

**Compito di oggi: spiegare cos'è una backdoor e perché è pericolosa.**

Backdoor: quei sistemi, software o hardware, che consentono un accesso privilegiato a dati o sistemi senza passare il vaglio del controllo accessi "standard". Quindi non è altro che una "porta di servizio" che permette di accedere indisturbati a un sistema informatico in maniera semplice e veloce. Una backdoor può essere installata dall'amministratore di sistema per entrare rapidamente in un dispositivo per effettuare operazioni di manutenzione in remoto, oppure, nella peggiore delle ipotesi, da un malware controllato da un hacker per prendere completamente il controllo di un sistema all'insaputa dell'utente. Anche utilizzando antivirus e antimalware, infatti, non è detto che questi sistemi di sicurezza "ordinari" siano in grado di chiudere questa porta di accesso "fantasma".

**Se lo scopo di una backdoor è consentire l'accesso rapido, veloce e indisturbato a un sistema informatico, è anche progettata per operare nell'ombra, sfuggendo al controllo degli antivirus, versatile al punto da essere in grado di sfruttare, all'occasione, le porte aperte da altri programmi più che legittimi.** È per questo motivo che cerca di nascondersi nel mare magnum dei file di sistema assumendo nomi e dimensioni che non diano nell'occhio al punto da insospettire un utente. Una volta scoperte infatti, permettono praticamente di tutto: dal controllo su tutti i processi attivi a quello sulla webcam, mouse e tastiera.

La pericolosità delle backdoor deriva dal fatto che consentono l'accesso non autorizzato ai dati sensibili, ai sistemi informatici o alle reti. Possono essere utilizzate da criminali informatici per rubare informazioni, danneggiare i dati, installare malware o compromettere la sicurezza in generale. Le backdoor possono essere inserite tramite vulnerabilità nel software, attraverso exploit o anche attraverso azioni umane fraudolente.

Inoltre, le backdoor possono essere sfruttate da governi o agenzie di sorveglianza per monitorare le attività di individui o organizzazioni senza il loro consenso o conoscenza. Questo solleva preoccupazioni legate alla privacy e alla sicurezza, poiché l'accesso non autorizzato può essere utilizzato per scopi non etici o illegali.

Spiegare i codici qui sotto dicendo cosa fanno e qual è la differenza tra i due.

### CODICE 1

```
1 import socket
2 import platform
3 import os
4
5 #Indirizzo IP del server e porta su cui ascoltare le connessioni
6 SRV_ADDR = "0.0.0.0"
7 SRV_PORT = 1234
8
9 #Creazione del socket
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.bind((SRV_ADDR, SRV_PORT))
12 s.listen(1) #Mette il server in ascolto per una connessione
13 connection, address = s.accept() #Accetta la connessione in arrivo
14
15 print ("client connected: ", address) #Stampa l'indirizzo del client connesso
16
17 while 1:
18     try:
19         data = connection.recv(1024) #Riceve i dati inviati dal client
20     except:continue #Se si verifica un errore, continua con il prossimo ciclo
21
22     #Decodifica i dati ricevuti dal client ed esegue azioni in base al messaggio ricevuto
23     if(data.decode('utf-8') == '1'):
24         #Se il messaggio è 1: invia informazioni sulla piattaforma e sulla macchina al client
25         tosend = platform.platform() + " " + platform.machine()
26         connection.sendall(tosend.encode())
27     elif (data.decode('utf-8') == '2'):
28         #Se il messaggio è 2: riceve ulteriori dati dal client presumibilmente un percorso di directory
29         data = connection.recv(1024)
30         try:
31             #Ottiene la lista dei file nella directory specificata
32             filelist = os.listdir(data.decode('utf-8'))
33             tosend = " "
34             for x in filelist:
35                 tosend += "," + x #Aggiunge i nomi dei file separati da virgola
36         except:
37             tosend = "Wrong path" #Se il percorso non è valido, invia "Wrong path"
38         connection.sendall(tosend.encode())
39     elif(data.decode('utf-8') == '0'):
40         #Se il messaggio è 0: chiude la connessione corrente e attende una nuova connessione
41         connection.close()
42         connection, address = s.accept()
43
44
```

Questo codice Python crea un **server** che si mette in ascolto su una porta specifica (SRV\_PORT = 1234) sull'indirizzo IP specificato (SRV\_ADDR). Quando un client si connette a questo server, il server accetta la connessione e stampa che il client si è connesso. Il codice quindi entra in un loop infinito (while 1:) per gestire la comunicazione con il client. Quando il server riceve dei dati dal client, decodifica il messaggio in UTF-8 e in base al contenuto del messaggio esegue delle azioni:

1. Se il messaggio ricevuto è 1, il server invia al client una stringa che contiene informazioni sulla piattaforma e la macchina utilizzate (platform.platform() e platform.machine()).
2. Se il messaggio ricevuto è 2, il server riceve ulteriori dati dal client, presumibilmente un percorso di directory. Il server tenta di ottenere la lista dei file nella directory specificata (os.listdir(data.decode('utf-8')) e invia al client la lista dei file separati da virgole. Se il percorso fornito non è valido, invia "Wrong path".
3. Se il messaggio ricevuto è 0, chiude la connessione corrente e si mette in attesa di accettare una nuova connessione.

Si tratta di un semplice server che fornisce informazioni sulla piattaforma e la macchina se riceve un certo input (1), elenca i file in una directory se riceve un altro input (2) e può chiudere la connessione se riceve un terzo input (0).

## CODICE 2

```
1  import socket
2
3  #Chiede all'utente di inserire l'indirizzo IP del server e la porta
4  SRV_ADDR = input("Type the server IP address: ")
5  SRV_PORT = int(input("Type the server port: "))
6
7  #Funzione per stampare il menu delle opzioni
8  def print_menu():
9      print("""\n\n0) Close the connection
10     1) Get system info
11     2) List directory contents""")
12
13  #Creazione del socket e connessione al server
14  my_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15  my_sock.connect((SRV_ADDR, SRV_PORT))
16
17  print("Connection established")
18  print_menu()
19
20  #Loop per la gestione del menu
21  while 1:
22      message = input("\n-Select an option: ") #Richiede all'utente di selezionare un'opzione
23
24      if(message == "0"):
25          #Se l'utente seleziona 0: invia il messaggio al server per chiudere la connessione
26          my_sock.sendall(message.encode())
27          my_sock.close() #Chiude la connessione
28          break #Esce dal loop
29
30      elif (message == "1"):
31          #Se l'utente seleziona 1: invia il messaggio al server per ottenere le informazioni di sistema
32          my_sock.sendall(message.encode()) #Invia il messaggio al server
33          data = my_sock.recv(1024) #Riceve i dati dal server
34          if not data: break #Se non ci sono dati, esce dal loop
35          print(data.decode('utf-8')) #Stampa le informazioni ricevute
36
37      elif (message == "2"):
38          #Se l'utente seleziona 2: richiede all'utente di inserire un percorso ed invia al server
39          path = input("Insert the path: ")
40          my_sock.sendall (message.encode()) #Invia il messaggio 2 al server
41          my_sock.sendall(path.encode()) #Invia il percorso al server
42          data = my_sock.recv(1024) #Riceve i dati dal server
43          data = data.decode('utf-8').split(",") #Decodifica e divide i dati ricevuti
44          print("*" * 40)
45          for x in data:
46              print(x) #Stampa i nomi dei file ricevuti dal server
```

Il codice è un **client** Python che si connette a un server remoto attraverso un socket TCP/IP. Preparazione della connessione: L'utente inserisce l'indirizzo IP del server e la porta a cui connettersi. Poi viene creato un socket di tipo SOCK\_STREAM per comunicare tramite TCP/IP (socket.AF\_INET).

I principali socket includono SOCK\_STREAM per la comunicazione affidabile orientata ai flussi (tipicamente utilizzato con TCP) e SOCK\_DGRAM per la comunicazione datagram (tipicamente utilizzato con UDP).

**SOCK\_STREAM:** È un tipo di socket che fornisce una connessione bidirezionale affidabile e ordinata tra due endpoint. Questo tipo di socket è orientato ai byte e gestisce la trasmissione di dati in sequenza, garantendo che i dati inviati vengano ricevuti nell'ordine corretto e senza perdita.

Il client si connette al server utilizzando l'indirizzo IP e la porta specificati.

L'interazione con l'utente ha 3 opzioni:

- Chiude la connessione al server e termina l'esecuzione del client (0)
- Invia una richiesta al server per ottenere informazioni di sistema (1)
- Chiede all'utente di inserire un percorso e invia questa richiesta al server per ottenere la lista dei file in quella directory (2)

Gestione delle richieste:

- Se l'utente seleziona 0, viene inviato un messaggio al server per chiudere la connessione e il client termina.
- Se l'utente seleziona 1, il client invia una richiesta al server per ottenere informazioni di sistema. Riceve i dati inviati dal server e li stampa.
- Se l'utente seleziona 2, il client richiede un percorso, lo invia al server e riceve una lista di file dalla directory specificata. Stampa i nomi dei file ricevuti.

In sostanza, **questo client si collega a un server remoto tramite socket e interagisce con esso inviando richieste specifiche, ricevendo i dati dal server e stampandoli a seconda dell'opzione scelta dall'utente. Il secondo codice è un client che interagisce con un server specifico.**

## DIFFERENZE

Le differenze principali tra i due codici:

Lato client e server:

Nel primo codice, viene implementato il codice per il server che accetta connessioni in entrata e gestisce le richieste dei client.

Nel secondo codice, viene implementato il codice per il client che si connette al server e invia richieste.

Logica di comunicazione:

Nel primo codice (lato server), il server aspetta richieste dal client e risponde in base al messaggio ricevuto.

Nel secondo codice (lato client), l'utente interagisce selezionando un'opzione dal menu e il client invia il messaggio corrispondente al server per ottenere informazioni di sistema o elencare i file.

Gestione delle connessioni:

Il primo codice gestisce più connessioni entranti poiché è un server che rimane in ascolto per accettare connessioni multiple.

Il secondo codice è progettato come client che si connette una sola volta al server, invia le richieste e poi chiude la connessione dopo aver terminato le operazioni.

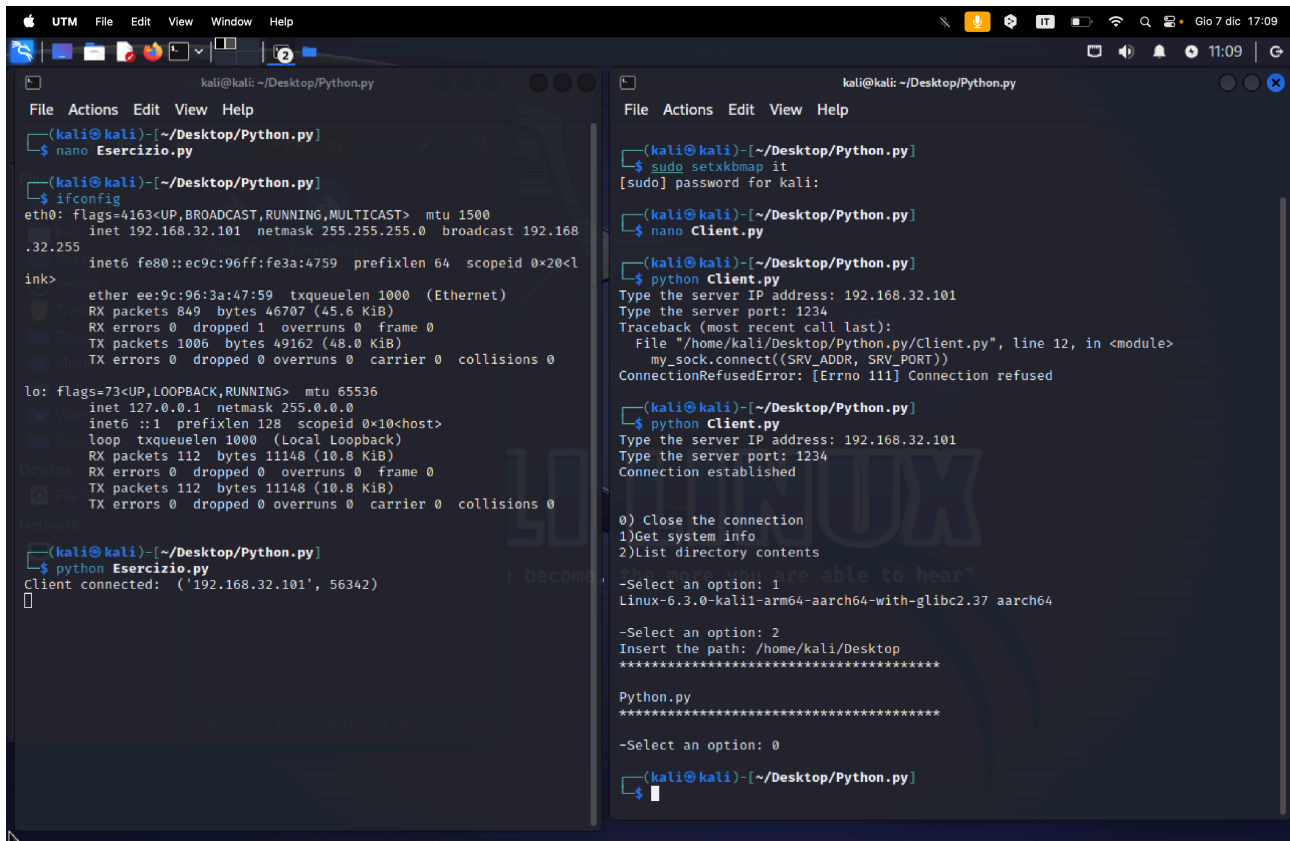
Interazione utente:

Nel primo codice, non c'è interazione diretta con l'utente del server; il server gestisce le richieste in modo automatico.

Nel secondo codice, il client interagisce con l'utente attraverso un menu testuale per selezionare le opzioni desiderate da inviare al server.

Il trojan è un software che può sembrare innocuo ma che appunto nasconde dietro una backdoor. I trojan possono nascondersi come server apparentemente innocui o possono sfruttare vulnerabilità nei server per eseguire azioni dannose. Il primo codice potrebbe essere un trojan.

## Testare praticamente il codice.



```
kali@kali: ~/Desktop/Python.py
File Actions Edit View Help
(kali@kali)~/Desktop/Python.py
$ nano Esercizio.py
(kali@kali)~/Desktop/Python.py
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.32.101 netmask 255.255.255.0 broadcast 192.168.32.255
    inet6 fe80::ec9c:96ff:fe3a:4759 prefixlen 64 scopeid 0x20<link>
    ether ee:9c:96:3a:47:59 txqueuelen 1000 (Ethernet)
    RX packets 849 bytes 46707 (45.6 KiB)
    RX errors 0 dropped 1 overruns 0 frame 0
    TX packets 1006 bytes 40162 (48.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 112 bytes 11148 (10.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 112 bytes 11148 (10.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

$ python Esercizio.py
Client connected: ('192.168.32.101', 56342)

(kali@kali)~/Desktop/Python.py
$ sudo setxkbmap it
[sudo] password for kali:
(kali@kali)~/Desktop/Python.py
$ nano Client.py
(kali@kali)~/Desktop/Python.py
$ python Client.py
Type the server IP address: 192.168.32.101
Type the server port: 1234
Traceback (most recent call last):
  File "/home/kali/Desktop/Python.py/Client.py", line 12, in <module>
    my_sock.connect((SRV_ADDR, SRV_PORT))
ConnectionRefusedError: [Errno 111] Connection refused

(kali@kali)~/Desktop/Python.py
$ python Client.py
Type the server IP address: 192.168.32.101
Type the server port: 1234
Connection established

0) Close the connection
1) Get system info
2) List directory contents
-Select an option: 1
Linux-6.3.0-kali1-arm64-aarch64-with-glibc2.37 aarch64

-Select an option: 2
Insert the path: /home/kali/Desktop
*****
Python.py
*****
-Select an option: 0

(kali@kali)~/Desktop/Python.py
$
```