# Day 4: Authentication & Authorization

# Section 1:
# Security & Design

# Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

# Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

# SQL Injection Prevention

```
// DO NOT USE - Vulnerable to SQL Injection
app.get('/user', (req, res) => {
  const userId = req.query.id; // e.g. 1 OR 1=1
  const query = `SELECT * FROM users WHERE id = ${userId}`;

  connection.query(query, (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

AMIT DHOJU

# SQL Injection Prevention

📌 If a user sends:

bash

```
/user?id=1 OR 1=1
```

The query becomes:

sql

```
SELECT * FROM users WHERE id = 1 OR 1=1;
```

This returns **all users**, bypassing authentication.

AMIT DHOJU

# SQL Injection: Safe Implementation

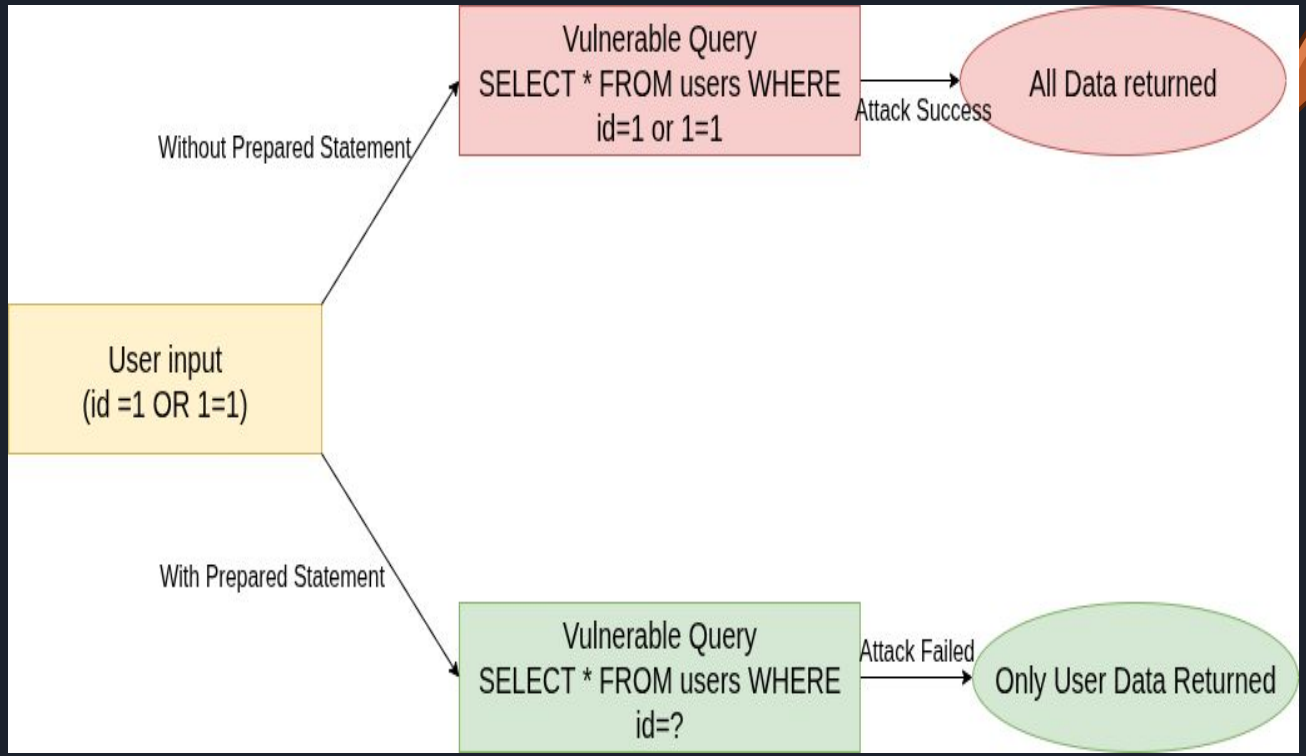
```
// SAFE - Using parameterized query
app.get('/user', (req, res) => {
  const userId = req.query.id;
  const query = 'SELECT * FROM users WHERE id = ?';

  connection.query(query, [userId], (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

**Why safe?**

- The `?` is replaced with escaped values — SQL code injection is impossible.

AMIT DHOJU

# SQL Injection: Safe Implementation




AMIT DHOJU

# XSS (Cross-Site Scripting)

🔴 **Vulnerable Example:**

```js
app.get('/search', (req, res) => {
  const term = req.query.q;
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

📌 If someone sends:

```javascript
/search?q=<script>alert('Hacked')</script>
```

The browser executes the script.

AMIT DHOJU

# XSS: Safe Implementation

```
const escapeHtml = (unsafe) =>
  unsafe.replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");

app.get('/search', (req, res) => {
  const term = escapeHtml(req.query.q);
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

AMIT DHOJU

# XSS: Safe Implementation

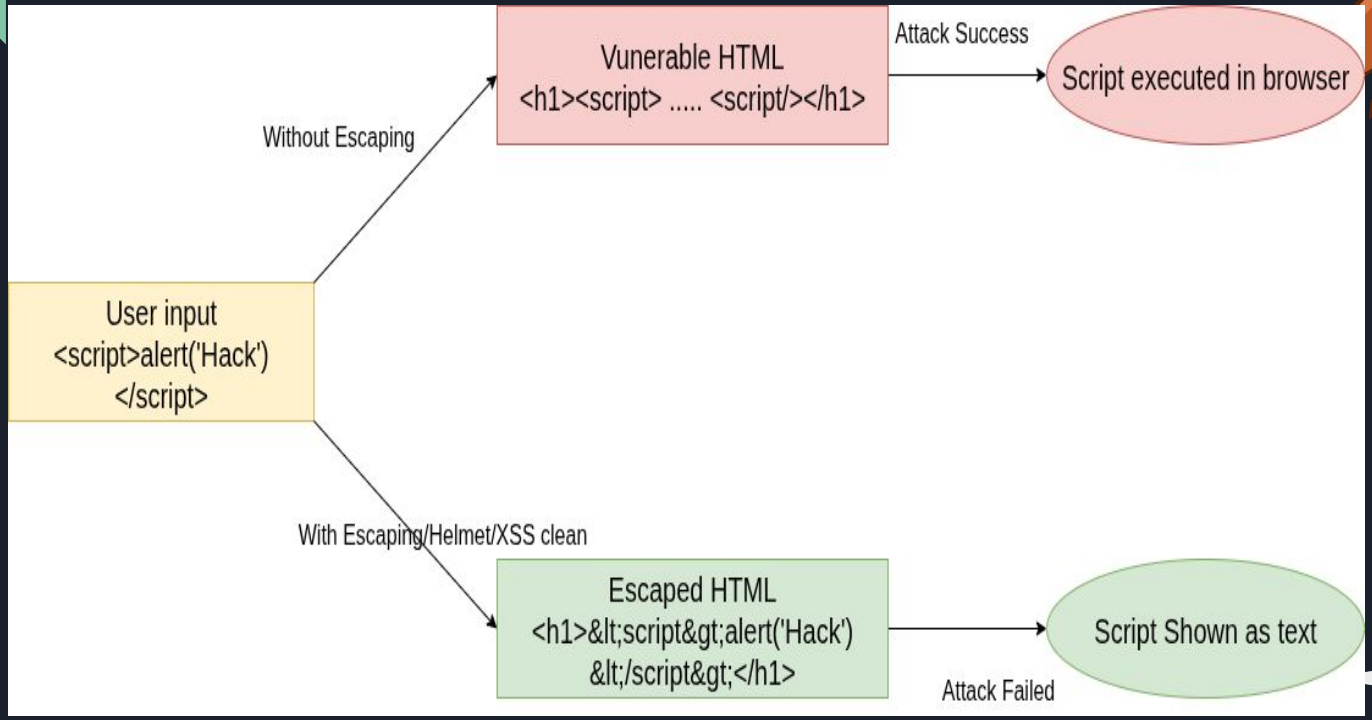
-

```bash
npm install helmet xss-clean
```

```js
const helmet = require('helmet');
const xss = require('xss-clean');

app.use(helmet()); // Security headers
app.use(xss());    // Prevent XSS
```

AMIT DHOJU

# XSS: Safe Implementation




AMIT DHOJU

# Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

# Section 2: Security Concepts

# Authentication – Who are you?

- Process of verifying the user’s identity.
-
- Common methods:
  - Username/password
  - OTP (One-Time Password)
  - Biometric (Fingerprint, Face ID)

# Authorization – What can you do?

- Process of checking the user's permissions after authentication.

- Example:
  - User A can view and edit their profile.
  - User B can only view their profile.

AMIT DHOJU

# Session Management: Server-side state (Stateful)

- Stores user data (login status, preferences) on the server.
- Usually implemented with cookies and session IDs.
- Requires server memory or database storage.
- **Stateful** — server needs to remember the user.

# Session Management: Server-side state (Stateful)

**Session-based Authentication Flow**

1. User logs in → sends credentials to Server

2. Server verifies credentials

3. Server creates session in Session Store

4. Server sends session ID in cookie

5. Client sends cookie with each request

6. Server fetches session from Session Store to validate user

AMIT DHOJU

# Token-based Authentication – Stateless authentication

- No server memory of the session.
- User receives a token after logging in.
- Token is sent with every request.
- Works well for scalable APIs.

# JWT (JSON Web Tokens)

- Compact, secure token format (base64-encoded).
- Contains:
  - Header (type & algorithm)
  - Payload (user data, claims)
  - Signature (ensures integrity)

- Example use:
  - Single Sign-On (SSO).
    - Google, Facebook etc login for sites
  - Token base login system
- Link: jwt.io

AMIT DHOJU

# Token-based Authentication

**Token-based JWT Authentication Flow**

1. User logs in → sends credentials to Server

2. Server verifies credentials

3. Server creates JWT and sends to Client

4. Client stores JWT (localStorage/cookie)

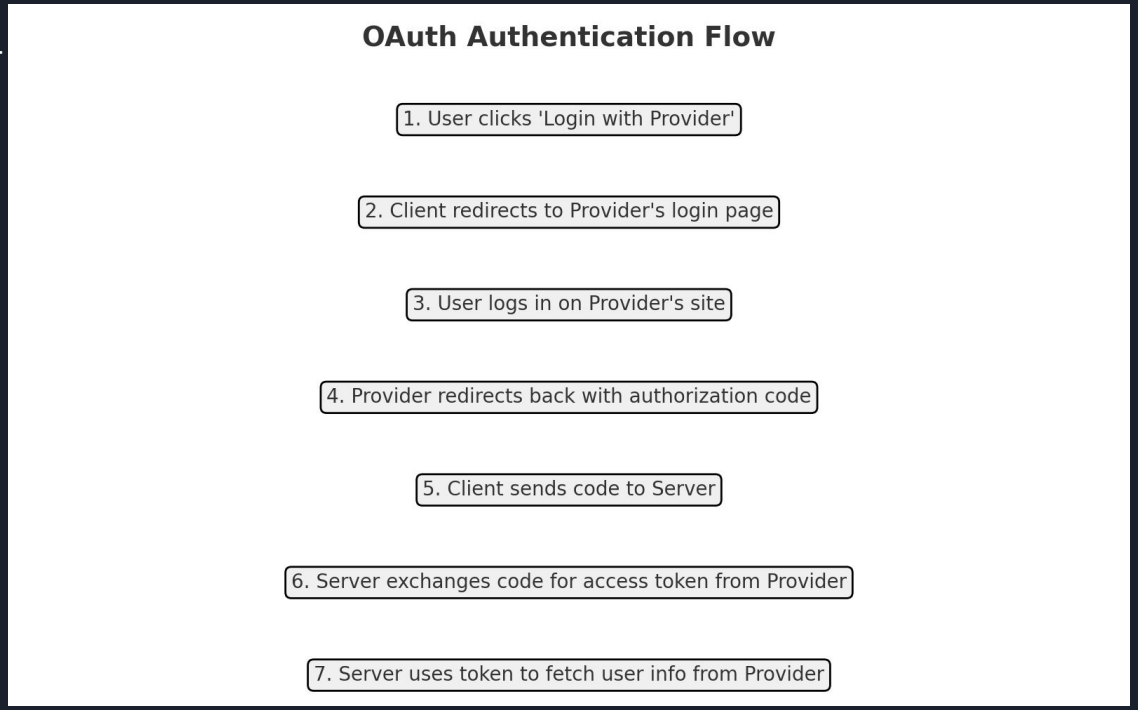
5. Client sends JWT in Authorization header with each request

6. Server verifies JWT signature and payload

# OAuth – Third-party authentication

- Lets users log in via other providers (Google, Facebook, GitHub).
- Avoids storing user passwords yourself.
- Uses **access tokens** and **refresh tokens**.

# OAuth – Third-party authentication




AMIT DHOJU