Guía para proteger una base de datos MySQL contra ataques SQL Injection y otros riesgos de seguridad

La seguridad de MySQL es esencial para prevenir ataques como **SQL Injection**, accesos no autorizados y pérdidas de datos. A continuación, te dejo una guía estructurada para proteger tu base de datos.

---

1. Protección contra SQL Injection

a) Usa consultas preparadas y declaraciones parametrizadas

Las consultas preparadas impiden que los atacantes inserten código SQL malicioso. Ejemplo en **Python (con MySQL y Flask)**:

```
import mysql.connector

db = mysql.connector.connect(host="localhost", user="user",
password="password", database="test_db")
cursor = db.cursor(prepared=True)

query = "SELECT * FROM users WHERE username = %s AND password = %s"
cursor.execute(query, (username, password))

result = cursor.fetchall()
```

Evita construir consultas dinámicas con format() o concatenación de strings:

```
query = "SELECT * FROM users WHERE username = '{}' AND password =
'{}'".format(username, password)  #
```

❌ ¡No hagas esto!

b) Usa ORM en lugar de consultas directas

Si usas Django, Sequelize (Node.js) o SQLAlchemy (Python), estos frameworks manejan internamente la seguridad:

- **Django (Python)**
- user = User.objects.filter(username=username, password=password).first()
- **Sequelize (Node.js)**
- User.findOne({ where: { username, password } })

---

2. Configuración Segura de MySQL

a) Restringe el acceso a la base de datos

- Usa contraseñas seguras y evita usuarios predeterminados (root).
- Crea un usuario con permisos mínimos:
- CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'ContraseñaSegura!';

- GRANT SELECT, INSERT, UPDATE, DELETE ON my_database.* TO 'usuario'@'localhost';
- FLUSH PRIVILEGES;

b) No expongas MySQL en internet

- **Cambia el puerto por defecto (3306)** en /etc/mysql/my.cnf:
- [mysqld]
- port = 3307
- **Deshabilita el acceso remoto** si no es necesario:
- bind-address = 127.0.0.1

c) Habilita TLS/SSL

Para cifrar la comunicación entre cliente y servidor:

[mysqld]

require_secure_transport = ON

Y asegúrate de usar --ssl-mode=REQUIRED en las conexiones.

---

3. Seguridad en los Datos

a) Cifra datos sensibles

No almacenes contraseñas en texto plano. Usa **bcrypt** para el hashing:

Python con bcrypt

```
import bcrypt
password = "mi_contraseña"
hashed_password = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
```

b) Usa permisos en nivel de tabla o columna

Si solo un usuario necesita acceso a una tabla específica:

```
GRANT SELECT ON my_database.usuarios TO 'usuario_limitado'@'localhost';
```

---

4. Protección contra Fuerza Bruta y DDoS

a) Usa un firewall de base de datos

Configura reglas en **iptables** o **UFW**:

sudo ufw allow 3306

sudo ufw deny from 192.168.1.100

b) Limita intentos de conexión fallidos

Usa **fail2ban** para bloquear ataques de fuerza bruta:

1. Crea un archivo /etc/fail2ban/jail.local con:
2. [mysqld-auth]
3. enabled = true
4. filter = mysqld-auth
5. logpath = /var/log/mysql/error.log
6. maxretry = 5
7. Reinicia fail2ban:
8. sudo systemctl restart fail2ban

---

5. Auditoría y Monitoreo

a) Habilita logs de consultas lentas y errores

Edita /etc/mysql/my.cnf:

[mysqld]

log_error = /var/log/mysql/error.log

slow_query_log = 1

long_query_time = 2

slow_query_log_file = /var/log/mysql/slow_queries.log

b) Revisa accesos sospechosos

Consulta conexiones activas:

SHOW PROCESSLIST;

O revisa el historial de autenticaciones:

SELECT * FROM mysql.general_log WHERE command_type = 'Connect';

Conclusión

Siguiendo estas prácticas, puedes proteger tu base de datos MySQL contra SQL Injection y otros ataques.
Prioriza: ✅ Uso de consultas preparadas.
✅ Usuarios con permisos mínimos.
✅ Cifrado de datos sensibles.
✅ Firewall y monitoreo activo.

Si necesitas más detalles, dime qué parte te interesa reforzar. 🚀