ESERCIZIO S3/L4 MARTA ARAMU

Consegna: L'esercizio di oggi consiste nel commentare/spiegare questo codice che fa riferimento ad una backdoor. Inoltre spiegare cos'è una backdoor.

Svolgimento: in informatica, una "**backdoor**" (traducibile in italiano come "porta posteriore" o "accesso secondario") si riferisce a una vulnerabilità o a un meccanismo segreto che consente l'accesso non autorizzato a un sistema o a un'applicazione. Una backdoor può essere intenzionalmente inserita da un programmatore o un utente autorizzato per scopi di manutenzione o di accesso di emergenza, ma può anche essere sfruttata malevolmente da terze parti. Le backdoor possono assumere diverse forme, come una password nascosta, un'applicazione segreta, una vulnerabilità del software o una porta di rete non documentata. Spesso, gli attaccanti cercano di sfruttare backdoor per ottenere accesso non autorizzato a sistemi informatici al fine di comprometterli, rubare dati sensibili o eseguire attività dannose.

Questo codice python implementa un semplice server che ascolta su una porta specifica (1234) per connessioni TCP. Il server gestisce comandi inviati da un client e risponde di conseguenza.
La riga di codice import socket, platform, os è un'espressione di importazione in python. In questo contesto, sta dicendo al programma di rendere disponibili le funzionalità fornite dai moduli socket, platform e os. Ecco cosa fa ciascun modulo:
socket: questo modulo fornisce un'interfaccia per la creazione e la gestione di socket, che sono oggetti utilizzati per la comunicazione tra processi su reti. In questo codice, il modulo socket viene utilizzato per creare un server che ascolta su una porta specifica per le connessioni in arrivo.
platform: questo modulo offre funzionalità per ottenere informazioni sulla piattaforma su cui python è in esecuzione. In questo codice, il modulo platform viene utilizzato per ottenere informazioni sulla piattaforma (ad esempio, il sistema operativo e l'architettura della macchina) e inviarle al client quando viene ricevuto il comando '1'.
os: questo modulo fornisce una serie di funzioni per interagire con il sistema operativo. In questo codice, il modulo os viene utilizzato per ottenere la lista dei file in una directory specifica, in risposta al comando '2' inviato dal client.

```
import socket, platform, os

# Impostazione dell'indirizzo IP del server e della porta
SRV_ADDR = ""
SRV_PORT = 1234
```

Successivamente viene creato un oggetto socket e questo viene associato all'indirizzo e alla porta specificati precedentemente. Poi il socket viene impostato in modalità ascolto e quell'1 tra parentesi indica che può gestire una sola connessione in questo caso. Si accetta la connessione in arrivo e con la funzione print si stampa l'indirizzo del client appena connesso.

```
# Creazione di un oggetto socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Associazione del socket all'indirizzo e alla porta specificati
s.bind((SRV_ADDR, SRV_PORT))

# Impostazione del socket in modalità ascolto (può gestire una sola cor
s.listen(1)

# Accettazione di una connessione in arrivo
connection, address = s.accept()

# Stampa dell'indirizzo del client appena connesso
print("client connected:", address)
```

Si inizia un loop infinito per gestire i comandi del client. Il loop infinito è implementato per mantenere attivo il server in modo continuo, consentendo di gestire più comandi inviati da client diversi o dallo stesso client nel corso del tempo. Quando il server riceve una connessione da un client, entra nel loop e inizia ad ascoltare costantemente i comandi inviati dal client corrente. Questo programma consente di ricevere massimo 1024 byte alla volta.

```
# Inizio di un loop infinito per gestire i comandi del client
while 1:
    try:
        # Ricezione dei dati inviati dal client (max 1024 byte alla vol
        data = connection.recv(1024)
    except:
        continue
```

Ora è il momento di verificare il comando ricevuto dal client. I comandi possibili sono:
'1': Restituisce informazioni sulla piattaforma e sull'architettura della macchina.
'2': Riceve un percorso dal client e restituisce la lista dei file in quel percorso.
'0': Chiude la connessione corrente e accetta una nuova connessione in arrivo.
In dettaglio, il blocco try inizia provando ad ottenere la lista dei file nella directory specificata dal client. Questo viene fatto utilizzando la funzione os.listdir(), che restituisce una lista di nomi di file nella directory specificata. Se l'operazione ha successo, il codice all'interno del blocco try viene eseguito senza errori. Viene quindi inizializzata una stringa vuota tosend, e viene iterato attraverso la lista dei file. Ogni nome di file viene aggiunto alla stringa tosend, preceduto da una virgola. Se si verifica un'eccezione (ad esempio, se il percorso specificato non esiste o non è accessibile), il blocco except viene eseguito. In questo caso, la stringa tosend viene impostata su "Wrong path" per indicare al client che il percorso fornito non è

valido. Infine, la stringa tosend viene convertita in un formato codificato (in questo caso, utilizzando UTF-8) e inviata al client tramite la connessione attiva (connection.sendall(tosend.encode())).

```
# Verifica del comando ricevuto dal client
if data.decode('utf-8') == '1':
    # Invio delle informazioni sulla piattaforma al client
    tosend = platform.platform() + " " + platform.machine()
    connection.sendall(tosend.encode())
elif data.decode('utf-8') == '2':
    # Ricezione di un percorso dal client e invio della lista di file n
    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':
    # Chiusura della connessione con il client
    connection.close()
    # Accettazione di una nuova connessione
    connection, address = s.accept()
```

Il codice potrebbe essere un esempio di backdoor poiché, pur sembrando inizialmente un server che risponde a comandi inviati da un client, potrebbe contenere vulnerabilità che consentono accesso non autorizzato o esecuzione di comandi malevoli.
Ad esempio, il blocco di codice che gestisce il comando '2' sembra ricevere un percorso dal client, tenta di ottenere la lista dei file in quella directory e invia la lista al client. Tuttavia, se un utente malintenzionato riesce a inviare un percorso dannoso, potrebbe essere possibile sfruttare questa parte del codice per eseguire comandi malevoli sul sistema.