

# Relatório de Prática: SQL Injection e Mitigação

Alunos:

Amanda Prado	RA 235343
Kalled Abdala	RA 234846
Leandro Lanzoni	RA 234840
Maria Neri	RA 234910

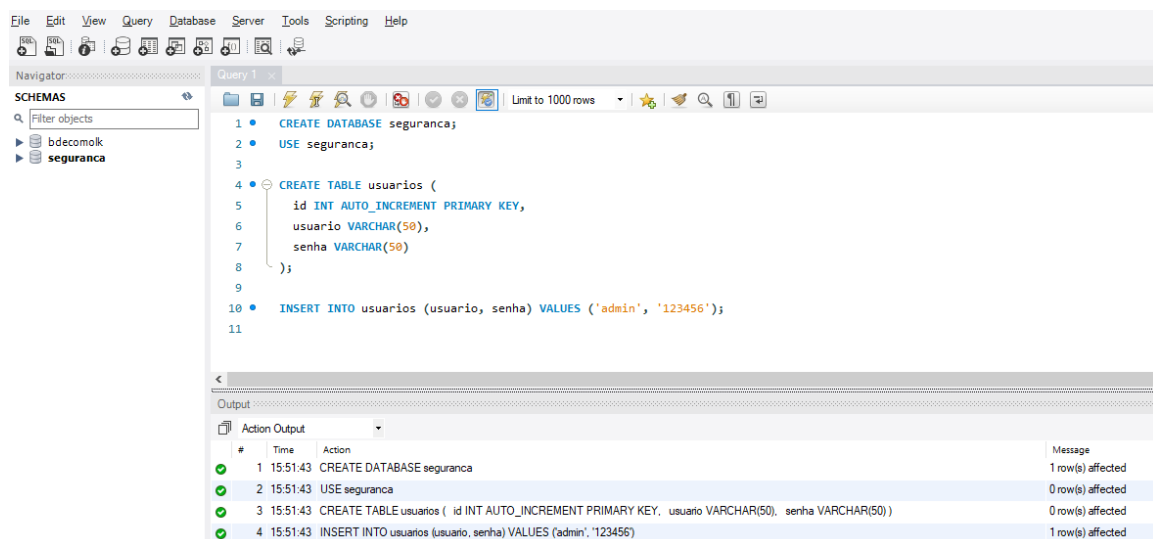
Repositório no GitHub: [https://github.com/KalledAbdala/SQL\\_INJECTION](https://github.com/KalledAbdala/SQL_INJECTION)

## Introdução

Neste relatório apresento a execução prática de uma vulnerabilidade clássica em sistemas: a SQL Injection, seguida da respectiva correção utilizando Prepared Statements. A prática foi realizada utilizando Node.js, Express e MySQL.

## 1. Configuração do Banco de Dados MySQL

Criação do banco de dados seguranca. Criação da tabela usuarios com campos usuario e senha. Inserção de um usuário de teste: admin / 123456.



The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying the following SQL script:

```
1 • CREATE DATABASE seguranca;
2 • USE seguranca;
3
4 • CREATE TABLE usuarios (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     usuario VARCHAR(50),
7     senha VARCHAR(50)
8 );
9
10 • INSERT INTO usuarios (usuario, senha) VALUES ('admin', '123456');
11
```

The 'Output' tab at the bottom shows the execution results of the queries:

#	Time	Action	Message
✓ 1	15:51:43	CREATE DATABASE seguranca	1 row(s) affected
✓ 2	15:51:43	USE seguranca	0 row(s) affected
✓ 3	15:51:43	CREATE TABLE usuarios ( id INT AUTO_INCREMENT PRIMARY KEY, usuario VARCHAR(50), senha VARCHAR(50) )	0 row(s) affected
✓ 4	15:51:43	INSERT INTO usuarios (usuario, senha) VALUES ('admin', '123456')	1 row(s) affected

## 2. Desenvolvimento da Aplicação Node.js

Criação do projeto Node.js. Instalação das dependências: express, mysql2, body-parser.

Implementação inicial de código vulnerável à SQL Injection com template literals.

```
PROBLEMAS SAÍDA CONSOLE DE DEPUÇÃO TERMINAL PORTAS QUERY RESULTS POSTMAN CONSOLE
PS D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION> mkdir sql-injection-demo
Diretório: D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION

Mode                LastWriteTime         Length Name
----                -
d-----          03/06/2025   15:56             sql-injection-demo

PS D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION> cd sql-injection-demo
PS D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION\sql-injection-demo> npm init -y
>>
Wrote to D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION\sql-injection-demo\package.json:

{
  "name": "sql-injection-demo",
  "version": "1.0.0",
  "main": "index.js",
  "test": "echo \"Error: no test specified\" && exit 1",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

PS D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION\sql-injection-demo> npm install express mysql2 body-parser
>>
added 77 packages, and audited 78 packages in 4s

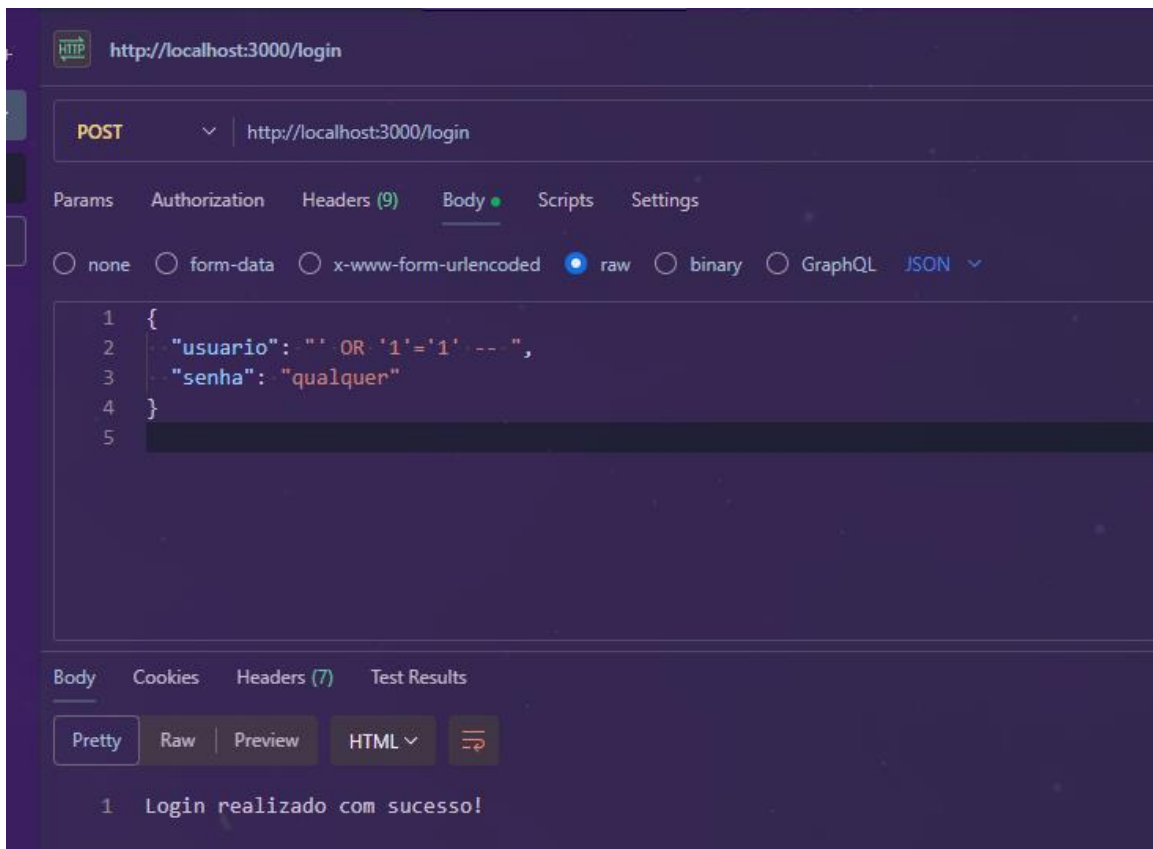
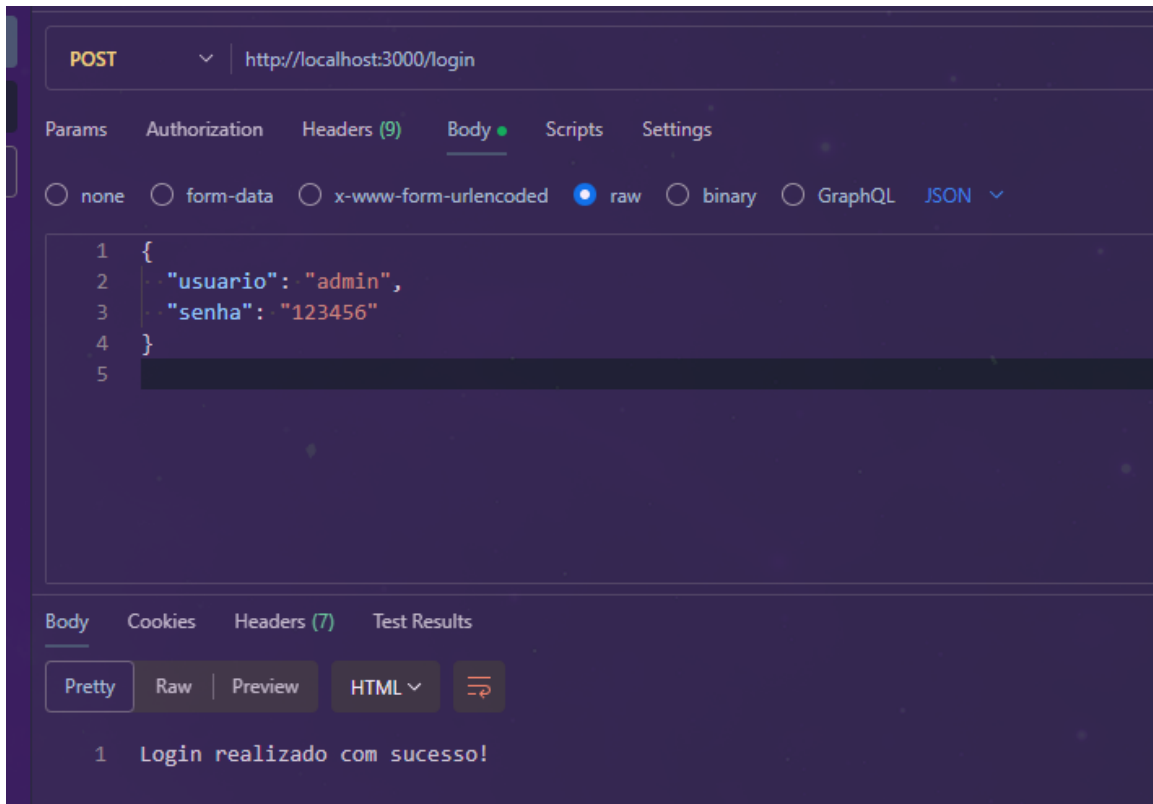
15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Documentos\Pessoais\Facull\trabalhos\5 semestre\Cyber Segur\SQL INJECTION\SQL_INJECTION\sql-injection-demo>
```

```
Bem-vindo JS app.js http://localhost:3000 Wallpaper Engine League of Legends VALORANT Cliente Riot Estudos Visual Studio Code
sql-injection-demo > JS app.js > ...
1  const express = require('express');
2  const mysql = require('mysql2');
3  const bodyParser = require('body-parser');
4
5  const app = express();
6  app.use(bodyParser.urlencoded({ extended: true }));
7  app.use(bodyParser.json());
8
9  const connection = mysql.createConnection({
10     host: 'localhost',
11     user: 'root',
12     password: '@LavaMe3FibraDeCarbono#', // trocar a senha se for rodar na sua maquina
13     database: 'seguranca'
14 });
15
16 app.post('/login', (req, res) => {
17     const { usuario, senha } = req.body;
18
19     const query = `SELECT * FROM usuarios WHERE usuario = '${usuario}' AND senha = '${senha}'`;
20
21     connection.query(query, (err, results) => {
22         if (err) throw err;
23         if (results.length > 0) {
24             res.send('Login realizado com sucesso!');
25         } else {
26             res.send('Usuário ou senha incorretos.');
```

### 3. Testes da Aplicação Vulnerável

Teste com credenciais corretas: Login realizado com sucesso! E tentativa de Injection: Erro!

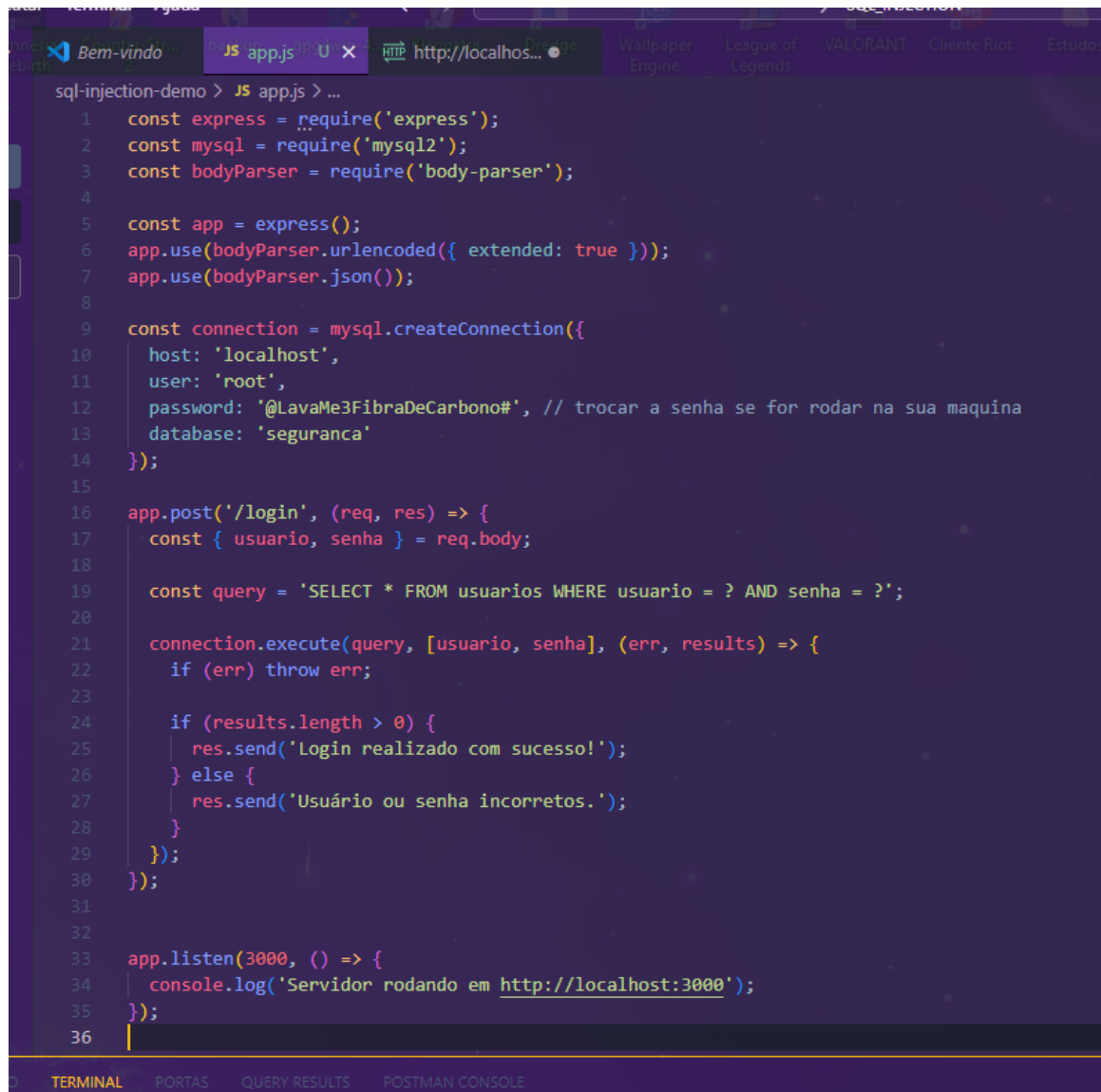


## 4. Análise da Vulnerabilidade

A falha ocorreu porque a aplicação concatenava diretamente os parâmetros de entrada na query SQL. O invasor consegue modificar a estrutura da consulta, burlando autenticação.

## 5. Correção da Vulnerabilidade

Alteração para uso de Prepared Statements, eliminando a possibilidade de injeção.



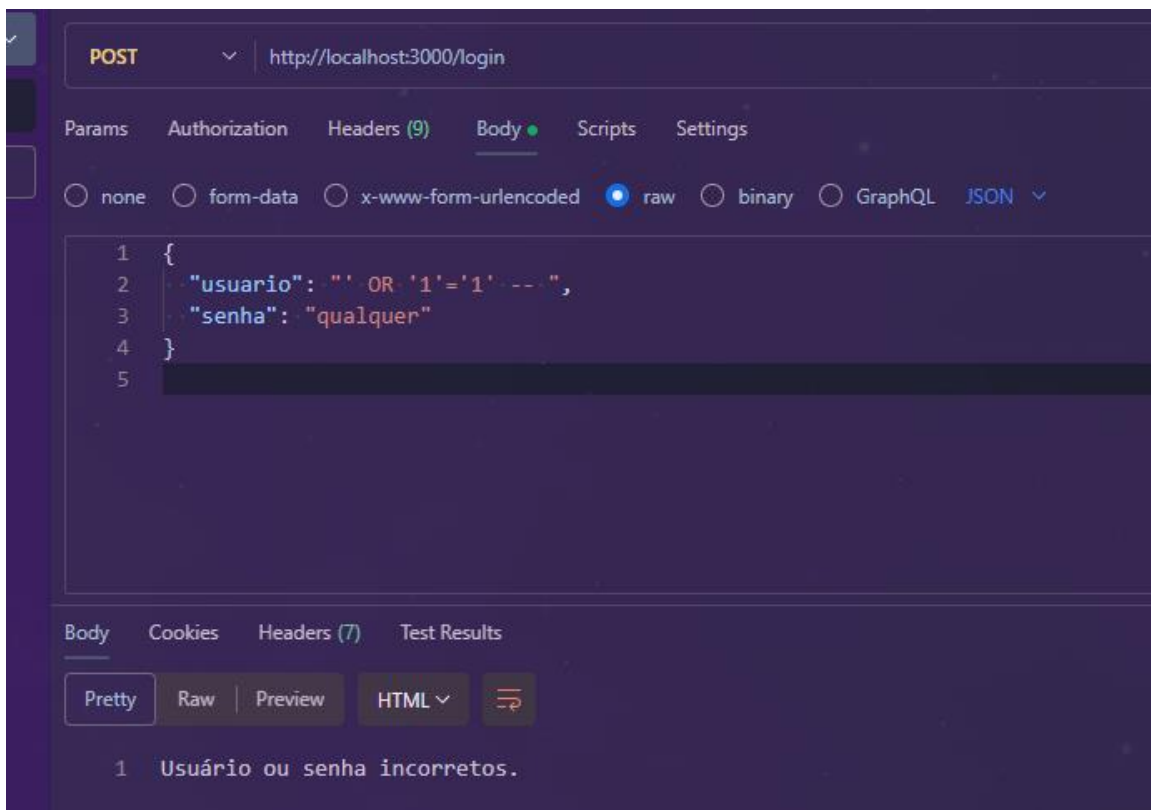
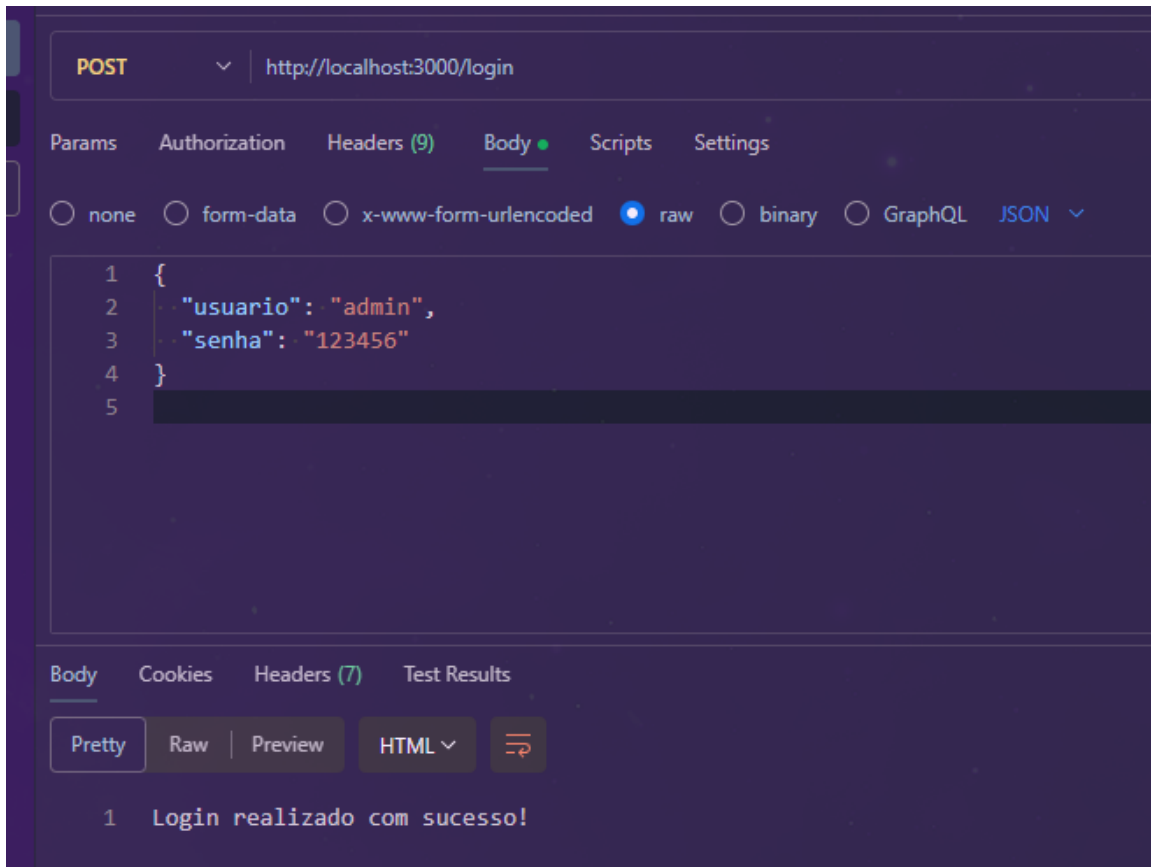
```
sql-injection-demo > JS app.js > ...
1  const express = require('express');
2  const mysql = require('mysql2');
3  const bodyParser = require('body-parser');
4
5  const app = express();
6  app.use(bodyParser.urlencoded({ extended: true }));
7  app.use(bodyParser.json());
8
9  const connection = mysql.createConnection({
10   host: 'localhost',
11   user: 'root',
12   password: '@LavaMe3FibraDeCarbono#', // trocar a senha se for rodar na sua maquina
13   database: 'seguranca'
14 });
15
16 app.post('/login', (req, res) => {
17   const { usuario, senha } = req.body;
18
19   const query = 'SELECT * FROM usuarios WHERE usuario = ? AND senha = ?';
20
21   connection.execute(query, [usuario, senha], (err, results) => {
22     if (err) throw err;
23
24     if (results.length > 0) {
25       res.send('Login realizado com sucesso!');
26     } else {
27       res.send('Usuário ou senha incorretos.');

TERMINAL PORTAS QUERY RESULTS POSTMAN CONSOLE


```

## 6. Testes após a Correção

Teste com credenciais corretas: Login realizado com sucesso! E tentativa de Injection: Erro!



## Conclusão

Esta prática demonstrou na prática como uma simples falha de codificação pode expor sistemas a riscos sérios de segurança. Utilizar prepared statements é uma das medidas fundamentais para garantir a integridade e segurança das aplicações.

## Respostas às Questões

### 1) O que aconteceu quando realizou o ataque de SQL Injection?

O login foi realizado com sucesso, mesmo utilizando credenciais incorretas. Isso ocorreu porque a consulta SQL foi manipulada para sempre retornar verdadeiro.

### 2) Por que o ataque funcionou?

Funcionou porque a aplicação concatenava diretamente as entradas do usuário na consulta SQL, sem nenhum mecanismo de proteção ou validação.

### 3) Como você corrigiu a vulnerabilidade?

Implementei Prepared Statements utilizando? como parâmetros e passando os valores em um array, garantindo que o driver do MySQL escape automaticamente qualquer conteúdo malicioso.

### 4) O ataque ainda funciona após a correção?

Não, após a correção o ataque de SQL Injection não funciona mais. A aplicação responde corretamente com "Usuário ou senha incorretos."