PortScanner
Il port-scanner, come suggerisce la parola, prende in input un IP ed un intervallo di porte e controlla se queste sono aperte o meno. Per un Hacker è fondamentale sapere quali porte sono aperte e quali servizi sono attivi.

Il primo passo è quello di importare la libreria "socket" che mette a disposizione tutte le funzioni necessarie per creare, configurare e utilizzare una connessione TCP o UDP grazie il quale è possibile stabilire una connessione tra un punto sorgente e un punto destinatario, attraverso la quale vengono trasmessi dati sotto forma di pacchetti.
il programma richiede all'utente di fornire due informazioni fondamentali: l'indirizzo IP del target da analizzare (target) e un intervallo di porte da sottoporre a scansione (portRange).

```
1 import socket
2
3 target = input("Enter the IP address to scan: ")
4 portRange = input("Enter the port range to scan (es. 5-200): ")
5
```

Una volta ricevuto l'intervallo, il programma utilizza il metodo .split('-') per dividere la stringa in due parti, e poi le converte in numeri interi tramite la funzione int(), ottenendo così i due estremi dell'intervallo su cui lavorare.

```
5
6 lowPort = int(portRange.split('-')[0])
7 highPort = int(portRange.split('-')[1])
8
```

Dopo aver definito i parametri, lo script stampa un messaggio per indicare che sta per iniziare la scansione del target specificato, all'interno del range di porte fornito. Questa fase non ha impatto sulla logica del programma, ma è utile per l'utente in modo da sapere cosa aspettarsi e su quale macchina e intervallo di porte si sta lavorando.

La parte più importante del codice è il ciclo for, che esegue un'iterazione su ogni porta compresa nell'intervallo fornito. Per ogni porta, il programma crea un nuovo socket TCP (socket.AF_INET indica che si sta usando IPv4, mentre socket.SOCK_STREAM indica l'uso del protocollo TCP) e tenta di stabilire una connessione con il metodo connect_ex(). Questo metodo restituisce 0 se la connessione ha successo (quindi la porta è aperta e probabilmente un servizio è in ascolto), oppure un altro valore se la porta è chiusa o filtrata. In base al valore restituito, lo script stampa un messaggio in base al risultato di un ciclo if che indica se la porta è aperta, evidenziando il messaggio con asterischi e se la porta è chiusa, viene notificato in un modo più neutro.

```
10
11 for port in range(lowPort, highPort):
12         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13         status = s.connect_ex((target, port))
14         if(status == 0):
15                 print("*** Port ", port, " OPEN ***")
16         else:
17                 print("Port ", port, " Closed")
```

Dal punto di vista della cybersecurity, strumenti come questo sono spesso usati durante la fase di ricognizione attiva nei penetration test, dove si vuole scoprire quali servizi sono esposti da una macchina. Ogni porta aperta rappresenta potenzialmente una superficie d'attacco, e conoscere quali sono accessibili può aiutare a determinare la configurazione del sistema, l'eventuale presenza di servizi vulnerabili o non autorizzati, e a preparare le successive fasi dell'attacco simulato.

Per poter testare il codice e dimostrare il suo funzionamento, utilizzo una macchina target metasploitable con sistema operativo linux con indirizzo IP 192.168.20.121 inserendo un range di porte che vadano da 0 a 500:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python portScanner.py
Enter the IP address to scan: 192.168.20.121
Enter the port range to scan (es. 5-200): 0-200
scanning host 192.168.20.121  from port  0  to port  200
Port  0  Closed
Port  1  Closed
Port  2  Closed
Port  3  Closed
Port  4  Closed
Port  5  Closed
Port  6  Closed
Port  7  Closed
Port  8  Closed
Port  9  Closed
Port  10  Closed
Port  11  Closed
Port  12  Closed
Port  13  Closed
Port  14  Closed
Port  15  Closed
Port  16  Closed
Port  17  Closed
Port  18  Closed
Port  19  Closed
Port  20  Closed
*** Port  21  OPEN ***
*** Port  22  OPEN ***
*** Port  23  OPEN ***
Port  24  Closed
*** Port  25  OPEN ***
```

Otterremo così una lunga lista di porte aperte e chiuse con la possibilità di impostare il codice in maniera tale che vengano mostrate in output solo le porte aperte semplicemente commentando le ultime due righe del codice:

```
11 for port in range(lowPort, highPort):
12         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13         status = s.connect_ex((target, port))
14         if(status == 0):
15                 print("*** Port ", port, " OPEN *
16         #else:
17         #        print("Port ", port, " Closed")
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python portScanner.py
Enter the IP address to scan: 192.168.20.121
Enter the port range to scan (es. 5-200): 0-200
scanning host 192.168.20.121  from port  0  to port  200
*** Port  21  OPEN ***
*** Port  22  OPEN ***
*** Port  23  OPEN ***
*** Port  25  OPEN ***
*** Port  53  OPEN ***
*** Port  80  OPEN ***
*** Port  111  OPEN ***
*** Port  139  OPEN ***
```

In conclusione, questo script rappresenta un buon punto di partenza per comprendere le basi del port scanning in Python, e può essere ulteriormente migliorato per diventare uno strumento più robusto, efficiente e sicuro, adatto anche ad ambienti più complessi e professionali.

Ecco il codice finale:

```python
import socket

target = input("Enter the IP address to scan: ")
portRange = input("Enter the port range to scan (es. 5-200): ")

lowPort = int(portRange.split('-')[0])
highPort = int(portRange.split('-')[1])

print("scanning host", target, " from port ", lowPort, " to port ", highPort)

for port in range(lowPort, highPort):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        status = s.connect_ex((target, port))
        if(status == 0):
                print("*** Port ", port, " OPEN ***")
        else:
                print("Port ", port, " Closed")
```