Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Počítačové a komunikačné siete

Zadanie 1 – analýzator sieťovej komunikácie

Petra Miková

Petra Miková
ID: 120852

Obsah

Petra Miková
ID: 120852

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.
2. Výpis IP adries a vnorených protokol na 2-4 vrstve pre rámce Ethernet II.
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. Ak 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**
   b) 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.
   c) 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. Za kompletnú TFTP komunikáciu považujeme takú komunikáciu, kde veľkosť posledného datagramu s dátami je menšia ako dohodnutá veľkosť bloku pri vytvorení spojenia a zároveň odosielateľ tohto paketu prijme ACK od druhej strany.
   d) Ak je na vstupe zadaný protokol ICMP:
      - Program identifikuje všetky typy ICMP správ. Echo request a Echo reply (vrátane aj Time exceeded) rozdeľte do kompletných komunikácii na základe nasledujúceho princípu. Najprv je potrebné identifikovať dvojice IP source a IP destination, ktoré si vymieňali ICMP správy a priradiť každej dvojici ich ICMP správy. Následne, Echo request a Echo reply obsahujú v hlavičke polia Identifier a Sequence. Pole Identifier označuje číslo komunikácie v rámci dvojice IP adries a Sequence označuje poradové číslo postupnosti v rámci komunikácie. Obe polia môžu byť rovnaké pre rôzne dvojice IP source a IP destination. Z čoho vyplýva, že nová komunikácia je identifikovaná dvojicou IP adries a ICMP polom Identifier.

Všetky ostatné typy ICMP správ a ICMP správy Echo request/reply bez páru vypíše ako nekompletnú komunikáciu.

- Pri každom rámci ICMP uveďte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod. Pri kompletných komunikáciach vypíšte aj ICMP polia Identifier a Sequence.

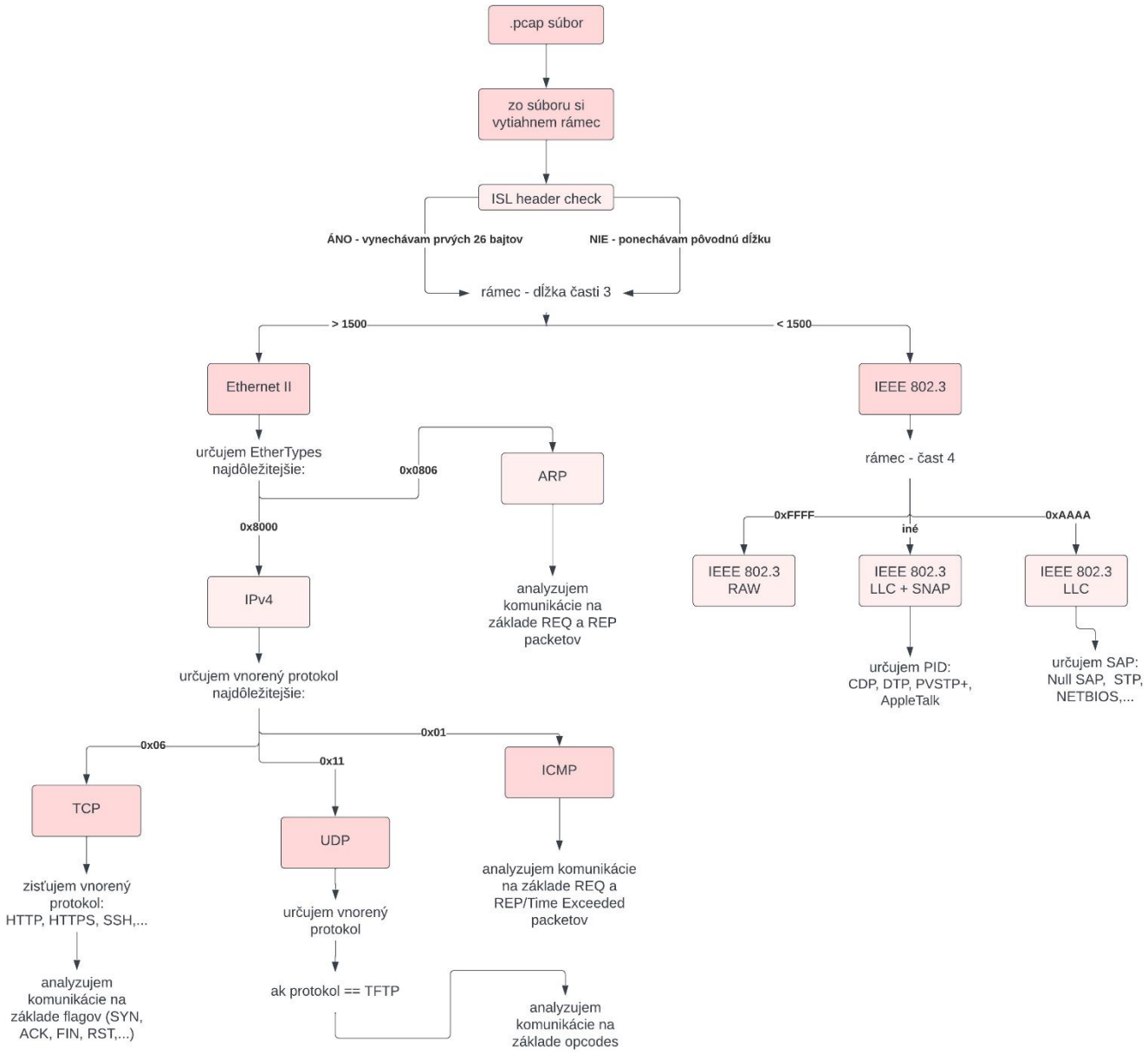
e) Ak je na vstupe zadaný protokol ARP:
   - Vypíšte všetky ARP dvojice (request -- reply), uveďte aj IP adresu, ku ktorej sa hľadá MAC (fyzická) adresa a pri ARP-Reply uveďte konkrétny pár - IP adresa a nájdená MAC adresa. V prípade, že bolo poslaných niekoľko rámcov ARP-Request na rovnakú IP adresu, najprv identifikujte všetky ARP dvojice a vypíšte ich do jednej kompletnej komunikácie bez ohľadu na zdrojovú adresu ARP-Requestu. Následne všetky ARP requesty bez ARP reply vypíšte v jednej nekompletnej komunikácii. Rovnako, ak identifikuje viac ARP reply ako ARP request správ na rovnakú IP, tak všetky ARP reply bez ARP request vypíšte v jednej nekompletnej komunikácii. Ostatné typy ARP správ ignorujeme v rámci filtra.

f) Ak je IP paket fragmentovaný:
   - Ak veľkosť IP paketu presiahne MTU, tak paket je rozdelený do niekoľko menších paketov tzv. fragmentov pred odoslaním a následne po prijatí všetkých fragmentov na strane príjemcu je opäť poskladaná celá správa. Pre ICMP filter, identifikujte všetky fragmentované IP pakety a vypíšte pre každý takýto paket všetky rámce s jeho fragmentami v správnom poradí. Pre spájanie fragmentov študujte polia Identification, Flags a Fragment Offset v IP hlavičke a uveďte ich aj pri paketoch v takej komunikácii, ktorá obsahuje fragmentované pakety ako id, flags_mf a frag_offset.

V mojom prípade sa mi podarilo splniť všetky úlohy okrem riešenia IP fragmentácie.

Petra Miková
ID: 120852

Diagram riešenia



Petra Miková
ID: 120852

Mechanizmus analyzovania protokolov na vstvách

Ako prvé určujem základné atribúty každého rámca – poradové číslo, dĺžku rámca, typ rámca. Taktiež sledujem či rámec neobsahuje ISL hlavičku:

```
if ISL_header_check_hex == "01000c0000" or ISL_header_check_hex ==
"0c000c0000":
    # Sledujeme či má rámec ISL header, ak áno posúvame sa o jeho celú
dĺžku
    data_in_bytes = data_in_bytes[26:]
```

Určovanie typu rámca:

Typ rámca určujem v samostatnej funkcii **frame_type:**

```
def frame_type(data_in_bytes):
    frame_part3_hex = data_in_bytes[12:14].hex()
    part_check_raw = data_in_bytes[14:16].hex()
    part_check_snap = data_in_bytes[14:15].hex()

    decimal_part3_length = int(frame_part3_hex, 16)

    if decimal_part3_length > 1500:
        frame_type_string = "ETHERNET II"
    else:
        if part_check_raw == "ffff":
            frame_type_string = "IEEE 802.3 RAW"
        elif part_check_snap == "aa":
            frame_type_string = "IEEE 802.3 LLC & SNAP"
        else:
            frame_type_string = "IEEE 802.3 LLC"
    return frame_type_string
```

Určujem následovne:

Ak na tretej časti rámca (bajty [12:14]) máme časť obsahujúcu hodnotu viac ako 1500, ide určite o EtherType a teda typ rámca je Ethernet II. Ak je tam hodnota menšia, ide o IEEE 802.3, a je potrebné určiť jeho typ. Pre RAW sledujeme bajty [14:16], na ktorých musí byť hodnota FFFF. Pre SNAP je to naopak hodnota AA na bajte 15. Ak nie je splnená ani jedna z týchto podmienok, ide o IEEE 802.3 LLC.

Už mimo funkcie určujem ďalšie vnorené protokoly následovne (pomocou externých súborov s vypísanými protokolmi a ich reprezentáciou v hexdumpe):

1. pre LLC SNAP určujem PID na bajtoch [20:22]
2. pre LLC určujem SAP na bajtoch [14:15]

Pri type Ethernet II určujem najprv jeho EtherType a to z bajtov [12:14] – medzi najzákladnejšie patria napr. IPv4, IPv6, ARP,… a určím pre tieto rámce aj zdrojovú a cieľovú IP adresu.

Ak mám protokol IPv4, určujem ďalší vnorený protokol z bajtu [23:24] – medzi najčastejšie patria TCP, UDP, ICMP,…

Pre TCP protokol určujem pomocou source alebo destination portu well known server porty, na základe ktorých potom filtrujem komunikácie v úlohe 4. To isté robím aj pre UDP protokol, s rozdielom že tam neskôr pri komunikáciach figuruje len jeden port a to TFTP. Porty sa nachádzajú na bajtoch [34:36] a [36:38].

Pre ICMP kontrolujeme dĺžku hlavičky IP pre korektné určenie atribútov a to ICMP typu a polí Identifier a Sequence. ICMP typ znova určíme pomocou externého súboru kde máme vypísané potrebné codes a nachádza sa na bajtoch [(14 + IHL):(14 + IHL + 1)].

Mechanizmus analyzovania komunikácii

1. TCP komunikácia so zadaným protokolom packetov
   Ako prvé si vyfiltrujem zo všetkých rámcov len tie, ktoré zodpovedajú požadovanému protokolu. Ďalej si tieto rámce "zgrupím" do komunikácii podľa toho ktoré ráamce so sebou komunikujú na základe ich IP adries a portov. Ďalej už nasleduje analýza každej komunikácie a určenie toho, či je kompletná alebo nie.
   Ak je komunikácia nad TCP kompletná, tak:
   a) otvorenie buď pomocou klasického 3-way handshaku (prvé tri packety maju určené flagy SYN, SYN-ACK, ACK) alebo 4 packety s flagmi SYN, SYN, ACK, ACK (menej časté)
   b) zatvorenie pomocou 4-way handshake (FIN, ACK, FIN, ACK / FIN-ACK, ACK, FIN-ACK, ACK) alebo FIN, FIN, ACK, ACK alebo FIN, ACK, FIN / FIN-ACK, ACK, FIN-ACK. Komunikácia taktiež môže byť uzavretá flagom RST.

   Tento prístup mi zabezpečil korekné rozpoznávanie kompletnosti daných komunikácií.

2. TFTP komunikácia
   Do funkcie pre TFTP komunikáciu si posielam už vyfiltrované len UDP packety. Princíp určovavania komunikácii nad TFTP spočíva v nájdení prvého TFTP rámca (hodnota 69) a analýzy rámcov nasledujúcich po ňom. Sledujeme výmenu rámcov s opcode hodnotami 3 (data packet) a 4 (ACK). Za behu si ukladáme dĺžky dát, keďže ukončenie spočíva v odoslaní kratšieho data packetu a prijatí ACK od druhej strany. Ak narazíme na packet ktorý je kratší, hľadáme k nemu teda ACK packet. Stále myslíme na to, že aj tu konverzácia vie byť ukončená opcodom 0005 – reset.

3. ICMP komunikácia
   Rovnako ako pri TCP komunikácií, aj tu si najprv komunikácie zgrupím podľa toho ktoré rámce so sebou komunikujú na základe ich IP adries a portov. Ďalej si v každej komunikácii vytváram dvojice REQ – REPLY/TIME EXCEEDED a zo zvyšných nespárovaných vytváram nekompletnú komunikáciu.

4. ARP komunikácia
   Z už vyfiltrovaných ARP packetov si vyfiltrujem len tie, ktoré sú buď Request alebo Reply na základe ich opcodes. Nasledovne si ich zgrupím do komunikácií bez ohľadu na zdrojovú adresu Requestu. Neskôr v daných komunikáciach hľadám páry REQ-REPLY a roztriedim ich na kompletné a nekompletné.

Všetky tieto komunikácie sú vypisované do príslušných YAML súborov.

Petra Miková
ID: 120852

Štruktúra externých súborov

Pre určovanie protokolov využívam externé .txt súbory. Toto je ukážka niektorých z nich (zdieľajú rovnaký formát):

```
# Ether Types
0x0200 XEROX PUP
0x0201 PUP Addr Trans
0x0800 IPv4
0x0801 X.75 Internet
0x0805 X.25 Level 3
0x0806 ARP
0x8035 Reverse ARP
0x809B Appletalk
0x80F3 AppleTalk AARP
0x8100 IEEE 802.1Q VLAN-tagged frames
0x8137 Novell IPX
0x86DD IPv6
0x880B PPP
0x8847 MPLS
0x8848 MPLS with upstream-assigned label
0x8863 PPPoE Discovery Stage
0x8864 PPPoE Session Stage
0x88CC LLDP
0x9000 Loopback
```

```
# ICMP codes
0 ECHO REPLY
3 Destination Unreachable
4 Source Quench
5 Redirect
8 ECHO REQUEST
9 Router Advertisement
10 Router Selection
11 Time Exceeded
12 Parameter Problem
13 Timestamp
14 Timestamp Reply
15 Information Request
16 Information Reply
17 Address Mask Request
18 Address Mask Reply
30 Traceroute
```

Tieto súbory si nasledovne v programe parsujem do listov a využívam ich na ďalšiu analýzu rámcov.

Používateľské rozhranie

Súčasťou zadania bolo implementovať buď prepínač alebo menu. Ja som teda konkrétne implementovala používateľské rozhranie formou menu (využitím inputu). Užívateľ je najprv vyzvaný zadať názov súboru, ktorý chce analyzovať. Hneď na to je privítaný názvom programu a jeho autorom a je mu zobrazený guide, ktorý obsahuje všetky možné filtre, ktoré užívateľ môže použiť. Aj bez zadania filtra sa automaticky na pozadí generuje yaml so všetkými packetmi. Po voľbe filtra je po ukončení vygenerovaný aj yaml pre konkrétny filter.

```
C:\Users\petra\AppData\Local\Programs\Python\Python310\python.exe C:/Users/petra/PycharmProjects/PKS_zadanie_1/main.py
Zadajte cestu súboru na analýzu so súborom v tvare názovsúboru.pcap:trace-12.pcap

*-------------------------------------------------------------------------*
|                    Analyzátor sieťovej komunikácie                      |
|                         Autor: Petra Miková                             |
*-------------------------------------------------------------------------*

-----------------------------------GUIDE-----------------------------------
HTTP - výpis HTTP komunikácii
HTTPS - výpis HTTPS komunikácii
TELNET - výpis TELNET komunikácii
SSH - výpis SSH komunikácii
FTP-CONTROL - výpis FTP riadiaci protokol komunikácii
FTP-DATA - výpis FTP dátový protokol komunikácii
TFTP - pre výpis TFTP komunikácii
ICMP - pre výpis ICMP komunikácii
ARP - pre výpis ARP komunikácii

Zadajte filter (skratku protokolu):HTTP

Process finished with exit code 0
```

Taktiež je pri zadaní nesprávneho protokolu vypísana chybová hláška.

Implementačné prostredie

Na implementovanie som použila Python a IDE PyCharm. Na obe veci som totižto už zvyknutá a taktiež je pre mňa Python atraktívny jednoduchou manipuláciou s poľami a taktiež obsahuje štruktúru dict, ktorú som v tomto zadaní niekoľkokrát využívala. Pri implementácií som použila aj knižnice:

Scapy – na spracovanie dát z pcap súboru pomocou príkazu rdpcap

Ruamel – na výpis do YAMLu

Zhodnotenie

Moja implementácia spĺňa skoro všetky požiadavky zadania. Dokáže z hex dumpu rámca určiť jeho základné atribúty, jeho vnorené protokoly a taktiež vyfiltrovať komunikácie na základe protokolov. Kvôli časovej tiesni sa mi nepodarilo implementovať poslednú úlohu, kde sa rozpoznávajú fragmentované IP packety. Zadanie bolo podľa mňa veľmi pracné, ale človek sa pri ňom veľa naučil a pochopil základ analýzy sieťovej komunikácie. Implementácia zadania v Pythone mi prišla príjemnejšia v porovnaní s programovaním komplexnejších zadaní v iných jazykoch napr. typu C a pod., aspoň z mojej skúsenosti. Zadanie sa určite dá rozšíriť o nejaké mikro funkcie, ale aj ešte viac komplexnú analýzu, ak by bolo treba.