Zadanie 1
Analyzátor sieťovej komunikácia

Laura Fulajtárová

Fakulta informatiky a informačných technológií STU

xfulajtarova@stuba.sk

ID: 120782

PKS – Utorok 18:00

14.10.2023

Contents

1. Zadanie úlohy

https://github.com/fiit-ba/pks-course/blob/main/202324/assignments/1_network_communication_analyzer/README.md

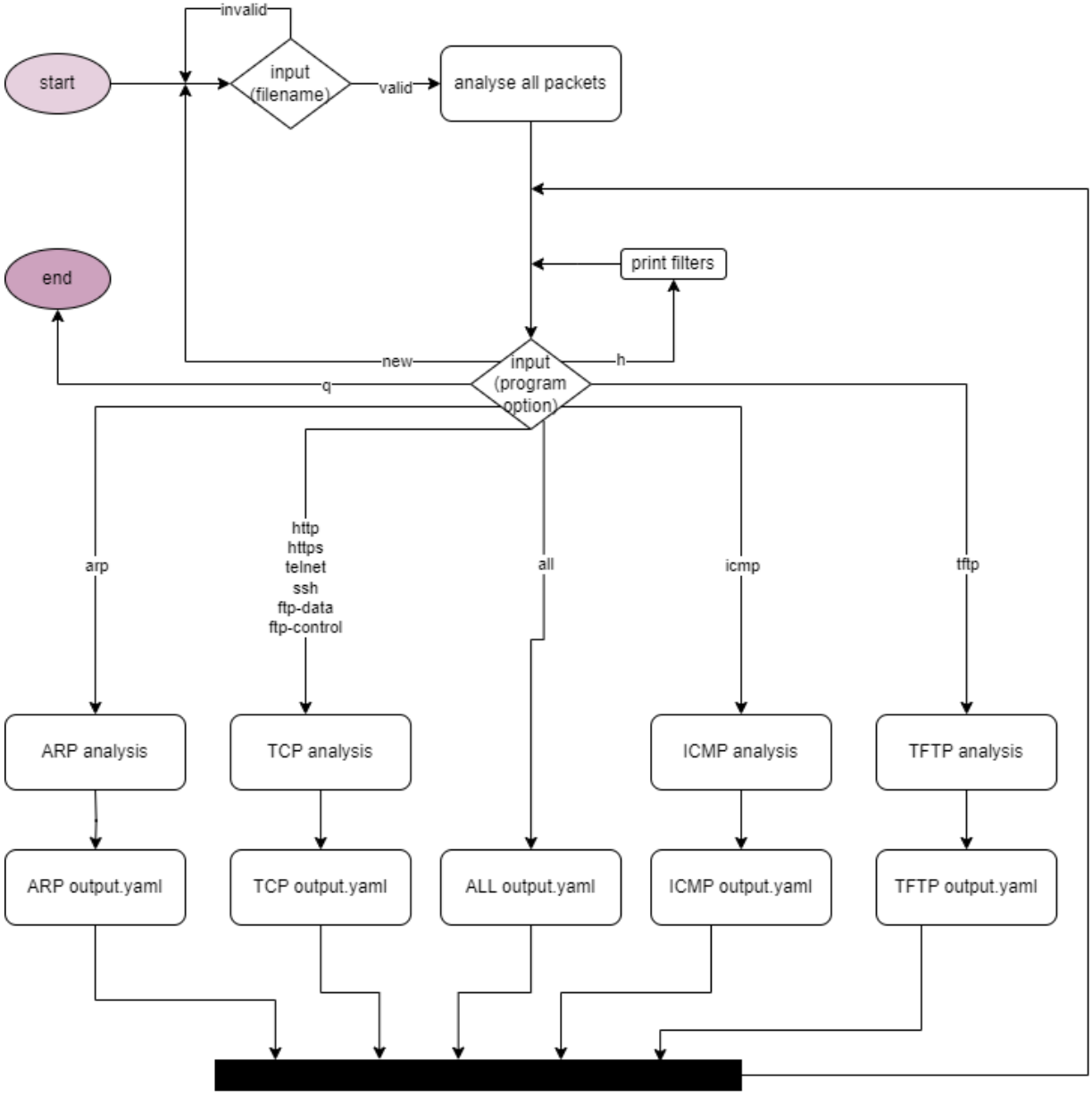
2. Diagram spracovávania a fungovanie riešenia

Pri vytvorení tohto programu pouužívateľ zadá názov súboru, ktorý chce analyzovať. Po identifikovaní tohto súboru program začne s analýzou všetkých paketov v ňom. Pre každý paket sa vytvorí objekt so všetkými spracovanými údajmi, a tieto objekty sa ukladajú do listu.

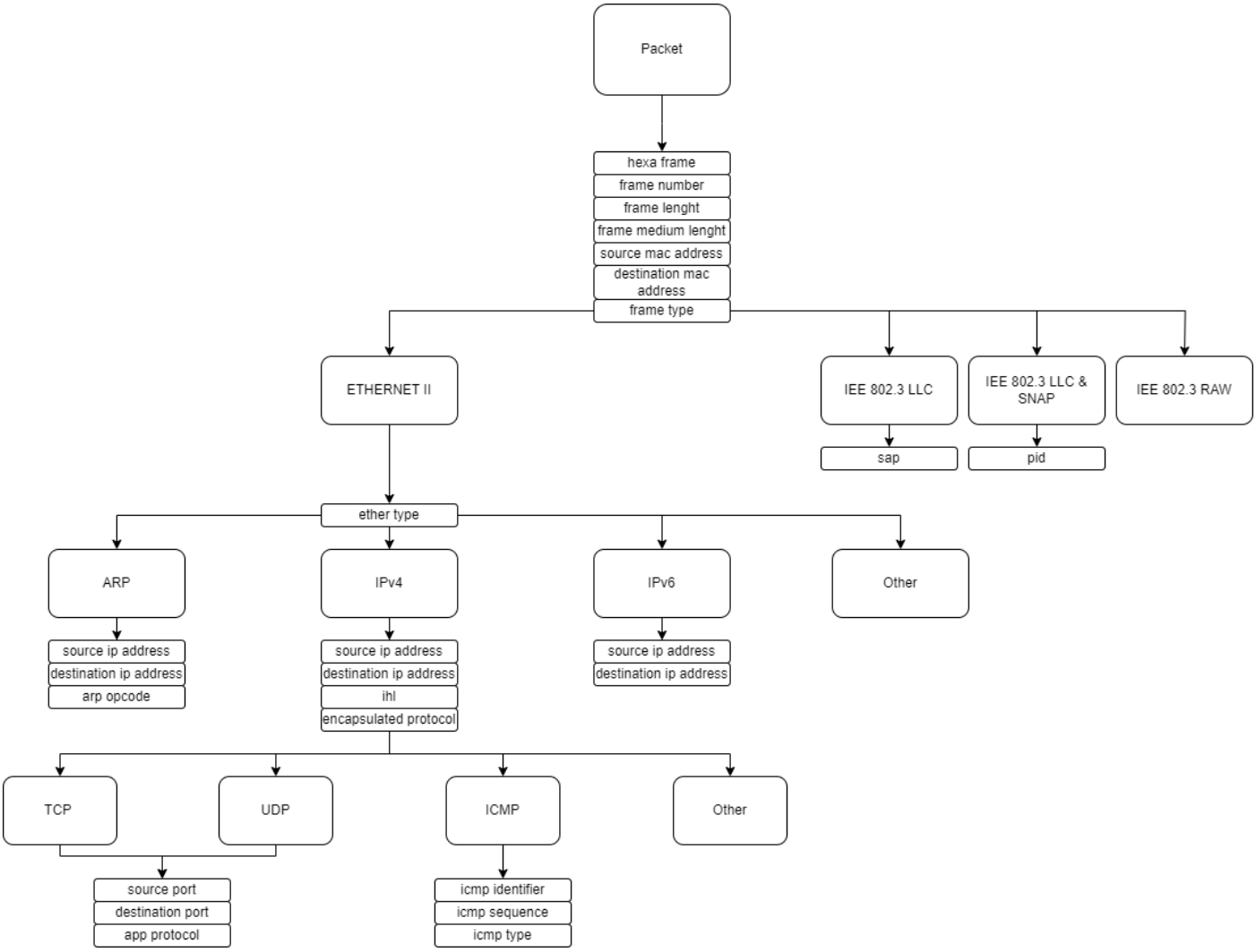
Následne sa používateľ rozhodne, čo chce s programom robiť. Ak používateľ zvolí "q" program sa ukončí. Pokiaľ vyberie možnosť "all" všetky spracované pakety sa vypíšu do súboru output.yaml. V prípade možnosti "new" sa program vráti na výber súboru. Okrem toho, ak zadá "h" program vypíše všetky akceptované filtre.

Používateľ má možnosť zadať filter, ako je HTTP, HTTPS, SSH, Telnet, ARP, ICMP, TFTP a ďalšie, pre výber týchto pakiet a následne ich analyzovanie z hľadiska kompletných alebo nekompletných komunikácií. Výsledky tejto analýzy sú vypísané a zobrazené v súbore output.yaml.

Po ukončení analýzy program vypisuje správu o jej ukončení a používateľ môže pokračovať v interakcii s programom na základe ponuky, ktorú som vytvorila.



3. Mechanizmus analyzovania protokolov na jednotlivých vrstvách



4. Implementácia riešenia

Moje riešenie pozostáva z 6 .py súborov:

- main.py
- analyze_packets.py
- tcp_assignment.py
- udp_assignment.py
- icmp_assignment.py
- arp_assignment.py

4.1. main.py

Slúži ako rozhranie medzi počítačom a užívateľom. Poskytuje interaktívne vstupy, číta .pcap súbory, extrahuje protokoly z externých zdrojov a generuje výstupný súbor output.yaml pre všetky pakety. Taktiež umožňuje detailnejšiu analýzu komunikácií na základe filtrov definovaných užívateľom. Bližšie informácie o fungovaní main nájdte tu: Diagram spracovávania a fungovanie riešenia.

Hlavná časť v main je načítanie všetkých pakiet, prechádzanie individuálnych pakiet a vytvorenie objektov k nim, uloženie všetkých objektov do listu:

```
try:
        packets = scapy.rdpcap(file_path)
    except FileNotFoundError:
        print(f"File not found: {file_path}")
```

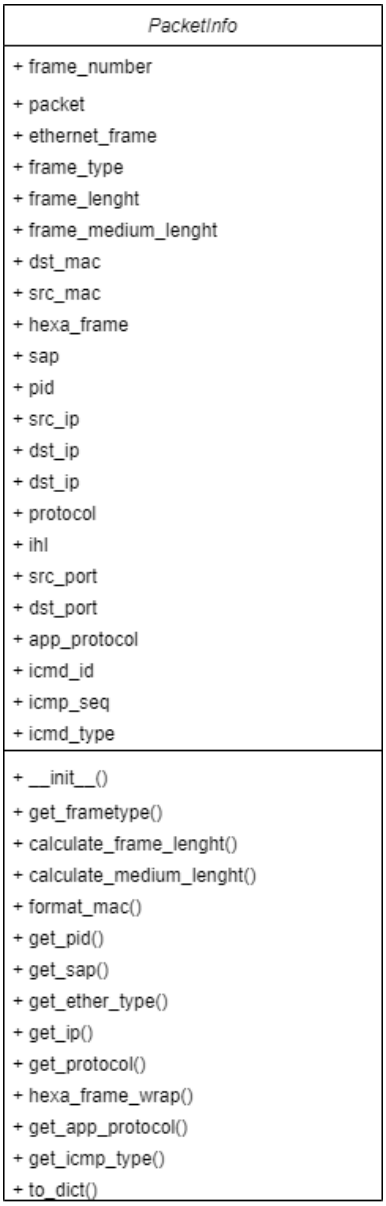
```
        main()

    packet_info_list = []
    all_packets = []

    for i, packet in enumerate(packets):
        packet_info = analyze_packets.PacketInfo(
            packet,
            i + 1,
            ethertypes,
            protocols,
            tcps_udps,
            pids,
            saps,
            icmp_types,
            senders,
        )
        all_packets.append(packet_info)
        packet_info_list.append(packet_info.to_dict())
```

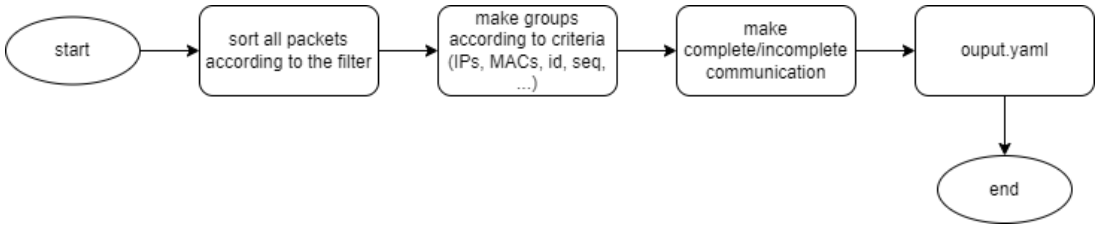
4.2. analyze_packets.py

Tento súbor obsahuje kľúčovú triedu PacketInfo, ktorá slúži na uchovávanie informácií o jednotlivých paketoch v objekte. Jej úlohou je podrobne charakterizovať každý paket a zaznamenať všetky príslušné vlastnosti. Pre charakterizovanie jednotlivých metód odporúčam pozrieť celý analyze_packets.py súbor.



4.3. tcp_assignment.py, udp_assignment.py, icmp_assignment.py, arp_assignment.py

V týchto súboroch sme sa zameriavali na individuálnu analýzu sieťovej komunikácie pre zvolené filtre. Začali sme tým, že sme filtrovali a vybrali relevantné pakety pre daný filter zo všetkých dostupných paketov, čo nám umožnilo následne pracovať s nimi. Potom sme tieto pakety triedili do skupín podľa rôznych kritérií, ako sú zdrojová IP adresa, cieľová IP adresa, MAC adresy, identifikátory a ďalšie. Tieto skupiny sme postupne prechádzali a analyzovali, či sú kompletné alebo nekompletné, a tieto informácie sme ukladali do listov. Nakoniec sme tieto dva listy výsledkov exportovali do výstupného súboru output.yaml s príslušným formátovaním. Týmto spôsobom sme umožnili detailnú analýzu a záznam sieťovej komunikácie pre zvolené filtre.



Príklad pre analýzu TFTP komunikácie:

main.py:

```
elif user_input in udp_values:
            keys_list = []
            start_list = []
            for packet in all_packets:
                if (
                    packet.frame_type == "ETHERNET II"
                    and packet.ether_type == "IPv4"
                    and packet.protocol == "UDP"
                    and packet.app_protocol == user_input
                ):
                    keys_list.append(packet.src_port)
                    start_list.append([packet])
            udp_assignment.udp_quest(
                start_list, keys_list, all_packets, file_name, user_input
            )
            print(Orange + "UDP analysis done" + reset)
```

udp_assignment.py

```
def dump_yaml(complete_list, incomplete_list, file_name, user_input):
       # funkcia iba vypisuje komunikaci vo zvolenom formatovani
       # nedala som ju sem, kvoli jej zdlhavosti a nepotrebnosti vzhladom na logiku programu
       # mozete ju najst v pribalenom subore

def udp_quest(packet_list, keys_list, all_packets, file_name, user_input):
    for i, item in enumerate(keys_list):
        des = None
        for j, packet in enumerate(all_packets):
            if (
                packet.frame_type == "ETHERNET II"
                and packet.ether_type == "IPv4"
                and packet.protocol == "UDP"
            ):
                if des == None and packet.dst_port == item:
                    des = packet.src_port
                    packet_list[i].append(packet)

                elif des != None and (
                    (packet.src_port == item and packet.dst_port == des)
                    or (packet.src_port == des and packet.dst_port == item)
                ):
                    packet_list[i].append(packet)

    complete_list, incomplete_list = check_valid(packet_list)

    dump_yaml(complete_list, incomplete_list, file_name, user_input)
```

4.4. Podmienky pre určenie komunikácií

Skratky na pochopenie:

- ACK – acknowledgement field value valid
- RST – reset connection
- SYN – synchronize sequence numbers
- FIN – no more data; finish connection

4.4.1. TCP filtre (HTTP, HTTPS, TELNET, SSH, FTP-CONTROL, FTP-DATA)

Pre úplnú komunikáciu je potrebné správne otvorenie a uzatvorenie pomocou handshaku. Klient musí poslať SYN a po jeho prijatí server mu odošle tiež SYN. Ak klient poslal SYN, môže prijať ACK od servera. Ak server poslal SYN, musí prijať ACK od klienta. Pre uzatvorenie klient posiela FIN a potom prijme ACK od servera (môže to byť aj kombinácia FIN, ACK). Server po prijatí FIN od klienta pošle ACK, pošle FIN klientovi a následne prijme ACK od klienta. V prípade nejakeho zlyhania môže nastať RST od servera na ukončenie komunikácie. Takéto ukončenie taktiež zaraďujeme do kompletných.

```
def find_communication(used_flags, sorted_list):
    complete_list = []
    incomplete_list = []

    for j, group in enumerate(used_flags):
        opening = False
        closing = False

        opening_syn1 = False
        opening_syn2 = False
        opening_ack1 = False
        opening_ack2 = False

        closing_fin1 = False
        closing_fin2 = False
        closing_ack1 = False
        closing_ack2 = False

        for i, item in enumerate(group):
            if "SYN" in item[1]:
                if not opening_syn1:
                    first_o = item[0]
                    opening_syn1 = True
                if (
                    first_o != item[0] and opening_syn1

                    opening_syn2 = True
            if "ACK" in item[1] and opening_syn1 and opening_syn2:
                if first_o != item[0]:
                    opening_ack1 = True
                elif (
                    first_o == item[0] and opening_ack1
                ):
                    opening_ack2 = True

            if opening_syn1 and opening_syn2 and opening_ack1 and opening_ack2:
                opening = True

            if opening:
                if "RST" in item[1]:
                    closing = True
                    break
                if "FIN" in item[1]:
                    if not closing_fin1:
                        first_c = item[0]
                        closing_fin1 = True
                    if first_c != item[0] and closing_fin1:
                        closing_fin2 = True
                if "ACK" in item[1] and closing_fin1:
```

```
                    if first_c != item[0]:
                        closing_ack1 = True
                    elif first_c == item[0] and closing_ack1:
                        closing_ack2 = True

                if closing_fin1 and closing_fin2 and closing_ack1 and closing_ack2:
                    closing = True

        if not (opening and closing):
            incomplete_list.append(sorted_list[j])
        else:
            complete_list.append(sorted_list[j])

    return complete_list, incomplete_list
```

4.4.2. TFTP filter

Komunikácia musí prebiehať na rovnakých portoch, avšak prvý packet má dst_port 69 a src_port náhodný. Packety s dst_port rovnakým ako src_port prvého packetu zaradíme do skupiny a zistíme src_port druhého packetu. Komunikácia bude prebiehať na týchto nových portoch. Keď máme jednotlivé packety rozdelené do skupín, pozrieme sa na predposledný packet, ktorý musí mať menej ako 512 bytov (menší ako dohodnutá veľkosť bloku pri vytvorení spojenia), zároveň odosielateľ tohto paketu a posledný packet musia obsahovať ACK.

```
def check_valid(packet_list):
    complete_list = []
    incomplete_list = []

    for group in packet_list:
        len_check = False
        ack_check = False
        penultimate = group[-2]
        last = group[-1]
        penultimate_len = int(
            penultimate.ethernet_frame[
                14 + penultimate.ihl + 4 : 14 + penultimate.ihl + 6
            ].hex(),
            16,
        )
        last_ack = last.ethernet_frame[14 + last.ihl + 8 : 14 + last.ihl + 10].hex()
        if penultimate_len < 512:
            len_check = True
            if last_ack == "0004":
                ack_check = True
        if len_check and ack_check:
            complete_list.append(group)
        else:
            incomplete_list.append(group)

    return complete_list, incomplete_list
```

4.4.3. ICMP filter

Pre vytvorenie komunikácie som jednotlivé packety zoskupila do groups podľa spoločných IPs, identifieru a sequence number. Takto som vytvorila dvojice Echo request + Echo reply alebo Echo request + Time exceeded, prípadé iné nekompletné packety. Avšak pri TTL som musela správne priradiť IP adresy, pretože túto správu nám odosiela router. Ak komunikácia splní jednu z týchto dvojíc, považujem ju za úplnú. V iných prípadoch je komunikácia neúplná.

```
def check_valid(packet_list):
    complete_list = []
    incomplete_list = []

    for group in packet_list:
        request = False
        reply = False
        request_index = None
        reply_index = None
        complete_group = []

        for i, packet in enumerate(group):
            if packet.icmp_type == "Echo Request":
```

```
                request = True
                request_index = i
            elif (
                (
                    packet.icmp_type == "Echo Reply"
                    or packet.icmp_type == "Time Exceeded"
                )
            ) and request == True:
                reply = True
                reply_index = i

            if reply:
                complete_group.append(group[request_index])
                complete_group.append(group[reply_index])
                reply = False
                request = False

        if complete_group:
            complete_list.append(complete_group)

    for group in packet_list:
        incomplete_group = []
        for item in group:
            if not any(item in complete for complete in complete_list):
                incomplete_group.append(item)
        if incomplete_group:
            incomplete_list.append(incomplete_group)

    return complete_list, incomplete_list
```

4.4.4. ARP filter

Packety som zoskupila podľa IP a MAC adries. Najskôr som musela nájsť adresu, ktorá sa viaže k našemu requestu, pretože dst_mac bola broadcast. Po nájdení tejto MAC adresy som skúmala, či celková komunikácia obsahuje Request a príslušný Reply. Ak sa poslalo viacero requests alebo prijalo viacero replys, túto kombináciu som zaznamenala iba raz do úplnej komunikáciie, ostatné packety som priradila k neúplnej.

```
def check_valid(packet_list):
    complete_list = []
    incomplete_list = []

    for group in packet_list:
        request = False
        request_index = None
        reply = False
        reply_index = None
        complete_group = []

        for i, packet in enumerate(group):
            if packet.arp_opcode == "Request":
                request = True
                request_index = i
            elif packet.arp_opcode == "Reply" and request:
                reply_index = i
                reply = True

            if reply:
                complete_group.append(group[request_index])
                complete_group.append(group[reply_index])
                reply = False
                request = False

        if complete_group:
            complete_list.append(complete_group)

    for group in packet_list:
        incomplete_group = []
        for item in group:
            if not any(item in complete for complete in complete_list):
                incomplete_group.append(item)
```

```
        if incomplete_group:
            incomplete_list.append(incomplete_group)

    return complete_list, incomplete_list
```

5. Štruktúra externých súborov pre určenie protokolov a portov

Všetky typy údajov uchovávam v súbore data.txt

Ukážka z data.txt:

```
#ethertypes
0800:IPv4
0806:ARP
88cc:LLDP
86dd:IPv6
9000:ECTP
#protocol
1:ICMP
2:IGMP
6:TCP
```

6. Používateľské rozhranie

V mojom programe používateľské rozhranie prebieha cez terminál. Na začiatku, používateľ zadá názov súboru. Ak program nevie nájsť tento súbor, zobrazí chybové hlásenie a požiada o znovu zadaný názov súboru. Potom program zobrazí niekoľko možností v termináli, ako pokračovať, viď obrázok nižšie. Ak používateľ zadá nesprávny vstup, program zobrazí "Wrong input" a požiada o nový vstup. Pre platný vstup program vykoná danú funkciu a zobrazí hlášku o jej dokončení. Výstupy sú prezentované v YAML súbore s názvom "output.yaml". Program sa neskončí automaticky, ale namiesto toho ponúkne používateľovi možnosť pokračovať alebo ho ukončiť.

```
Enter file name: trace-100
File not found: project1/vzorky_pcap_na_analyzu/trace-100.pcap

Enter file name: trace-27

Enter:
   1) "all"
   2) "filter name"
   3) "new" to enter a new file name
   4) "h" print valid filters
   5) "q" to quit
Enter your choice: abcd
Wrong input

Enter:
   1) "all"
   2) "filter name"
   3) "new" to enter a new file name
   4) "h" print valid filters
   5) "q" to quit
Enter your choice: h
Valid filters: HTTP, HTTPS, TELNET, SSH, FTP-CONTROL, FTP-DATA, TFTP, ICMP, ARP

Enter:
   1) "all"
   2) "filter name"
   3) "new" to enter a new file name
   4) "h" print valid filters
   5) "q" to quit
Enter your choice: https
TCP analysis done

Enter:
   1) "all"
   2) "filter name"
   3) "new" to enter a new file name
   4) "h" print valid filters
   5) "q" to quit
Enter your choice: q
Goodbye
```

7. Implementačné prostredie

Rozhodla som sa implementovať riešenie svojho projektu v programovacom jazyku Python kvôli jeho jednoduchej syntaxi a intuitívnemu programovaniu, čo mi umožnilo rýchlo a efektívne pracovať na ňom. Aktívne som pracovala s

rôznymi dátovými štruktúrami, ako sú listy a dictionaries, čo mi umožnilo efektívne manipulovať a analyzovať dáta. Taktiež som potrebovala pracovať s externými súbormi, čo je nevyhnutné pre prácu s uloženými dátami. Python mi tiež umožnil importovať moje vlastné .py súbory, čo mi poskytlo flexibilitu a efektívnosť pri práci na projekte. Vďaka týmto výhodám sa Python ukázal byť optimálnou voľbou pre realizáciu môjho projektu.

8. Zhodnotenie

Projekt je komplexným analyzátorom sieťovej komunikácie. Okrem základnej analýzy rámcov poskytuje aj detailné informácie o Ethernet II a vyšších vrstvách, vrátane rozpoznania známych protokolov a portov. Vyniká svojou schopnosťou rozpoznať úplnú aj neuplnú komunikáciu a umožňuje detailné špecifikácie pre konkrétne protokoly.

V projekte by som mohla ďalej doimplementovať fragmentáciu paketov, čo by umožnilo spracovanie a rekonštrukciu fragmentovaných paketov. Bohužiaľ, v pôvodnom časovom rámci som to nestihla. Okrem toho by som sa mohla zamerať na optimalizáciu výkonu programu, čo by zabezpečilo rýchlejšiu analýzu veľkých súborov .pcap. Tieto ďalšie vylepšenia by umožnili rozšíriť funkčnosť projektu a zlepšiť jeho výkon.