SQL-инъекции: примеры и защита от них

SQL-инъекция – это один из самых известных и опасных видов атак на базы данных. Она возникает, когда программа подставляет данные пользователя прямо в SQL-запрос **без проверки и экранирования**.

Если приложение уязвимо к SQL-инъекциям, злоумышленник может:

- читать и изменять данные в базе;
- удалять таблицы;
- обходить авторизацию;
- полностью контролировать систему.

SQLite не является исключением: он тоже подвержен таким атакам, если код написан небезопасно.

Пример уязвимого кода

Рассмотрим простой пример на Python:

```
import sqlite3

conn = sqlite3.connect("students.db")
cursor = conn.cursor()

email = input("Введите email: ")

query = f"SELECT * FROM students WHERE email =
'{email}'"
cursor.execute(query)

print(cursor.fetchall())
```

Что здесь происходит:

- Пользователь вводит email.
- Email напрямую подставляется в SQL-запрос через f-строку.
- Если ввести корректный адрес — всё работает.

⚠ Но если ввести такую строку:

```
' OR 1=1 --
```

SQL-запрос станет:

```
SELECT * FROM students WHERE email = '' OR 1=1 --'
```

- `OR 1=1` всегда возвращает истину.
- `--` превращает оставшийся код в комментарий.
- В итоге злоумышленник получает **всех студентов**, а не только одного.

Другие примеры атак

Удаление таблицы

Пусть программа делает такой запрос:

```
query = f"DELETE FROM students WHERE name = '{name}'"
cursor.execute(query)
```

Если злоумышленник введёт:

```
'; DROP TABLE students; --
```

Запрос превратится в:

```
DELETE FROM students WHERE name = '';
DROP TABLE students; --
```

⚠ Таблица будет удалена.

Обход авторизации

Предположим, есть простая проверка логина и пароля:

```
query = f"SELECT * FROM users WHERE login = '{login}'
AND password = '{password}'"
```

Злоумышленник вводит:

- логин: `admin`
- пароль: `' OR '1'='1`

Запрос станет:

```
SELECT * FROM users WHERE login = 'admin' AND password
= '' OR '1'='1'
```

Так как 1=1 всегда верно, вход будет разрешён без пароля.

Как защищаться от SQL-инъекций

1. Использовать параметризованные запросы

Это основной и надёжный способ. В Python с sqlite3 это выглядит так:

```
cursor.execute("SELECT * FROM students WHERE email =
?", (email,))
```

- Символ ? — это плейсхолдер.
- Значение передаётся отдельно и обрабатывается безопасно.
- Даже если пользователь введёт `'; DROP TABLE students; --`, это будет воспринято как **строка**, а не как SQL-команда.

2. Никогда не конкатенировать строки

⚠ Нельзя писать так:

```
query = "SELECT * FROM students WHERE email = '" +
email + "'"
```

или так:

```
query = f"SELECT * FROM students WHERE email =
'{email}'"
```

Это делает код уязвимым.

3. Использовать ORM (например, SQLAlchemy)

В реальных проектах часто применяют ORM (Object Relational Mapping). Пример:

```
from sqlalchemy import create_engine, Table, Column,
Integer, String, MetaData

engine = create_engine("sqlite:///students.db")
metadata = MetaData()

students = Table(
    "students", metadata,
    Column("id", Integer, primary_key=True),
    Column("name", String),
    Column("age", Integer),
    Column("email", String, unique=True),
```

```
)

conn = engine.connect()
result =
conn.execute(students.select().where(students.c.email
== "ivan@example.com"))
for row in result:
    print(row)
```

ORM автоматически экранирует параметры и защищает от инъекций.