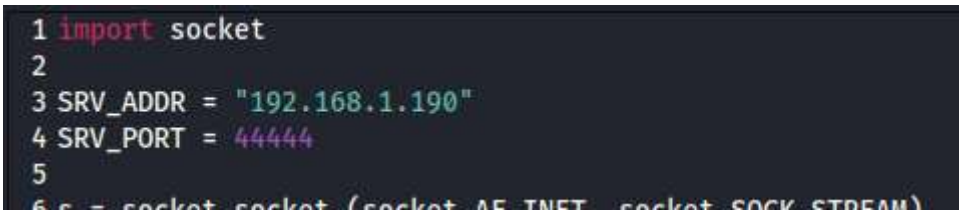
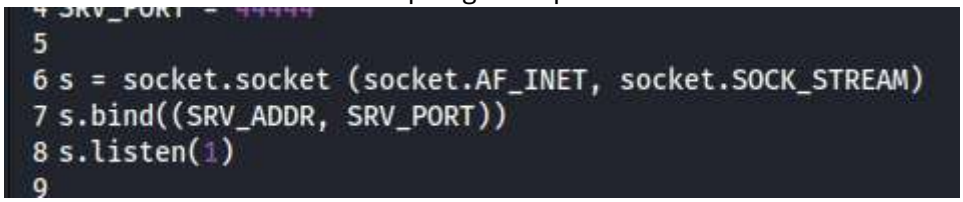
Socket di rete
I socket di rete sono uno degli strumenti fondamentali per permettere la comunicazione tra computer all'interno di una rete. Grazie a essi è possibile stabilire una connessione tra un punto sorgente e un punto destinatario, attraverso la quale vengono trasmessi dati sotto forma di pacchetti. In Python, per poter utilizzare i socket, è necessario importare il modulo standard chiamato socket. Questo modulo mette a disposizione tutte le funzioni necessarie per creare, configurare e utilizzare una connessione TCP o UDP. È importante non nominare il file in cui scriviamo il codice come socket.py, perché altrimenti Python interpreterebbe il nostro file come se fosse il modulo originale, causando un conflitto che impedirebbe l'importazione corretta delle funzioni di rete.`

Nel nostro esempio, vogliamo scrivere un piccolo server TCP che rimane in ascolto in attesa che un client si connetta. Per farlo, definiamo due variabili: una contiene l'indirizzo IP del computer su cui il server deve rimanere attivo (SRV_ADDR), mentre l'altra specifica la porta su cui riceverà le connessioni(SRV_PORT). In questo modo si stabilisce un punto di accesso univoco nella rete che permetterà ai client di collegarsi.

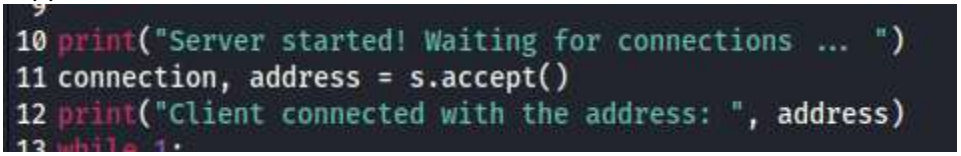


Dopo aver configurato indirizzo e porta, si procede creando un oggetto socket utilizzando la sintassi socket.socket(). Questa istruzione chiama una funzione, contenuta nel modulo socket, che restituisce un oggetto di tipo socket. In questo contesto, passiamo due parametri fondamentali. Il primo è socket.AF_INET, dove AF sta per Address Family e INET indica che stiamo utilizzando il protocollo IPv4. Se si volesse usare IPv6, si dovrebbe passare socket.AF_INET6. Il secondo parametro è socket.SOCK_STREAM, che specifica che il socket è di tipo stream, quindi userà il protocollo TCP. Questo tipo di socket garantisce una connessione affidabile e orientata alla comunicazione continua tra i due endpoint.

Una volta creato il socket, dobbiamo associarlo all'indirizzo IP e alla porta precedentemente definiti. Questo avviene tramite il metodo bind(), che permette di riservare la porta e l'indirizzo per il nostro server. Subito dopo, attiviamo la modalità di ascolto con il metodo listen(). Questo fa sì che il server si metta in attesa di connessioni da parte di client. Il parametro passato a listen(1) specifica il numero massimo di connessioni in coda che il server può gestire prima di iniziare a rifiutare nuove richieste.



Il passo successivo è attendere effettivamente una connessione. Questa operazione si effettua chiamando il metodo accept() sull'oggetto socket. Si tratta di una chiamata bloccante, cioè il programma si ferma e resta in attesa finché un client non tenta di connettersi. Quando questo accade, accept() restituisce due valori: il primo rappresenta un nuovo oggetto socket che verrà utilizzato per comunicare direttamente con quel client specifico; il secondo contiene l'indirizzo IP e la porta del client che si è appena connesso.



Una volta che un client ha stabilito la connessione, il server entra in un ciclo infinito dove rimane in attesa di ricevere dati. Questo ciclo inizia con un while 1, che equivale a dire "esegui per sempre, finché non ti dico di fermarti". All'interno del ciclo, il metodo recv(1024) prova a leggere fino a 1024 byte inviati dal client. Se il client chiude la connessione, recv() restituirà un oggetto vuoto e a quel punto il ciclo verrà interrotto. Quando invece i dati vengono ricevuti correttamente, sono in formato binario, quindi è necessario decodificarli con decode('utf-8') per poterli leggere come testo. Una volta decodificati, i dati vengono stampati a schermo tramite la funzione print.

```
13 while 1:
14         data = connection.recv(1024)
15         if not data: break
```

Quando il client chiude la connessione o il server riceve un segnale vuoto, il ciclo si interrompe e viene eseguita la chiusura del socket di comunicazione con il client, utilizzando il metodo close() sull'oggetto restituito da accept(). In questo modo si rilascia la risorsa associata alla connessione, e il programma può terminare o tornare in ascolto per una nuova connessione, se modificato per farlo.

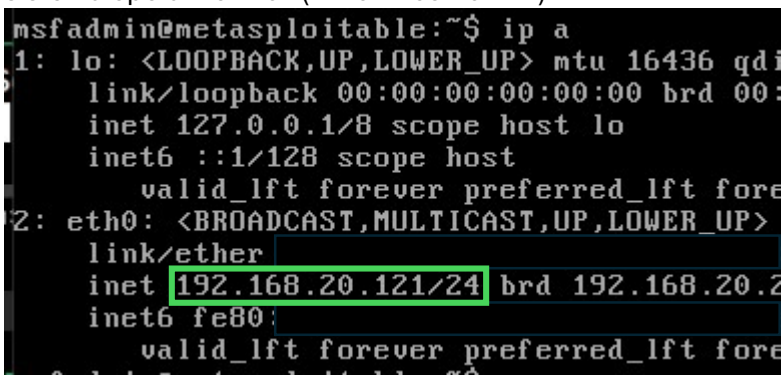
```
17         print(data.decode('utf-8'))
18 connection.close()
```

È presente anche una stringa commentata "#connection.sendall(b'-- Message Receved --\n')" che, se dovesse essere decommentata rimuovento il cancelletto, invia il messaggio "-- Message Receved --" al client che invia qualche dato al server.

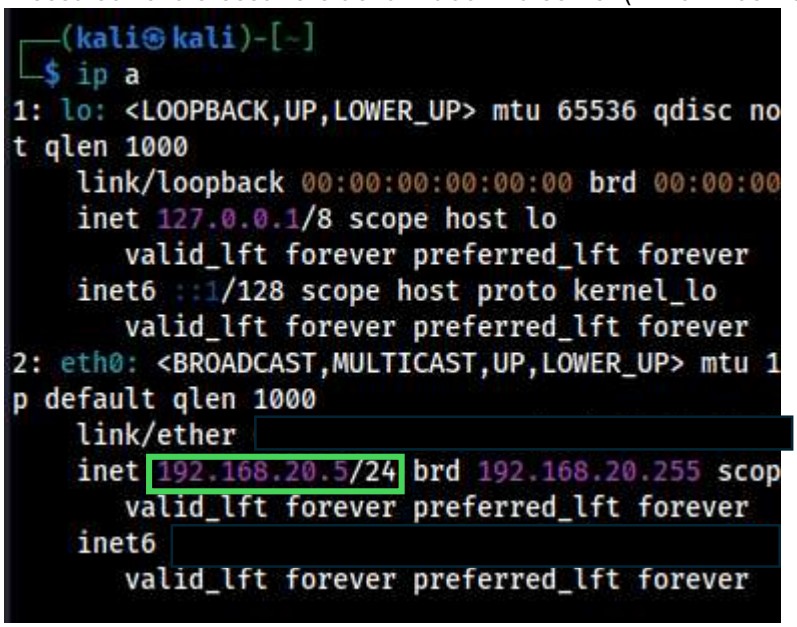
Ecco il codice completo:

```
 1 import socket
 2
 3 SRV_ADDR = "192.168.1.190"
 4 SRV_PORT = 44444
 5
 6 s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
 7 s.bind((SRV_ADDR, SRV_PORT))
 8 s.listen(1)
 9
10 print("Server started! Waiting for connections ... ")
11 connection, address = s.accept()
12 print("Client connected with the address: ", address)
13 while 1:
14         data = connection.recv(1024)
15         if not data: break
16         #connection.sendall(b'-- Message Receved --\n')
17         print(data.decode('utf-8'))
18 connection.close()
```

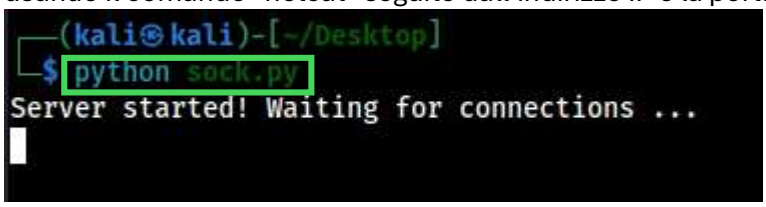
Per poter utilizzare il socket dal punto di vista client, utilizzo una macchina metosploitable con sistema operativo linux (IP 192.168.20.121):

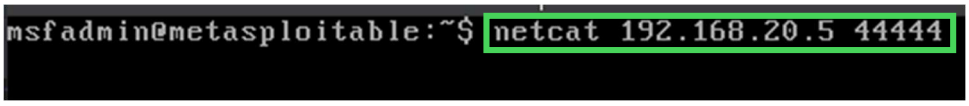


messa sotto la stessa rete della macchina server (IP 192.168.20.5).



Dopo aver avviato il codice sulla macchina server, sulla macchina client effettuerò una connessione usando il comando “netcat” seguito dall’indirizzo IP e la porta della macchina server





Possiamo anche vedere come sulla console del server, vengono stampati informazioni riguardo il client connesso: