Faculty of Informatics and Information Technologies STU, Bratislava Ilkovičova 2, 842 16 Bratislava 4

Zadanie 1
Analyzátor sieťovej komunikácie

Počítačové a komunikačné siete

Frederik Duvač
3. semester
15.10. 2023

Obsah

1. Technologické pozadie

1.1. Hardware

Zariadenie, na ktorom bol skompilovaný zdrojový kód a testovaný program, je notebook Lenovo Legion S7, 12. generácia Intel® Core™ i5-12500H, 16 GB RAM.

1.2. Programovací jazyk

Celý projekt je implementovaný v programovacom jazyku Python 3. V programe je importovaná externá knižnica Scapy, použitá výhradne za účelom otvorenia "pcap" súborov na analýzu, použitím funkcie `rdpcap("example.pcap")`. Žiadne ďalšie funkcionality tejto knižnice neboli v programe použité. Externé súbory potrebné pre fungovanie programu sú v štandardizovanom formáte Yaml, tak ako aj výstupy z analyzovaných dát pomocou nášho analyzátora sieťovej komunikácie. O výstupy sa stará funkcia `dump()` poskytnutá knižnicou Ruamel.

1.3. Vývojové prostredie

Analyzátor sieťovej komunikácie bol počas svojej evolúcie plne prítomný a verný jedinému vývojovému prostrediu, ktoré zabezpečovalo hladkú a programátorsky príjemnu interakciu pri zostavovaní, interpretovaní a spúšťaní programu. Áno, reč nie je o žiadnom inom vývojovom prostredí ako PyCharm v balíku Professional vo verzii 2023.2.3 od spoločnosti JetBrains.

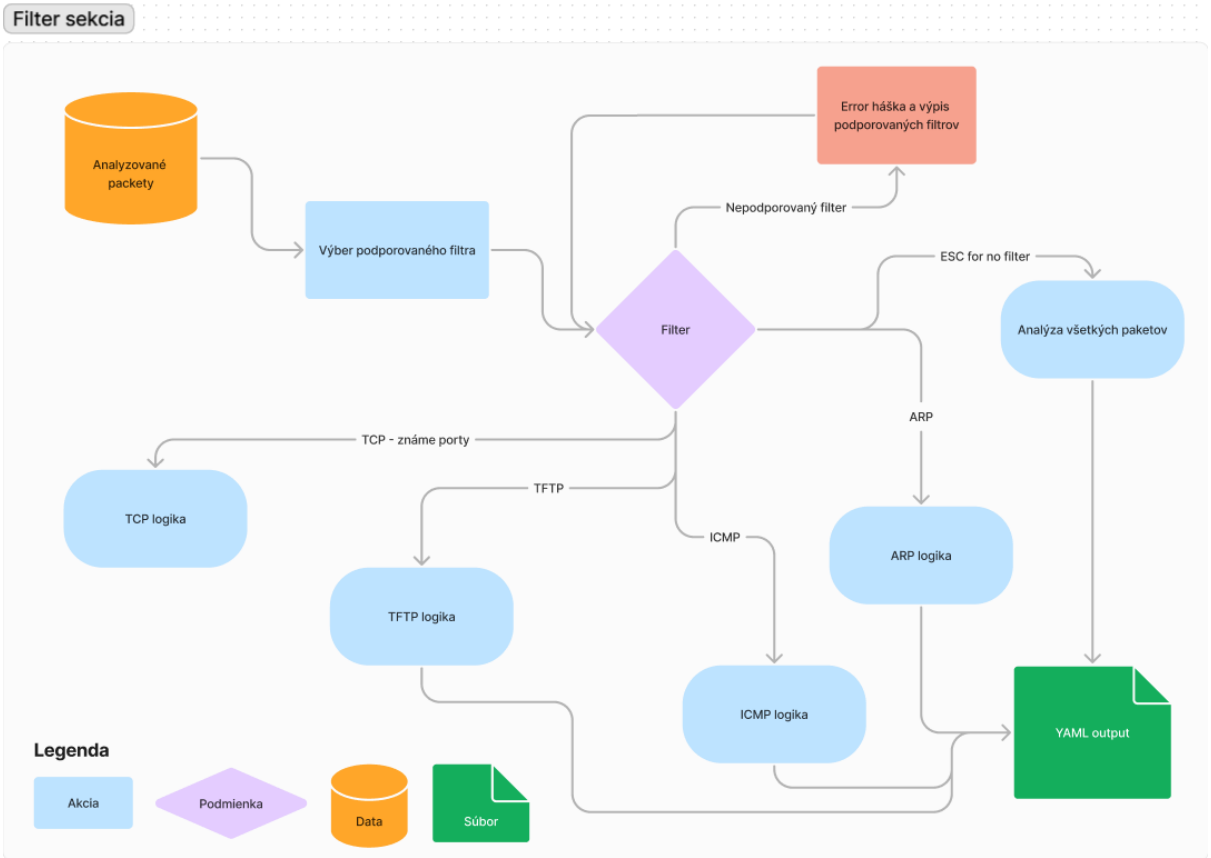
2. Implementácia

2.1. Algoritmus - vývojový diagram

2.1.1. Sekcia filter

Filter sekcia slúži na rozdelenie spracovaných sieťových paketov na základe vstupu v podobe špecifického protokolového filtra, ako napríklad “ARP”, “ICMP”, “TFTP” alebo podporovaného filtra pre TCP komunikáciu. Výstupom sú spracované pakety a rozdelené do úplných a čiastočných komunikácií.

Program načíta protokoly z YAML súboru, kde sú definované filtre pre známe protokoly TCP komunikácie ako sú HTTP, HTTPS, TELNET, SSH, FTP-CONTROL, FTP-DATA. Dalšími podporovanými protokolmi sú TFTP, ICMP a ARP.



2.1.2. Výstup bez filtra

Program prejde celým polom paketov získaných zo súboru. Každý paket je prevedený na pole hexadecimálnych bytov. Získame informácie o ethernetovej hlavičke z binárnych dát paketu, ako sú zdrojová MAC adresa, cieľová MAC adresa a typ ethernetu. Vypočítame dĺžku rámca v pcap formáte a dlzku na mediu.

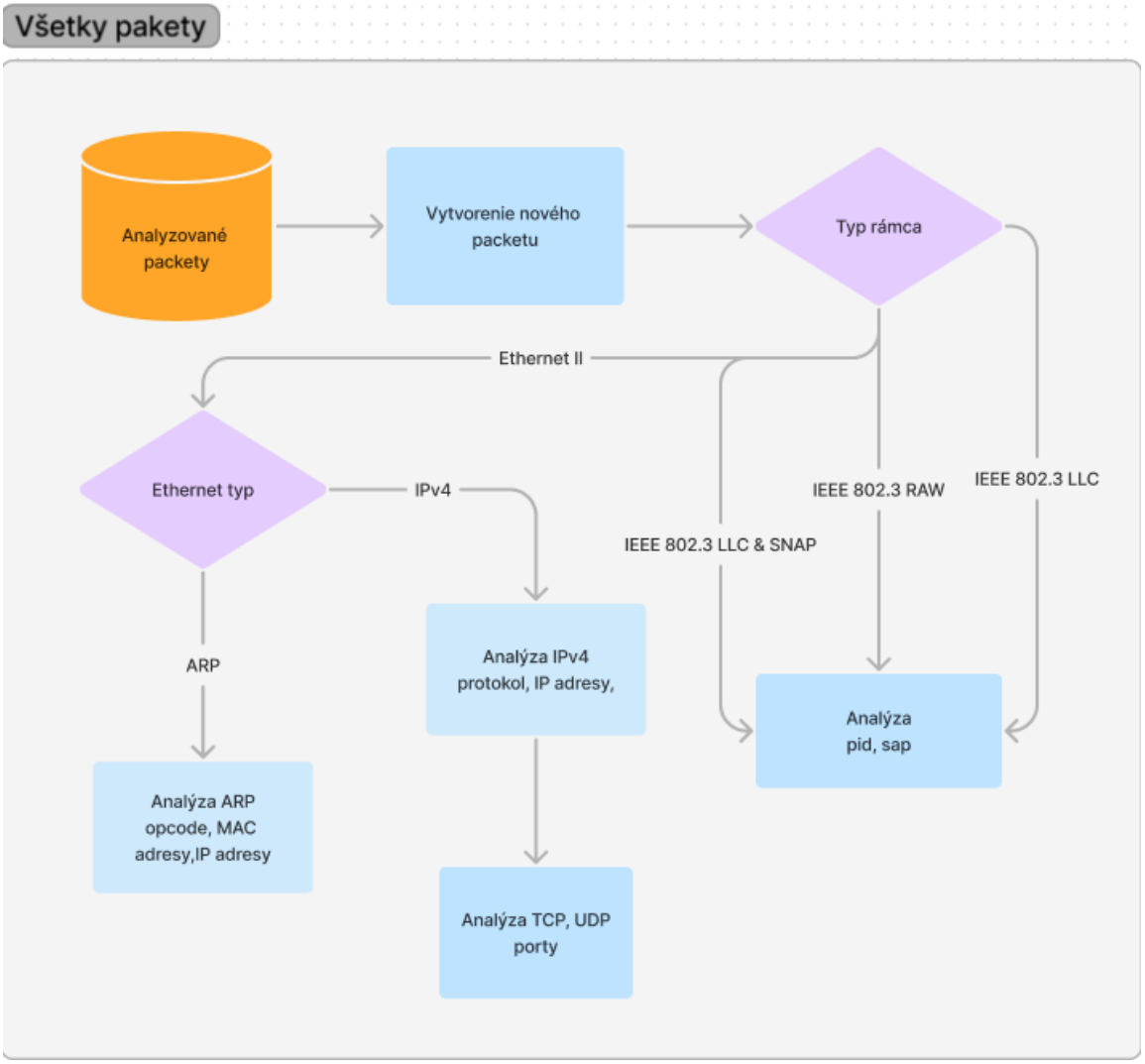
Následne sa rozpozná, či ide o rámec v Ethernet II formáte alebo IEEE 802.3 formáte podľa hodnoty "ether_type". Ak ide o IEEE 802.3 formát, detekujeme rôzne typy IEEE 802.3 rámca podľa hodnoty "dsap" (Destination Service Access Point).

Ak je rámec typu Etherne II, získame informácie o ARP rámcoch a IP rámcoch. Ak ide o rámec s IPv4 protokolom, analyzujeme ďalšie informácie vrátane IP adries, typu prenosu (TCP alebo UDP) a portov.

Ďalej zaznamenávame všetky IP adresy a počítame výskyty jednotlivých odoslaných paketov.

Ak ide o rámec s protokolmi TCP alebo UDP, získame porty a zistíme informácie o protokoloch, ktoré komunikujú na týchto portoch.

Informácie o pakete sa uložia do python dictionary. Postupne zostavíme štruktúru dát pre každý paket a pridáme tieto dáta do zoznamu "packets_to_yaml".



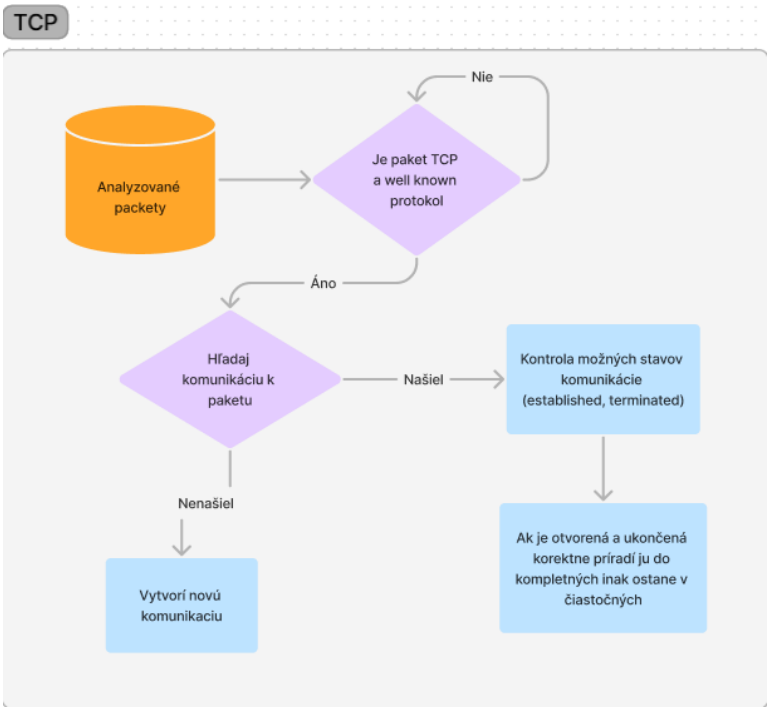
2.1.3. TCP - známe porty

Program prechádza všetky pakety, ktoré sú uložené v zozname "packets_to_yaml". Sleduje len tie pakety, ktoré majú protokol TCP a zároveň zodpovedajú vybranému vstupnému filtru.

Následne program prehľadáva čiastočné komunikácie, kde sa pre každý paket nájsť komunikáciu, ku ktorej patrí. V prípade, že sa komunikácia prislúchajúca paketu medzi čiastočnými komunikáciami nenašla, je potrebné vytvoriť novú komunikáciu a priradiť k nej aktualny paket. Naopak ak sa komunikácia pre paket našla, algoritmus skontroluje, či komunikácia už prešla do stavu "established" (est), čo sa bežne deje pomocou 3-way handshake, keď nastane určitá výmena flagov medzi zariadeniami vzor SYN, SYN-ACK a ACK. Ak áno, nastaví "est" na True pre túto komunikáciu. Táto podmienka je výlučne na zefektívnenie programu.

Ak je paket súčasťou existujúcej komunikácie, pridá sa do zoznamu paketov danej komunikácie.

Ak obsahuje paket informáciu o tom, že chce zariadenie ukončiť komunikáciu, v takom prípade program začína sledovať či nastane korektné ukončenie komunikácie pomocou 4-way handshake alebo pomocou resetu, iba v prípade, ak bola komunikácia správne nadviazaná. V prípade, ak nastanú takéto situácie komunikácia sa vyradí z čiastočných a zaradí sa medzi kompletné komunikácie.



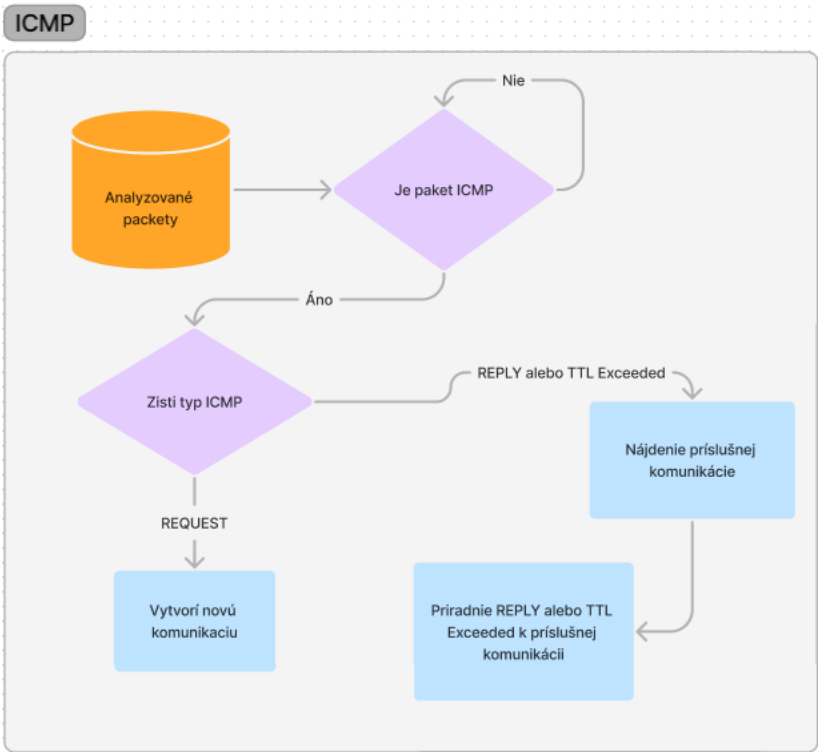
2.1.4. ICMP sekcia

Program prejde všetkými paketmi a skontroluje, či "protocol" paketu je nastavený na "ICMP". Ak áno, pokračuje ďalej. Analyzujeme hlbšie rámec a získame základné informácie o pakete, ako sú zdrojová a cieľová IP, prípadný offset IPv4 hlavičky a icmp typ.

Skontrolujeme, či je "icmp_type" rovné 8, čo zodpovedá ICMP request. V takom prípade vytvára novú komunikáciu a pridá do nej tento paket.

Ak "icmp_type" je rovné 0 alebo 11, čo zodpovedá ICMP reply alebo ICMP TTL Exceeded, algoritmus skúma existujúce komunikácie v "partial_communications" a priradí tento paket k príslušnej komunikácii na základe zdrojovej a cieľovej IP adresy a "icmp_id". Ak sa príslušná komunikácia nenašla, vytvárame novú komunikáciu a pridáme do nej tento paket.

Pre všetky ostatné hodnoty "icmp_type" algoritmus vytvára novú komunikáciu. Takto rozdelíme ICMP pakety podľa typu a spárujeme ich s príslušnými komunikáciami.



2.1.5. ARP sekcia

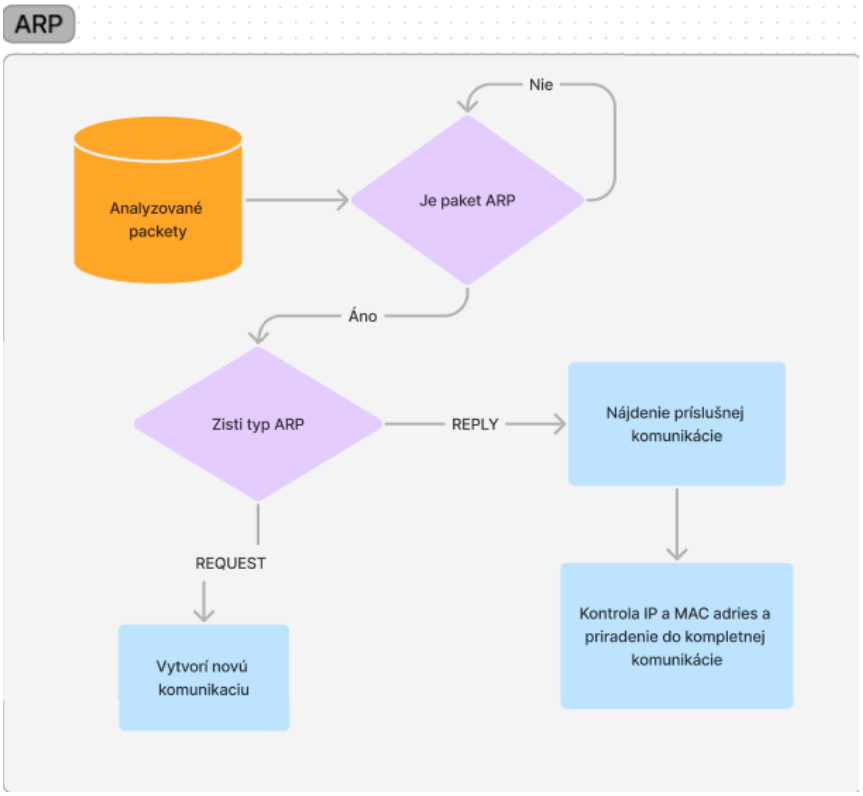
Pre každý paket v zozname "packets_to_yaml" skontroluje, či "ether_type" paketu je nastavený na "ARP". Ak áno, pokračujeme ďalej. Skontrolujeme hodnotu "arp_opcode" paketu.

Ak je nastavená na REQUEST, paket sa pridá do zoznamu "partial_communications". Ak "arp_opcode" paketu je nastavený na

REPLY, algoritmus skontroluje, či existuje príslušný REQUEST paket v zozname "partial_communications". Ak áno, paket je považovaný za odpoveď na existujúci REQUEST paket a oba sú pridané do zoznamu "complete_communications". Potom sú tieto dva pakety odstránené z "partial_communications".

Ak sa v "partial_communications" nenašiel príslušný REQUEST paket, paket je pridaný do "partial_communications".

Týmto spôsobom algoritmus spracováva ARP pakety a zabezpečuje, že ARP reply pakety sú spojené s príslušnými ARP request paketmi, aby bolo možné sledovať ARP komunikáciu v sieti.



2.1.6. TFTP sekcia

Tento filter spracováva pakety s protokolom UDP a s protokolom TFTP (Trivial File Transfer Protocol). Algoritmus prechádza všetky pakety a vyberá tie pakety, ktoré majú protokol UDP.

Pre každý UDP paket sa sleduje, či ide o TFTP komunikáciu. Ak ide o TFTP komunikáciu, vytvorí sa nová komunikácia. Premenná counter sleduje 1 alebo 2 paket po TFTP pakete s dohodnutou veľkosťou dát, pretože v prípade write requestu velkost dat sa zisti z paketu, až keď server potvrdil request. V prípade read requestu je veľkosť dát hned v prvom dátovom pakete.

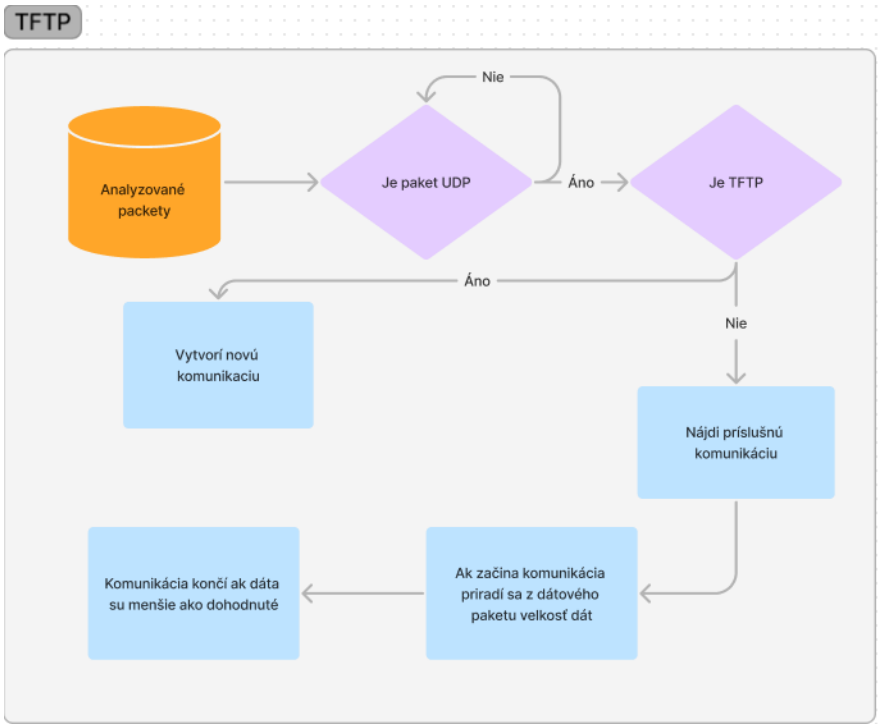
Získame dôležité informácie z TFTP paketu, ako sú typ TFTP operácie, zdrojová a cieľová IP adresa, zdrojový a cieľový port a ďalšie

informácie. Program hľadá existujúce komunikácie, ku ktorým paket patrí podľa zdrojovej a cieľovej IP adriesy a portov.

Ak sa komunikácia nájde, zmeníme cieľový port 69(TFTP) komunikácie na zdrojový port tohto paketu, pretože nasledujúca komunikácia bude prebiehať len cez tieto dva porty.

Kontrolujeme, či sme zistili dĺžku dát komunikácie (pole "data_size") a ak nie, určíme ju na základe typu TFTP operácie ("tftp_type").

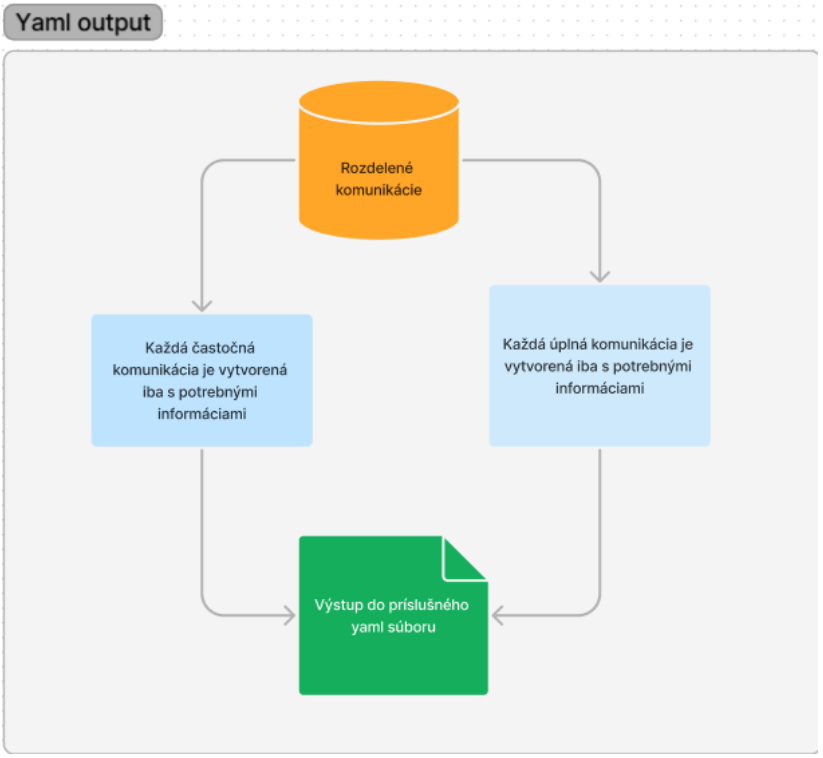
Paket sa pridá do zoznamu paketov danej komunikácie, zároveň sa sleduje, či je komunikácia ukončená. Komunikácia sa ukončí v prípade, ak posledný dátový paket obsahuje mensiu dĺžku dát ako na zaciatku. Ak je komunikácia ukončená, pridáme ju do zoznamu úplných komunikácií.



2.1.7. YAML výstup

Program formátuje dáta pre každú komunikáciu do vhodného výstupného formátu, vrátane relevantných údajov, ako sú poradové čísla komunikácií, IP adresy a pakety, pri čiastočných komunikáciách sa IP adresy nevypisujú.

Spracované dáta sa zapíšu do príslušných YAML súborov. Výstupné súbory sú pomenované podľa vybraného protokolového filtra, ako napríklad "packets-http.yaml", "packets-icmp.yaml" atď.



3. Používateľské rozhranie

Používateľské rozhranie bolo implementované ako jednoduché menu so vstupom v konzole, kde sa najskor program spýta na názov “pcap” súboru. Na základe poskytnutého názvu (podpora aj bez typu súboru example.pcap alebo example) program vyhľadá súbor v projektovej štruktúre a overí či sa tam nachádza, ak sa nachádza viackrát, program vypíše upozornenie na odstránenie duplikátov. V prípade ak sa našiel správny súbor program ponúkne možnosť filtra, ktorý si prajeme aplikovať na vyfiltrovanie analyzovanej komunikácie.

```
Input "pcap" file >> trace-10
Supported filters: HTTP | HTTPS | TELNET | SSH | FTP-CONTROL | FTP-DATA | TFTP | ICMP | ARP
Apply some protocol filter (ESC for no filter) >> random
Error: protocol RANDOM is not supported as filter!
Supported filters: HTTP | HTTPS | TELNET | SSH | FTP-CONTROL | FTP-DATA | TFTP | ICMP | ARP
Apply some protocol filter (ESC for no filter) >> http

Data has been successfully streamed to packets-http.yaml
```

Obr. 1: Príklad možnej interakcie s programom

4. Štruktúra externých súborov

V programe sú používané 3 externé súbory. 2 externé súbory sú vo formáte YAML a ďalší súbor obsahuje konštanty používané programom.

main.py protocol_filters.yml

```yaml
protocol_filters:
  - HTTP
  - HTTPS
  - TELNET
  - SSH
  - FTP-CONTROL
  - FTP-DATA
  - TFTP
  - ARP
  - ICMP

tcp_filters:
  - HTTP
  - HTTPS
  - TELNET
  - SSH
  - FTP-CONTROL
  - FTP-DATA
```

main.py protocol_filters.yml info_data.yaml

```yaml
  34: IPv6 I-Am-Here
  35: Mobile Registration Request
  36: Mobile Registration Reply
  37: Domain Name Request
  38: Domain Name Reply
  39: SKIP
  40: Photuris

tftp_type:
  1: Read request (1)
  2: Write request (2)
  4: Acknowledgement (4)
  5: Error code (5)

arp_opcode:
  1: REQUEST
  2: REPLY

wellknown_ports:
  20: FTP-DATA
  21: FTP-CONTROL
  22: SSH
  23: TELNET
  25: SMTP
  53: DNS
```

Obr. 2: Ukážka formátovania externých súborov

5. Záver

5.1. Zhodnotenie

Implementácia projektu mala veľmi pozitívny vplyv v oblasti analýzy komunikácie, prebiehajúcej medzi rôznymi používateľmi a zariadeniami pripojenými v siete. Naučili sme sa dekódovať údaje o jednotlivých posielaných paketoch, ako prebieha komunikácia na transportnej vrstve, ako o sebe informujú zariadenia, tak aby vedeli kto sa kde nachádza a v neposlednom rade ako správne chápať a filtrovať rôzne typy komunikácií aj s použitím programu Wireshark.

5.2. Možnosti rozšírenia

Vďaka architektúre programu a zvolenému prístupu k filtrom vieme napríklad kedykoľvek rozšíriť náš analyzátor o další požadovaný filter. Reprezentácia komunikácii, paketov a ich údajov je pomocou python dictionary, čo znamená, že máme možnosť doplniť a zobrazovať akékoľvek ďalšie požadované informácie o týchto premenných medzi výstupmi analyzovaných komunikácii.