

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Stefan Otto Novak, ID: 106078

ZADANIE 1:
ANALYZÁTOR SIEŤOVEJ KOMUNIKÁCIE
Počítačové a komunikačné siete

Študijný program: Informatika 4
Prednášajúci: prof. Ing. Ivan Kotuliak, PhD.
Cvičiaci: Ing. Lukáš Mastiľak
október 2021

Obsah:

1. Zadanie úlohy	3
2. Blokový návrh (konceptia) fungovania riešenia	4
3. Navrhnutý mechanizmus analyzovania protokolov na jednotlivých vrstvách	4
a) Body zadania 1-3.....	4
b) Bod zadania 4	7
4. Príklad štruktúry externých súborov pre určenie protokolov a portov	9
5. Opísanie používateľského rozhrania	10
6. Voľbu implementačného prostredia	10

1 Zadanie úlohy

Navrhните a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácie o komunikáciách. Vypracované zadanie musí spĺňať nasledujúce body:

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.
- c) Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 – Raw).
- d) Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.

Vo výpise jednotlivé **bajty rámca usporiadajte po 16 alebo 32 v jednom riadku**. Pre prehľadnosť výpisu je vhodné použiť neproporcionálny (monospace) font.

2) Pre rámce typu **Ethernet II a IEEE 802.3 vypíšte vnorený protokol**. Študent musí vedieť vysvetliť, aké informácie sú uvedené v jednotlivých rámcoch Ethernet II, t.j. vnáranie protokolov ako aj ozrejmiť dĺžky týchto rámcov.

3) Analýzu cez vrstvy vykonajte pre rámce Ethernet II a protokoly rodiny TCP/IPv4:

Na konci výpisu z bodu 1) uveďte pre IPv4 pakety:

- a) Zoznam IP adries všetkých odosielaajúcich uzlov,
- b) IP adresu uzla, ktorý sumárne odoslal (bez ohľadu na prijímateľa) najväčší počet paketov a koľko paketov odoslal (berte do úvahy iba IPv4 pakety).

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

4) V danom súbore analyzujte komunikácie pre zadané protokoly:

- a) HTTP
- b) HTTPS
- c) TELNET
- d) SSH
- e) FTP riadiace

- f) FTP dátové
- g) TFTP, **uvedte všetky rámce komunikácie**, nielen prvý rámec na UDP port 69
- h) ICMP, uvedte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.
- i) **Všetky ARP dvojice (request – reply)**, uvedte aj IP adresu, ku ktorej sa hľadá MAC (fyzická) adresa a pri ARP-Reply uvedte 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.

Vo všetkých výpisoch treba uviesť aj IP adresy a pri transportných protokoloch TCP a UDP aj porty komunikujúcich uzlov.

V prípadoch komunikácií so spojením vypíšte iba jednu kompletnú komunikáciu - obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia a aj prvú nekompletnú komunikáciu, ktorá obsahuje iba otvorenie spojenia. Pri výpisoch vyznačte, ktorá komunikácia je kompletná.

Ak počet rámcov komunikácie niektorého z protokolov z bodu 4 je väčší ako 20, vypíšte iba 10 prvých a 10 posledných rámcov tejto komunikácie. **(Pozor: toto sa nevzťahuje na bod 1, program musí byť schopný vypísať všetky rámce zo súboru podľa bodu 1.)** Pri všetkých výpisoch musí byť poradové číslo rámca zhodné s číslom rámca v analyzovanom súbore.

2 Blokový návrh (konceptia) fungovania riešenia

Zadanie sa zameriava na vytvorenie programu pre analyzovanie Ethernet sietí. Zvolil som programovací jazyk Python, a vytvoril som program, ktorý postupne číta rámce z *.pcap* súborov a postupne ich analyzuje po jednotlivých bajtoch. Pre načítanie súborov program používa knižnicu *scapy* a priebežne rozdelí obdržanú informáciu do rámcov a analyzuje ich podľa zadania. Program číta *.pcap* súbory z priečinku *subory_na_analyzu*. Do budúcej aktualizácie programu, v priečinku musí byť iba jeden *.pcap* súbor.

3 Navrhnutý mechanizmus analyzovania protokolov na jednotlivých vrstvách

- a) Body zadania 1-3

Po úspešnom načítaní rámcov, všetky sú detailne analyzované po spustení funkcie *analyze_files()*. Ako prvé, program vypíše **poradové číslo rámca** pomocou premennej *index_frame*, ktorá vždy pri začatí analyzovania nového rámca je inkrementálna a takým sobom stále získa novú hodnotu. Všetky výpisy rámcov v tomto bode (a) sú vypísané do súboru *vypis.txt*.

Následne, program vypíše **dĺžku rámca poskytnutú pcap API**, ako aj dĺžku tohto rámca **prenášaného po médiu**. Dĺžku rámca v bajtoch poskytnutú pcap API je získaná pomocou príkazu *len()*, ktorá vráti presnú hodnotu žiadanej dĺžky, čo reprezentuje dĺžku v bajtoch. Dĺžka tohto rámca prenášaného po médiu je získaná taktiež pomocou príkazu *len()*, ku ktorému sa pripočítajú

4 bajty. Dĺžka rámca prenášaného po médiu je vždy minimálne 64 bajtov, čo znamená, že hore získaná hodnota musí byť porovnaná so hodnotou 64, a bude vypísaná hodnota, ktorá je väčšia.

Typ rámca vypisuje funkcia *write_type_of_frame()*, ktorá dostáva ako parameter rámec v hexadecimálnej forme a vráti vnorený protokol.

```
def write_type_of_frame(hex_packet):
    str1 = hex_packet[24:28]
    str2 = hex_packet[28:30]
    vnoreny_protokol = ""

    if int(str1, 16) > 1500:
        FILE_VYPIS.write("Ethernet II\n")
        vnoreny_protokol = find_ether_type(hex_packet)

    elif str2.decode() == "ff" or str2.decode() == "FF":
        FILE_VYPIS.write("IEEE 802.3 - Raw\n")
        vnoreny_protokol = "IPX"

    elif str2.decode() == "aa" or str2.decode() == "AA":
        FILE_VYPIS.write("IEEE 802.3 s LLC a SNAP\n")
        vnoreny_protokol = find_lsap_type(hex_packet)

    else:
        FILE_VYPIS.write("IEEE 802.3 LLC\n")
        vnoreny_protokol = find_lsap_type(hex_packet)

    return vnoreny_protokol
```

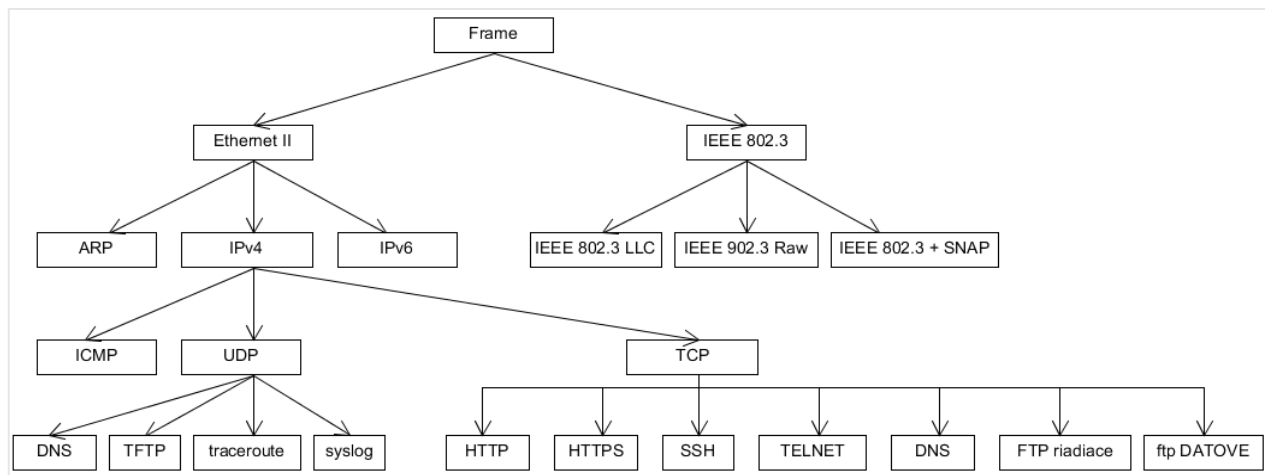
Ak hodnota prevedená decimálnej forme na bitoch od 24 až po 28 (premenná *str1*) je väčšia než 1500, rámec je typu Ethernet II a je zavolaná funkcia *find_ether_type()*, ktorá prijíma ako parameter rámec v hexadecimálnej forme a vracia Ethernet II typ vnorený protokol.

```
def find_ether_type(hex_packet):
    global ries_kom
    str1 = hex_packet[24:28]
    index_dictionary = int(str1.decode(), 16)

    if ETHER_types.__contains__(index_dictionary):
        if ETHER_types[index_dictionary] == "IPv4" and ries_kom == 0:
            add_source_IPv4_adress_to_list(hex_packet)
        return ETHER_types[index_dictionary]
    else:
        return "Tento EtherType nie je uvedený v databaze"
```

Ak hodnota je menšia než 1500, je typu IEEE 802.3 a sa prechádza do ďalšej verifikácie. Ak *str1* hodnota je menšia než 1500 a hodnota na bitoch rámca na bitoch 28 a 29 (premenná *str2*) je *ff*, tak je vždy vnorený protokol bude IPX, ak hodnota je *aa*, je vnorený protokol IEEE 802.3 s LLC a SNAP a sa zavolá funkcia *find_lsap_type()*, ktorá prijíma ako parameter rámec v hexadecimálnej forma a vracia LSAP vnorený protokol. Ak hodnota nie je *ff*, a nie je ani *aa*, je typu IEEE 802.3 LLC a rovnako sa zavolaná funkcia *find_lsap_type()*.

Tieto LSAP porty, taktiež ako Ethernet, IPv4, TPC a UDP, sú čítané na začiatku programu pomocou funkcie *proctocl_initialization()* zo súboru *protokoly.txt* a sú zapísané do špecifických globálnych slovníkov pre uľahčenie hľadania.



Následne, **MAC adresa** je vypísaná pomocou funkcie *write_MAC_adress()*, ktorá prijíma rámec v hexadecimálnom tvare, a číslo 12 alebo 0, čo predstavujú bitovú pozíciu odkiaľ ma byť čítaná MAC adresa (12 je pre zdrojovú MAC adresu a 0 je pre cieľovú MAC adresu).

Následne, program vypíše **vnorený protokol**, hodnota ktorá bola vrátená skôr funkciou *write_type_of_frame()* a je zapísaná do premennej *vnoreny_protokol*.

V prípade ak vnorený protokol je typu *IPv4*, je vypísaná zdrojová IP adresa (od bitu 52 po 60) a cieľová IP adresa (od bitu 60 po 68) pomocou funkcie *transforme_to_IP_adress()*. Taktiež, v prípade vnoreného protokolu typu *IPv4*, je zavolaná funkcia *write_IPv4_type_port*, ktorá prijíma rámec v hexadecimálnej forme a číslo rámca pre vypísanie vnorenejšieho protokolu. Ak vnorený protokol daného rámca je zapísaný do globálneho slovníka *IPprotocols*, tak je vypísaný presný názov, ak sa nenachádza v slovníku, je vypísaná sprava je nie je uvedený v databáze. Ak vnorený protokol je typu TCP alebo UDP, je volaná funkcia *write_TCP_type_port()*, respektíve *write_UDP_type_port()*, ktorá vypíše presný názov vnoreného protokolu (ak existuje v slovníku) a **cieľový a zdrojový port**. V prípade vnoreného portu typu TCP, je riešene pridanie komunikácii do triedene podľa typu vnoreného protokolu do špecifických globálnych slovníkov pomocou funkcie *add_communication()*. Funkcia *add_communications()* funguje takým spôsobom, že prijíma ako parameter slovník komunikácii daného vnoreného protokolu a postupne pridáva komunikácie (môže byť podľa cieľového zariadenia, alebo zdrojového) do slovníka pomocou kľúča typu *src_ip + dst_ip + src_port + dst_port*. Pomocou tohto kľúča sú triedene komunikácie všetkých vnorených protokolov typu TCP.

```
def add_communication(communications, index_frame, hex_packet):

    src_ip = transforme_to_IP_address(hex_packet[52:60])
    dst_ip = transforme_to_IP_address(hex_packet[60:68])

    src_port = int(hex_packet[68:72], 16)
    dst_port = int(hex_packet[72:76], 16)

    # kluc pre kazdu komunikaciu vytvoreny z str(src_ip) + str(dst_ip) + str(src_port) + str(dst_port)
    src_key = str(src_ip) + str(dst_ip) + str(src_port) + str(dst_port)
    dst_key = str(dst_ip) + str(src_ip) + str(dst_port) + str(src_port)

    if not communications.__contains__(src_key) and not communications.__contains__(dst_key):
        communications[src_key] = list()
        communications[src_key].append([index_frame, hex_packet])

    elif communications.__contains__(src_key):
        communications[src_key].append([index_frame, hex_packet])

    elif communications.__contains__(dst_key):
        communications[dst_key].append([index_frame, hex_packet])
```

Program **vypíše celý rámec** pomocou funkcie *write_entire_packet()*, ktorá prijíma rámec v hexadecimálnej forme a dĺžku rámca.

Cod sa týka 3. bod zadania, vypísanie zoznamu IP adries všetkých odosielaajúcich uzlov a vypísanie IP adresy uzla, ktorý sumárne odoslal najväčší počet paketov a koľko paketov odoslal sú riešene v dolnej časti funkcie *analyze_files()*.

Pre **výpis zoznamu všetkých odsol'ujúcich IP** adries používa for loop, ktorý postupne prechádza všetky zaznamenané adresy zo slovníku *source_IPv4_adresses()*, v ktorom ako kľúč je zaznamenaná IP adresa, a ako parameter počet vyskytnutí. Tieto adresy sú zapísané do slovníka pomocou funkcie *add_source_IPv4_adress_to_list()*, ktorá je volaná ešte pri zistení vnoreného protokolu Ethernet II pre všetkých rámcov v funkcii *find_ethet_type()*. Ale pred for loopom, sú zoradené zostupne všetky IP adresy podľa počtu výskytov pomocou príkazu *dic(sorted(source_IPv4_adress.items(), key = operator.itemgetter(1), reverse=True))*. Takýmto spôsobom sú zostupne vypísané všetky IP adresy a počet vyskytnutí. Pre výpis IP zdrojovej IP adresy ktorá sa vyskytla najčastejšie, používa príkaz *max(source_IPv4_adress, key=source_IPv4_adress.get)*, a potomok vrátená hodnota je vypísaná do súboru spolu s počtom vyskytnutím.

b) Bod zadania 4

Analýza komunikácii protokolov HTTP, HTTPS, TELNET, SSH, FTP riadiace FTP dátové je uskutočnená pomocou funkcie *write_complete_and_incomplete_TCP_communication()*, ktorá priam ako parameter slovník protokolu (napr. *HTTP_communications*) a názov protokolu ako string. V globálnych slovníkoch jednotlivých protokol sú pridané komunikácie pomocou funkcie *add_communicaiton()* ešte vo fáze zistenia TCP vnoreného protokolu, čiže v tomto bode už sú všetky komunikácie zapísané (táto funkcia je stručnejšie opísaná ešte v časti Body zadania 1-3). For cyklus prechádza postupne všetky komunikácie daného protokolu a je zavolaná funkcia *find_start_communication()*, ktorá preveruje ak komunikácie vôbec začína (3 way handshake) a následne, ak komunikácia začala, tak sa preverí úspešnosť ukončenia komunikácie pomocou funkcie *find_end_communication()*.


```

def write_complete_and_incomplete_TCP_communication(communications, protocol_type):
    uspesna_vypisana_kompletne = 0
    uspesna_vypisana_nekompletne = 0

    for commun in communications:
        start = find_start_communication(commun, communications)
        if start[0] == "complete":
            print("3wh")
            ramec_pokracovat = start[1]
            end = find_end_communication(commun, ramec_pokracovat, communications)
            if end == "complete":
                print("COMPLETE")
                if uspesna_vypisana_kompletne == 0:
                    FILE_VYPIS.write("\nuspesna " + protocol_type + " komunikacia-----\n")
                    write_frame(communications[commun])
                    uspesna_vypisana_kompletne = 1
            elif end == "incomplete":
                print("NOT C")
                if uspesna_vypisana_nekompletne == 0:
                    FILE_VYPIS.write("\nNEuspesna " + protocol_type + " komunikacia-----\n")
                    write_frame(communications[commun])
                    uspesna_vypisana_nekompletne = 1

```

Funkcia *find_start_communication()* hľadá postupnosť Flagov SYN - SYN zároveň ACK - ACK (3 way handshake). Tieto flagy sú zapísané od 92. po 96 bit v rámci. Ak sa na hore uvedená postupnosť nájde, znamená že komunikácia začala a vráti pomocou jedného póla sprava *compelte* a číslo nasledovného rámca.

Funkcia *find_end_communication()* funguje podobní spôsobom ako funkcia *find_start_communication()*, ale prijíma ako parameter aj číslo oramca odkiaľ musí začať vyhľadávanie. Tato funkcia hľadá tri možné soby ukončenia komunikácie pomocou flagov, a to: postupnosť FIN – ACK – FIN – ACK, postupnosť FIN – ACK – RST, alebo iba flag RST. Ak komunikácia je úspešne ukončená (sa našiel jedoš s hore uvedených prípadov), funkcia vráti správu *compelte*, alebo v opačnom prípade *incomplete*.

Späť ku funkcii *write_complete_and_incomplete_TCP_communication()*, po zistení začatej a ukončenej komunikácii, triedi komunikácie, takým spôsobom, že vypíše iba raz prvú úspešne ukončenú komunikáciu a iba raz prvú neúspešne ukončenú komunikáciu a ich vypíše pomocou funkcie *write_frame()*.

Funkcia *write_frame()* prijíma ako parameter slovník komunikácie ktorú ma vypísať. Najprv preverí počet rámcov komunikácii, ak počet rámcov komunikácii je väčší než 20, vypíše iba prvé 10 rámce a posledné 10 rámce. Ak počet rámcov je myši než 20, tak vypíše všetky rámce. Vypisovanie rámca sa uskutoční pomocou funkcie *wrire_frame_efectiv()*, ktorá prijíma ako parameter jednotlivý rámec a ho výpise podobným spôsobom ako funkcia *analyze__file()* (v časti Body riešenia 1-3).


```

def write_frame(packets):
    x = 0

    # ak su viac nez 19, vypise prve 10 a posledne 10
    if len(packets) > 19:
        x = 0
        for frame in packets:
            x += 1
            if x > 10:
                break
            write_frame_efectiv(frame)

        # posledne 10 ramce
        x = 0

        for frame in packets[len(packets)-20:]:
            x += 1
            if x > 10:
                break
            write_frame_efectiv(frame)

    # ak su menej nez 20, vypise vsetky
    if len(packets) < 20:
        dlzka = len(packets)

        for frame in packets:
            x += 1
            if x > dlzka:
                break
            write_frame_efectiv(frame)

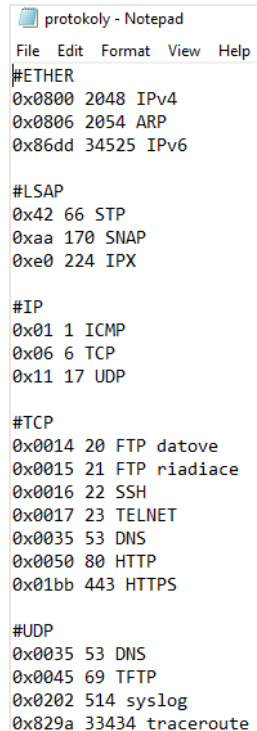
```

ICMP, ARP a TFTP komunikácie sú pridane do slovníkov a vypisane podobne ako TCP protokoly. ICMP komunikácie sú zapísane do slovníka *ICMP_communications* vo funkcii *write_CPM_type_port()*, pri detekcii takéhoto protokolu ktorá je volaná ešte vo funkcii *write_IPv4_type_port()* a vkladá rámce do knižnice pomocou kľúča *src_ip + dst_ip*.

ARP komunikácie sú pridane pomocou funkcie *add_ARP_communication()*, ktorá je volaná ešte vo funkcii *find_ether_type()* pri detekcii ARP protokolu. ARP rámce sú triedene porátala ARP typu request a reply. Rámec je typu request, kde na bitoch 42 až 44 sa nachádza 01, v prípade ak sa tam nachádza 02, je to typ reply. Komunikácie sú vládane do globálneho slovníka pomocou kľúča *src_ip + dst_MAC + dst_ip*. Keď rámec je typu request, do kľúča ako *src_ip* bude vložen IP nachádzajúci sa v rámci od bitu 56 po 64, *dst_ip* bude IP v rámci od bitu 76 po 84 a *src_Mac* sa nachádza v rámci od bitu 12 po 24. V prípade rámca ARP typu reply, *dst_ip* a *src_ip* budú naopak ako v prípade ARP request a *src_Mac* adresa bude od bitu 0 po 12. Posledný protokol, protokol typu TFTP je pridaný pomocou funkcie *add_TFTP_communication()* s kľúčom typu *src_ip + dst_ip*. Funkcia pre pridanie TFTP komunikácii do slovníka je volaná vo funkcii *write_UDP_type_port()*, keď v slovníku *UDPports* sa nachádza cieľový port nachádzajúci sa tiež v rámci od bitu 72 po 76. Keď tento typ portu je TFTP a už tento port sa nachádza v *UDPports* knižnici, znamená že je to skutočne typ TFTP.

4 Príklad štruktúry externých súborov pre určenie protokolov a portov

Program číta zo súboru *protokoly.txt* všetky určené protokoly a porty.



```
protokoly - Notepad
File Edit Format View Help
#ETHER
0x0800 2048 IPv4
0x0806 2054 ARP
0x86dd 34525 IPv6

#LSAP
0x42 66 STP
0xaa 170 SNAP
0xe0 224 IPX

#IP
0x01 1 ICMP
0x06 6 TCP
0x11 17 UDP

#TCP
0x0014 20 FTP datove
0x0015 21 FTP riadiace
0x0016 22 SSH
0x0017 23 TELNET
0x0035 53 DNS
0x0050 80 HTTP
0x01bb 443 HTTPS

#UDP
0x0035 53 DNS
0x0045 69 TFTP
0x0202 514 syslog
0x829a 33434 traceroute
```

5 Opísanie používateľského rozhrania

Po spustení programu používateľ nemusí robiť nič, všetky výsledky budú zapísané do súborov. Skôr ako by spustil program, používateľ musí vložiť do priečinka *subory_na_analyzu* súbor typu *.pcap*, ktorý chce analyzovať. Do budúcej aktualizácie programu, do priečinka musí byť vložiť iba jeden *.pcap* súbor. Po spustení programu, do súboru *vypis.txt* program zapíše analýzu zámkov (1-3 body zadania) a do súboru *vypis_komunikacie* program zapíše analýzu komunikácii (4 bod zadania).

[7]

6 Voľbu implementačného prostredia

Program bol písaný v programovacom jazyku Python 3.9 a boli použité knižnice *walk*, *bytes_hex* a *rdpcap* v bol písaný vo vývojové prostredí PyCharm.