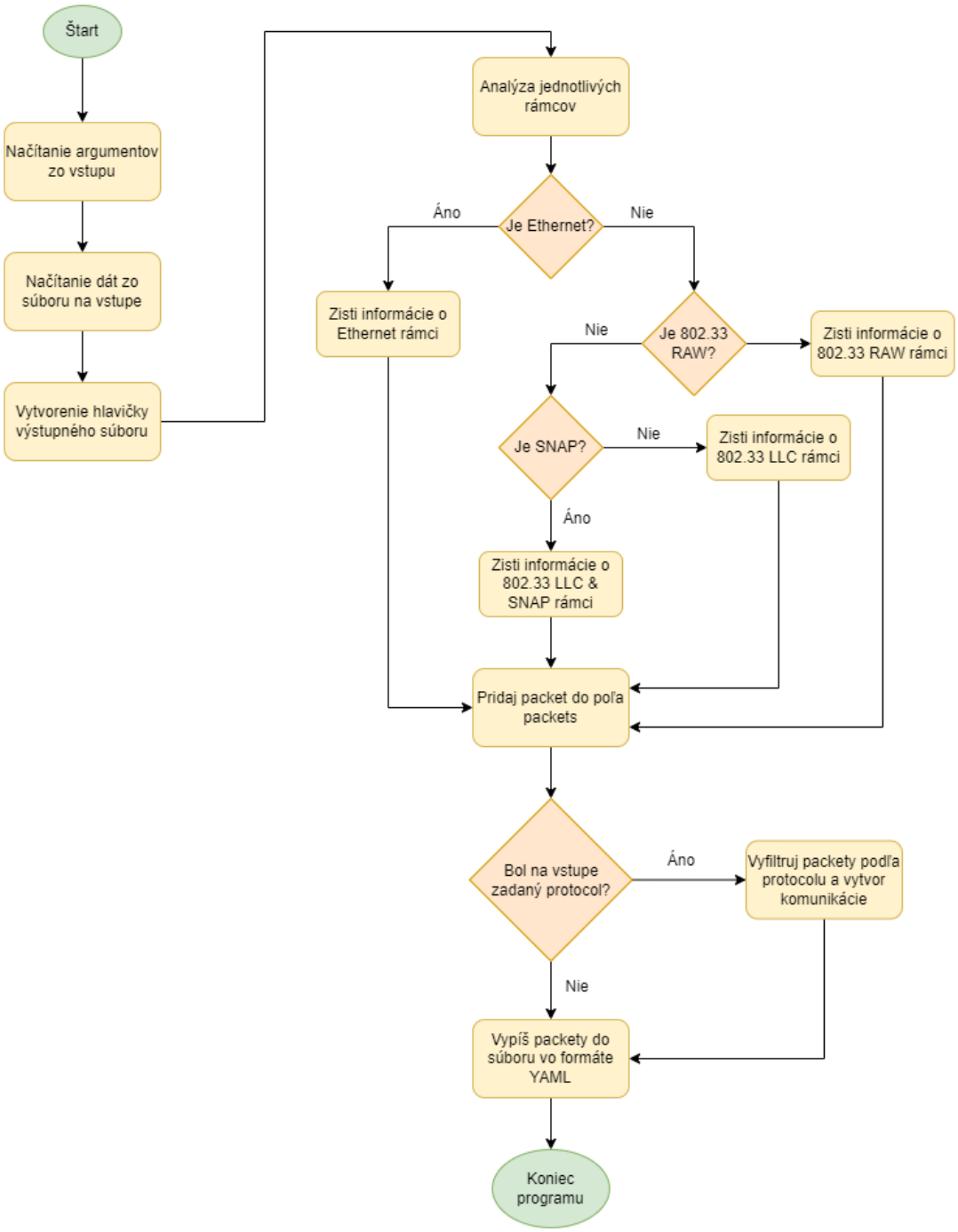
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKÁ TECHNICKÁ UNIVERZITA

Počítačové a komunikačné siete

Zadanie č.1

Meno: René Bukovina
Ročník: 3.
Cvičenie: utorok 18:00
Cvičiaci: Ing. Kristián Košťál, PhD.

Flowchart


Štart
Načítanie argumentov zo vstupu
Načítanie dát zo súboru na vstupe
Vytvorenie hlavičky výstupného súboru
Analýza jednotlivých rámcov
Je Ethernet?
Áno
Nie
Zisti informácie o Ethernet rámci
Je 802.33 RAW?
Nie
Zisti informácie o 802.33 RAW rámci
Je SNAP?
Nie
Zisti informácie o 802.33 LLC rámci
Áno
Zisti informácie o 802.33 LLC & SNAP rámci
Pridaj packet do poľa packets
Bol na vstupe zadaný protocol?
Áno
Vyfiltruj packety podľa protocolu a vytvor komunikácie
Nie
Vypíš packety do súboru vo formáte YAML
Koniec programu

Mechanizmus analýzy jednotlivých vrstiev

Typ rámca

1. Skontrolujeme či hodnota na 13. a 14. bajte je väčšia ako 1500. Ak áno, tak pokračujeme na analýzu rámca typu ETHERNET II. Ak nie tak ideme na krok 2.
2. Ak máme v rámci na 15. a 16. bajte hexadecimálnu hodnotu ‚FFFF', tak sa jedná od rámec typu Novell 802.33 RAW, a následne pokračujeme na analýzu rámca tohto typu. Ináč ideme na krok 3.
3. Ak sa v rámci na 15. a 16. bajte nachádza hexadecimálna hodnota ‚AAAA', tak vieme, že sa jedná o rámec typu IEEE 802.33 LLC & SNAP, a môžem pokračovať analýzou tohto typu rámca. Ak sa ale hodnoty líšia od danej hodnoty, tak predpokladáme, že sa jedná o rámec typu IEEE 802.33 LCC, preto budeme riešiť analýzu tohto typu.

```
4.  def getFrameType(hex):
5.      if(int(hex[12]+hex[13],16) >= 1500):
6.          return 'ETHERNET II'
7.
8.      if((hex[14]+hex[15]) == 'ffff'):
9.          return 'IEEE 802.3 RAW'
10.
11.     if ((hex[14]+hex[15]) == 'aaaa'):
12.         return 'IEEE 802.3 LLC & SNAP'
13.     else:
14.         return 'IEEE 802.3 LLC'
```

Analýza 2. vrstvy

Informácia o type protokolu sa nachádza na 13. a 14. bajte pri rámcoch typu Ethernet II, tieto dva bajty obsahujú 4-miestny hexadecimálny reťazec, ktorý označuje daný protokol na druhej vrstve. Možné typy protokolov na danej vrstve sa nachádzajú v súbore *ethernet-protocols.json*

```
    # analyzing ethernet II frames
     if 'ETHERNET II' in pkt['frame_type']:
         #find ether_type on 13th and 14th byte
         pkt['ether_type'] = getEtherType(hexDecoded[12]+hexDecoded[13])
```

Pokiaľ sa hodnota nerovná žiadnej v databáze, program skončí chybovou hláškou.

Analýza pokračuje iba pre rámce s protokolom ARP alebo IPv4.

```
if 'IPv4' in pkt['ether_type']:
                ihl = int(hexDecoded[14][1],16)
                pkt['src_ip'] = getIp(hexDecoded[26:30])
                pkt['dst_ip'] = getIp(hexDecoded[30:34])
                pkt['protocol'] = getIpProtocol(int(hexDecoded[23],16))
                #find source ports with ihl to know we need to look for them
                pkt['src_port'] =
int(hexDecoded[14+ihl*4]+hexDecoded[14+ihl*4+1],16)
                pkt['dst_port'] =
int(hexDecoded[14+ihl*4+2]+hexDecoded[14+ihl*4+3],16)
                #check to see if one of found ports is known port to protocol
IPv4 protocol
                if type(getAppProtocol(pkt['src_port'],pkt['protocol'])) ==
str or  type(getAppProtocol(pkt['dst_port'],pkt['protocol']))== str:
                    pkt['app_protocol'] =
getAppProtocol(pkt['src_port'],pkt['protocol'])

if type(getAppProtocol(pkt['src_port'],pkt['protocol'])) == str else
getAppProtocol(pkt['dst_port'],pkt['protocol'])
                #ipv4_senders counter
                if pkt['src_ip'] in ip_send:
                    ip_send[pkt['src_ip']] += 1
                else:
                    ip_send[pkt['src_ip']] = 1

            elif 'ARP' in pkt['ether_type']:
                pkt['arp_opcode'] =
getArpOp(int(hexDecoded[20]+hexDecoded[21],16))
                pkt['src_ip'] = getIp(hexDecoded[28:32])
                pkt['dst_ip'] = getIp(hexDecoded[38:42])
```

Za povšimnutie stojí fakt, že *hexDecoded* je náš vstupný reťazec rozdelený po bajtoch do poľa, kde jednotlivé indexy pre dané pole majú hodnotu *bajtu v rámci – 1*. Práve *-1*, kvôli tomu, že sa jedná o pole a to začína indexom *0*.

Analýza 3. vrstvy

Analýzu vnoreného protokolu na 3. vrstve vykonávame iba pre rámce s protokolom IPv4, nakoľko protokol ARP nedisponuje touto vlastnosťou. Databáza pre tieto protokoly sa nachádza v súbore *ipv4-protocols.json*

```
 pkt['protocol'] = getIpProtocol(int(hexDecoded[23],16))

 def getIpProtocol(dec):
    file = open("./protocols/ipv4-protocols.json","r")
    protocols = json.load(file)

    return protocols[str(dec)]
```

Analýza 4. vrstvy

Analýzu 4. vrstvy vykonávame iba pre protokoly TCP a UDP. Ak port nepatrí žiadnemu známemu protokolu, tak názov vo výpise neuvádzame. Databázy sa nachádzajú v súboroch *udp-protocols.json* a *tcp-protocols.json*

```
def getAppProtocol(dec,protocol):

   if 'UDP' in protocol:
       udp = open("./protocols/udp-protocols.json")
       udp_ports = json.load(udp)
       try:
           return udp_ports[str(dec)]
       except:
           return False
   elif 'TCP' in protocol:
       tcp = open("./protocols/tcp-protocols.json")
       tcp_ports= json.load(tcp)
       try:
           return tcp_ports[str(dec)]
       except:
           return False
   else:
       return False
```

Mechanizmus analýzy komunikácií

V mojej implementácii sa nachádza možnosť analyzovať komunikácie typu ARP a typu TFTP.

```
def analyzeArp(packets):
    complete_comms = []
    incomplete_comms = []
    arp_packets = list(filter(lambda packet: packet['ether_type'] == 'ARP',
packets))
    #get only ARP packets
    for packet in arp_packets:
        comm = {}
        pair = {}
        requests = []
        for item in arp_packets:
            if(packet['src_ip'] != item['src_ip'] and packet['dst_ip'] ==
item['dst_ip'] and packet['dst_mac'] == item['dst_mac'] and item['arp_opcode']
== 'REQUEST'):
                requests.append(item)

            if( item['src_ip'] == packet['dst_ip'] and packet['src_ip'] ==
item['dst_ip'] and packet['src_mac'] == item['dst_mac'] and item['arp_opcode']
== 'REPLY'):
                pair = item
                break;
        # if we found reply, it means there is no pair for it so it goes to
incomplete
        if packet['arp_opcode'] == 'REPLY' or not pair:
            comm['number_comm'] = len(incomplete_comms) +1
            comm['packets'] = []
            comm['packets'].append(packet)
            incomplete_comms.append(comm)
        else:
                comm['number_comm'] = len(incomplete_comms) +1
                comm['src_comm'] = packet['src_ip']
                comm['dst_comm'] = packet['dst_ip']
                comm['packets'] = []
                comm['packets'].append(packet)
                for req in requests:
                    comm['packets'].append(req)
                    arp_packets.remove(req)
                comm['packets'].append(pair)
                complete_comms.append(comm)

                #remove packets from list of packets to prevent using them
again by mistake
                arp_packets.remove(pair)
                arp_packets.remove(packet)

    data = {
        'complete_comms': complete_comms,
        'incomplete_comms': incomplete_comms
    }
```

```
    return data
```

```
def analyzeTftp(packets):
    comms = []
    #get only UDP packets
    udp_packets = list(filter(lambda packet: packet['ether_type'] == 'IPv4'
and packet['protocol'] == 'UDP', packets))

    for packet in udp_packets:
        comm={}
        # find packet with port of TFTP com. start point
        if packet['dst_port'] == 69:
            #format new communication header
            comm['number_comm'] = len(comms) +1
            comm['src_comm'] = packet['src_ip']
            comm['dst_comm'] = packet['dst_ip']
            comm['packets'] = []
            comm['packets'].append(packet)
            udp_packets.remove(packet)

            for pair in udp_packets:
                #find packet responding to first packets source IP
                if pair['dst_port'] == packet['src_port'] and packet['src_ip']
== pair['dst_ip'] and packet['dst_ip'] == pair['src_ip']:
                    comm['packets'].append(pair)
                    udp_packets.remove(pair)
                    flip = False
                    #cycle the rest of the packets to find all remaining
packets of communication when we know the source and dest ports
                    for rest in udp_packets:
                        #if statements to handle if we looking for req or
respond
                        if (not flip and pair['dst_port'] == rest['src_port']
and rest['src_ip'] == pair['dst_ip'] and rest['dst_ip'] == pair['src_ip']):
                            comm['packets'].append(rest)
                            udp_packets.remove(rest)
                            flip = True

                        if(flip and pair['src_port'] == rest['src_port'] and
rest['src_ip'] == pair['src_ip'] and rest['dst_ip'] == pair['dst_ip']):
                            comm['packets'].append(rest)
                            udp_packets.remove(rest)
                            flip = False

        if comm:
            comms.append(comm)

    data ={
        'comms': comms
    }
    return data
```

Výstup programu

Výstup programu je naformátovaný do súboru typu YAML. Na dosiahnutie tejto funkcionality bola použitá externá knižnica *ruamel.yaml*.

```
f = open("pks-output.yaml", "w")
yaml = ruamel.yaml.YAML()
yaml.default_flow_style = False
yaml.indent(mapping=2, sequence=4, offset=2)
yaml.dump(data, f)
```

Štruktúra udržiavania externých dát

Ako štruktúru som použil objekty vo formáte JSON, ktoré sú v tvare:

```
{
  ...
  "hexadecimálna hodnota / decimálna hodnota" : "názov protocolu/portu k danej hodnote"
  ...
}
```

Pri načítaní viem tieto dáta preformátovať na dátový typ dictionary v jazyku python a tak ihneď pristupovať k daným hodnotám.

Popis používateľského rozhrania

Program je ovládateľný prepínačmi -d/--destination a -p/--protocol

V termináli:

```
python main.py -p <názov protokolu, pre analýzu komunikácií> -d <cesta k súboru *.pcap s rámcami>
```

Ani jeden zo spomínaných prepínačov nie je nutné zadať, nakoľko ak nie je na vstupe zadný žiaden protokol cez prepínač *-p*, tak program vypíše všetky analyzované rámce, bez komunikácií a ak nezadáme argument pomocou prepínača *-d*, tak program použije defaultnú cestu a to *./vzorky/eth-1.pcap*

Implementačné prostredie

Pre riešenie svojho zadania som si vybral programovanie prostredie jazyka Python, na koľko je tento jazyk veľmi jednoduchý zo stránky syntaxe a zároveň veľmi schopný na procesovanie komplexných dát vďaka veľkému množstvu vstavaných modulov a možnosťou rýchlo a efektívne rozšírovať prostredie pomocou externých modulov. Vo svojej implementácií som použil z externých knižníc knižnicu menom Scapy, z ktorej som použil funkciu *rdpcap(),* ktorá slúži na čítanie rámcov zo súborov typu *.pcap* a knižnicu Binascii, ktorej úlohou je preložiť dáta načítané zo vstupného súboru na hexadecimálny reťazec, pomocou funkcie *hexlify()*. Ďalšou výhodou prostredia je fakt, že Python

nie je nízko-úrovňový programovací jazyk, ako napr. jazyk C/C++, čo by nám pri spracovaní zadania, vytváralo nadbytočnú prácu s pamäťou.

Zhodnotenie

Vytvorený program spĺňa očakávané účely a funguje správne. Možnosť rozšírenia programu je pomerne jednoduchá nakoľko je program naprogramovaný dynamickými funkciami no spolieha sa na presne definovanie označenia niektorých protokolov v podmienkach. Možnosť upravovať databázy nemení funkcionalitu programu pokiaľ budú zanechané správne označenia protokolov.