**PXL Secure Coding**

### RISK #1: SQL Injection

**Exploit**

' or '1'='1

' or '1'='1

**Message**: wireless charging is overrated -> message

**SQL sanitization:**

app.get('/authenticate/:username/:password', async (request, response) => {

const username = request.params.username;  
const password = request.params.password;

const query = 'SELECT \* FROM users WHERE user\_name=$1 and password=$2';  
console.log(query);

pool.query(query, [username, password], (error, results) => {

if (error) {

throw error

}

response.status(200).json(results.rows)});

});

### RISK #2: Insecure Storage

CREATE DATABASE pxldb;

\c pxldb;

CREATE USER secadv WITH PASSWORD 'ilovesecurity';

GRANT ALL PRIVILEGES ON DATABASE pxldb TO secadv;

CREATE TABLE users (

id SERIAL PRIMARY KEY,

user\_name TEXT NOT NULL UNIQUE,

password\_hash TEXT NOT NULL

);

GRANT ALL PRIVILEGES ON TABLE users TO secadv;

BEGIN;

INSERT INTO users (user\_name, password\_hash)

VALUES ('pxl-admin'**, crypt('insecureandlovinit'**, gen\_salt('bf', 8))),

('george', **crypt('iwishihadbetteradmins'**, gen\_salt('bf', 8)));

COMMIT;

Note that the password column has been replaced with a password\_hash column, which will store the bcrypt-hashed password. The crypt() function is used to perform the hashing, with a randomly generated salt of length 8 using the gen\_salt() function. The number 8 is the cost factor, which determines the computational cost of the hashing process. A higher cost factor makes the hashing process more computationally expensive, making it harder for attackers to crack the passwords.

### RISK #3: CORS

const pg = require('pg');

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

const cors = require('cors')

const port=3000;

const pool = new pg.Pool({

user: 'secadv',

host: 'db',

database: 'pxldb',

password: 'ilovesecurity',

port: 5432,

connectionTimeoutMillis: 5000

})

console.log("Connecting...:")

//CORS

const allowedOrigins = ['http://localhost:8080'];

const corsOptions = {

origin: function (origin, callback) {

if (allowedOrigins.indexOf(origin) !== -1 || !origin) {

callback(null, true);

} else {

callback(new Error('Not allowed by CORS'));

}

}

};

//CORS

app.use(cors(corsOptions));

app.use(bodyParser.json());

app.use(

bodyParser.urlencoded({

extended: true,

})

)

app.get('/authenticate/:username/:password', async (request, response) => {

const username = request.params.username;

const password = request.params.password;

const query = 'SELECT \* FROM users WHERE user\_name=$1 and password=$2';

console.log(query);

pool.query(query, [username, password], (error, results) => {

if (error) {

throw error

}

response.status(200).json(results.rows)});

});

// app.get('/authenticate/:username/:password', async (request, response) => {

// const username = request.params.username;

// const password = request.params.password;

// const query = `SELECT \* FROM users WHERE user\_name='${username}' and password='${password}'`;

// console.log(query);

// pool.query(query, (error, results) => {

// if (error) {

// throw error

// }

// response.status(200).json(results.rows)});

// });

app.listen(port, () => {

console.log(`App running on port ${port}.`)

})

### RISK #4: Credentials in Version Control

Methods: Commit with git-ignore, use ansible-vault, environment variables

### …

**PXL OAuth**