SQL-инъекции

Причина возникновения

class `sqlite3.`**`Cursor`**

A `Cursor` instance has the following attributes and methods.

`execute`(*`sql`*[, *`parameters`*])

Executes an SQL statement. Values may be bound to the statement using placeholders.

`execute()` will only execute a single SQL statement. If you try to execute more than one statement with it, it will raise a `Warning`. Use `executescript()` if you want to execute multiple SQL statements with one call.

Как надо делать:

```
@db.execute(
  "select rowid, name, password from Users where name = ?",
  [name]
```

Как не надо делать:

```
@db.execute(
  "select rowid, name, password from Users where name = #{name}"
) do |row|
```

Как не надо делать:

```ruby
@db.execute(
  "select rowid, name, password from Users where name = #{name}"
) do |row|
```

```
name = "(select password from users where name = 'admin')"
```

Как не надо делать

```
name_q = mysql_escape_string(name)
@db.execute(
  "select rowid, name, password from Users where name = #{name_q}"
) do |row|
```

Как не надо делать

```
name_q = mysql_real_escape_string(name, @db)
@db.execute(
  "select rowid, name, password from Users where name = #{name_q}"
) do |row|
```

Экранирование - это небезопасно

- Есть где ошибиться

```
O'Connor -> O\'Connor

O\'Connor -> O\\'Connor
```

Экранирование - это небезопасно

- Двойное экранирование

```
hello world -> "hello world"

`echo '${s}'` -> `echo '"hello world"'`
```

Экранирование - это небезопасно

- Оно бывает разным в разных местах

```
hello world -> hello%20world -> hello%%20world
```

Вывод: не нужно экранировать

- Нужно использовать Query Parameters

```
@db.execute(
  "select rowid, name, password from Users where name = ?",
  [name]
```

Вывод: не нужно экранировать

- Нужно использовать Query Parameters

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
              "VALUES (:id, :name, :salary)");
query.bindValue(":id", 1001);
query.bindValue(":name", "Thad Beaumont");
query.bindValue(":salary", 65000);
query.exec();
```

Query Parameters - это небезопасно

- Можно забыть добавить параметр

```
@db.execute(
  "insert into Users (name, password ) values (?, ?)",
  [name]
)
```

Query Parameters - это небезопасно

- Можно забыть добавить параметр

```
@db.execute(
  "insert into Users (name, password ) values (?, ?)",
  [name]
)
```

- Точно также можно забыть поменять местами параметры и т.д.

Query Parameters - это небезопасно

- Можно забыть добавить параметр

```
@db.execute(
  "insert into Users (name, password ) values (?, ?)",
  [name]
)
```

- Точно также можно забыть поменять местами параметры и т.д.
- Type Confusion - можно использовать число там где подразумевается строка, и т.п.

Вывод: не нужно использовать SQL напрямую

- ORM - object-relational mapping
- Для python: SQLAlchemy, peewee
- Для C++: Qt
- Для C#: LINQ, Entity Framework
- Для PHP: Laravel, YII

Как это выглядит

```
class Person(Model):
    name = CharField()
    birthday = DateField()

class Pet(Model):
    owner = ForeignKeyField(Person, backref='pets')
    name = CharField()
    animal_type = CharField()

grandma = Person.create(name='Grandma', birthday=date(1935, 3, 1))
mittens = Pet.create(owner=grandma, name='Mittens', animal_type='cat')
print(grandma.pets) # > ["Mittens"]
```

Виды sql-инъекций

- Инъекция с видимым результатом
  - Кто-то забыл выключить отладочный режим
  - Хотя бы видно select-нутый контент

Виды sql-инъекций

- Инъекция с видимым результатом
  - Кто-то забыл выключить отладочный режим
  - Хотя бы видно select-нутый контент
- Слепая инъекция

Слепая инъекция

```
insert into users (name, password) values ('user', '${pwd}')

pwd = ' || substring((select password from users where name = 'admin'), 1, 1) || '
```

Слепая инъекция

```
insert into users (name, password) values ('user', '${pwd}')

pwd = ' || select
    case when substring((select password from users where name = 'admin'), 1, 1) = 'a'
        then 1/0
        else 'a'
    end || '
```