SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

ZADANIE 1 - ANALYZÁTOR SIEŤOVEJ KOMUNIKÁCIE

Ctibor Kovalčík
FIIT STU
Cvičenie: Štvrtok 16:00
20.10.2021

1 Zadanie úlohy

Navrhnite a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácie okomunikáciách. Vypracované zadanie musí spĺňať nasledujúce body:

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 odosielajúcich uzlov,

   b) IP adresu uzla, ktorý sumárne odoslal (bez ohľadu na prijímateľa) najväčší počet paketov a koľko paketov odoslal (berte do úvahy iba IPv4 pakety).

   IP adresy a počet odoslaných / prijatých 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 ARP- Reply bez ARP-Request), vypíšte ich samostatne.

Vo všetkých výpisoch treba uviesť aj IP adresy a pri transportných protokoloch TCP a UDP aj porty komunikujúcich uzlov.
V prípadoch komunikácií so spojením vypíšte iba jednu kompletnú komunikáciu - obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia a aj prvú nekompletnú komunikáciu, ktorá obsahuje iba otvorenie spojenia. Pri výpisoch vyznačte, ktorá komunikácia je kompletná. Ak počet rámcov komunikácie niektorého z protokolov z bodu 4 je väčší ako 20, vypíšte iba 10 prvých a 10 posledných rámcov tejto komunikácie. **(Pozor: toto sa nevzťahuje na bod 1, program musí byť schopný vypísať všetky rámce zo súboru podľa bodu 1.)**Pri všetkých výpisoch musí byť poradové číslo rámca zhodné s číslom rámca v analyzovanom súbore.

5) Program musí byť organizovaný tak, aby čísla protokolov v rámci Ethernet II (pole Ethertype), IEEE 802.3 (polia DSAP a SSAP), v IP pakete (pole Protocol), ako aj čísla portov v transportných protokoloch boli programom **načítané z jedného alebo viacerých externých textových súborov.** Pre známe protokoly a porty (minimálne protokoly v bodoch 1) a 4) budú uvedené aj ich názvy. Program bude schopný uviesť k rámcu názov vnoreného protokolu po doplnení názvu k číslu protokolu, resp. portu do externého súboru. Za externý súbor sa nepovažuje súbor knižnice, ktorá je vložená do programu.

6) V procese analýzy rámcov pri identifikovaní jednotlivých polí rámca ako aj polí hlavičiek vnorených protokolov nie je povolené použiť funkcie poskytované použitým programovacím jazykom alebo knižnicou. **Celý rámec je potrebné spracovať postupne po bajtoch.**

7) Program musí byť organizovaný tak, aby bolo možné jednoducho rozširovať jeho funkčnosť výpisu rámcov pri doimplementovaní jednoduchej funkčnosti na cvičení.

8) Študent musí byť schopný preložiť a spustiť program v miestnosti, v ktorej má cvičenia. V prípade dištančnej výučby musí byť študent schopný prezentovať podľa pokynov cvičiaceho program online, napr. cez Webex, Meet, etc.

V danom týždni, podľa harmonogramu cvičení, musí študent priamo na cvičení doimplementovať do funkčného programu (podľa vyššie uvedených požiadaviek) ďalšiu prídavnú funkčnosť.

Program musí mať nasledovné vlastnosti (minimálne):

1) Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižnice pcap, skompilovateľný a spustiteľný v učebniach. Na otvorenie pcap súborov použite knižnice libpcap pre linux/BSD a winpcap/npcap pre Windows. Použité knižnice a funkcie musia byť schválené cvičiacim. V programe môžu byť použité údaje o dĺžke rámca zo struct pcap_pkthdr a funkcie na prácu s pcap súborom a načítanie rámcov:
   pcap_createsrcstr()
   pcap_open()
   pcap_open_offline()
   pcap_close()
   pcap_next_ex()
   pcap_loop()
   Použitie funkcionality *libpcap* na priamy výpis konkrétnych polí rámca (napr. ih->saddr) bude mať za následok nulové hodnotenie celého zadania.

2) Program musí pracovať s dátami optimálne (napr. neukladať MAC adresy do 6x int).

3) Poradové číslo rámca vo výpise programu musí byť zhodné s číslom rámca v analyzovanom súbore.

4) Pri finálnom odovzdaní, pre každý rámec vo všetkých výpisoch uviesť použitý protokol na 2. - 4. vrstve OSI modelu. (ak existuje)

5) Pri finálnom odovzdaní, pre každý rámec vo všetkých výpisoch uviesť zdrojovú a cieľovú adresu / port na 2. - 4. vrstve OSI modelu. (ak existuje)

Nesplnenie ktoréhokoľvek bodu minimálnych požiadaviek znamená neakceptovanie riešenia cvičiacim.

Súčasťou riešenia je aj dokumentácia, ktorá musí obsahovať najmä:

a) zadanie úlohy,

b) blokový návrh (koncepcia) fungovania riešenia,

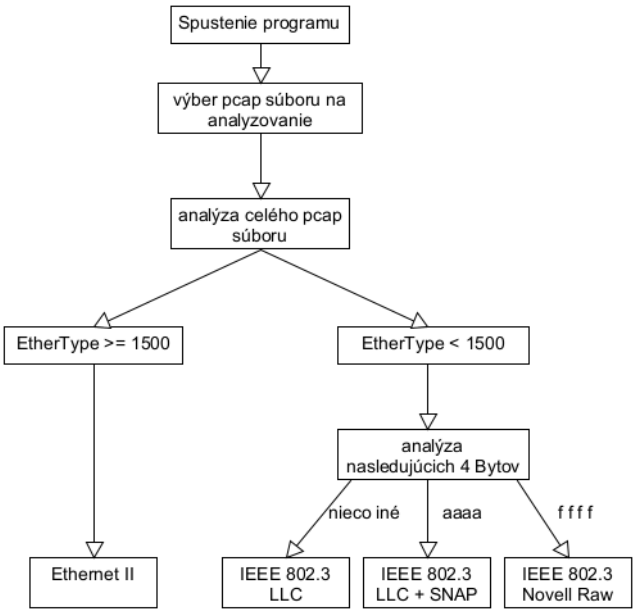
c) navrhnutý mechanizmus analyzovania protokolov na jednotlivých vrstvách,

d) príklad štruktúry externých súborov pre určenie protokolov a portov,

e) opísané používateľské rozhranie,

f) voľbu implementačného prostredia.

2 Blokový návrh



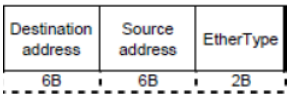
Obr. 1: Blokový návrh č.1

Program si na začiatku vypýta od používateľa konkrétny pcap súbor, ktorý následne bude analyzvaný. Zabezpečuje to jednoduché User Interface v konzole.

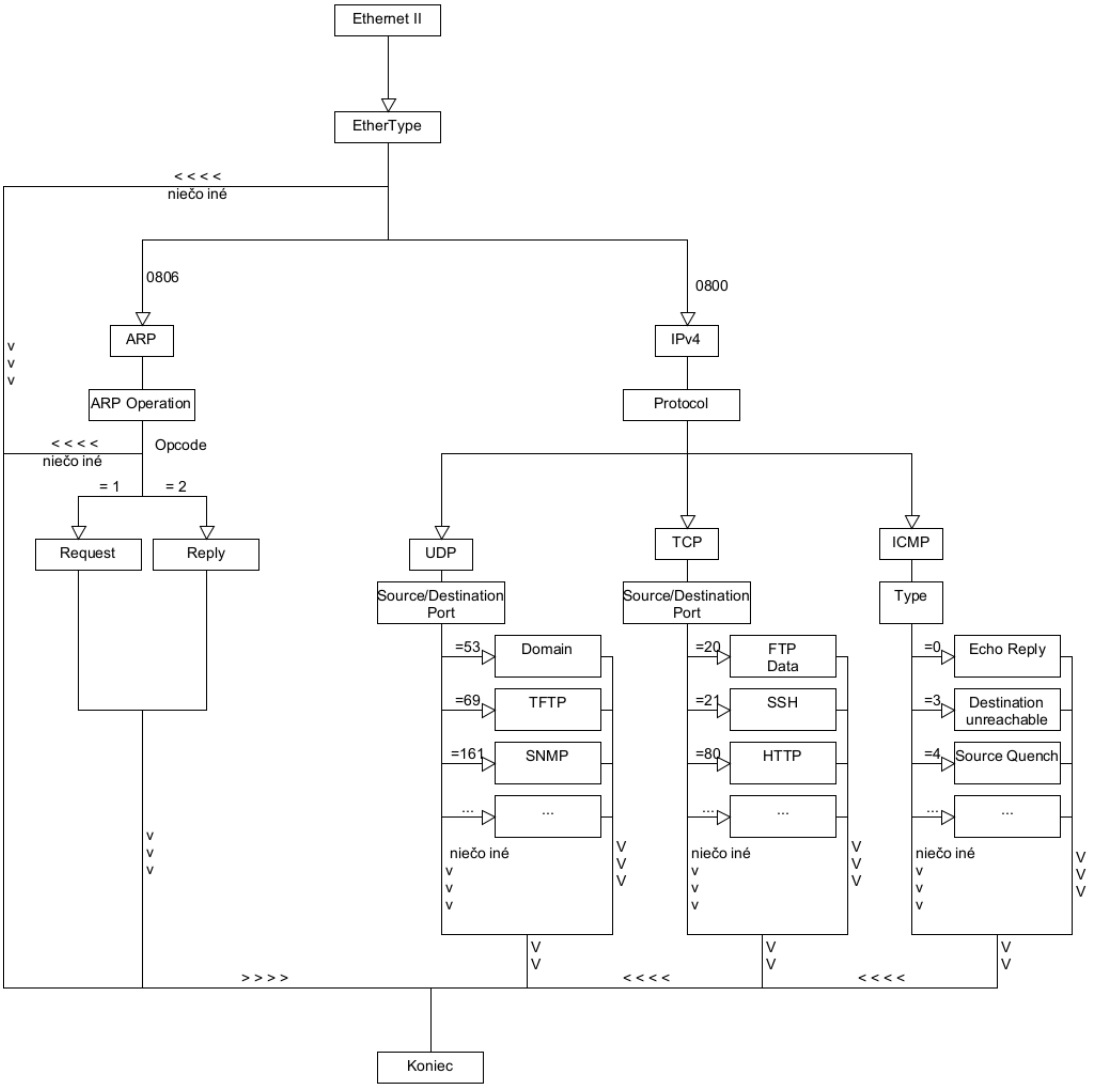
Na začiatku je potrebné zistiť o aký typ rámca ide. V postupnosti bytov jedného pcap súboru sa pozrieme na 12. a 13. byte. Dokopy nám dávajú EtherType ktorý ak je v 10-kovej sústave väčší ako 1500 (05DC v 16-kovej) tak ide o **Ethernet II** ak je to číslo menšie tak sa musíme pozrieť na ďalšie 2 Byty.

Nasledujúce 2 Byty po EtherType nám definujú aký typ **IEEE 802.3** ide. Ak nastane situácia kde sa tieto 2 Byty rovnajú **ffff** tak ide o **IEEE 802.3 Novell Raw** ak je to **aaaa** tak ide o **IEEE 802.3 LLC + SNAP** a ak je to niečo iné tak je to typu **IEEE 802.3 LLC**

Následne analyzujeme vo všetkých 4-och prípadoch MAC adresy (Source aj Destination) MAC adresy sú uložené v pcap súbore ako je vyznačené na obrázku.



Obr. 2



Obr. 3: Blokový návrh č.1

V zadaní máme bližšie analyzovať Ethernet II.
(číselné údaje sú uvedené v 10-tkovej sústave)
Ak je EtherType:

- **EtherType = 2048** -> tak protokol 3. vrstvy je **IPv4**
  Pri protokole IPv4 nás zaujíma dĺžka, ktorú potrebujeme na to aby sme sa vedeli posunúť v liste bytov na ďalšiu vrstvu (protokol). ďalej v IPv4 hlavičke máme informáciu ohľadom ip adries odosielajúcich a prijmajúcich uzlov a nás zaujíma aj vnorený protokol ktorý ak je :

  1. **Protocol = 6** -> tak ide o **TCP**
     Pri TCP nás zaujíma o aký typ komunikácie ide. Táto informácia je nám dostupná z portov, source a takisto aj destination port Ak je jeden z týchto portov:
     - 20 -> tak ide o FTP DATA
     - 21 -> tak ide o FTP CONTROL
     - 22 -> tak ide o SSH
     - 23 -> tak ide o TELNET
     - 80 -> tak ide o HTTP
     - 443 -> tak ide o HTTPS

     Sledujeme takisto aj pole FLAGS, pomocou ktorých určujem či daná komunikácia je valdiná. Konkrétne zachytávam tieto FLAGY:
     - ACK -> Ackowledgment
     - SYN
     - FIN
     - RST -> RESET
  2. **Protocol = 17** -> tak ide o **UDP**

     Pri UDP nás zaujíma o aký typ komunikácie ide. Táto informácia je nám dostupná z portov, source a takisto aj destination port Ak je jeden z týchto portov:
     - 69 -> tak ide o TFTP
  3. **Protocol = 1** -> tak ide o **ICMP**

     Pri ICMP nás zaujíma jeho typ:
     - Type = 0 -> Echo Reply
     - Type = 3 -> Destination Unreachable

  - Type = 4 -> Source Quench
  - Type = 5 -> Redirect
  - Type = 8 -> Echo
  - Type = 9 -> Router Advertisment
  - Type = 10 -> Router Selection
  - Type = 11 -> Time Exceeded
  - Type = 12 -> Parameter Problem
  - Type = 13 -> Timestamp
  - Type = 14 -> Timestamp Reply
  - Type = 15 -> Information Request
  - Type = 16 -> Information Reply
  - Type = 17 -> Address Mask Request
  - Type = 18 -> Address Mask Reply
  - Type = 30 -> Traceroute

- **EtherType = 2054** -> tak protokol 3. vrstvy je **ARP**
  Pri ARP nás zaujíma Operation code:

  - OPcode = 1 -> tak ide o Request
  - OPcode = 2 -> tak ide o Reply

  Ale pozerám takisto na

  - Sender MAC address
  - Sender IP address
  - Target MAC address
  - Target IP address

3 Mechanizmus

V programe mám vytvorenú triedu LISTPACKETOV do ktorej postupne vkladám ďalšiu mnou vytvorenú triedu PACKET

```
class PACKET:
    def __init__(self, position, length_real, length_media):
        self.position = position
        self.length_real = length_real
        self.length_media = length_media

    def set_text(self, ramec):
        self.ramec = ramec

    class Data_link_header:
        def __init__(self, destination_mac, source_mac, typ_prenosu):
            self.typ_prenosu = typ_prenosu
            self.source_mac = source_mac
            self.destination_mac = destination_mac

        def set_eth_type(self, protocol_type):
            self.eth_type = protocol_type

    class Protocol:
        def __init__(self):
            self.fragmented = False

        def getName(self):
            return "Unknown"

        def vypis(self):
            self.Protocol.vypis()
```

Obr. 4: Class Packet

PACKET nám v programe prezentuje každý jeden packet bez ohľadu na to o aký typ rámca

- position -> predstavuje pozíciu daného packet v celom pcap súbore
- length_real -> predstavuje dlžku packetu ktorá je poskytnúta pcap API
- length_media -> predstavuje dĺžku packetu prenášaného po médiu. + 4 Byty (FCS) ak je length real < 60

- Class **Data_link_header** predstavuje linkovú vrstvu.
  v tejto Časti si takisto aj ukladám o aký typ prenosu ide (Ethernet II, IEEE 802.3 LLC , ...)
  v tejto vrstve pozorujeme tieto 3 údaje :
  - Prvých 6 Bajtov -> Destination MAC address
  - Druhých 6-Bajtov -> Source MAC address
  - 13 a 14 Bajt (spolu) -> EtherType
- Class **Protocol**
  Do tejto triedy vkladám mnou navrhnuté 3 triedy v závislosti od toho aký typ prenosu ide a aký je EtherType.
  1. Ak je typ prenosu (IEEE 802.3 LLC, IEEE 802.3 Novell Raw, IEEE 802.3 LLC + SNAP) tak sa tam vloží trieda LLC_header [5].
  2. Ak je EtherType = 2054 tak sa tam vloží trieda ARP_header [5].
  3. Ak je EtherType = 2048 tak sa tam vloží trieda IP_header [5].

```
class LLC_header:
    def __init__(self, DSAP, SSAP):
        self.SSAP = SSAP
        self.DSAP = DSAP
        self.fragmented = False
        self.protocol = None

    def vypis(self):...

class ARP_header:
    def __init__(self, Opcode, sender_MAC, sender_IP, target_MAC, target_IP):
        self.sender_MAC = sender_MAC
        self.sender_IP = sender_IP
        self.target_MAC = target_MAC
        self.target_IP = target_IP
        self.Opcode = Opcode
        self.fragmented = False
        self.protocol = None

    def vypis(self):...

class IP_header:
    def __init__(self, protokol, source_adress, destination_adress, length):
        self.protocol = protokol
        self.source_adress = source_adress
        self.destination_adress = destination_adress
        self.length = length
        self.fragmented = False

    def vypis(self):...

    def set_fragmented(self, type):...
```

Obr. 5: LLC Class, ARP Class, IP class

LLC_header

V LLC_header si zaznamenávam údaje ako SSAP, DSAP, ale takisto aj premennú fragmented (ak by sa vyskytol nejaký fragmentovaný rámec, *v tejto triede to nieje celkom doiplementované*) a pre zjednodušenie výpisov rámca si nastavím protokol na ďalšej vrstve ako None

ARP_header

V ARP_header zaznamenávam údaje:

- sender_MAC
- sender_IP

- target_MAC
- target_IP
- Opcode
- fragmented *takisto ako v LLC_header, ani tu to nieje celkom doimplementované*
- protocol -> protokol na ďalšej vrstve nastavený na NONE pre zjednodušenie výpisov

funkcia vypis predstavuje základný výpis ARP_header

IP_header Premenná fragmented naznačuje či sa jedná o fragmentovaný Packet, čiže ak je rozdelený na 2 tak sa označí ako fragmented a nasledujúci packet bude čerpať informácie aj z predošleho packetu Do tejto triedy je možné vkladať dalšiu triedu v podobe ďalšieho protokolu. V mojom programe sú na to vytvorené ešte 3 triedy:

1. TCP_header
2. ICMP_header
3. UDP_header

ostatné protokoly sa uložia do premennej protocol nie ako trieda ale ako list("Názov", Protocol_number)

V programe využívam nasledovné funkcie:

1. **length_of_packet_media(length)**

   V tejto funkcii ako argument vstupuje dĺžka packetu poskytnú pcap API (reálne dĺžka) a vypočíta sa na základe toho dĺžka prenášaného packetu po médiu.

   if (length $<$ 60): media_length0 = 64
   else: media_length = length + 4

2. **dest_mac_adress(list_of_packet_bytes)**

   Do tejto funkcie vstupuje ako argument list bajtov jedného packetu už uložené po jednom ako string

   Výstupom je destination MAC address.

3. **source_mac_adress(list_of_packet_bytes)**

   Do tejto funkcie vstupuje ako argument list bajtov jedného packetu už uložené po jednom ako string.

   Výstupom je source MAC address.

4. **type_of_packet(list_of_packet_bytes)**

   Do tejto funkcie vstupuje ako argument list bajtov jedného packetu už uložené po jednom ako string.

   Výstupom je typ prenosu (Ethernet II/ IEEE 802.3 LLC/ IEEE 802.3 Novell RAW/ IEEE 802.3 LLC +SNAP.

5. **file_checker(number,ID)**

   Argument number predstavuje číselné označenie protokolu/typu a argument ID označuje ID veci ktorú chceme v externom súbore hľadať:

   - | -> EtherType
   - + -> SSAP
   - - -> IP protokol
   - / -> TCP typ
   - < -> UDP typ
   - > -> ICMP typ
   - * -> ARP Opcode

   Výstupom funkcie je buď 'Unknown' v prípade ak sa v externom súbore nenašla položka s rovnakým číslom ako je argument number.
   Alebo ak sa našla zhoda tak výstup funkcie je názov daného protokolu/typu.

6. **ARP_info(list_of_packet_bytes)**

   Do tejto funkcie vstupuje ako argument list bajtov jedného packetu už uložené po jednom ako string.

   V tele tejto funckie sa analyzuje bajt po bajte a zostavuje sa trieda ARP_header.

   Výstupom tejto funkcie je trieda ARP_header.

7. **IP_info(list_of_packet_bytes)**

   Do tejto funkcie vstupuje ako argument list bajtov jedného packetu už uložené po jednom ako string.

   V tele tejto funkcie sa zostavuje celá trieda IP_header vrátane ďalších protokolov. V prípade ak je to jeden z protokolov (TCP,UDP,ICMP) tak vytvorí ešte samotné triedy týchto protokolov. Ak je to iný protokol poprípade 'Unknown' tak funkcia si uloží do premennej protocol iba list("Názov", protokol_number)

   Výstupom tejto funkcie je IP_header

8. **LoadAllPackets(pcap,mylist)**

   Do tejto funkcie vstupujú ako argumenty :

   (a) pcap -> pcap súbor vo formáte bajtov

   (b) mylist -> list všetkých packetov

   V tele tejto funkcii prebieha spracovanie každého jedného packetu z pcap súboru. Zostavuje sa tu výpis celého rámcu vo formáte:

```
0000 | 00 30 c1 61 eb ed 00 08   74 4f 36 23 08 00 45 00
0010 | 00 4e 01 c2 00 00 80 11   00 00 c0 a8 01 66 c0 a8
0020 | 01 68 10 1d 00 a1 00 3a   30 ca 30 30 02 01 00 04
0030 | 06 70 75 62 6c 69 63 a0   23 02 02 18 31 02 01 00
0040 | 02 01 00 30 17 30 15 06   11 2b 06 01 04 01 0b 02
0050 | 03 09 04 02 01 02 02 02   01 00 05 00
```

   Celá funkcia nemá výstup slúži len ako funkcia na analýzu celého packetu a vytváranie príslušných tried.

9. **print_communication(stream)**

   Vstupným argumentom do tejto funkcie je list packetov 1 komunikácie. Funkcia slúži na výpis komunikácie, buď jednoduchým štýlom alebo vypisuje packety v celku (zakomentované v zdrojovm kóde)

10. **print_icmp(listpacketov)**

    Vstupným argumentom do tejto funkcie je list packetov ktoré obsahujú icmp protokoly, Funkcia slúži na výpis ICMP komunikácií

11. **print_tftp(coms)**

    Do funkcie vstupuje ako argument list všetkých tftp komunikácií Funkcia má jeden for cyklus v ktorom každú komunikáciu vypisuje cez spominánú funkciu print_communication(komunil

    Funkcia slúži na výpis všetkých tftp komunikácií v danom pcap súbore.

12. **communication(source,listpacketov)**

    Do tejto funkcie vstupujú ako argumenty source -> source port packetu ktorý pridávam Je to pomocná funkcia ktorá mi uľahčuje prácu s niektorými tcp komunikáciami, kontroluje porty a ip adresy a podľa toho daný packet začlení/nezačlení do komunikácie

    Výstupom je list[source, list_komunikácie]

    source je source port každej jednej komunikácie v list_komunikácie... značí nejaký identifikátor danej komunikácie.

13. **tftpcom(list,coms_tftp,info**$_protocol, i)$

    - Vstupom do tejto funkcie sú:

    - **list**
      list predstavuje list všetkých packetov v danom pcap súbore
    - **coms_tftp**
      predstavuje list komunikácií TFTP (do funkcie vstupuje prázdny)
    - **info_protocol**
      info_protocol predstavuje list[source port, destination port] jedného packetu kde sa vyskytol TFTP port
    - **i**
      i predstavuje index packetu v celom liste všetkých packetov kde sa vyskytol TFTP port

    Funkcia hľadá TFTP komunikáciu pričom hneď ako prvé tak pridá do danej novej TFTP komunikácie list[i] čo prdstavuje packet s TFTP portom a následne skúma všetky ostatné packety od packetu list[i].

14. **option1(list)**

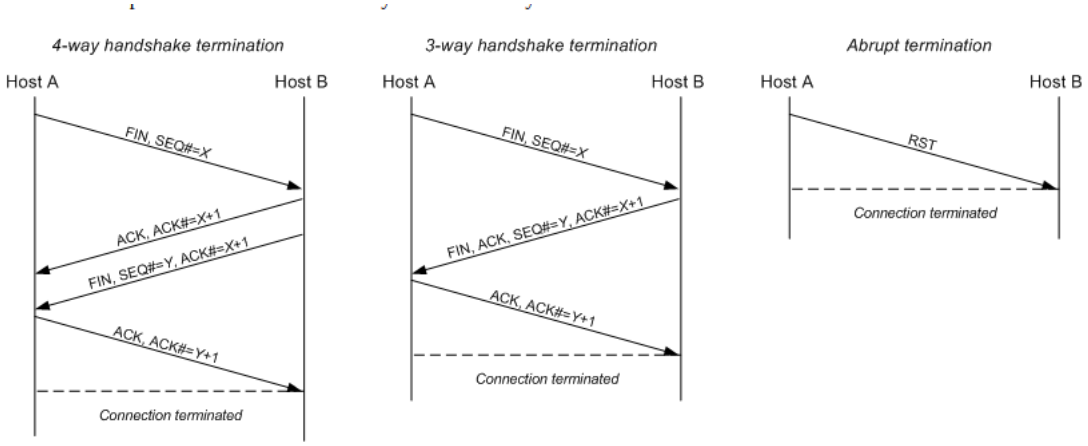
    - Vstupom do tejto funkcie je list všetkých packetov z daného pcap súboru

    Funkcia slúži na výpis každého rámca a na konci vypočíta ktorá IP adresa z rodiny TCP/IPv4 odoslala najviac packetov pre prehľadnosť je v tejto funkcii použitá ešte funkcia **print_p(packet)** ktorá vypíše packet podľa mojho naformátovania

15. **option2(list,keyword)**

    - Vstupom do tejto funkcie je list všetkých packetov z daného pcap súboru a keyword ktorý predstavuje aký typ protokolu chceme anylyzovať v danom pcap súbore.

    Funkcia slúži na výpis a triedenie packetov podľa bodu 4.a) až h) Vo funkcii sa takisto kontroluje 3-way handshake na začiatku každej komunikácie. ak tam nieje, nepovažujem danú komunikáciu za kompletnú/nekompletnú Ďalej vo funkcii sa tiež pozerá na ukončenie. V programe rozdelujem ukončenie na 3 prípady:



Obr. 6: Termination of TCP

V tejto sa odfiltruju aj ICMP A UDP packety ktoré už putujú do svojich funkcii ktoré sú spomínané vyššie.

16. **option3(list)** - Vstupom do tejto funkcie je list všetkých packetov z daného pcap súboru.

    Táto funkcia slúži na analyzovanie ARP komunikácií kedy sa najprv prejde celým listom packetov a vyfiltruju sa samostane Replies a Requests. Následne sa prechádza Requestami a hľadajú sa replies. Program potom odstráni prípady kedy može nastať to že, je viacero requestov na 1/viacero replies

    Na konci program vypíše všetky Komunikácie, vypíše aj replies ktoré nemajú svoj request

4 Externý súbor

V programe využívam externý txt súbor v ktorom si ukladám hodnoty všetkých Ethertypes, IPv4 protokolov, ...

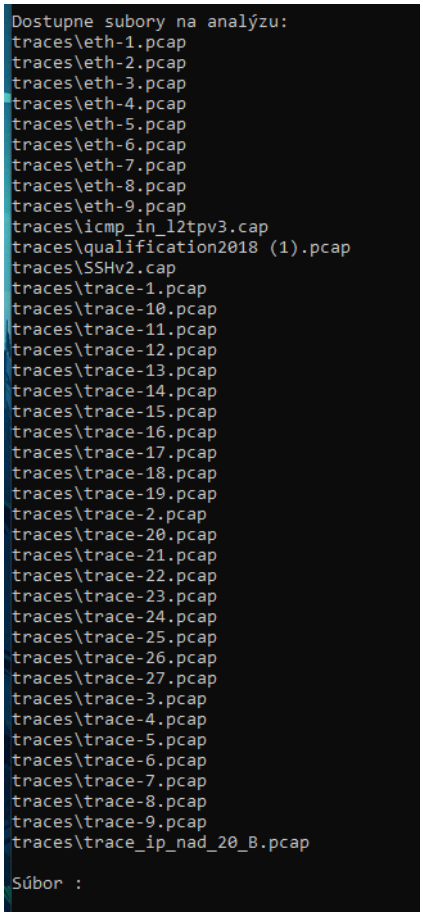
```
Ethernet typ
| 2048 = IPv4
| 2054 = ARP
| 512 = XEROX PUP
| 513 = PUP Addr Trans
| 2049 = X.75 Internet
| 2053 = X.25 Level 3
| 8192 = CDP
| 8196 = DTP (Dynamic Trunk Protocol)
| 32821 = Revers ARP
| 32923 = Appletalk
| 33011 = Appletalk AARP
| 33024 = IEEE 802.1Q VLAN tagged frames
| 33079 = Novell IPX
| 34525 = IPv6
| 34827 = PPP
| 34887 = MPLS
| 34888 = MPLS with upstream assigned label
| 34915 = PPPoE Discovery Stage
| 34916 = PPPoE Session Stage
SSAP
+ 0 = NULL SAP
+ 2 = LLC Sublayer Management or Individual
+ 3 = LLC Sublayer Management or Group
+ 6 = IP (DOD Internet Protocol)
+ 14 = PROWAY (IEC 955)
+ 66 = BPDU (Bridge PDU 802.1 Spanning tree)
+ 78 = MMS (Manufacturing Message Service)
+ 94 = ISI IP
+ 126 = X.25 PLP (ISO 8208)
+ 142 = PROWAY (IEC 955) Active Station List Maintenance
+ 170 = SNAP
+ 224 = IPX (Novell NetWare)
+ 244 = LAN Management
+ 254 = ISO Network Layer Protocols
+ 255 = Global DSAP
IP
- 1 = ICMP
- 2 = IGMP
- 6 = TCP
- 9 = IGRP
- 17 = UDP
- 47 = GRE
- 50 = ESP
- 51 = AH
- 57 = SKIP
- 88 = EIGRP
- 89 = OSPF
- 115 = L2TP
TCP
/ 7 = ECHO
/ 19 = CHARGEN
/ 20 = FTP DATA
/ 21 = FTP CONTROL
/ 22 = SSH
/ 23 = TELNET
/ 25 = SMTP
/ 53 = DOMAIN
/ 79 = FINGER
/ 80 = HTTP
/ 110 = POP3
/ 111 = SUNRPC
/ 119 = NNTP
/ 139 = NETBIOS SSN
/ 143 = IMAP
/ 179 = BGP
/ 389 = LDAP
/ 443 = HTTPS
/ 445 = MICROSOFT DS
/ 1080 = SOCKS
UDP
< 7 = Echo
< 19 = Chargen
< 37 = Time
< 53 = Domain
< 67 = Bootps DHCP
< 68 = Bootpc DHCP
< 69 = TFTP
< 137 = Netbios ns
< 138 = Netbios dgm
< 161 = SNMP
< 162 = SNMP trap
< 500 = Isakmp
< 514 = syslog
< 520 = RIP
< 33434 = Traceroute
ICMP
> 0 = Echo Reply
> 3 = Destination Unrechab
> 4 = Source Quench
> 5 = Redirect
> 8 = Echo
> 9 = Router Advertisment
> 10 = Router Selection
> 11 = Time Exceeded
> 12 = Parameter Problem
> 13 = Timestamp
> 14 = Timestamp Reply
> 15 = Information Request
> 16 = Information Reply
> 17 = Address Mask Reques
> 18 = Address Mask Reply
> 30 = Traceroute
ARP
* 1 = Request
* 2 = Reply
```

Obr. 7: Externý súbor

Rozlišujem to podľa symbolov pred každým 10-kovým číslom. funkcia **file_checker()** prechádza každým riadkom tohto súboru a hľadá najprv aby sa symbol ktorý vstúpil do tejto funkcie rovnal symbolu na riadku ak na taký riadok narazí tak kontroluje 10-kové čislo v súbore či sa zhoduje s číslom ktoré vstúpilo do funkcie (16-kové) po prevedení na 10-kové.

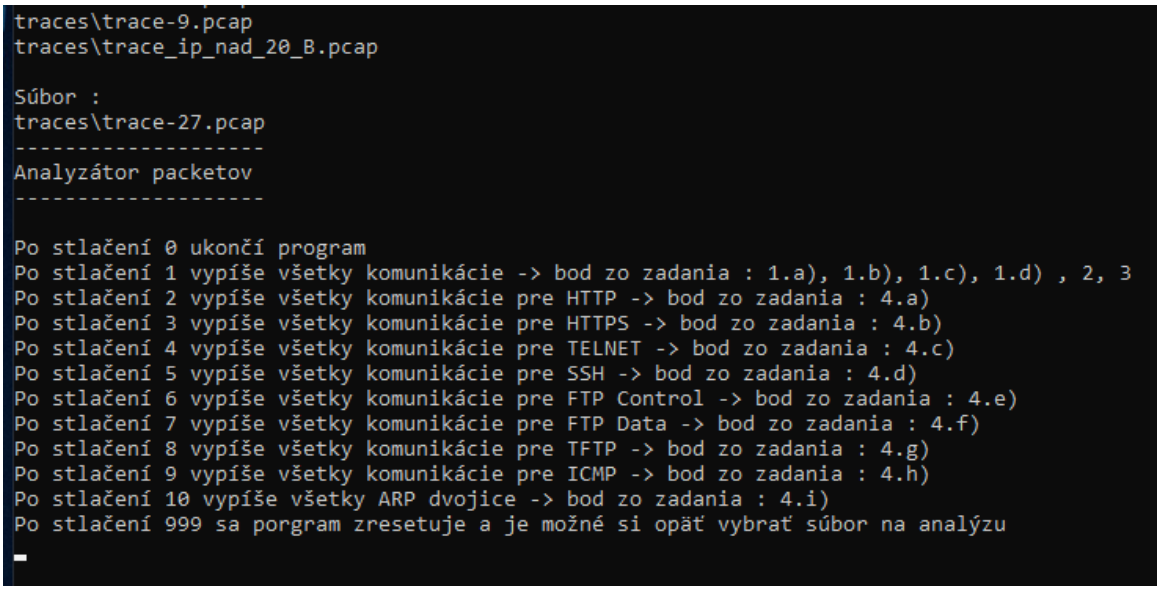
5 Používateľské rozhranie

Program po spustení vypíše pcap súbory ktoré je možno aktuálne analyzovať



Obr. 8: Program po spustení

Súbor stačí skopírovať z vypísaného zoznamu a vložiť za **Súbor:** + ENTER

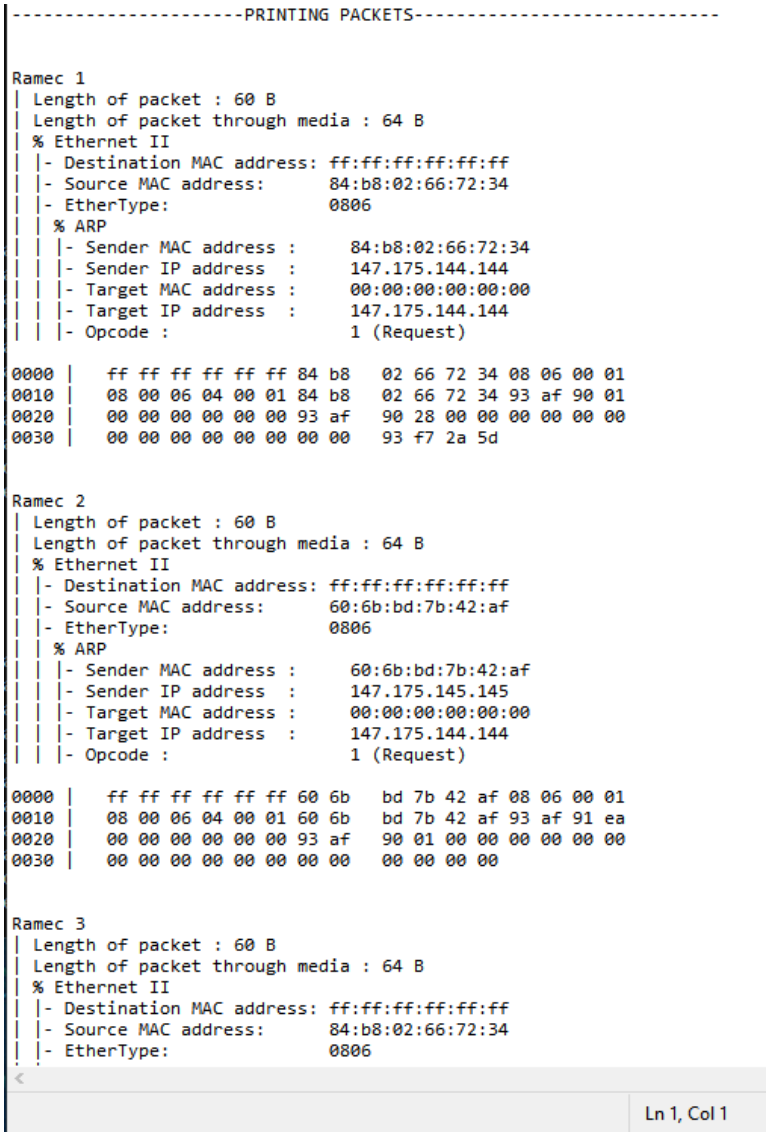


Obr. 9: Program po vybratí súboru

Takto program vyzerá po vybratí súboru. Vypíše sa základne menu ako možeme zvolený súbor analyzovať. Tento vypis menu sa zobrazí po každom stlačenú jednej z možností okrem možnosti 0 -> vypne program a možnosti 999 -> reset programu, zvolenie iného súboru. Po každej inej možnosti sa zobrazí toto menu znova a možeme vyerať iné možnosti. Výpisy sa exportujú do súboru program_output.txt tento súbor sa prepisuje za každým, po zvolení jednej z možností z menu

6 Výpisy zo súboru

Po stlačení 1 sa vypíšu všetky packetu z pcap súboru :



```
----------------------PRINTING PACKETS----------------------------


Ramec 1
| Length of packet : 60 B
| Length of packet through media : 64 B
| % Ethernet II
| |- Destination MAC address: ff:ff:ff:ff:ff:ff
| |- Source MAC address:      84:b8:02:66:72:34
| |- EtherType:               0806
| | % ARP
| | |- Sender MAC address :     84:b8:02:66:72:34
| | |- Sender IP address  :     147.175.144.144
| | |- Target MAC address :     00:00:00:00:00:00
| | |- Target IP address  :     147.175.144.144
| | |- Opcode :                 1 (Request)

0000 |   ff ff ff ff ff ff 84 b8   02 66 72 34 08 06 00 01
0010 |   08 00 06 04 00 01 84 b8   02 66 72 34 93 af 90 01
0020 |   00 00 00 00 00 00 93 af   90 28 00 00 00 00 00 00
0030 |   00 00 00 00 00 00 00 00   93 f7 2a 5d


Ramec 2
| Length of packet : 60 B
| Length of packet through media : 64 B
| % Ethernet II
| |- Destination MAC address: ff:ff:ff:ff:ff:ff
| |- Source MAC address:      60:6b:bd:7b:42:af
| |- EtherType:               0806
| | % ARP
| | |- Sender MAC address :     60:6b:bd:7b:42:af
| | |- Sender IP address  :     147.175.145.145
| | |- Target MAC address :     00:00:00:00:00:00
| | |- Target IP address  :     147.175.144.144
| | |- Opcode :                 1 (Request)

0000 |   ff ff ff ff ff ff 60 6b   bd 7b 42 af 08 06 00 01
0010 |   08 00 06 04 00 01 60 6b   bd 7b 42 af 93 af 91 ea
0020 |   00 00 00 00 00 00 93 af   90 01 00 00 00 00 00 00
0030 |   00 00 00 00 00 00 00 00   00 00 00 00


Ramec 3
| Length of packet : 60 B
| Length of packet through media : 64 B
| % Ethernet II
| |- Destination MAC address: ff:ff:ff:ff:ff:ff
| |- Source MAC address:      84:b8:02:66:72:34
| |- EtherType:               0806
```

Ln 1, Col 1

Obr. 10: Výpis po zvolení možnosti 1

Na konci výpisu je takisto aj informácia ohľadom IP adries všetkých odosielajúcich uzlov rodiny TCP/IPv4

```
Zoznam IP adries všetkých odosielajúcich uzlov rodiny TCP/IPv4:
147.175.98.238 - 20
147.175.99.30 - 20
147.175.98.217 - 1
147.175.98.54 - 11
147.175.98.4 - 1

Najviac packetov odoslala IP :
147.175.98.238 - 20
```

Obr. 11: Výpis po zvolení možnosti 1 - úplne na konci

Po stlačení 2 sa vypíšu HTTP komunikácie (ak sú) Zvolil som jednoduchý výpis pre lepšiu čitateľnosť a lepšie vystvetľovanie. V programe ja ale takisto vo funkcii **print_communication** zakomentovaný druhý spôsob výpisu, ktorý vypisuje celé packety ako v možnosti 1.

```
There are : 3 communication(s) for HTTP


Prvá nekompletná komunikácia
#007   192.168.1.102 -> 128.119.245.12  (4307 -> 80)
#008   128.119.245.12 -> 192.168.1.102  (80 -> 4307)
#009   192.168.1.102 -> 128.119.245.12  (4307 -> 80)
#010   192.168.1.102 -> 128.119.245.12  (4307 -> 80)
#011   128.119.245.12 -> 192.168.1.102  (80 -> 4307)
#012   128.119.245.12 -> 192.168.1.102  (80 -> 4307)
#027   192.168.1.102 -> 128.119.245.12  (4307 -> 80)


Prvá kompletná komunikácia
#.014   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.018   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.019   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.020   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.026   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.029   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.030   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.031   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.032   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.033   134.241.6.82 -> 192.168.1.102  (80 -> 4309)

........

#.048   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.049   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.050   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.051   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.052   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.053   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.054   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
#.055   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.056   192.168.1.102 -> 134.241.6.82  (4309 -> 80)
#.057   134.241.6.82 -> 192.168.1.102  (80 -> 4309)
```

Obr. 12: Výpis po zvolení možnosti 2

Po stlačení 8 sa vypíšu TFTP komunikácie (ak sú) Takisto je zvolený jednoduchy Štýl výpisu vo forme :
ramec sourceIP -> destinationIP sourcePORT -> destinatinoPORT
ale v programe je zakomentovaný celý výpis pre potrebu.

```
TFTP KOMUNIKÁCIE :

Komunikacia 1
ramec 001   172.26.0.4 -> 172.26.0.20  (42301 -> 69)
ramec 002   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 003   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 004   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 005   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 006   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 007   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 008   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 009   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 010   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)

........

ramec 110   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 111   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 112   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 113   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 114   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 115   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 116   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 117   172.26.0.4 -> 172.26.0.20  (42301 -> 55064)
ramec 118   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)
ramec 121   172.26.0.20 -> 172.26.0.4  (55064 -> 42301)


Komunikacia 2
ramec 123   172.26.0.4 -> 172.26.0.20  (58802 -> 69)
ramec 124   172.26.0.20 -> 172.26.0.4  (46201 -> 58802)
ramec 125   172.26.0.4 -> 172.26.0.20  (58802 -> 46201)
ramec 126   172.26.0.20 -> 172.26.0.4  (46201 -> 58802)
```

Obr. 13: Výpis po zvolení možnosti 8

7 Implementačné prostredie

Ako prostredie som si vybral Pycharm (Jetbrains). IDE(čka) od Jetbrains som už používal (Clion, PHPStorm, Intellij) a vyhovuje mi najmä kvalitne spracované debugovanie, vďaka čomu chyby v programe odstránim rýchlo. z knižníc som si naimportoval len

dpkt -> na otváranie pcap súborov

os -> vypis súborov po spustení programu

from contextlib import redirectstd_out - > presmerúvavam výstup z konzole do mojho txt súboru