Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

Počítačové a komunikačné siete

Dokumentácia

Zadanie 1 – Analyzátor sieťovej komunikácie

Cvičenie: Stvrtok 18:00

Cvičiaci: Ing. Lukáš Mastiľak

Obsah:

1 Zadanie úlohy

Navrhnite a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácieo 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:

   a) Poradové číslo rámca v analyzovanom súbore.
   b) Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu. (tieto hodnoty nemusia byť rovnaké)
   c) Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE802.3 – Raw).
   d) Pre IEEE 802.3 s LLC uviesť aj Service Access Point (SAP) napr. STP, CDP,IPX, SAP....
   e) Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.

   Ostatné požiadavky:

   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.
   g) Výstup musí byť v **YAML.**
   h) Riešenie tejto úlohy musí byť **prezentované na 4. cvičení.**

   Hodnotenie: 3 body

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:

   a) Vnorený protokol v hlavičke rámca. (ARP, IPv4, IPv6 .....)
   b) Zdrojovú a cieľovú IP adresu paketu.
   c) Pre IPv4 uviesť aj vnorený protokol. (TCP, UDP ...)
   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:

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

Hodnotenie: 1 bod

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.

Pozn.: IP adresy a počet odoslaných / prijatých paketov sa musia zhodovať s IP adresami vo výpise Wireshark -> Statistics -> IPv4 Statistics -> Source and Destination Addresses.

Hodnotenie: 1,5 boda

4) Váš program **rozšírte o analýzu komunikácie** pre vybrané protokoly:
Predpríprava:
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):
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.
c) Vypíšte **prvú nekompletnú** komunikáciu, ktorá obsahuje iba otvorenie alebo iba zatvorenie spojenia.
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 L2 a L3).

Pozn.1: Otvorenie spojenia sa štandardne deje pomocou 3-way handshake, pošlú sa spolu 3 správy, ale môže nastať prípad, že sa spolu 4 správy, pre viac informácií pozrite celú kapitolu: TCP Connection Establishment Process: The "Three-Way Handshake".

Pozn.2: Zatvorenie spojenia sa deje pomocou 4-way handshake, ale môžu tam nastať dve situácie, pozri celú kapitolu: TCP Connection Termination alebo pomocou 3-way handshake. Spojenie tiež môže byť ukončené pomocou flagu RST.

Pozn.3: Paket, ktorý iniciuje začiatok procesu ukončenia spojenia, môže okrem príznaku FIN mať nastavené aj iné príznaky ako napríklad PUSH.

Hodnotenie: 2,5 body

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

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.*

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

Hodnotenie: 2 body

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

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.

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

Hodnotenie: 1 bod

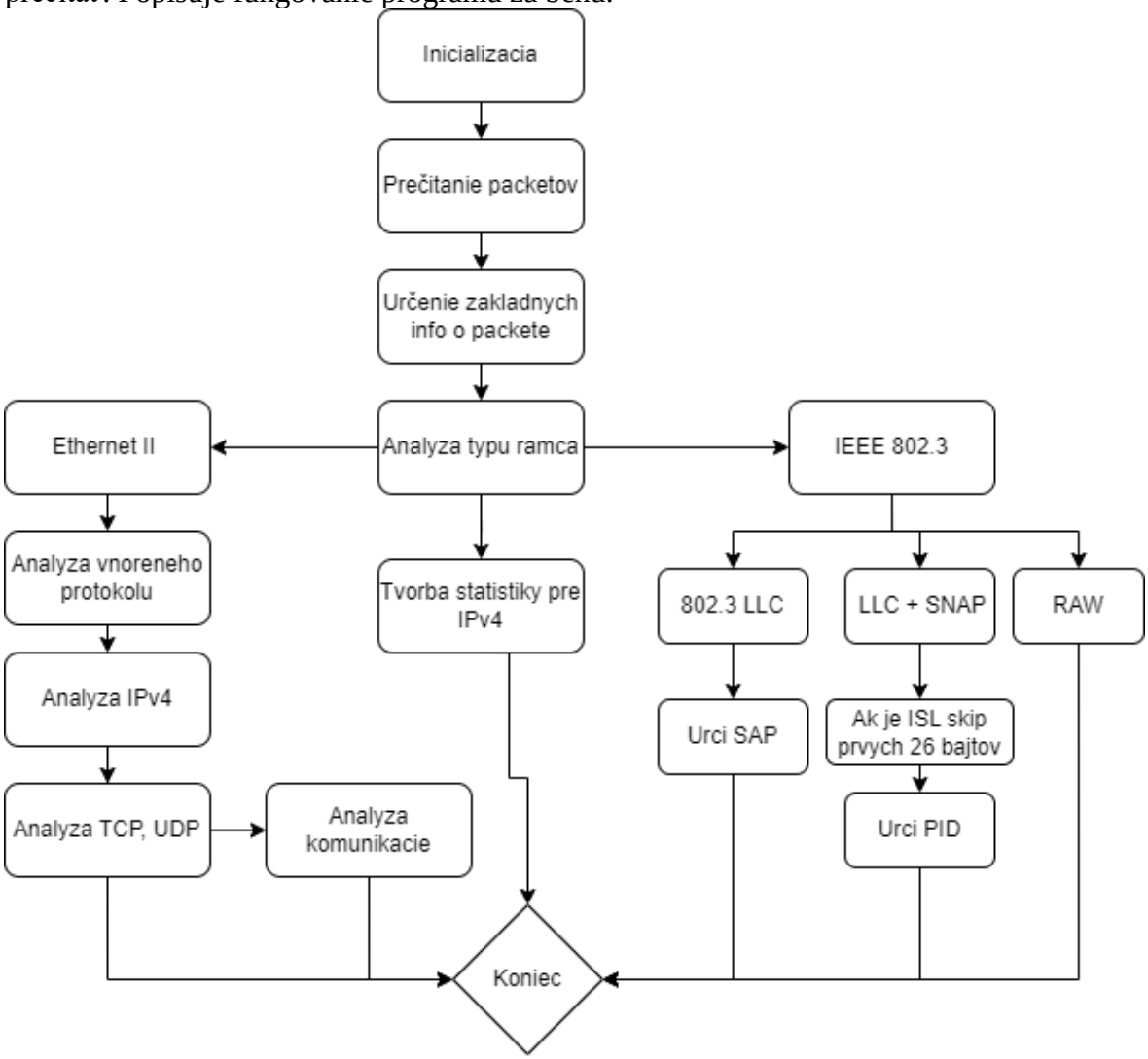
Ak je na vstupe zadaný protokol **ARP:**

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.

Hodnotenie: 1 bod

2 Blokovy návrh

Blokový návrh je urobený zjednodušene, aby nebol príliš veľký a dalo sa ho prehľadne prečítať. Popisuje fungovanie programu za behu.



3 Mechanizmus analyzovania

Analyzovanie prebieha jedným spôsobom. Podobne ako v blokovom návrhu spustíme program ktorý pomocou funkcie rdpcap načítame .pcap súbor do premennej ktorý následne analyzujeme. Tento pcap prechádzam po jednom packete kde na začiatok uvediem jeho číslo, základné údaje ako dĺžku rámca, zdrojovú a cieľovú MAC. Ďalej zisťujem typ rámca teda či ide o Ethernet II alebo IEEE 802.3. Pri 802.3 iba vypíšem podľa bajtu DSAP a SSAP o aký protokol ide čiže buď RAW, LLC, LLC+SNAP. Pri LLC a LLC+SNAP ešte určíme PID a SAP. Ethernet II analyzujeme podľa zadania takže vypíšem vnorený protokol a vypíšem ak sa jedna o IPv4 pokračujem ďalej v analýze. Kde zisťujem dĺžku IP Headera, ďalej sa pozerám na protokol ktorý vypíšem a ak sa jedna o TCP/UDP pokračujem v analýze, ak nie vypíšem zdrojovú a cieľovú IPv4. Ak je to TPC/UDP pozriem sa na zdrojový a cieľový port ktoré vypíšem a podľa nich vypíšem ešte vnorený protokol.

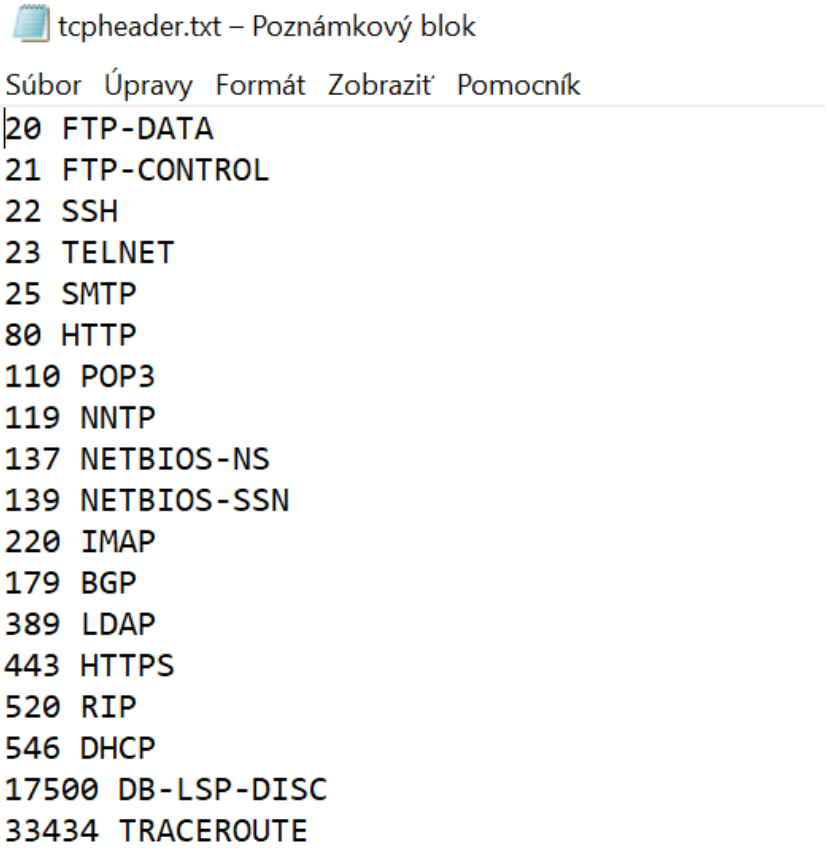
Komunikácia je analyzovaná za behu programu. Teda postupne ako prechádzam packety po jednom ak sa jedná o nejakú komunikáciu zavolá sa funkcia ktorá zistí či ide o komunikáciu. Ak áno uloží si potrebné informácie ako IP adresy ktoré komunikujú alebo porty záleží o akú komunikáciu sa jedná. Následne ak príde odpoveď alebo komunikácia prebieha uloží sa potom vždy len index packatu.

4 Externé súbory

Program používa 2 foldery s názvom packets a heads, v packets sú uložené pakety na analýzu, v heads sú externé textové súbory pre protokoly ako bolo určené podľa zadania.

Program využíva 7 externých súborov, pre uloženie čísel a názvov protokolov a portov. Všetky súbory majú rovnaký formát. Na začiatku riadku sa nachádza číslo, ktoré identifikuje daný protokol/port. Po týchto číslach sa nachádza zvyšok riadku, ktorý program načítava ako dictionary, čiže rozdelí riadok na keys a values, každému key prislúcha value, key je v podstate číslo protokolu/portu a value je text ktorý jej prislúcha.

Ukážka súboru:



5 Popis časti zdrojového kódu

Zdrojový kód je rozdelený do viacero funkcií. Ktoré nie sú dlhé ale pre lepšiu orientáciu v kóde sa rozhodol ho takto rozdeliť. Väčšina funkcií sú triviálne, taktiež každá funkcia je okomentovaná na začiatku čo robí. Preto kvôli lepšej orientácií v dokumentácií sem pridám časti zdrojového kódu ktoré môžu byť zaujímavé alebo v ktorých používam nejakú funkciu z pythonu.

```
while True:
    file_name = input("Write name of file: ")
    try:
        pcap = rdpcap('packets/' + file_name + '.pcap')
    except IOError:
        print('There was an error opening the file!')
        break
    counter = 1
```

Tu používam funkciu rdpcap z knižnice scapy pre otvorenie súboru .pcap a jeho ďalšej analýze.

```
#funkcia pre vyformatovanie mac adresy
def form_mac(packet, begin, end):
    bts = packet.__bytes__()[begin:end]
    #odstrani b''
    str_data = str(hexlify(bts))[2:-1]
    out = ""
    i = 0
    for c in str_data:
        out += c
        i += 1
        #pridanie : do mac adresy
        if i % 2 == 0:
            out += ":"
    if i % 2 == 0:
        out = out[:-1]
    return out.upper()
```

V tomto prípade sa snažím vypísať MAC. Keďže funkcia rdpcap spomínaná vyššie síce pcap otvorí ale každý je obalený v class. A výpis v tom prípade začínal b‘‘ rozhodol som sa tieto znaky odstrániť. A bajty boli reprezentované ako 0x rozhodol som sa to ošetriť funkciou hexlify ktoré to prekonvertuje iba čisto na 2- miestne hexadecimálne zobrazenie, potom ho skonvertujem na string.

Podobnú funkciu používam aj pre formátovanie hexdumpu aby to bolo správne reprezentované aj v .yaml súbore. Taktiež aj pre naformatovanie IPv4, funkcie sú len upravene podľa potreby. Z toho dôvodu ich tu nebudem pridávať aby dokumentácia nebola zbytočne dlhá.

```
with open(file_name + '.yaml', 'w') as outfile:
    yaml = ruamel.yaml.YAML()
    yaml.default_flow_style = False
    yaml.dump(yaml_file, outfile)

packet_dict["hexa_frame"] = ruamel.yaml.scalarstring.LiteralScalarString(form_hexdump(packet))
counter += 1
packety.append(packet_dict)
```

Ukladanie výstupu do .yaml súboru, pre správne formátovanie hexdumpu som použil ruamel.yaml.scalarstring.

```
len = hex(packet.__bytes__()[14])
len = BitArray(hex=len)
ip_len = len.bin[4:]
ip_len = int(ip_len, 2) * 4
```

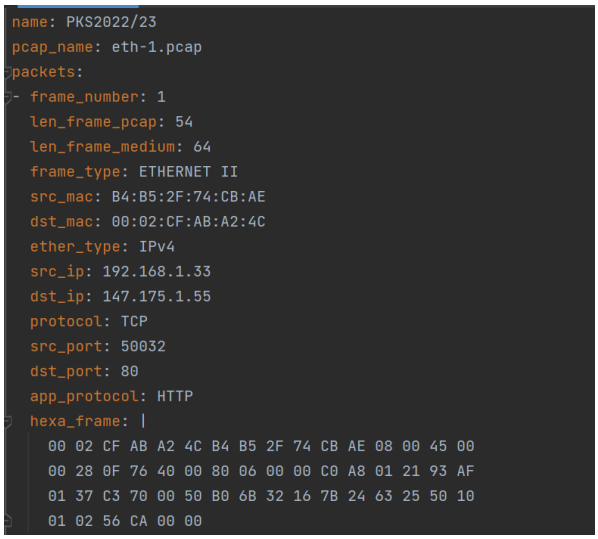
Posledná vec ktorú používam je BitArray. V tomto prípade si vytiahnem bajt ktorý reprezentuje verziu ipv4 a dĺžku IP Headera. Keďže tieto údaje sa nachádzajú v jednom bajte rozhodol som sa ho pomocou BitArray rozdeliť na bity a následne sa pozrieť na tie ktoré označujú dĺžku Headera uložiť si ich a skonvertovať na int.

Každá z funkcií je v krátkosti okomentovaná aby sa používateľ vedel rýchlo z orientovať alebo si pozrieť na čo slúži.

Všetky funkcie ktoré vyžadujú externé knižnice:
Rdpcap z **scapy.utils**
Hexlify z **binascii**
BitArray z **bitstring**
import ruamel.yaml.scalarstring

6 Vystup

Výstup programu je formovaný do 2 súborov. Jeden je v tomto prípade základná analýza packetov ktory bude niesť názov zadaného packetu + .yaml. Druhy súbor je zoznam komunikácií pre zvolený protokol.



```yaml
name: PKS2022/23
pcap_name: eth-1.pcap
packets:
- frame_number: 1
  len_frame_pcap: 54
  len_frame_medium: 64
  frame_type: ETHERNET II
  src_mac: B4:B5:2F:74:CB:AE
  dst_mac: 00:02:CF:AB:A2:4C
  ether_type: IPv4
  src_ip: 192.168.1.33
  dst_ip: 147.175.1.55
  protocol: TCP
  src_port: 50032
  dst_port: 80
  app_protocol: HTTP
  hexa_frame: |
    00 02 CF AB A2 4C B4 B5 2F 74 CB AE 08 00 45 00
    00 28 0F 76 40 00 80 06 00 00 C0 A8 01 21 93 AF
    01 37 C3 70 00 50 B0 6B 32 16 7B 24 63 25 50 10
    01 02 56 CA 00 00
```

Ukážka ako vyzerá výstup pre analýzu packetov.

```yaml
name: PKS2022/23
pcap_name: tftp.pcap
filter_name: TFTP
complete_comms:
- number_comm: 1
  src_comm: 12.0.0.1
  dst_comm: 12.0.0.5
  packets:
  - frame_number: 23
    len_frame_pcap: 70
    len_frame_medium: 74
    frame_type: ETHERNET II
    src_mac: CC:08:09:D4:00:00
    dst_mac: 02:00:4C:4F:4F:50
    ether_type: IPv4
    src_ip: 12.0.0.1
    dst_ip: 12.0.0.5
    protocol: UDP
    src_port: 49917
    dst_port: 69
    app_protocol: TFTP
    hexa_frame: |
      02 00 4C 4F 4F 50 CC 08 09 D4 00 00 08 00 45 00
      00 38 00 00 00 00 FF 11 A3 AF 0C 00 00 01 0C 00
```

Ukážka výstupu pre komunikácie, kde je rozdiel v tom že pre každú komunikáciu je vypísane len je číslo a packety v nej.

7 Voľba implementačného prostredia

Program je implementovaný v jazyku Python a ako implementačné prostredie som si vybral PyCharm ktorý mi rovno zobrazí aj .yaml súbory a teda pre kontrolu som nemusel používať niečo iné.

8 Používateľské rozhranie

Používateľské rozhranie je implementované spôsobom, že používateľ ktorý spusti program najskôr zadá názov .pcap súboru ktorý chce analyzovať a následne na to mu poskytne možnosti ktoré môže analyzovať.

```
Zadaj nazov suboru: eth-8
```

Zadávanie názvu súbora ktorý chceme analyzovať

```
1 - pre analyzu packetov
2 - pre analyzu packetov a komunikacie (TFTP)
3 - ukonci program
```

Ponuka možnosti pre analýzu súboru. 1 je len analýza packetov a následný zápis do .yaml súboru. Pri možnosti 2 spraví sa udeje to isté no taktiež je vypísaná aj komunikácia TFTP do samostatného .yaml súboru. Pre možnosť 3 sa program len ukončí.

Program funguje v cykle teda nie je potrebné ho spúšťať zakaždým nanovo. Používateľ vždy zadá len názov súboru a jednu z možností, následne na to môže zadať iný súbor a taktiež inú možnosť.

9 Zhodnotenie a prípadne možnosti rozširenia

Zadanie nám rozšírilo vedomosti v rámci práce s programovacím jazykom Python. Taktiež sme sa naučili pracovať s programom Wireshark ktorý dokáže komunikáciu zaznamenávať a následne do nej nahliadnuť, rovnako sme ho používali aj pre kontrolu či naše všetky údaje sedia.

Keďže sa nám nepodarilo stihnúť implementovať všetky body zadania, dalo by sa rozšíriť presne o tieto body. Ďalšou možnosťou rozšírenia programu by bolo grafické rozhranie kde pri otvorení by si vedel používateľ zobraziť všetky informácie o packete prípadne komunikácií.

Celkovo bola implementácia zadania pre nás prínosom, nahliadli sme lepšie na to ako taký packet vyzerá a taktiež ako rôzne komunikácie prebiehajú a aké pravidlá používajú.