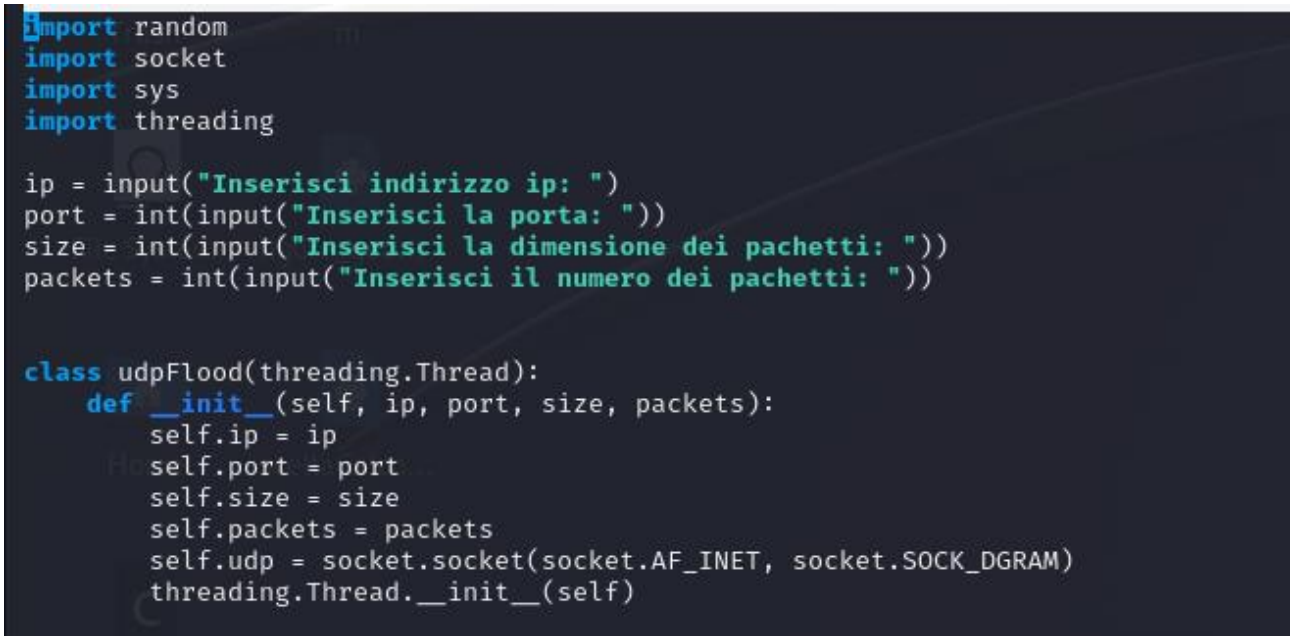
ESERCIZIO UPD



```
import random
import socket
import sys
import threading

ip = input("Inserisci indirizzo ip: ")
port = int(input("Inserisci la porta: "))
size = int(input("Inserisci la dimensione dei pachetti: "))
packets = int(input("Inserisci il numero dei pachetti: "))


class udpFlood(threading.Thread):
    def __init__(self, ip, port, size, packets):
        self.ip = ip
        self.port = port
        self.size = size
        self.packets = packets
        self.udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        threading.Thread.__init__(self)
```

Questo è il metodo di inizializzazione (`__init__`) della classe `udpFlood`. Esso viene chiamato automaticamente quando crei un nuovo oggetto della classe. Ora spieghiamo ciascuna riga:

- `self.ip = ip`: Imposta l'attributo `ip` dell'istanza dell'oggetto sulla variabile `ip` passata come argomento al costruttore.

- `self.port = port`: Analogo a sopra, imposta l'attributo `port` dell'istanza dell'oggetto sulla variabile `port` passata come argomento.

- `self.size = size`: Ancora analogo, imposta l'attributo `size` sulla variabile `size`.

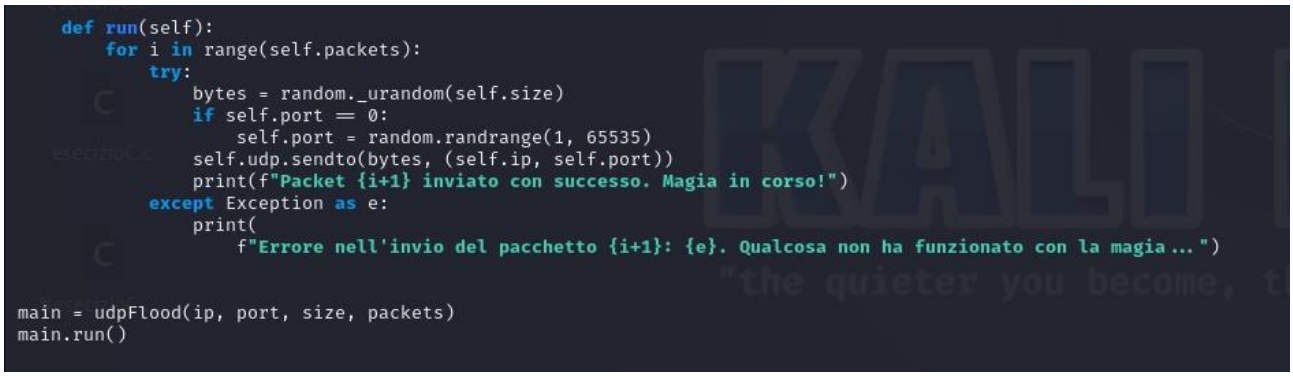
- `self.packets = packets`: Simile agli altri, imposta l'attributo `packets` sulla variabile `packets`.

- `self.udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`: Crea un oggetto socket UDP (`self.udp`) e lo assegna all'attributo `udp` dell'istanza

dell'oggetto. `socket.AF_INET` indica l'uso di IPv4, e `socket.SOCK_DGRAM` indica che stiamo utilizzando un socket di tipo UDP.

- `threading.Thread.__init__(self)` : Richiama il metodo di inizializzazione della classe padre `Thread`. La classe `udpFlood` eredita da `Thread`, e questa riga assicura che l'inizializzazione della classe padre venga eseguita correttamente.

In sintesi, questo metodo si occupa di inizializzare gli attributi dell'oggetto con i valori forniti come argomenti al momento della creazione dell'oggetto, e di configurare un socket UDP per l'invio dei pacchetti.



```
    def run(self):
        for i in range(self.packets):
            try:
                bytes = random._urandom(self.size)
                if self.port == 0:
                    self.port = random.randrange(1, 65535)
                self.udp.sendto(bytes, (self.ip, self.port))
                print(f"Packet {i+1} inviato con successo. Magia in corso!")
            except Exception as e:
                print(
                    f"Errore nell'invio del pacchetto {i+1}: {e}. Qualcosa non ha funzionato con la magia ...")


main = udpFlood(ip, port, size, packets)
main.run()
```

Questa porzione di codice è legata al metodo `run` della classe `udpFlood` e all'istanziazione di un oggetto di questa classe.

Il metodo `run` è un metodo speciale che viene richiamato quando si esegue `main.run()`. In questo contesto, il metodo `run` è responsabile dell'effettivo invio dei pacchetti. Analizziamo le singole righe:

- Il ciclo `for i in range(self.packets)` itera `self.packets` volte, inviando un pacchetto ad ogni iterazione.

- `bytes = random._urandom(self.size)` : Genera una sequenza di byte casuali di lunghezza `self.size`. Questi byte costituiranno il carico utile dei pacchetti UDP.

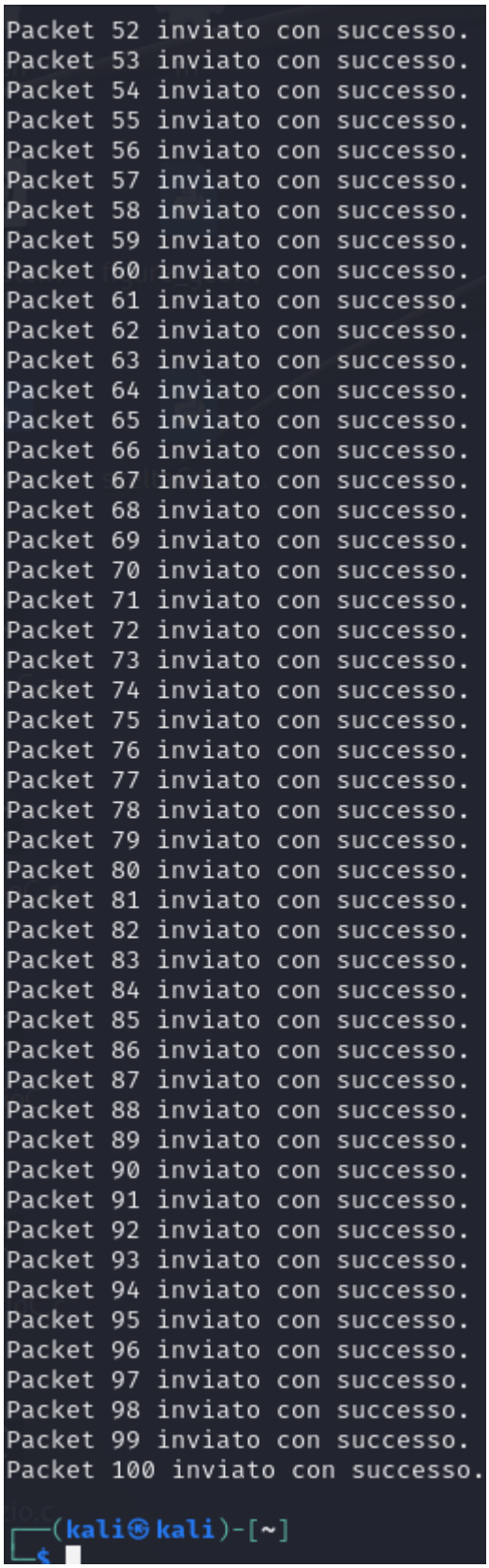
- `if self.port == 0: self.port = random.randrange(1, 65535)`: Controlla se la porta è 0. In caso affermativo, assegna una porta casuale compresa tra 1 e 65534.

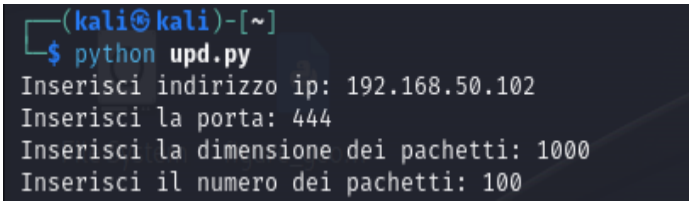
- `self.udp.sendto(bytes, (self.ip, self.port))`: Invia il pacchetto UDP al destinatario specificato da `self.ip` e `self.port`.

- `except: pass`: Questo blocco `except` cattura qualsiasi eccezione che potrebbe verificarsi durante l'invio del pacchetto, ma non esegue alcuna azione in caso di errore.

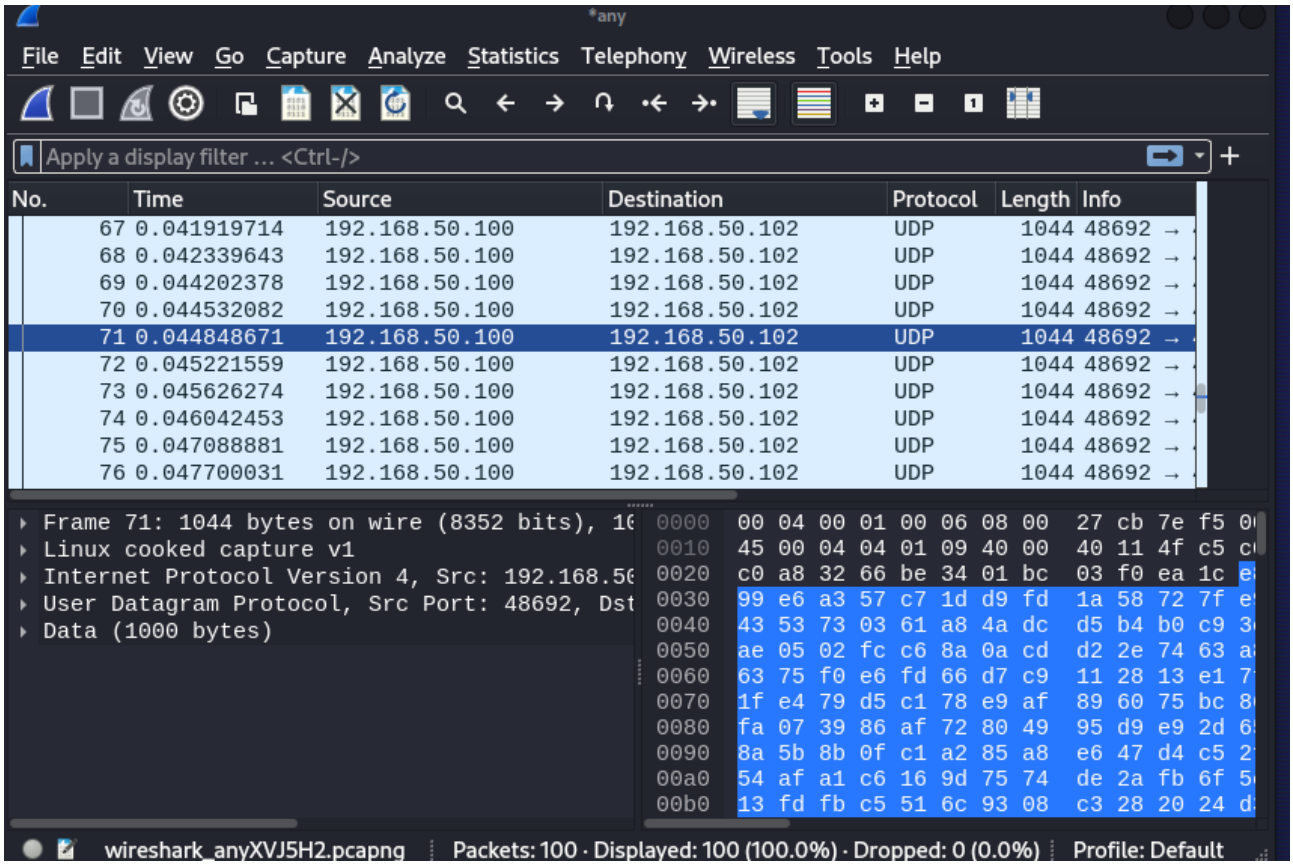
Infine, riguardo alle ultime due righe del codice:

L'istruzione `main = udpFlood.__init__(ip, port, size, packets)` è un uso non convenzionale e potenzialmente errato. L'istanziazione corretta dovrebbe essere fatta direttamente come `main = udpFlood(ip, port, size, packets)`. Inoltre, è consigliabile avviare il thread usando `main.start()` anziché chiamare esplicitamente `main.run()`. Va notato che chiamare il costruttore (`__init__`) direttamente non è una pratica comune, poiché il costruttore viene implicitamente chiamato quando si crea un nuovo oggetto della classe.





L’immagine a destra mostra l’invio dei pacchetti, ed l’immagine a sinistra mostra dove mando i pacchetti.



se si utilizza un tool come quello descritto nel codice fornito (un attacco di tipo UDP flood), Wireshark potrebbe mostrare i seguenti aspetti:

1. **Alto volume di pacchetti UDP**: Wireshark registrerebbe un elevato numero di pacchetti UDP inviati dal mittente al destinatario specificato.
2. **Payload randomico**: Analizzando i pacchetti, si noterebbe un payload casuale generato dalla funzione **random._urandom(self.size)**, il quale costituirebbe il contenuto effettivo dei pacchetti UDP inviati.
3. **Porte UDP casuali**: Wireshark evidenzierebbe l'uso di porte UDP casuali, poiché il codice controlla se la porta è 0 e, in tal caso, assegna una porta casuale tra 1 e 65534.
4. **Risposta del destinatario**: Potrebbe essere possibile osservare la risposta del destinatario ai pacchetti UDP inviati. La natura del codice suggerisce che il destinatario non risponde attivamente, ma Wireshark comunque mostrerebbe i pacchetti inviati al destinatario.