PROGRAMMA PYTHON UDP FLOOD

Introduzione

Un attacco **UDP Flood** è una tipologia di attacco **DoS (Denial of Service)** volumetrico che mira a saturare le risorse di una rete o di un server bersaglio inondando con una quantità enorme di pacchetti UDP (User Datagram Protocol). Per capire l'attacco, bisogna ricordare che l'UDP è un protocollo **"connectionless"** (senza connessione). A differenza del TCP, non prevede un "handshake" (stretta di mano) per stabilire una sessione. I dati vengono inviati senza verificare se il destinatario è pronto o se il pacchetto è arrivato correttamente. Questa sua "leggerezza" lo rende perfetto per lo streaming o il gaming, ma anche facile da sfruttare per scopi malevoli.

In un attacco UDP Flood, l'attaccante invia un numero massiccio di pacchetti UDP a **porte casuali** del server vittima. Ecco cosa accade "dietro le quinte":

-Ricezione del pacchetto: Il server riceve il pacchetto UDP su una specifica porta.

-Verifica dell'applicazione: Il sistema operativo controlla se c'è un'applicazione in ascolto su quella porta.

-Risposta di errore (ICMP): Quando il server vede che nessuna applicazione è associata a quella porta, genera un pacchetto **ICMP "Destination Unreachable"** per informare il mittente.

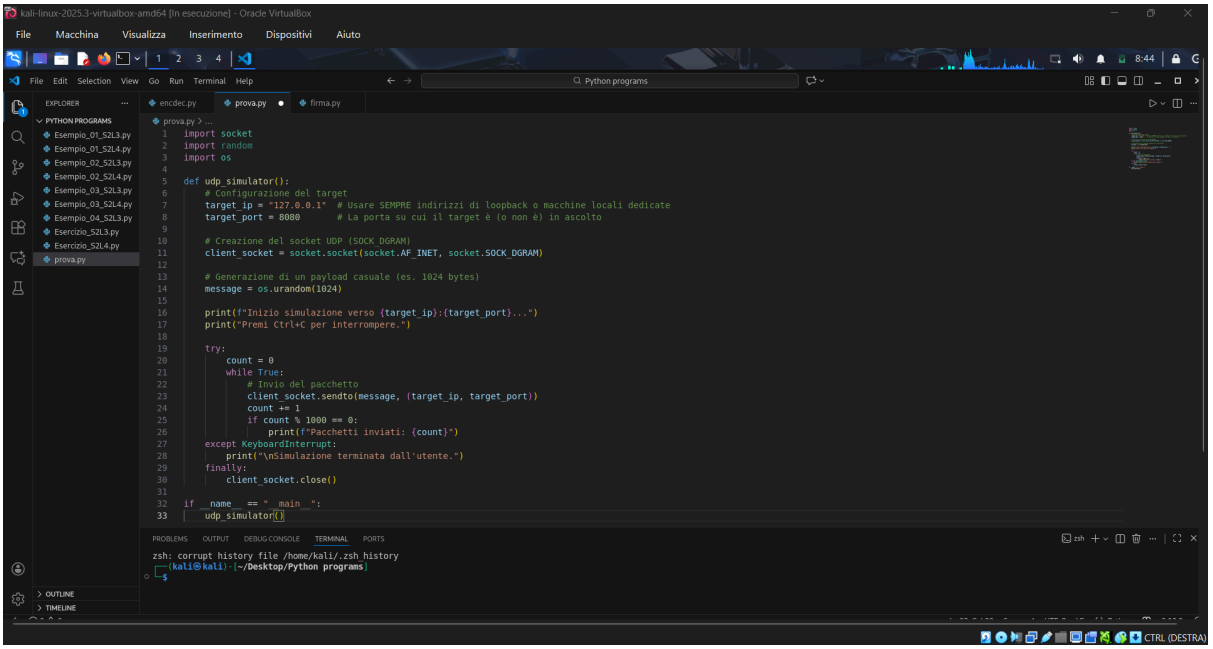
-Esaurimento risorse: Se arrivano migliaia di pacchetti al secondo, il server consuma tutta la sua larghezza di banda e potenza di calcolo (CPU) solo per elaborare i pacchetti in entrata e inviare le risposte di errore.

Obiettivo:

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 sulla porta UDP casuale.

1) Creazione dello scenario

Spiegazione del passaggio: Creiamo il nostro codice in python su visual studio code.



2) Spiegazione dello scenario

Spiegazione del codice: Questo codice utilizza delle **librerie** (una libreria è una collezione di moduli che contengono codice già scritto da altri programmatori, pronto per essere riutilizzato).

-Import socket: La libreria fondamentale per inviare e ricevere dati in rete.

-Import random: Carica la libreria standard di Python dedicata alla generazione di numeri e scelte casuali.

-Import os: Viene usata qui per accedere a funzioni del sistema operativo, specificamente per generare dati casuali.

```
1   import socket
2   import random
3   import os
```

Questa sezione del codice si occupa della **configurazione iniziale** e della **preparazione delle risorse** necessarie per avviare lo scambio di dati. Queste variabili definiscono "dove" verranno inviati i dati:

```
target_ip = "127.0.0.1"  # Usare SEMPRE indirizzi di loopback o macchine locali dedicate
target_port = 8080       # La porta su cui il target è (o non è) in ascolto
```

-target_ip: L'indirizzo IP. Il valore "127.0.0.1" è fondamentale: è l'indirizzo di **loopback**. Indica al computer di inviare i dati a se stesso. È la pratica standard per testare il codice in totale sicurezza senza influenzare altri dispositivi o la rete internet.

-**target_port**: La porta specifica. In rete, le porte sono come i citofoni di un palazzo: ogni numero corrisponde a un servizio diverso (es. la 80 per il web). Qui si usa la 8080, spesso usata per test di server web alternativi.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Questa riga "apre la comunicazione".

-**socket.AF_INET**: Specifica che useremo indirizzi **IPv4** (il formato classico tipo 127.0.0.1).

-**socket.SOCK_DGRAM**: Specifica che il protocollo è **UDP**. A differenza del TCP, l'UDP invia "datagrammi" (pacchetti isolati) senza aspettare conferme di ricezione. È questa scelta che rende l'attacco di tipo "**flood**" (inondazione) possibile, perché non c'è perdita di tempo nel verificare la connessione.

```
message = os.urandom(1024)
```

Qui viene creato il "carico" da spedire:

-os.urandom(1024): Chiede al sistema operativo di generare **1024 byte** (1 Kilobyte) di dati casuali binari.

-In una simulazione DoS, il contenuto dei dati non è importante; quello che conta è la **dimensione**. Inviando pacchetti da 1KB ripetutamente, si cerca di riempire la larghezza di banda del destinatario.

```
print(f"Inizio simulazione verso {target_ip}:{target_port}...")
print("Premi Ctrl+C per interrompere.")
```

Queste righe sono semplici istruzioni per chi esegue il programma, conferma che il processo è partito.

```
count = 0
while True:
    # Invio del pacchetto
    client_socket.sendto(message, (target_ip, target_port))
    count += 1
```

Questo è il cuore della simulazione:

-while True: Dice al computer di ripetere le istruzioni all'infinito.

-sendto: Invia il pacchetto UDP. In questo ciclo, viene richiamato migliaia di volte al secondo, saturando la capacità di elaborazione.

-count += 1: Incrementa un contatore ogni volta che un pacchetto viene spedito.

```
if count % 1000 == 0:
    print(f"Pacchetti inviati: {count}")
```

Senza queste righe, vedresti solo un cursore lampeggiante e non sapresti se il programma sta funzionando. L'operatore **%** (modulo) verifica se il numero è un multiplo di 1000. Se lo è, stampa il totale. Questo serve a non rallentare troppo il programma stampando ogni singolo pacchetto (operazione che richiederebbe molto tempo alla CPU).

```
except KeyboardInterrupt:
    print("\nSimulazione terminata dall'utente.")
```

Quando premi **Ctrl+C** sulla tastiera, Python genera un errore chiamato KeyboardInterrupt. Invece di mostrare una sfilza di errori tecnici rossi, il programma intercetta il comando e stampa un messaggio pulito di chiusura.

```
if __name__ == "__main__":
    udp_simulator()
```

Questa è una convenzione standard di Python. Dice al computer: "Se questo file viene eseguito direttamente (e non importato come modulo in un altro script), allora avvia la funzione udp_simulator()". È come l'interruttore "ON" del tuo script.

Conclusione:

In conclusione, l'attività di simulazione ha permesso di analizzare concretamente le dinamiche di un attacco **DoS (Denial of Service)** di tipo volumetrico. Attraverso l'esecuzione dello script Python, sono emersi i seguenti punti fondamentali:

-Vulnerabilità del protocollo UDP: È stato dimostrato come l'assenza di un meccanismo di controllo della sessione (handshake) renda l'UDP un protocollo facilmente sfruttabile per generare grandi volumi di traffico in breve tempo con un dispendio minimo di risorse da parte dell'attaccante.

-Impatto sulle risorse: Durante la simulazione, è stato possibile osservare come la continua generazione di pacchetti casuali causi un incremento nell'utilizzo della **CPU** e della **larghezza di banda** del sistema target (localhost), dimostrando il potenziale rischio di interruzione dei servizi legittimi.

-Importanza del monitoraggio: L'uso di contatori nel codice e di strumenti di analisi (come il Task Manager o analizzatori di pacchetti) è risultato essenziale per quantificare l'entità dell'invio dei dati e comprendere la velocità di propagazione dell'attacco.

-Prospettive di difesa: L'analisi del codice evidenzia la necessità di implementare contromisure adeguate a livello di rete, quali il **Rate Limiting** (limitazione della frequenza), la configurazione di firewall avanzati e l'uso di sistemi **IPS** (Intrusion Prevention System) capaci di identificare e bloccare pattern di traffico anomali.