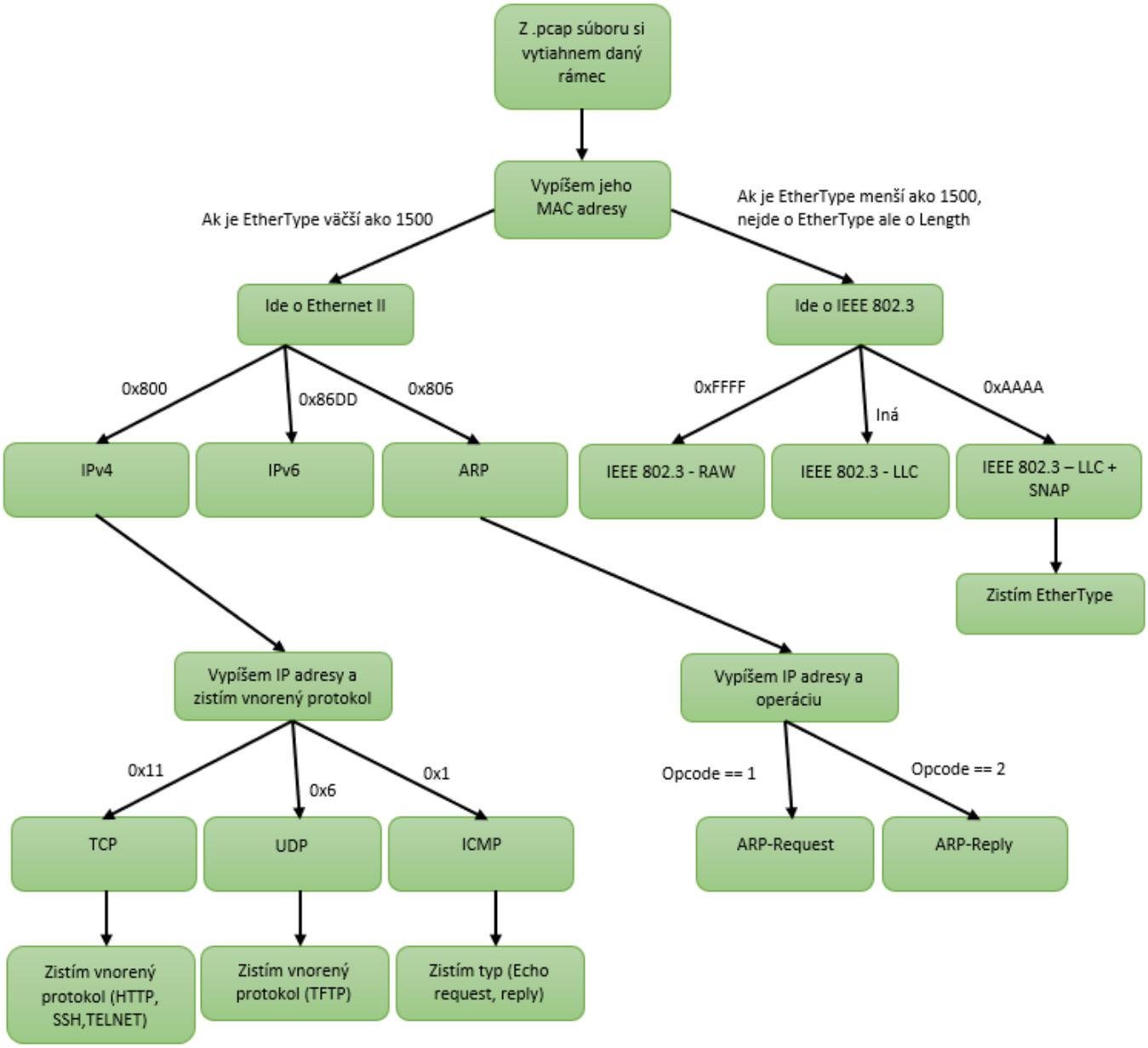
3) Analýzu cez vrstvy vykonajte pre rámce Ethernet II a protokoly rodiny TCP/IPv4:
   Na konci výpisu z bodu 1) uveďte pre IPv4 pakety:
   a)Zoznam IP adries všetkých prijímajúcich uzlov,
   b) IP adresu uzla, ktorý sumárne prijal (bez ohľadu na odosielateľa) najväčší počet paketov a koľko paketov prijal (berte do úvahy iba IPv4 pakety).

   IP adresy a počet poslaných paketov sa musia zhodovať s IP adresami vo výpise Wireshark -> Statistics -> IPv4 Statistics -> Source and Destination Addresses.

4) V danom súbore analyzujte komunikácie pre zadané protokoly:
   a) HTTP
   b) HTTPS
   c) TELNET
   d) SSH
   e) FTP riadiace
   f) FTP dátové
   g) TFTP, **uveďte všetky rámce komunikácie**, nielen prvý rámec na UDP port 69
   h) ICMP, uveďte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.
   i) **Všetky** ARP dvojice (request – reply), uveďte aj IP adresu, ku ktorej sa hľadá MAC (fyzická) adresa a pri ARP-Reply uveďte konkrétny pár- IP adresa a nájdená MAC adresa. V prípade, že bolo poslaných viacero rámcov ARP-Request na rovnakú IP adresu, vypíšte všetky. Ak sú v súbore rámce ARP-Request bez korešpondujúceho ARP-Reply (alebo naopak ARPReply bez ARP-Request), vypíšte ich samostatne.

Matej Delinčák, ID: 102895

Postup spracovania jedného rámca



Matej Delinčák, ID: 102895

Mechanizmus analýzy protokolov

Celý mechanizmus analýzy mam rozdelený na niekoľko častí. Vo funkcii *process_data()* na základe užívateľovej voľby riešim buď všetky rámce, alebo iba komunikácie typu TCP, UDP, ICMP, ARP. Najskôr rozpíšem ako analyzujem jednotlivé rámce.

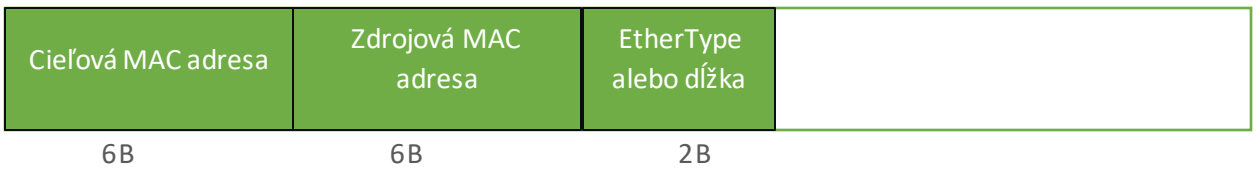
Moja najdôležitejšia funkcia je *get_data_from_frame()*, ktorá mi vráti údaje z rámca na určitom mieste. Ako náhle potrebujem nejakú informáciu, tak použijem túto funkciu.

Vo funkcií *out_to_terminal()* potom postupne volám funkcie a na výpis informácií o rámci. Najprv vypíšem číslo rámca a jeho dĺžku. Toto mi zabezpečí funkcia *print_len_of_frame()* – ide v podstate len o dĺžku rámca s tým, že minimum je 64 B. Ako ďalšie vypíšem, o aký typ rámca sa jedná.

```
# vypise typ ramca
def print_frame_type(p_ethertype_value, p_ieeetype_value):
    print("Typ rámca: ", end='')
    if p_ethertype_value >= 1500:
        print("Ethernet II")
    elif p_ieeetype_value == 0xAAAA:
        print("IEEE 802.3 LLC + SNAP")
    elif p_ieeetype_value == 0xFFFF:
        print("IEEE 802.3 raw")
    else:
        print("IEEE 802.3 LLC")
```

-ak je na mieste pre EtherType (12-13 B) je to Ethernet II,
ak je tam niečo iné, tak sa pozriem na ďalšie dva bajty (14-15 B)
a na základe neho rozhodnem o aký IEEE typ sa jedná-

Ďalej vypíšem cieľovú MAC adresu a zdrojovú MAC adresu. Na to mi slúži funkcia *print_mac_adr()*. Tieto informácie sa nachádzajú v rámci na miestach od 0 – 5 B pre cieľovú a 6-11 pre zdrojovú. Nižšie uvádzam obrázok ako vyzerá taká hlavička rámca:



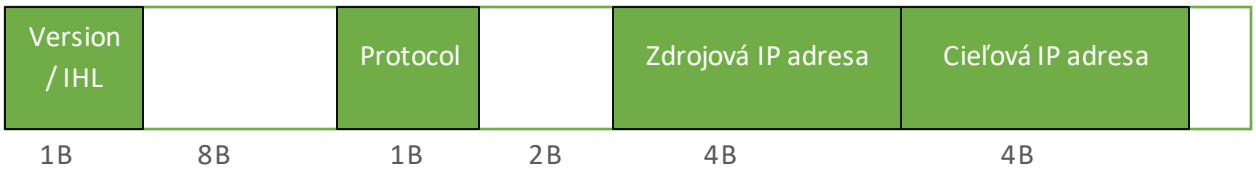
Následne idem zistiť vnútorné protokoly a na základe typu rámca sa rozhodnem, čo ďalej. Ak ide o:

- IEEE 802.3 – raw: vypíšem jeho vnútorný protokol IPX a končím
- IEEE 802.3 LLC + SNAP: pozriem sa na miesto v rámci 20 – 21 B a prečítam odtiaľ EtherType a taktiež tu analýza končí.
- IEEE 802.3 LLC: na základe hodnôt na mieste 14 B sa pozriem do súboru *llc_saps.txt* a vypíšem príslušný vnútorný protokol. Napr.(IPX, LAN management, Null SAP, ...)
- Ethernet II: tu sa pozriem do súboru *ether_types.txt* a vypíšem vnútorný protokol (IPv4, ARP, ...)

Z týchto 4 typov rámcov analyzujem ďalej len Ethernet II a to pre protokoly IPv4 a ARP. Ku ARP sa vrátim, keď budem rozoberať analýzu komunikácií.

Teda ak nájdem protokol IPv4, tak najskôr vypíšem jeho zdrojovú a cieľovú IP adresu. Toto uskutočním vo funkcií *print_ipv4_adr()* do ktorej vložím hodnoty na mieste 26 -29 B pre zdrojovú a 30 - 33 B pre cieľovú. Následne idem zistiť vnorený protokol pre IPv4. Z rámca si vytiahnem na mieste 23 B hodnotu *Protocol*. A pozriem sa, či takú hodnotu poznám (súbor *ip_protocols.txt*). Ale prv než idem ešte rozanalyzovať tento vnútorný protokol, musím zistiť, akú veľkú hlavičku má IPv4. Toto zistím na 14 B a informácia sa vola *IHL*. Kedže na tomto bajte sa nachádza aj *Version* , pomocou funkcie *get_transport_protocol()* si 14 B rozdelím a vytiahnem len *IHL*. Táto funkcia mi nakoniec vráti len protokol transportnej vrstvy.

IPv4:



Ako predposlednú časť analýzy jedného rámca si idem ešte rozobrať protokol transportnej vrstvy. Zo súborov si načítam známe porty pre UDP alebo TCP. Vypíšem zdrojový a cieľový port a taktiež aj názov vnoreného protokolu (pre TCP: HTTP, HTTPS, TELNET,... pre UDP: TFTP).

TCP:

Ako poslednú vec vypíšem celý rámec byte po byte. V riadku sa bude nachádzať priamo 16 B ako je to aj v program Wireshark.

Mechanizmus analýzy komunikácií

Vyššie som opísal ako analyzujem jeden rámec. Teraz rozpíšem ako analyzujem komunikácie z bodu 4 zadania. Užívateľ si na začiatku rozmyslí, akú komunikáciu si chce pozrieť.

Pre TCP komunikáciu rozlišujem tieto typy: HTTP, HTTPS, TELNET, SSH, FTP -riadiace, FTP - dátové. Na začiatku si celý súbor .pcap prejdem a roztriedim si jednotlivé komunikácie daného typu. To, že daný rámec patrí k danej komunikácií zisťujem na základe toho, že dva rámce komunikujú medzi sebou, čo sa týka, IP adries tak aj portov. To znamená, že cieľová adresa jedného rámca bude zdrojová adresa iného rámca a naopak. Čiže mám unikátne páry IP adries a portov.

Keď vyfiltrujem a roztriedim rámce na „tcp streamy“, tak idem analyzovať, či ide o kompletnú komunikáciu alebo nie:

1) Kompletná komunikácia musí spĺňať správne otvorenie komunikácie. To zistím z prvých troch rámcov. Kde postupne potrebujem SYN, SYN ACK a ACK. Toto považujem za správne otvorenie.

SYN, ACK, PSH, FIN a RST sú tzv. flagy a tie zistím z 13 B transportnej vrstvy. A aby bola komunikácia kompletná , musí byť aj správne ukončená. A to rôznymi spôsobmi:

a) 3 way FIN-om: posledné tri rámce od konca potrebujú ACK, FIN ACK, FIN ACK
b) 4 way FIN-om: posledné štyri rámce od konca potrebujú ACK, FIN ACK, ACK, FIN ACK
c) Alebo RST: posledný rámec musí byť ukončený buď RST alebo RST ACK

2) Nekompletná komunikácia je taká, ktorá je správne začatá, ale nemusí byť dobre ukončená.

Z oboch typov vypíšem vždy jednu korešpondujúcu komunikáciu, ak taká samozrejme existuje.

Pre UDP komunikáciu spracovávam iba TFTP protokol. Podobne ako pri TCP si roztriedim „udp streamy". Komunikácia začne na cieľovom porte 69 a ďalej sa začnú striedať ip adresy a ich porty, s tým, že port 69 je už vynechaný a nahradený iným. Ak nenájdem už takúto kombináciu ip adries a portov, viem že je komunikácia ukončená. Ak nájdem ďalší port 69, viem, že to je už druhá komunikácie. Neviem, či som to podal správne, ale snažil som sa, to čo najlepšie vysvetliť.

Pre ICMP komunikáciu len filtrujem rámce a vypisujem navyše správu, ktorej typ sa nachádza na 0 B transportnej vrstvy. Tieto správy mám uložené v súbore *icmp_types.txt*.

A ako poslednú dokážem spracovať ARP komunikáciu. Tuto v podstate opakujem princíp streamov z predošlých bodov. Ak sa rámcu zhodujú IP adresy a jeho MAC adresa a zároveň ide o request, tak ho pridam do otvorenej komunikácie. Ako náhle nájdem totožný reply, vložím ho do tejto komunikácie a stream uzavriem. Takto my môžu vzniknúť aj viaceré requesty po sebe a jeden reply na konci.

Vytvorím si takéto komunikácie a na konci vypíšem všetky, ktoré sú uzavreté (číže majú aspoň jeden request a práve jeden reply). Po týchto komunikáciách ešte vypíšem za sebou všetky zbytkové rámce za sebou bez párov.

Príklad štruktúry externých súborov

Pre každý externý súbor mám rovnakú formu a to v prvom stĺpci je hexadecimálna hodnota daného protokolu/portu a ako druhý stĺpec mám prislúchajúci názov. Všetky súbory môžem načítať z jednej funkcie *load_from_file()*.

```
0x1 ICMP
0x2 IGMP
0x6 TCP
0x9 IGRP
0x11 UDP
0x2F GRE
0x32 ESP
0x33 AH
0x39 SKIP
0x58 EIGRP
0x59 OSPF
0x73 L2TP
```

```
0x800 Internet IP (IPv4)
0x801 X.75 Internet
0x805 X.25 Level 3
0x806 ARP (Address Resolution Protocol)
0x8035 Reverse ARP
0x809B Appletalk
0x80F3 AppleTalk AARP (Kinetics)
0x8100 IEEE 802.1Q VLAN-tagged frames
0x8137 NovellIPX
0x86DD IPv6
0x880B PPP
0x8847 MPLS
0x8848 MPLS wit upstream-assigned label
0x8863 PPPoE Discovery Stage
0x8864 PPPoE Session Stage
```

- toto sú príklady formátu súborov, vľavo je súbor ether_types.txt a vpravo ip_protocols.txt-

Matej Delinčák, ID: 102895

Opis používateľského rozhrania

V mojom analyzátore som si vybral interakčné prostredie. Užívateľa program privíta peknou hlavičkou s názvom súboru a menom autora. Následne sa ho opýta, z akého .pcap súboru chce načítať. Po načítaní sa vylistuje zoznam filtrov. Ak chce vypísať celý pcap súbor, tak si zvolí all voľbu. A ak chce nejakú špeciálnu komunikáciu z bodu 4, tak napíše príslušný filter.

Ako poslednú vec sa užívateľa spýta, že či chce výstup vypísať na obrazovku a ak nie, tak do akého súboru sa ma výstup zapísať. A tu sa po spracovaní dát, program ukončí.

Výpis rámca vyzerá napríklad takto:

```
Rámec č. 2
Dĺžka rámca poskytnutá pcap API - 110 B
Dĺžka rámca prenášaného po médiu - 114 B
Typ rámca: Ethernet II
Zdrojová MAC adresa: 00 14 38 06 e0 93
Cieľová  MAC adresa: 00 02 cf ab a2 4c
Typ vnoreného protokolu: Internet IP (IPv4)
Zdrojová IP adresa: 192.168.1.33
Cieľová IP adresa: 158.195.4.138
ICMP
Správa: Echo Request
00 02 cf ab a2 4c 00 14   38 06 e0 93 08 00 4e 00
00 60 bb 9c 00 00 80 01   c7 c4 c0 a8 01 21 9e c3
04 8a 44 24 05 01 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 08 00   36 5c 03 00 14 00 61 62
63 64 65 66 67 68 69 6a   6b 6c 6d 6e 6f 70 71 72
73 74 75 76 77 61 62 63   64 65 66 67 68 69
```

Voľba implementačného prostredia

Na implementovanie som si vybral jazyk Python 3. Je to preto, lebo má veľa zabudovaných funkcií, ako ľahká práca s poľami a taktiež obsahuje štruktúru dictionary, ktorú veľmi často používam. Taktiež je prehľadný a ľahko sa s ním pracuje.
Použité knižnice:

- Scapy – na spracovanie dát z .pcap súboru – použil som odtiaľ len načítanie dát - rdpcap()
- Sys – na presmerovanie výstupu do súboru
- Binascii – na premenu dát typu bytes na ASCII znaky

Zoznam mojich funkcií:
def extract_data()
- extrahuje dáta z .pcap súboru

def load_from_file(p_filename, p_dictionary)
- načíta externé dáta

def get_data_from_frame(p_frame, p_start, p_end, p_result_in_integer=False)
- vráti dáta z daného rozsahu z frame-u

def print_data(p_seq_data, p_connection='', p_to_int=False)
- vypíše dáta v danom formáte

def print_frame(p_frame_data)
- vypíše celý rámec

def print_ipv4_nodes(p_nodes)
- vypíše zoznam ipv4 uzlov (bod 3)

def get_transport_protocol(p_frame)
- vráti len dáta z transportnej vrstvy

def print_frame_type(p_ethertype_value, p_ieeetype_value)
- vypíše typ rámca

def print_ipv4_adr(src, dst)
- vypíše IP adresy

def choose_and_print_inside_protocol(p_frame, p_ethertype_value, p_ieeetype_value, p_force_udp)
- tu prebieha rozhodovanie o aké protokoly sa jedná

def print_len_of_frame(p_frame)
- vypíše dĺžku rámca

def print_mac_adr(p_frame)
- vypíše MAC adresy

def out_to_terminal(p_frame, p_num, p_force_udp='', ports=False)
- volá funkcie na výpisy

def insert_ipv4_to_dict(p_frame, p_dictionary)
- vloží novu IP adresu do zoznamu

def print_arp_header(p_item, p_request=True)
- vypíše ARP hlavičku komunikácie

def is_ip_dest_in_stream(p_streams, p_ip_dest, p_ip_src, p_prt_src=0, p_prt_dest=0)
- zistí, či už daná kombinácia IP adries a portov už existuje

def insert_to_streams(p_arr_of_streams, p_frame_id, p_ip_src, p_ip_dest, p_port_src=0, p_port_dest=0)
- vloží novú TCP konverzáciu do zoznamu na základe výsledku z predchádzajúcej funkcie

def is_arp_in_stream(p_streams, p_ip_dest, p_ip_src, p_mac_src)
- zistí, či už daná kombinácia IP adries a MAC adresy už existuje

def insert_to_arpstreams(p_arr_of_streams, p_frame_id, p_ip_src, p_ip_dest, p_mac_src, arp_reply=1)
- vloží novú ARP konverzáciu do zoznamu na základe výsledku z predchádzajúcej funkcie

def print_communication(p_data, frames, force_udp='', flags=True)
- vypíše celú komunikáciu

def process_data(p_data, p_menu)
- na základe voľby spracuje dané dáta

Záver

Môj program dokáže zistiť IP adresy, MAC adresy, typy protokolov aj portov a všetko, čo zadanie požadovalo. Dokáže zistiť a vypísať dané komunikácie, či už TCP, UDP alebo ARP. Výstup môže byť ako do konzoly tak aj do súboru. Toto zadanie bolo celkom zaujímavé, ale ak toto čítate musím sa posťažovať. Tým, že som musel odovzdať v nedeľu, nestihol som prednášku o TCP a všetko som si musel naštudovať sám. Taktiež aj v ostatných prípadoch z bodu 4 som mal chaos, čo ako funguje. Z mojej strany nebolo dostatočne vysvetlené učivo a 90 % som si musel doštudovať sám. Čo ma dosť mrzí.