SLOVENKSÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 19 Bratislava 4

PKS – 1. Zadanie: Analyzátor sieťovej komunikácie

Martin Beňa
FIIT STU
Cvičenie: Utorok 16:00
18.10.2022

Obsah

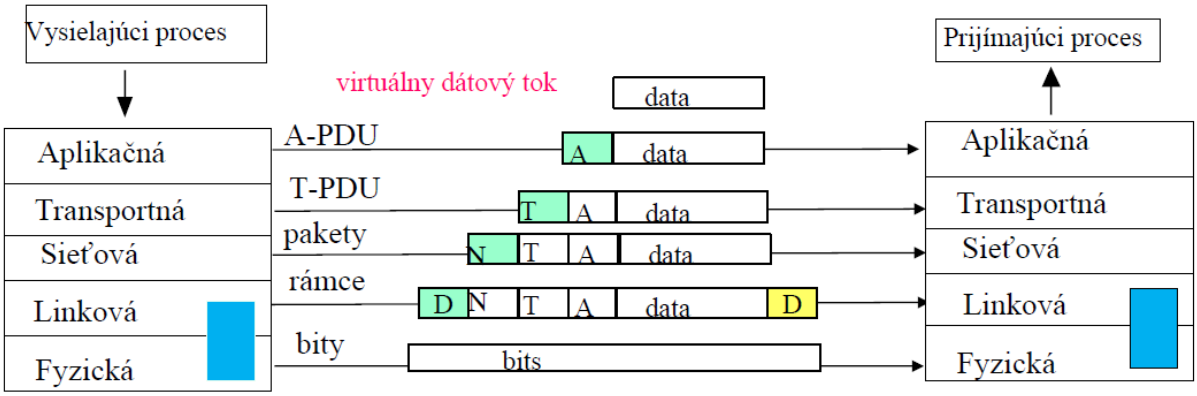
1. Zadanie

Navrhnite a implementujte program analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje informácie o komunikáciách.

2. Popis problému a postup riešenia

Popis problému

Dáta prenášané po sieti musia byt obalené takzvanými hlavičkami. Každá vrstva ma viacero protokolov ako obaliť dáta svojou hlavičkou. Ak budeme chcieť dáta zanalyzovať tak ich najprv musíme postupne „po rozbaľovať“ ale ešte pretým musíme vedieť dáta čítať.



Obr. Je prevzatý z 1.prednášky predmetu PKS na FIIT

Postup riešenia

Dáta sa načítajú postupne zo zadaného pcap zložky, v kóde sú následne rozdelené na jednotlivé rámce, ktoré sú uložené ako jedno dimenzionálne hexadecimálne pole. Na otvorenie a spracovanie pcap súborov sme použili knižnicu `scapy.all` a znej príkazy `rdpcap` a `raw`.

Funkcia `rdpcap` slúži na čítania dát zo súborov s koncovkou .pcap a funkcia slúži `raw` na vyňatie hexadecimálneho poľa, ktorý reprezentuje dáta z poslaného packetu.

Cez toto pole prechádzame for cyklom v main funkcií a následne zapisujeme informácie do predpripravenej python štruktúry dictionary (slovník). Dáta sú zabalené hlavičkami jednotlivých vrstiev. V tomto zadaní sme sa stretli s týmito vrstvami:

- **Linková:**
  - Ethernet II
    - type >= 1536 alebo je rovný 512 (XEROX PUP) alebo 513 (PUP Addr)
  - IEEE 802.3 s LLC + RAW
    - Prvé 2 byty sú ff ff
  - IEEE 802.3 s LLC + SNAP
    - Prvé 2 byty sú aa aa
  - IEEE 802.3 s LLC
- **Sieťová:**
  - Ethernet II (ETH)

        - IPv4
        - ARP – Request/Reply
        - IPv6
        - LLDP
        - ECTP

    - IEEE 802.3 s LLC + SNAP (PID)
        - PVSTP+
        - CDP
        - DTP
        - AppleTalk
    - IEEE 802.3 s LLC (PID)
        - STP
        - SNAP
        - IPX
        - NETBIOS
- **Transportná**:
    - ICMP
    - IGMP
    - PIM
    - TCP
    - UDP
    - ICMPv6
- **Aplikačná:**
    - Protokoly tejto vrstvy sa nachádzajú v súbore *Protocols\l4\ *.txt*

Niektoré protokoly majú v sebe takzvané pod protokoly, ktoré treba taktiež zapísať.

3. Popis a spustenie programu

Moje riešenie som rozložil do dvoch hlavných časti. Prvá časť sa spúšťa v každom chode programu a druhá iba ak je pri spustení zavolaný príkaz „-p [názov protokolu] *.pcap”. Po spustení ja vypíše úvodná veta, ktorá vyzýva používateľa aby zadaj korektný vstup.

Prvá časť (zadanie body 1-3)

Zaobstaráva jednoduchý výpis naformátovaných dát s prislúchajúcimi protokolmi aj ak exitujú aj pod protokolmi do jedného *yaml* súboru. Táto časť na svojom vstupe očakáva iba názov súboru odkiaľ bude dáta čítať. Okrem prečítaných dát sa vo výpise ďalej nachádza aj zoznam IP adries odosielateľov s počtom packáte, ktoré odoslali a taktiež je vypísaný IP adresa používateľa, ktorý odoslal najviac packetov.
O vykonanie prvej časti sa postará *main.py*.

Spustenie prvej časti

Program postúpení vypíše do terminálu vetu:

Pre klasicku analyzu zadaj iba nazov suboru ".pcap", pre filtorvanie komunikacie zadaj "-p [TFTP, ARP, ICMP, ...]" a nazov suboru "*.pcap":*

Kde užívateľ zadá neexistujúci súbor tak je upozornení a požiadaný o validný vstup.
Príklady validných vstupov: trace-14.pcap, eth-1.pcap, eth-4.pcap...

Po ukončení bude na konzole vypísaná hláška o úspešnom vykonaní a programu a vráti nechybový kód(0).

Druhá časť (zadanie bod 4)

Táto časť programu sa venuje filtrácií dát podľa zadaného protokolu. Tieto dáta sú následne uložené do externého súboru *yaml*. Pri zadaní protokolov ARP, ICMP a TFTP sú vypísane aj jednotlivé komunikácie aj s prislúchajúcimi packetmi.

Na túto časť zadania sa používajú python súbory *arp.py*, *icmp.py*, *rest.py* a *udp.py*.

Spustenie druhej časti

Program sa spúšťa obdobne ako pri prvej časti ale je pred súbor daný príkaz o vykonaní filtrácie.

Príklad validného vstupu: -p tftp trace-14.pcap

Ak by používateľ zadal chybový vstup, tak sa vypíše chybová hláška a program požiada opätovný zadanie.

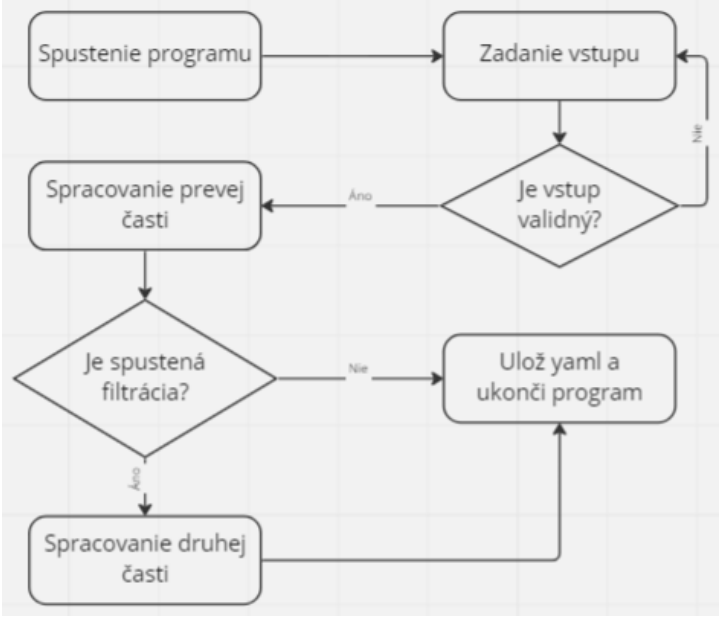
Po ukončení bude používateľ informovaný správou, ktorá ho odkáže k súboru s vyfiltrovanými dátami.

4. Postup riešenie

Po otvorení a prevedení soboru na *raw formát* som postupne prechádzal jednotlivé rámce. Vďaka dostupným materiálom z dokumentového servera som dokázal určovať čo by mali dané skupiny bajtov reprezentovať. Tak som dokázal určiť IP a MAC adresy, typy protokolov.

Pri každom narazení na IPv4 si do predpripraveného slovníka uložím alebo aktualizuje stav odoslaných packetov a potom viem určiť ich veľkosť.

V kóde sa ku každej funkcií nachádza dokumentácia, ktorá podrobne vysvetľuje chod celého programu.

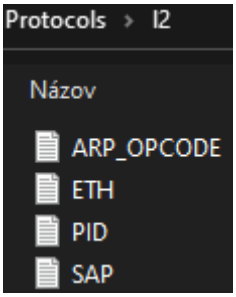


Flowchart ako funguje môj prgram

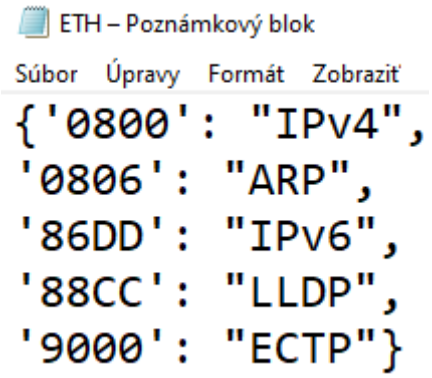
5. Externé súbory a implementačne prostredie

Na kontrolu protokolov som si vytvoril pár textových zložiek rozdelených podľa vrstvy, ktoré v sebe obsahujú vnútro python *slovníka* kde *kľúč* je hexadecimálna hodnota a *hodnota* je názov protokolu. Všetky tieto súbory sú uložené v súbore s názvom *Protocols*.

Textové súbory sú uložené do separátnych súborov podľa vrstvy na ktorej sa používajú ich názvy sú totožne s názvami v yaml výstupe.



Súbor l2 s protokolmi na sieťovej vrstve



Obsah ETH.txt zložky

Cele zadanie som programoval v jazyku Python lebo má dobrý podporu slovníkov, ktorých som v riešení pomerne často využíval a aj kvôli veľmi užitočným knižniciam na prácu s pcap a yaml súbormi.

Použité knižnice: copy, os, yaml a scapy.all

6. Záver

Táto úloha mi priblížila ako funguje posielanie paketov a taktiež som vďaka nej nadobudol nové skúsenosti.
Ďalším možným rozšírením je doimplementácia 4. body odrážky b), c), d) a e) nakoľko som sa kvôli časovej tiesni.