**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №5**

**по дисциплине «Базы данных»**

**Тема: Тестирование БД на безопасность**

| Студентка гр. 2384 |  | Соц Е.А. |
| --- | --- | --- |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2024

## **Цель работы**

Создать web-сервер для выполнения запросов. Изучить понятие sql-инъекций, их виды и протестировать базу данных на безопасность.

## **Задание**

Вариант 18

1. Сделать простой web-сервер для выполнения запросов из ЛР3, например с express.js. Не обязательно делать авторизацию и т.п., хватит одного эдпоинта на каждый запрос, с параметрами запроса как query parameters.
2. Намеренно сделайте несколько (2-3) запроса, подверженных SQL-инъекциям
3. Проверьте Ваше API с помощью sqlmap (или чего-то аналогичного), передав эндпоинты в качестве целей атаки. Посмотрите, какие уязвимости он нашёл (и не нашёл), опишите пути к исправлению.
4. \*+2 балла, если напишете эндпоинт с уязвимостью, которая не находится sqlmap-ом.

## 

## **Выполнение работы**

Был создан простой web-сервер для выполнения запросов из третьей лабораторной работы с помощью express.js. Для выполнения эдпоинтов на каждый запрос написаны функции get() с параметрами запроса как query parameters.

Код и тестирование запросов приведены ниже.

Код для создания сервера находится в приложении А.

1. Автор текста, композитор и дата создания песни с данным названием? В репертуар какой группы она входит?

async getTrackInfo(trackName) {

const track = await Track.findOne({

where: { track\_name: trackName },

include: {

model: MusicalGroup,

attributes: ['group\_name']

}

});

if (!track) {

console.log(`Track not found: ${trackName}`);

return { error: 'Track not found' };

}

const result = {

composer: track.composer,

lyricsAuthor: track.lyrics\_author,

releaseDate: track.release\_date,

groupName: track.MusicalGroup.group\_name

};

console.log(`Track Info:`, result);

return result;

},

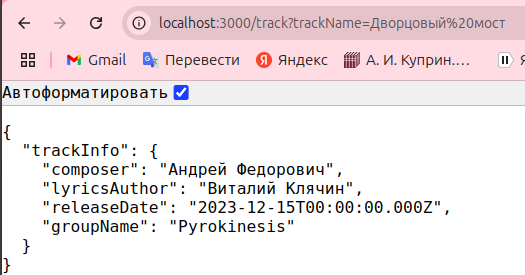


Рисунок 1 – Информация о треке

1. Репертуар наиболее популярной группы?

async getMostPopularGroupRepertoire() {

const mostPopularGroup = await MusicalGroup.findOne({

order: [['chart\_position', 'ASC']]

});

if (!mostPopularGroup) {

console.log('No popular group found');

return { error: 'No popular group found' };

}

const tracks = await Track.findAll({

where: { musical\_group\_id: mostPopularGroup.musical\_group\_id }

});

const result = tracks.map(track => ({

trackName: track.track\_name,

composer: track.composer,

lyricsAuthor: track.lyrics\_author,

releaseDate: track.release\_date

}));

console.log(`Repertoire for ${mostPopularGroup.group\_name}:`, result);

return result;

},

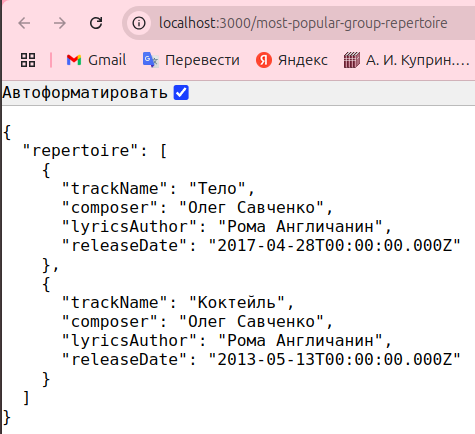


Рисунок 2 – Репертуар популярной группы

1. Цена билета на последний концерт указанной группы?

async getLastConcertTicketPrice(groupName) {

const group = await MusicalGroup.findOne({

where: { group\_name: groupName }

});

if (!group) {

console.log(`Group not found: ${groupName}`);

return { error: 'Group not found' };

}

const tours = await Tour.findAll({

where: { musical\_group\_id: group.musical\_group\_id }

});

const concerts = await ConcertInTour.findAll({

where: { tour\_id: tours.map(tour => tour.tour\_id) },

include: {

model: Concert,

attributes: ['date\_concert', 'ticket\_price']

},

order: [[Concert, 'date\_concert', 'DESC']]

});

if (concerts.length === 0) {

console.log(`No concerts found for group: ${groupName}`);

return { error: 'No concerts found' };

}

const result = concerts[0].Concert.ticket\_price;

console.log(`Last Concert Ticket Price for ${groupName}:`, result);

return result;

},

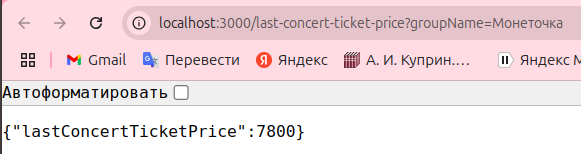


Рисунок 3 – Цена на последний концерт

1. Состав исполнителей группы с заданным названием, их возраст и амплуа?

async getGroupMembers(groupName) {

const group = await MusicalGroup.findOne({

where: { group\_name: groupName }

});

if (!group) {

console.log(`Group not found: ${groupName}`);

return { error: 'Group not found' };

}

const members = await RoleOfMember.findAll({

where: { musical\_group\_id: group.musical\_group\_id },

include: [

{

model: Member,

attributes: ['member\_name', 'surname', 'birth\_date']

},

{

model: MusicalRole,

attributes: ['role\_name']

}

]

});

const result = members.map(member => ({

name: member.Member.member\_name,

surname: member.Member.surname,

age: new Date().getFullYear() - new Date(member.Member.birth\_date).getFullYear(),

role: member.MusicalRole.role\_name

}));

console.log(`Members of ${groupName}:`, result);

return result;

},

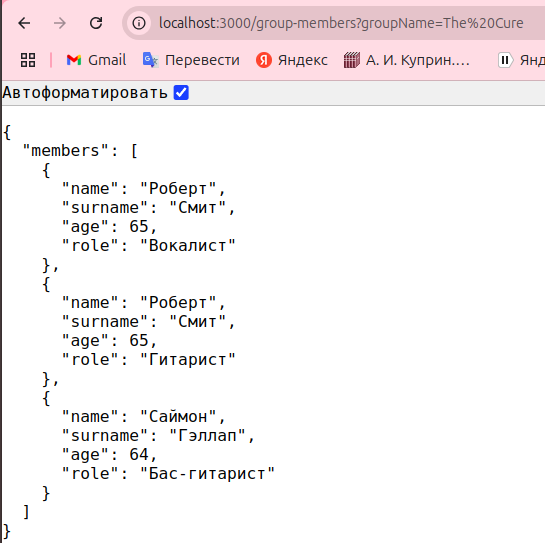


Рисунок 4 – Состав исполнителей

1. Место и продолжительность гастролей группы с заданным названием?

async getGroupTourInfo(groupName) {

const group = await MusicalGroup.findOne({

where: { group\_name: groupName }

});

if (!group) {

console.log(`Group not found: ${groupName}`);

return { error: 'Group not found' };

}

const tours = await Tour.findAll({

where: { musical\_group\_id: group.musical\_group\_id },

include: {

model: ConcertInTour,

include: {

model: Concert,

attributes: ['city']

}

}

});

const result = tours.map(tour => {

const duration = new Date(tour.end\_day) - new Date(tour.start\_day);

return {

tourName: tour.tour\_name,

duration,

cities: tour.ConcertInTours.map(concertInTour => concertInTour.Concert.city)

};

});

console.log(`Tour info for ${groupName}:`, result);

return result;

},

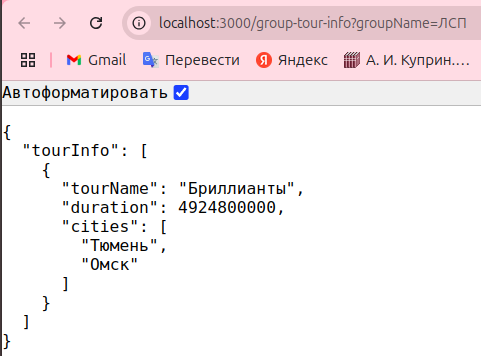


Рисунок 5 – Гастроли определенной группы

1. Какие группы в текущем году отмечают юбилей?

async getAnniversaryGroups() {

const currentYear = new Date().getFullYear();

const groups = await MusicalGroup.findAll({

where: sequelize.literal(`(${currentYear} - year\_foundation) % 5 = 0 AND ${currentYear} - year\_foundation > 0`)

});

const result = groups.map(group => ({

groupName: group.group\_name,

foundationYear: group.year\_foundation,

anniversary: currentYear - group.year\_foundation

}));

console.log(`Anniversary groups for ${currentYear}:`, result);

return result;

},

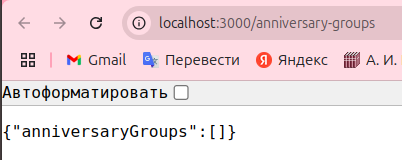


Рисунок 6 – Юбилей в текущем году

На выход поступает пустота, так как в текущем году по тестовым наполнениям таблиц действительно никакие группы в текущем году не отмечают юбилей. Для проверки работоспособности запроса был написан запрос, в который передается конкретный год (описан ниже).

1. Группы, отмечающие юбилей в конкретном году

async getAnniversaryGroupsTest(myYear) {

const groups = await MusicalGroup.findAll({

where: sequelize.literal(`(${myYear} - year\_foundation) % 5 = 0 AND ${myYear} - year\_foundation > 0`)

});

const result = groups.map(group => ({

groupName: group.group\_name,

foundationYear: group.year\_foundation,

anniversary: myYear - group.year\_foundation

}));

console.log(`Anniversary groups for ${myYear}:`, result);

return result;

},

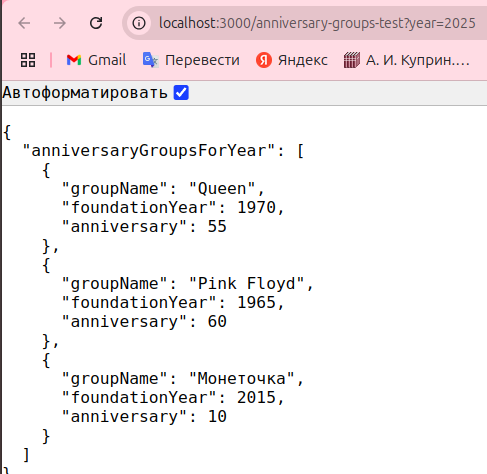


Рисунок 7 – Юбилей в 2025 году

1. Самый молодой вокалист? Какую группу он представляет?

async getYoungestVocalist() {

const vocalist = await RoleOfMember.findOne({

where: { musical\_role\_id: 1 },

include: [

{

model: Member,

attributes: ['member\_name', 'surname', 'birth\_date']

},

{

model: MusicalGroup,

attributes: ['group\_name']

}

],

order: [[Member, 'birth\_date', 'DESC']]

});

if (!vocalist) {

console.log('No vocalist found');

return { error: 'No vocalist found' };

}

const age = new Date().getFullYear() - new Date(vocalist.Member.birth\_date).getFullYear();

const result = {

name: vocalist.Member.member\_name,

surname: vocalist.Member.surname,

age,

groupName: vocalist.MusicalGroup.group\_name

};

console.log('Youngest Vocalist:', result);

return result;

}

};

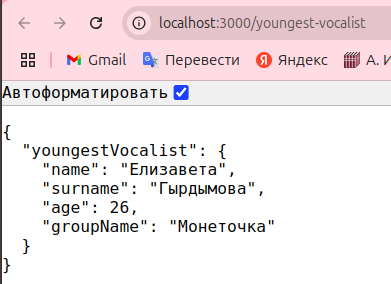


Рисунок 8 – Самый молодой вокалист

Все запросы успешно отработали на сервере.

Далее намеренно были созданы несколько запросов с sql-инъекциями. Код запросов, результаты инъекций, а также тестирование sqlmapом приведены ниже.

1. Автор текста, композитор и дата создания песни с данным названием? В репертуар какой группы она входит?



Рисунок 9 – Тестирование написанного запроса

Теперь попытаемся сломать запрос:

async getTrackInfo(trackName) {

const query = `SELECT \* FROM track WHERE track\_name = '${trackName}'`;

const result = await sequelize.query(query, { type: QueryTypes.SELECT });

return result;

},

Выполнение самостоятельного тестирования:

*' OR 1=1 –*

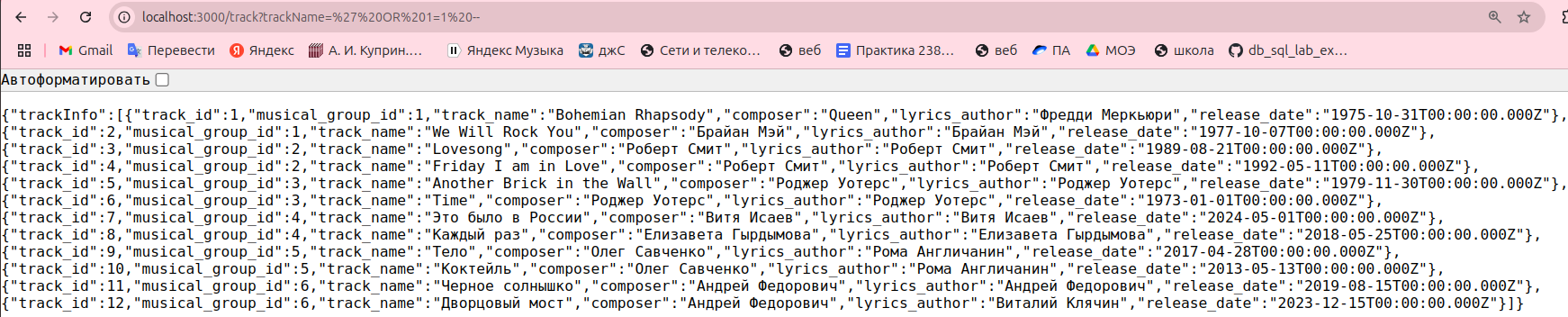


Рисунок 10 – Выполнение инъекции boolean-based

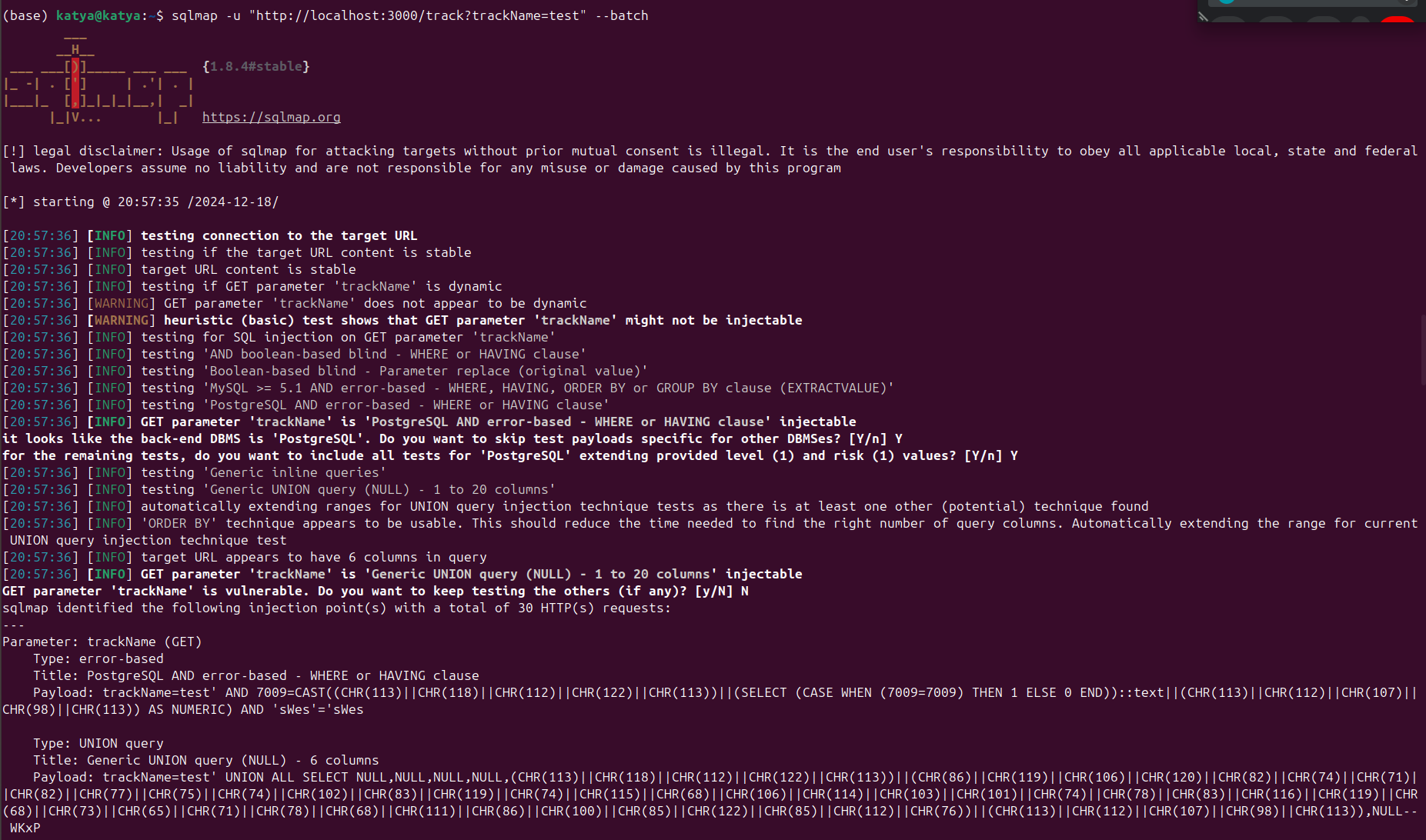


Рисунок 11 – Тестирование map-oм

Можно заметить, что sqlmap не нашел именно инъекцию boolean-based, значит, четвертое задание в данной работе можно считать выполненным. SQLmap — это отличный инструмент, но он не совершенен. Для сложных сценариев может потребоваться ручное тестирование и глубокий анализ структуры запросов и приложения.

1. Цена билета на последний концерт указанной группы?

Запрос, написанный ранее, оказался безопасным, поэтому, будет также ломать его:

async getLastConcertTicketPrice(groupName) {

const query = `

SELECT c.ticket\_price

FROM concert AS c

JOIN concert\_in\_tour AS cit ON c.concert\_id = cit.concert\_id

JOIN tour AS t ON cit.tour\_id = t.tour\_id

JOIN musical\_group AS mg ON t.musical\_group\_id = mg.musical\_group\_id

WHERE mg.group\_name = '${groupName}'

ORDER BY c.date\_concert DESC

LIMIT 1

`;

try {

const result = await sequelize.query(query, { type: QueryTypes.SELECT });

console.log(`Last Concert Ticket Price for ${groupName}:`, result);

return result;

} catch (error) {

console.error('Error executing query:', error);

throw new Error('Database query failed');

}

},

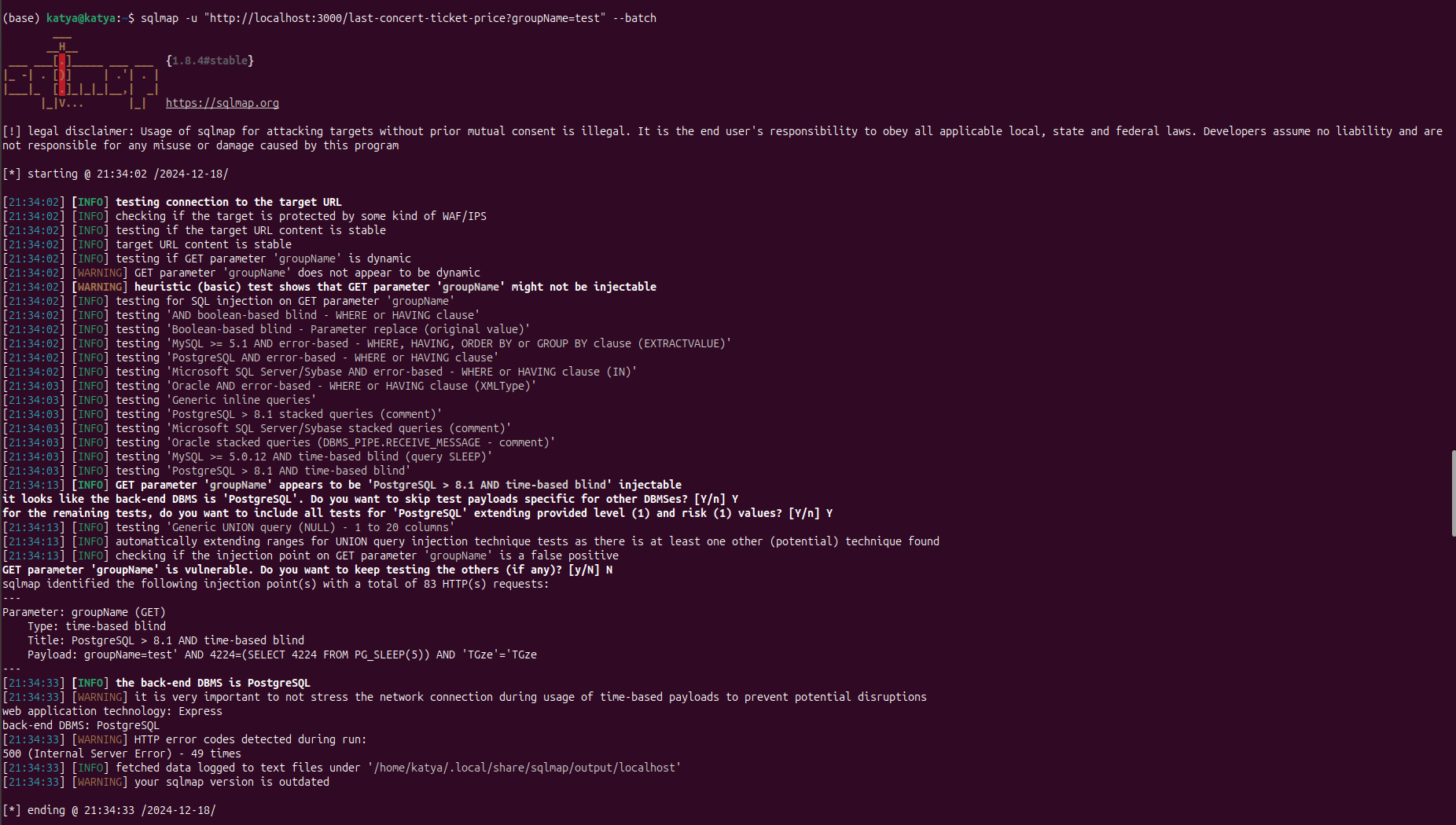


Рисунок 12 – Тестирование sqlmap

Можно заметить, что утилита нашла time-based инъекцию. Действительно, при запуске такой инъекции время ожидания ответа возрастает – написанный запрос подвергается инъекции.

1. Информация о туре

Переписанный запрос:  
async getGroupTourInfo(groupName) {

const query = `

SELECT \* FROM musical\_group WHERE group\_name = '${groupName}'

`;

try {

const groups = await sequelize.query(query, { type: QueryTypes.SELECT });

if (groups.length === 0) {

console.log(`Group not found: ${groupName}`);

return { error: 'Group not found' };

}

const group = groups[0];

const tours = await Tour.findAll({

where: { musical\_group\_id: group.musical\_group\_id },

include: {

model: ConcertInTour,

include: {

model: Concert,

attributes: ['city']

}

}

});

const result = tours.map(tour => {

const duration = new Date(tour.end\_day) - new Date(tour.start\_day);

return {

tourName: tour.tour\_name,

duration,

cities: tour.ConcertInTours.map(concertInTour => concertInTour.Concert.city)

};

});

console.log(`Tour info for ${groupName}:`, result);

return result;

} catch (error) {

console.error('Error retrieving group tour info:', error.message);

throw new Error('Database query failed');

}

},

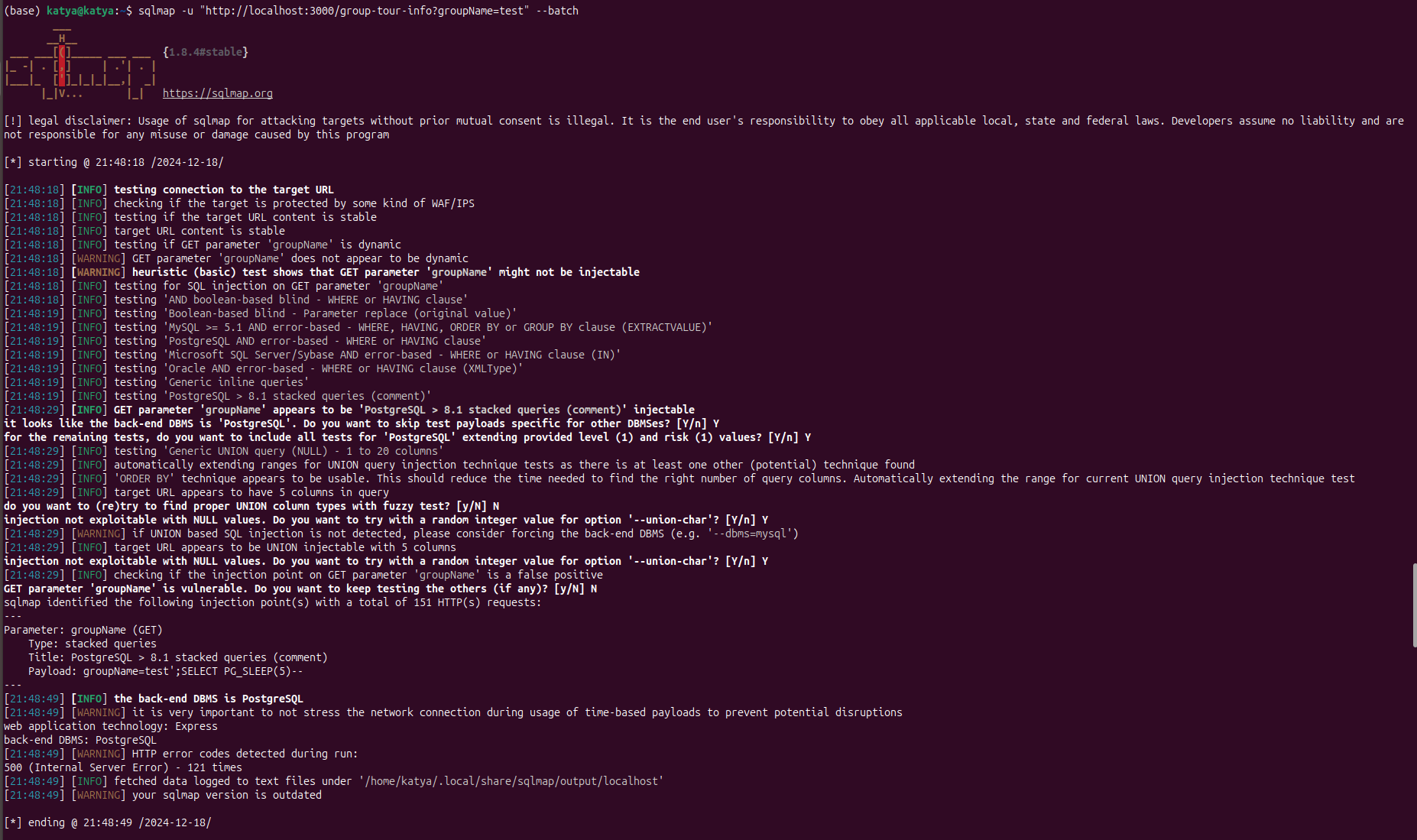


Рисунок 13 – Тестирование sqlmap

Видно, что утилита нашла инъекцию stacked, а значит пользователь может поставить “;” после названия группы и написать любое действие в запросе, например: DROP TABLE или, как приведено в терминале, “groupName=test';SELECT PG\_SLEEP(5)--”.

## **Вывод**

В ходе данной лабораторной работы был разработан web-сервер для выполнения запросов, каждый запрос был протестирован. Была изучена утилита SQLmap, помогающая тестировать запросы на SQL-инъекции. Также было замечено, что она работает не идеально: не всегда определяет все виды инъекций, а значит каждый запрос все еще требует ручного тестирования.

## **ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

const express = require('express');

const queries = require('./unsafe\_queries');

const app = express();

const PORT = 3000;

// Endpoint: Get track info

app.get('/track', async (req, res) => {

try {

const { trackName } = req.query;

//if (!trackName) return res.status(400).send({ error: 'trackName query parameter is required' });

const result = await queries.getTrackInfo(trackName);

res.json({ trackInfo: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get repertoire of the most popular group

app.get('/most-popular-group-repertoire', async (req, res) => {

try {

const result = await queries.getMostPopularGroupRepertoire();

res.json({ repertoire: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get last concert ticket price

app.get('/last-concert-ticket-price', async (req, res) => {

try {

const { groupName } = req.query;

//if (!groupName) return res.status(400).send({ error: 'groupName query parameter is required' });

const result = await queries.getLastConcertTicketPrice(groupName);

res.json({ lastConcertTicketPrice: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get group members

app.get('/group-members', async (req, res) => {

try {

const { groupName } = req.query;

//if (!groupName) return res.status(400).send({ error: 'groupName query parameter is required' });

const result = await queries.getGroupMembers(groupName);

res.json({ members: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get group tour info

app.get('/group-tour-info', async (req, res) => {

try {

const { groupName } = req.query;

//if (!groupName) return res.status(400).send({ error: 'groupName query parameter is required' });

const result = await queries.getGroupTourInfo(groupName);

res.json({ tourInfo: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get anniversary groups

app.get('/anniversary-groups', async (req, res) => {

try {

const result = await queries.getAnniversaryGroups();

res.json({ anniversaryGroups: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get anniversary groups for a specific year

app.get('/anniversary-groups-test', async (req, res) => {

try {

const { year } = req.query;

//if (!year) return res.status(400).send({ error: 'year query parameter is required' });

const result = await queries.getAnniversaryGroupsTest(Number(year));

res.json({ anniversaryGroupsForYear: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Endpoint: Get youngest vocalist

app.get('/youngest-vocalist', async (req, res) => {

try {

const result = await queries.getYoungestVocalist();

res.json({ youngestVocalist: result });

} catch (error) {

console.error(error);

res.status(500).send({ error: error.message });

}

});

// Start the server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

// http://localhost:3000/track?trackName=Дворцовый мост

// http://localhost:3000/most-popular-group-repertoire

// http://localhost:3000/last-concert-ticket-price?groupName=Монеточка

// http://localhost:3000/group-members?groupName=The Cure

// http://localhost:3000/group-tour-info?groupName=ЛСП

// http://localhost:3000/anniversary-groups

// http://localhost:3000/anniversary-groups-test?year=2025

// http://localhost:3000/youngest-vocalist

## **ПРИЛОЖЕНИЕ В**

**ССЫЛКА НА PR**

<https://github.com/moevm/sql-2024-2384/pull/28>