**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA**

Ilkovičova 2, 842 16 Bratislava 4

**2022/2023**

Počítačové a komunikačné siete

**Zadanie č.1**

**Analyzátor sieťovej komunikácia**

**Cvičiaci: Ing. Miroslav Bahleda, PhD. Vypracoval: Šimon Valíček**

**Čas cvičení: Piatok 8:00 – 9:40 AIS ID: 116320**

Šimon Valíček AIS ID: 105345

Obsah

1. **Úvod........................................................................................................................................3**
2. **Implementácia.........................................................................................................................6**
   1. *Úvod..................................................................................................................................6*
   2. *Prvá úloha.........................................................................................................................6*
   3. *Druhá úloha......................................................................................................................8*
   4. *Tretia úloha.....................................................................................................................10*
   5. *Štvrtá úloha.....................................................................................................................11*
3. **Záver......................................................................................................................................14**
   1. *Problémy pri implementácii.............................................................................................14*
   2. *Záverečné slová...............................................................................................................14*

**Analyzátor sieťovej komunikácie**

1. **Úvod**

**Zadanie:**

Navrhnite a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácie o komunikáciách. Kompletne vypracované zadanie spĺňa nasledujúce úlohy.

**1) Výpis všetkých rámcov v hexadecimálnom tvare** postupne tak, ako boli zaznamenané v súbore.

Pre každý rámec uveďte:

1. a) Poradové číslo rámca v analyzovanom súbore.
2. b) Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu. (tieto hodnoty nemusia byť rovnaké)
3. c) Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 – Raw).
4. d) Pre IEEE 802.3 s LLC uviesť aj Service Access Point (SAP) napr. STP, CDP,IPX, SAP....
5. e) Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.

Ostatné požiadavky:

1. f) Vo výpise jednotlivé **bajty rámca usporiadajte po 16 v jednom riadku**. Každý riadok je ukončený znakom nového riadku. Pre prehľadnosť výpisu je vhodné použiť neproporcionálny (monospace) font.
2. g) Výstup musí byť v **YAML.**
3. h) Riešenie tejto úlohy musí byť **prezentované na 4. cvičení.**

**2**) **Výpis IP adries a vnorených protokol na 2-4 vrstve** pre rámce Ethernet II.

Pre každý rámec pridajte nasledujúce informácie k výpisu z úlohy 1:

1. a) Vnorený protokol v hlavičke rámca. (ARP, IPv4, IPv6 .... )
2. b) Zdrojovú a cieľovú IP adresu paketu.
3. c) Pre IPv4 uviesť aj vnorený protokol. (TCP, UDP ...)
4. d) Pre 4. vrstvu, tj. vo vnútri TCP a UDP, uviesť zdrojový a cieľový port komunikácie a zároveň, ak niektorý z portov patrí medzi “známe porty”, tak uviesť aj názov aplikačného protokolu.

Ostatné požiadavky:

1. e) Čísla protokolov v rámci Ethernet II (pole Ethertype), v IP pakete (pole Protocol) a čísla portov pre transportné protokoly musia byť **načítané z jedného alebo viacerých externých textových súborov** (body a, c, d v úlohe 2).
2. f) Pre **známe protokoly a porty** (minimálne protokoly v úlohách 1) a 2) budú **uvedené aj ich názvy**. Program bude schopný uviesť k rámcu názov vnoreného protokolu aj po doplnení nového názvu k číslu protokolu, resp. portu do externého súboru.

g) Za externý súbor sa nepovažuje súbor knižnice, ktorá je vložená do programu.

**3) Na konci výpisu z úlohy 2) uveďte pre IPv4 packety nasledujúcu štatistiku:**

a) Zoznam IP adries všetkých odosielajúcich uzlov a koľko paketov odoslali.

b) IP adresu uzla, ktorý sumárne odoslal (bez ohľadu na prijímateľa) najväčší počet paketov a koľko paketov odoslal, ak ich je viac, tak uviesť všetky uzly.

**4) Váš program rozšírte o analýzu komunikácie pre vybrané protokoly:**

a) Implementujte prepínač “-p” (ako protokol), ktorý bude nasledovaný ďalším argumentom a to skratkou protokolu braného z externého súboru, napr. *analyzator.py -p HTTP. A*k prepínač nebude nasledovaný ďalší argumentom alebo zadaný argument bude neexistujúci protokol, tak program vypíše chybové hlásenie a vráti sa na začiatok. Ako alternatíva môže byť implementované menu, ale **výstup musí byť zapísaný do súboru YAML**.

**Ak je na vstupe zadaný protokol s komunikáciou so spojením (tj. nad TCP):**

1. b) Vypíšte **všetky kompletné** komunikácie aj s poradovým číslom komunikácie - obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia. Otvorenie spojenia môže nastať dvomi spôsobmi a zatvorenie štyrmi spôsobmi.
2. c) Vypíšte **prvú nekompletnú** komunikáciu, ktorá obsahuje iba otvorenie alebo iba zatvorenie spojenia.
3. d) Na vstupe musíte podporovať všetky nasledujúce protokoly so spojením: **HTTP, HTTPS, TELNET, SSH, FTP radiace, FTP dátové.**

e) Výpis každého rámca komunikácie musí spĺňať požiadavky kladené v úlohách 1 a 2 (analýza

1. (analýza L2 a L3).

**Ak je na vstupe zadaný protokol s komunikáciou bez spojenia (nad UDP):**

1. f) Pre protokol **TFTP uveďte všetky rámce a prehľadne ich uveďte v komunikáciách**, nielen prvý rámec na UDP porte 69, ale *identifikujte všetky rámce každej TFTP komunikácie a prehľadne ukážte, ktoré rámce patria do ktorej komunikácie.*
2. g) Výpis každého rámca komunikácie musí spĺňať požiadavky kladené v úlohách 1 a 2 (analýza L2 a L3).

**Ak je na vstupe zadaný protokol ICMP:**

1. h) Program identifikuje všetky rámce jednej ICMP komunikácie a bude vedieť vo výpise prehľadne ukázať, ktoré rámce patria do ktorej komunikácie. Ak identifikujete nekompletnú ICMP komunikáciu tak ju vypíšete ako nekompletnú.
2. i) Pri každom rámci ICMP uveďte aj typ ICMP správy (pole Tye v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.

**Ak je na vstupe zadaný protokol ARP:**

1. j) Vypíšte 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 ako nekompletné komunikácie.

**5) Súčasťou riešenia je aj dokumentácia:**

* 1. a) Vyžaduje sa prehľadnosť a zrozumiteľnosť odovzdanej dokumentácie ako aj kvalita spracovania celkového riešenia. Za túto časť získa plný bodový zisk študent, ktorý má v dokumentácií uvedené všetky podstatné informácie o fungovaní jeho programu vrátane diagramu spracovávania \*.pcap súborov a popis jednotlivých častí zdrojového kódu (knižnice, triedy, metódy, ...).

b) Musí **obsahovať** najmä: • Úvodnú stranu,

• Diagram (activity, flowchart) spracovávania (koncepcia) a fungovania riešenia,

• Navrhnutý mechanizmus analyzovania protokolov na jednotlivých vrstvách,

• Príklad štruktúry externých súborov pre určenie protokolov a portov,

• Opísané používateľské rozhranie,

• Voľbu implementačného prostredia,

• Zhodnotenie a prípadné možnosti rozšírenia.

1. **Implementácia**
   1. **Úvod**

Pre implementáciu tohto programu som si zvolil jazyk Python a programovacie prostredie PyCharm od spoločnosti JetBrains.

* 1. **Prvá úloha**

Cieľom úlohy je výpis všetkých rámcov v hexadecimálnom tvare. Moja implementácia vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

Na začiatku som si zadefinoval triedu **Packet**, v ktorej som zadefinoval všetky premenné, ktoré sa nachádzali v súbore **schemal-all.yaml**, ktorý bol súčasťou validátora na dokumentovom serveri. Všetky hodnoty som prednastavil na *None*a budem ich prepisovať za behu programu. Ďalej som si otvoril súbor file, ktorý predstavuje komunikácie vo Wiresharku. Pre budúcu prácu s týmto súborom vyzerá moja implementácia nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis



Na otvorenie tohto súboru som využil knižnicu **scapy.all.** Jednotlivé riadky z otvoreného súboru som následne vyťahoval pomocou for-cyklu, kde som s nimi potom pracoval. Takto prevzaté rámce som si následne upravil do hexadecimálneho tvaru, aby sa mi s ním lepšie pracovalo.

Obrázok, na ktorom je text

Automaticky generovaný popis

Výstupom je premenná frame, ktorá predstavuje rámec v hexadecimálnom tvare.

Pre každý riadok zo súbor, resp. rámec, bolo treba uviesť nasledovné:

1. **Poradové číslo rámca v analyzovanom súbore.**

Poradové číslo rámca mi určuje číslo **i** z for-cyklu, implementované o 1, nakoľko cyklus začína od nuly

1. **Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu. (tieto hodnoty nemusia byť rovnaké)**

Obrázok, na ktorom je text

Automaticky generovaný popis

Dĺžka rámca je v mojej implementácii veľkosť stringu vydeleným hodnotou 2, nakoľko jeden Byte vo *Wiresharku* predstavuje dve políčka v mojom reťazci. Vieme, že minimálna dĺžka je 64, preto ak bol rámec kratší, nastavil som mu hodnotu na 64. Následne som pričítaním 4 (predstavuje 4 Byty) získal dĺžku rámca prenášaného po médiu, nakoľko každý rámec obsahuje „4-bytovú“ pätu, ktorá sa tam nepíše.

1. **Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 – Raw).**

Pri tejto úlohe bolo nutné pozrieť sa na 12.-13. Byty rámca. Následné číslo sa prevedie do decimálneho tvaru. Ak je väčšie ako 1500, vieme, že sa jedná o Ethernet rámec. Ak je menšie ako 1500, vieme, že číslo predstavuje dĺžku dátového poľa rámca a teda sa jedná o jeden z ďalších typov rámcov. V takom prípade je nutné pozrieť na 14. Byt v reťazci a rozlíšiť, o aký typ rámca sa jedná. Hodnoty Bytov, podľa ktorých rozlišujeme sú nasledovné, kde *string* predstavuje hodnotu 14. Bytu v rámci:

Obrázok, na ktorom je text

Automaticky generovaný popis

1. **Pre IEEE 802.3 s LLC uviesť aj Service Access Point (SAP) napr. STP, CDP,IPX, SAP....**

Overím si, o aký rámec sa jedná, a následne načítavam hodnotu z externého súboru, ktorá sa zhoduje s hodnotou na pozeraných Bytoch. Obrázok, na ktorom je text

Automaticky generovaný popis

Ak sa jedná o typ rámca *LLC & SNAP,* hľadáme pid, ktorý sa nachádza na 20. – 21. Byte.

Ak sa jedná o typ rámca *LLC,* hľadáme sap, ktorý sa nachádza na 14. Byte.

1. **Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.**

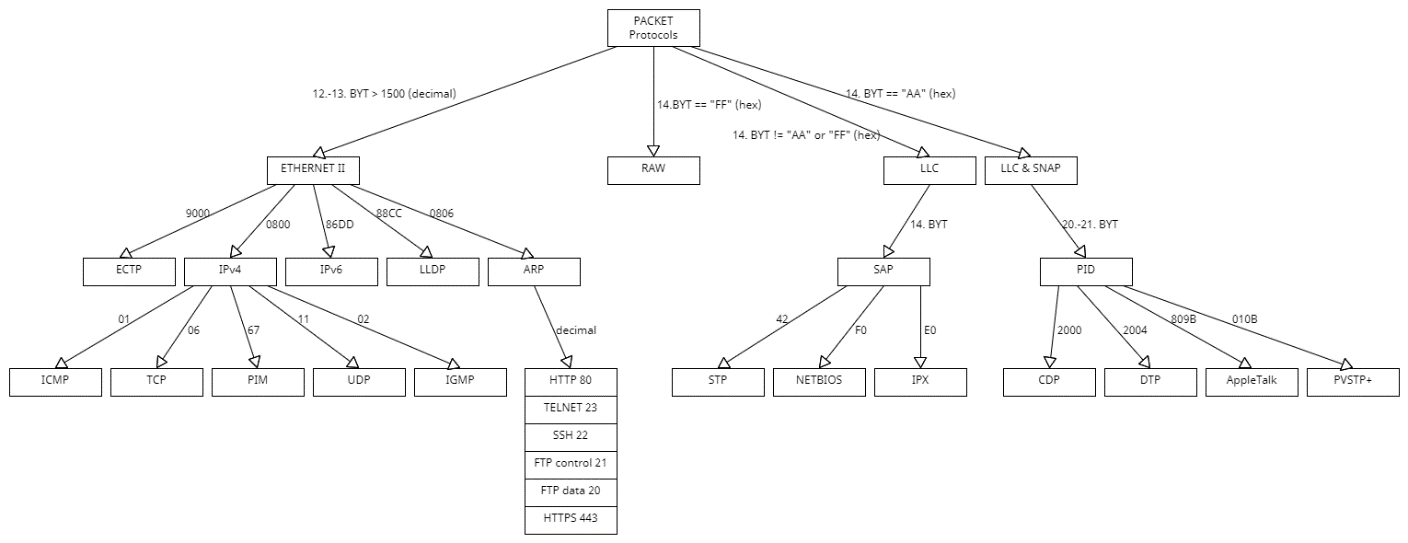
Hľadáme hodnoty na 0.-5. Byte a na 6.-11. Byte. Tieto hodnoty následne upravíme na formát, v akom sa píšu MAC adresy.

Obrázok, na ktorom je text

Automaticky generovaný popis

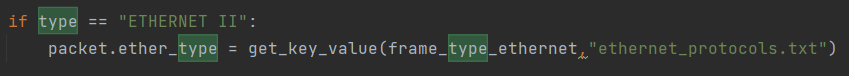
* 1. **Druhá úloha**

Zadaním druhej úlohy bolo vypísať IP adresy a ich vnorené protokoly na 2-4 vrstve pre rámce Ethernet II. Nakoľko táto úloha bola predovšetkým o identifikovaní jednotlivých protokolov, zameral som svoj diagram špeciálne na ňu. V diagrame sú zobrazené jednotlivé hodnoty, ktoré hľadáme aby sme zistili, o aký protokol sa jedná. Čísla sú uvedené v hexadecimálnom tvare, pokiaľ nie je uvedené inak, nakoľko mi to prišlo jednoduchšie. Jednotlivé čísla hľadáme na rôznych bytoch v rámci packetu, no to je popísané nižšie v tejto kapitole. Môj diagram zobrazuje iba jednotlivé názvy protokolov a hodnoty na jednotlivých bytoch, podľa ktorých sú priradené. V prípade, že by ho nebolo dobre vidno v dokumentácii, posielam ho aj v zip súbore spolu s ostatnými požadovanými dokumentami.



Následne bolo treba vypísať nasledujúce informácie:

1. **Vnorený protokol v hlavičke rámca. (ARP, IPv4, IPv6 .... )**

Vprvom rade bolo treba skontrolovať, či sa daný typ rámca rovná „Ethernet II“. Ak áno, pozeráme 12.-13. Byt v rámci a v závislosti od jeho hodnoty mu pridelíme názov vnoreného protokolu, ktorý získame z externého súboru.

1. **Zdrojovú a cieľovú IP adresu paketu.**

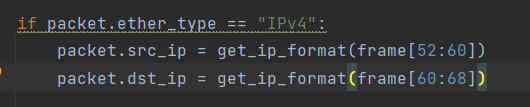
Tieto adresy sa nachádzajú na rôznych Bytoch rámca, v závislosti od toho, o aký vnorený protokol sa jedná. Ako príklad uvediem ukážku z kódu pre rámec ARP. Jedná sa o Byty 28-31 pre source ip a 38-41 pre destination IP.

Obrázok, na ktorom je text

Automaticky generovaný popis

IP adresy pre zvyšné protokoly sú zobrazené v nasledujúcich screenshotoch:

**IPv4** – source IP na 26-29 BYTE a destination IP na 30-33 BYTE



1. **Pre IPv4 uviesť aj vnorený protokol. (TCP, UDP ...)**

Takýto protokol nájdeme na 23. Byte v rámci. Jeho hexadecimálnej hodnote priradíme názov podľa externého súboru.



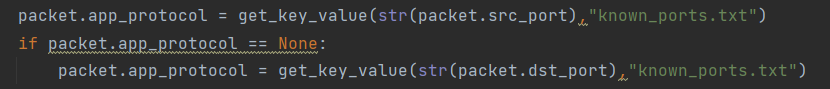
1. **Pre 4. vrstvu, tj. vo vnútri TCP a UDP, uviesť zdrojový a cieľový port komunikácie a zároveň, ak niektorý z portov patrí medzi “známe porty”, tak uviesť aj názov aplikačného protokolu.**

Najprv treba rozlíšiť, či sa jedná o porty TCP a UDP, ak áno, pozrieme Byty 34-35 pre source port a Byty 36-37 pre destination port.

Obrázok, na ktorom je text

Automaticky generovaný popis

Následne vezmeme source port, a porovnáme ho so známymi portami v externom súbore. Ak nájdeme zhodu, píšeme ho, ak nie, vezmeme destination port a spravíme s ním to isté.



1. **Čísla protokolov v rámci Ethernet II (pole Ethertype), v IP pakete (pole Protocol) a čísla portov pre transportné protokoly musia byť načítané z jedného alebo viacerých externých textových súborov (body a, c, d v úlohe 2).**

Táto úloha je splnená automaticky, nakoľko sa v mojom program nachádza viacero externých súborov, ako som spomínal v úlohách vyššie.

1. **Pre známe protokoly a porty (minimálne protokoly v úlohách 1) a 2) budú uvedené aj ich názvy. Program bude schopný uviesť k rámcu názov vnoreného protokolu aj po doplnení nového názvu k číslu protokolu, resp. portu do externého súboru.**

Túto úlohu zabezpečuje funkcia **get\_key\_value()**, ktorú som si za týmto účelom vytvoril. Vstupom je číslo v hexadecimálnom tvare a výstupom je hodnota priradená k tomuto číslu z externého súboru, pre príklad uvediem súbor **ethernet\_protocols.txt**:

Obrázok, na ktorom je text

Automaticky generovaný popis

Čísla v hexadecimálnom tvare a hodnoty k nim priraďované sú oddelené znakom &.

1. **Za externý súbor sa nepovažuje súbor knižnice, ktorá je vložená do programu.**



* 1. **Tretia úloha**

Úlohou bolo vypísať nasledujúcu štatistiku.

1. **Zoznam IP adries všetkých odosielajúcich uzlov a koľko paketov odoslali.**

Výpis z môjho programu je interpretovaný nasledovne. Obrázok, na ktorom je text

Automaticky generovaný popis

Pre prehľadnejší výpis do konzole využívam pomocné funkcie, ktoré sú pre chod programu viac-menej nepodstatné. Nakoľko som si nebol istý interpretáciou tohto zadania, celý výpis do konzole som zakomentoval a nechal to vypísané len v yaml súbore vo formáte takom, ako bolo v ukážkovom yaml súbore, ktorý bol súčasťou zip súboru s validátorom.

1. **IP adresu uzla, ktorý sumárne odoslal (bez ohľadu na prijímateľa) najväčší počet paketov a koľko paketov odoslal, ak ich je viac, tak uviesť všetky uzly.**

Opäť raz musím zhodnotiť, že nakoľko zo zadania nie je úplne jasné, čo treba spraviť, urobil som to pomocou výpisu do yaml súboru ako je to v ukážkovom súbore, ktorý spomínam vyššie.

* 1. **Štvrtá úloha**

Cieľom úlohy bolo analyzovať komunikáciu pre jednotlivé protokoly:

1. **Implementujte prepínač “-p” (ako protokol), ktorý bude nasledovaný ďalším argumentom a to skratkou protokolu braného z externého súboru, napr. *analyzator.py -p HTTP. A*k prepínač nebude nasledovaný ďalší argumentom alebo zadaný argument bude neexistujúci protokol, tak program vypíše chybové hlásenie a vráti sa na začiatok. Ako alternatíva môže byť implementované menu, ale výstup musí byť zapísaný do súboru YAML.**

Túto problematiku som riešil pomocou terminálu v konzole. Moja implementácia vyzerá nasledovne:

Obrázok, na ktorom je text

Automaticky generovaný popis

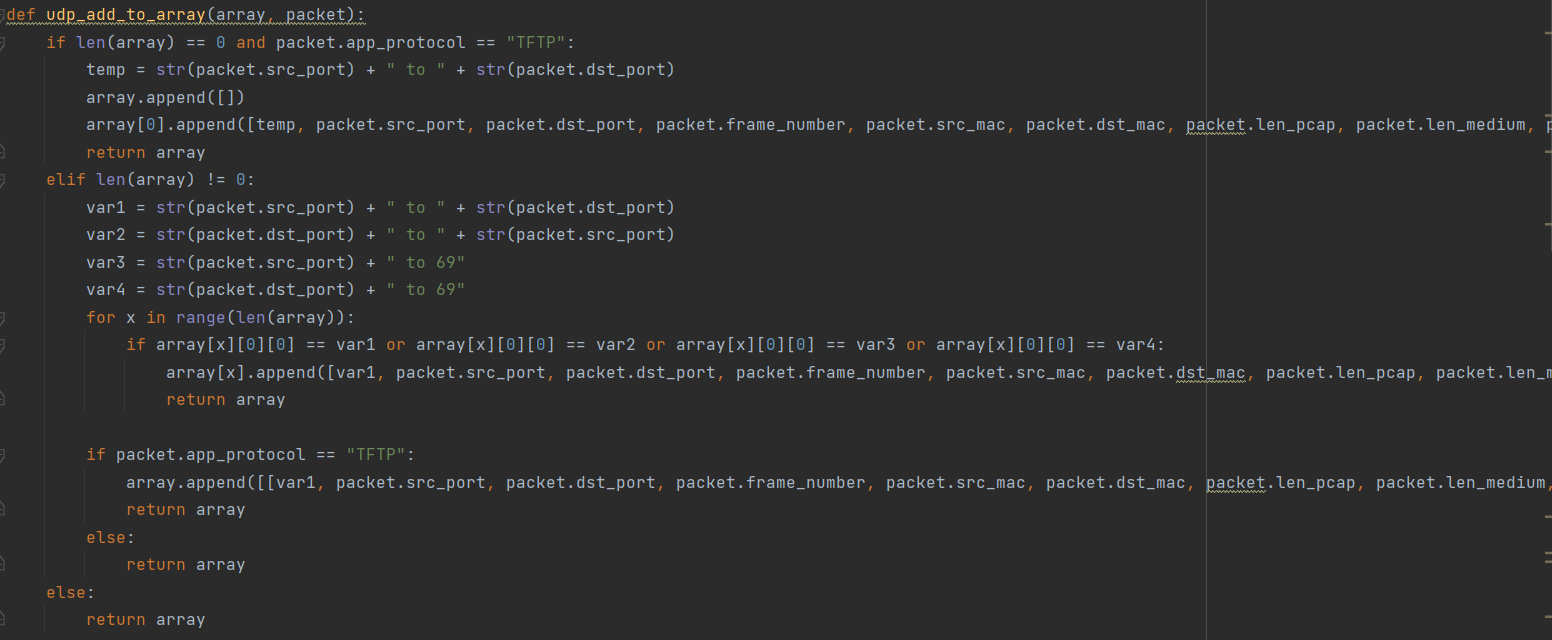
Znamená to, že môj prepínač volá filtrovacie funkcie pre jednotlivé protokoly, v závislosti od vstupu. V prípade, že do konzoly vpíšeme „**break“,** filter sa ukončí a vytvoria sa súbory, ktoré sme sa pomocou filtra snažili získať. **Ak príkaz break nezadáme, jednotlivé súbory sa nevytvoria, zároveň sa tak nevytvorí ani hlavný súbor s jednotlivými packetmi.**

V tejto podkapitole by som chcel zdôrazniť a vysvetliť princíp fungovania mojich filtrov, nakoľko sú založené na tej istej báze a menia sa len detaily. Tieto filtre som dotiahol do rôznych štádií, pričom UDP a ARP filtre by mali spĺňať podmienky kladené v zadaní, TCP filter funguje čiastočne, avšak nespĺňa podmienky v zadaní a ICMP filter som z mojej implementácie radšej vymazal, nakoľko som mu vôbec nevenoval pozornosť.

Vo for-cykle v main súbore som postupne prechádzal jednotlivé riadky a vyťahoval z nich packety, ktoré som rozparsoval a získal z nich jednotlivé informácie ako bolo treba v úlohách 1 a 2. Následne som vytvoril 4 polia – pre packety typu UDP, ARP, ICMP, TCP a ukladal som packety do týchto polí podľa toho, aký vnorený protokol mali. Následne som zavolal funkcie, ktoré sú takmer totožné pre všetky 4 polia a tie mi následne pridali packet do poľa, ku ktorému patril. Tieto polia som riešil pomocou 3D formátu, kde index **x** predstavuje číslo komunikácie, index **y** prestavuje pozíciu packetu vrámci komunikácie **x**, a keďže na indexoch **y** sú uložené polia, znamená to, že index **z**predstavuje konkrétny údaj/informáciu z packetu **y** v komunikácii **x**.

Z tejto pozície sa následne už zoradené komunikácie filtrovali vcelku jednoduchým spôsobom, napriek tomu som do úspešného konca doťiahol iba dva typy filtrov.

Nakoľko sa nebudem v ďalších podkapitolách venovať jednotlivým filtrom a ich kódom potrobne, pretože sú takmer rovnaké, uvediem jeden príklad v tejto podkapitole. Ako príklad som si zobral filter UDP.



Najprv sa pozriem, či dané pole už existuje, ak nie, vložím doňho prvý prvok v podobe potrebných údajov packetu, ktorý mi prišiel na vstupe. Ak dané pole už existuje, pozerám vo for-cykle pre prvú vrstvu pola, či sa adresy prvého prvku v komunikácii už nachádzajú. Ak áno, pridám dané informácie o packete do danej vrstvy. Ak nie, pridám novú prvú vrstvu, predstavujúcu novú komunikáciu. Takto zadefinované a implementované pole už ľahko rozdeľujem podľa požiadaviek v zadaní.

1. **Ak je na vstupe zadaný protokol s komunikáciou so spojením (tj. nad TCP):**

**Vypíšte všetky kompletné komunikácie aj s poradovým číslom komunikácie - obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia. Otvorenie spojenia môže nastať dvomi spôsobmi a zatvorenie štyrmi spôsobmi.**

**Vypíšte prvú nekompletnú komunikáciu, ktorá obsahuje iba otvorenie alebo iba zatvorenie spojenia.**

**Na vstupe musíte podporovať všetky nasledujúce protokoly so spojením: HTTP, HTTPS, TELNET, SSH, FTP radiace, FTP dátové.**

**Výpis každého rámca komunikácie musí spĺňať požiadavky kladené v úlohách 1 a 2 (analýza L2 a L3).**

Túto úlohu sa mi nepodarilo vyriešiť. Dokázal som síce vyfiltrovať jednotlivé TCP komunikácie a vypísať ich v konzole, avšak nezaimplementoval som filter pre jednotlivé protokoly. Napriek tomu, že by takýto filter bolo veľmi jednoduché spraviť, nepustil som sa do toho, nakoľko som mal problém s rozlišovaním medzi kompletnou a nekompletnou komunikáciou, a teda výstup v *yaml* súbore by nebol korektný. Pre lepšie pochopenie môjho daného problému, odporúčam zadať si pri filtrovaní „**-p tcp“** do konzoly.

1. **Ak je na vstupe zadaný protokol s komunikáciou bez spojenia (nad UDP):**

**Pre protokol TFTP uveďte všetky rámce a prehľadne ich uveďte v komunikáciách, nielen prvý rámec na UDP porte 69, ale *identifikujte všetky rámce každej TFTP komunikácie a prehľadne ukážte, ktoré rámce patria do ktorej komunikácie.***

**Výpis každého rámca komunikácie musí spĺňať požiadavky kladené v úlohách 1 a 2 (analýza L2 a L3).**

K tejto časti zadania asi nie je čo dodať, treba si len zadať do konzoly **-p udp**, následne **break** a pozrieť si výsledný súbor na výstupe.

1. **Ak je na vstupe zadaný protokol ICMP:**

**Program identifikuje všetky rámce jednej ICMP komunikácie a bude vedieť vo výpise prehľadne ukázať, ktoré rámce patria do ktorej komunikácie. Ak identifikujete nekompletnú ICMP komunikáciu tak ju vypíšete ako nekompletnú.**

**Pri každom rámci ICMP uveďte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.**

So zármutkom v srdci musím skonštatovať, že túto úlohu som vôbec neriešil.

1. **Ak je na vstupe zadaný protokol ARP:**

**Vypíšte 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 ako nekompletné komunikácie.**

Opäť ako pri UDP filtri, aj tu musím poznamenať rovnako. Najlepšie bude, ak si zadáte **-p ARP** do konzole, následne zadáte **break** a pozriete sa na výsledok v yaml súbore.

1. **Záver**
   1. **Problémy pri implementácii**

Najväčším problémom pri implementácii daného analyzátora bola jeho časová náročnosť. Nakoľko sme podobnú problematiku na iných predmetoch doposiaľ neriešili, bola to ozajstná skúška odhodlania a pevnej vôli. Dovolím si zhodnotiť, že časovú náročnosť tohto zadania som značne podcenil, čo sa odrazilo aj na výsledku, nakoľko sa mi nepodarili implementovať všetky filtre.

* 1. **Záverečné slová**

Nakoľko sa jednalo o vôbec prvú moju skúsenosť s Pythonom, ale aj so sieťovou komunikáciou, musím skonštatovať, že ma toto zadanie posunulo vpred ako budúceho programátora. Z tohto dôvodu ho hodnotím pozitívne, avšak nedá mi povedať len plus a nezohľadniť mínus, a preto by som chcel aj poukázať na fakt, že zadanie bolo v jeho priebehu niekoľkokrát upravované, čo na mňa nepôsobilo zrovna najšťastnejším dojmom.