Advanced Web Programming Part 07 – Authentication and Authorization Fundamentals

Dr. Amjad AbuHassan

Understanding Authentication and Authorization

Authentication and Authorization

- Definition of Authentication:
  - Confirms **who the user is.**
  - Examples: Logging in with a username/password, biometric scans, social login (Google, FB).
- Definition of Authorization:
  - Defines **what resources the user can access** once authenticated.
  - Examples: Access control lists (ACL), role-based access control (RBAC).
- Key Difference:
  - Authentication verifies identity.
  - Authorization sets permissions.

Introduction to Authentication

- What is Authentication?
  - The process of verifying the identity of a user, device, or system.
  - Ensures that the entity is who or what it claims to be.
- Why is it Important?
  - Protects sensitive data, systems, and resources from unauthorized access.
  - Builds trust in digital interactions.
- Example:
  - Logging into an email account with a username and password.

Authentication Methods

- 1. Password-based Authentication:
  - Users provide a secret password to verify their identity.
  - **Pros:** Simple and widely used.
  - **Cons:** Vulnerable to weak passwords, phishing, and brute force attacks.
  - **Example:** Logging into a social media account.
- 2. Multi-factor Authentication (MFA):
  - Requires two or more verification factors (e.g., password + SMS code).
  - **Pros:** Adds an extra layer of security.
  - **Cons:** Can be inconvenient for users.
  - **Example:** Using Google Authenticator for logging into a bank account.

Authentication Methods cont.

- 3. Biometric Authentication:
  - Uses unique biological traits (e.g., fingerprint, facial recognition).
  - **Pros:** Highly secure and user-friendly.
  - **Cons:** Expensive to implement and privacy concerns.
  - **Example:** Unlocking a smartphone with a fingerprint.
- 4. Token-based Authentication:
  - Uses tokens (e.g., JWTs, OAuth) to verify identity.
  - **Pros:** Scalable and stateless.
  - **Cons:** Requires secure token storage.
  - **Example:** Logging into a web app using a Google account.

Challenges in Authentication

- Security Risks:
  - Weak passwords, and phishing attacks.
  - Example: Hackers using stolen credentials to access accounts.
  - Solution: Enforce strong password policies and use MFA.
- User Experience:
  - Balancing security with ease of use (e.g., avoiding complex password requirements).
  - Example: Users may forget passwords if they are too complex.
  - Solution: Implement passwordless authentication or biometrics.

Challenges in Authentication cont.

- Scalability:
  - Managing authentication for millions of users in large systems.
  - Example: A global e-commerce platform handling user logins.
  - Solution: Use scalable authentication services like Auth0 or Okta.
- Emerging Threats:
  - AI-powered attacks and deepfake-based impersonation.
  - Example: Hackers using AI to mimic a user's voice for authentication.
  - Solution: Stay updated on security trends and use advanced authentication methods.

Introduction to Authorization

- What is Authorization?
  - Granting or denying access to specific resources or actions based on user permissions
  - Determines what an authenticated user can do.
- Why is it Important?
  - Ensures users only access resources they are permitted to use.
  - Prevents unauthorized actions (e.g., deleting files, accessing sensitive data).
- Example:
  - An admin can delete user accounts, while a regular user can only view their profile.

Authentication vs. Authorization

- Authentication:
  - Confirms "who you are."
  - Example: Logging into a bank account.
- Authorization:
  - Determines "what you can do."
  - Example: Accessing account details or making transactions.
- Key Difference:
  - First Authentication, then authorization.
- Real-World Example:
  - A hotel guest is authenticated at check-in (ID verification) and authorized to access their room (key card).

Authorization Models

- 1. Role-Based Access Control (RBAC):
  - Users are assigned roles (e.g., admin, editor, viewer) with specific permissions.
  - Example: Admins can delete posts, while editors can only edit them.
  - Pros: Simple and easy to manage.
  - Cons: Inflexible for complex scenarios.

Authorization Models cont.

- 2. Attribute-Based Access Control (ABAC):
  - Access is granted based on attributes (e.g., user department, time of day).
  - Example: Only HR employees can access payroll data during work hours.
  - Pros: Highly flexible and granular.
  - Cons: Complex to implement and manage.

Authorization Models cont.

- 3. Policy-Based Access Control (PBAC):
  - Uses policies to define access rules.
  - Example: A policy might allow access only if the user is in a specific location.
  - Pros: Combines the best of RBAC and ABAC.
  - Cons: Requires advanced policy management tools.

Authentication Best Practices

- Use strong, unique passwords and enforce password policies.
- Implement multi-factor authentication (MFA).
- Regularly update and patch authentication systems.
- Monitor for suspicious login attempts.

Authorization Best Practices

- Follow the principle of least privilege (grant minimal necessary access).
- Use role-based or attribute-based access control.
- Regularly review and update permissions.
- Log and audit access to sensitive resources.
- **Example:** A bank reviewing employee access to customer data annually.

Tools and Technologies

- Authentication Tools:
  - Auth0: A platform for managing user authentication.
  - Okta: Provides identity and access management solutions.
  - Firebase Authentication: A Google service for app authentication.
- Authorization Frameworks:
  - OAuth 2.0: A protocol for delegated authorization.
  - OpenID Connect: A layer on top of OAuth 2.0 for authentication.
  - JWT (JSON Web Tokens): A compact way to securely transmit information.

Real-World Applications

- E-commerce:
  - Authenticate users for secure transactions.
  - Authorize access to order history and payment details.
- Healthcare:
  - Authenticate doctors and patients.
  - Authorize access to sensitive patient records.
- Banking:
  - Use MFA for added security.
  - Authorize transactions based on user roles.
- Social Media:
  - Authenticate users and authorize access to posts and messages.

Emerging Trends

- Passwordless Authentication:
  - Uses biometrics or magic links instead of passwords.
  - Example: Logging into an app with a fingerprint or facial recognition.
- Zero Trust Architecture:
  - Assumes no user or device is trusted by default.
  - Example: Continuously verifying user identity and device security.

Emerging Trends cont.

- AI and Machine Learning:
  - Detects anomalies and prevents unauthorized access.
  - Example: Using AI to flag suspicious login attempts.
- Decentralized Identity:
  - Users control their identity data using blockchain technology.
  - Example: Storing identity information on a blockchain for secure access.

JSON Web Tokens (JWTs)

What is a JWT?

- Definition:

```
Header: {"alg": "HS256", "typ": "JWT"}
Payload: {"sub": "12345", "name": "John Doe", "role": "admin", "exp": 1735689600}
Signature: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

  - A compact, URL-safe token format for securely transmitting information between parties.
  - Commonly used for authentication and authorization in web applications.
- Structure:
  - Header: Contains metadata (e.g., token type and signing algorithm).
  - Payload: Contains claims (e.g., user ID, roles, expiration time).
  - Signature: Ensures the token’s integrity (created using a secret key).

How JWTs Work

- **User Login:** The user provides credentials (e.g., username and password).
- **Server Validation:** The server verifies the credentials and generates a JWT.
- **Token Issuance:** The JWT is sent to the client and stored (e.g., in localStorage or cookies).
- **Token Usage:** The client includes the JWT in the Authorization header of subsequent requests.
- **Server Verification:** The server verifies the JWT's signature and extracts the payload to authorize the request.

Step-by-Step Flow

- Login Request:
  - The user sends a POST request with their credentials:

```
{ "username": "john_doe", "password": "securepassword123" }
```

- Server Validation:
  - The server verifies the credentials and generates a JWT:

```
{
  "sub": "12345",
  "name": "John Doe",
  "role": "user",
  "exp": 1735689600
}
```

Step-by-Step Flow cont.

- Token Issuance:
  - The server sends the JWT to the client:

```
{
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NSIsIm
5hbWUiOiJKb2huIERvZSIsInJvbGUiOiJ1c2VyIiwiZXhwIjoxNzM1Njg5Nj
AwfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o"
}
```

- Accessing Protected Resources:
  - The client includes the JWT in the Authorization header:

```
GET /api/protected-resource HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

- Server Verification:
  - The server verifies the JWT’s signature and checks the payload for permissions.
  - If valid, the server returns the requested resource.

Benefits of Using JWTs

- Stateless:
  - The server doesn’t need to store session data, making it scalable.
- Compact:
  - JWTs are small and can be easily transmitted via URLs, headers, or cookies.
- Secure:
  - The signature ensures the token hasn’t been tampered with.
- Flexible:
  - Can include custom claims (e.g., user roles, permissions).

Challenges and Best Practices

- Challenges:
  - **Token Expiry:** JWTs must have an expiration time to prevent misuse.
  - **Token Storage:** Storing JWTs securely on the client (e.g., HTTP-only cookies).
  - **Token Size:** Large payloads can increase token size and overhead.
- Best Practices:
  - Use short expiration times and refresh tokens for long-lived sessions.
  - Encrypt sensitive data in the payload if necessary.
  - Use strong signing algorithms (e.g., HS256 or RS256).

Tools and Libraries for JWTs

- Node.js:
  - jsonwebtoken library for creating and verifying JWTs.
- Python:
  - PyJWT library for working with JWTs.
- Java:
  - jjwt library for JWT handling.

jsonwebtoken Setup and Installation

- Install the jsonwebtoken Library:
  - Run the following command in your Node.js project:

```
npm install jsonwebtoken
```

- Import the Library:

```
1  const jwt = require('jsonwebtoken');
```

Creating a JWT

- jwt.sign() creates a JWT with the payload, secret key, and options.
- The payload contains user information (e.g., userId, username, role).
- The expiresIn option sets the token's expiration time.

```
1  const payload = {
2      userId: 12345,
3      username: "yahia_",
4      role: "admin"
5  };
6
7  const secretKey = "your_secret_key";
8  const options = { expiresIn: "1h" }; // Token expires in 1 hour
9
10 const token = jwt.sign(payload, secretKey, options);
11 console.log("Generated Token:", token);
```

```
Generated Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEyMzQ1LCJ
1c2VybmFtZSI6ImpvaG5fZG9lIiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNjI
5NzQwMjIyLCJleHAiOjE2Mjk3NDM4MjJ9.XbPfbIHMI6arZ3Y922BhjWgQzW
XcXNrz0ogtVhfEd2o
```

Verifying a JWT

- jwt.verify() checks the token's validity and decodes it.
- If the token is invalid or expired, an error is thrown.

```
1  const verifyToken = (token) => {
2      try {
3        const decoded = jwt.verify(token, secretKey);
4        console.log("Decoded Token:", decoded);
5        return decoded;
6      } catch (error) {
7        console.error("Token verification failed:", error.message);
8        return null;
9      }
10 };
11
12 const decodedToken = verifyToken(token);
```

```
Decoded Token: {
  userId: 12345,
  username: "john_doe",
  role: "admin",
  iat: 1629740222,
  exp: 1629743822
}
```

Decoding a JWT (Without Verification)

- jwt.decode() extracts the payload without verifying the token's signature.
- Use this only for debugging or when verification is handled elsewhere.

```
1  const decodedToken = verifyToken(token);
2
3  const decoded = jwt.decode(token);
4  console.log("Decoded Token (Unverified):", decoded);
```

```
Decoded Token (Unverified): {
  userId: 12345,
  username: "john_doe",
  role: "admin",
  iat: 1629740222,
  exp: 1629743822
}
```

Real-World Use Case

```
const express = require('express');
const app = express();
app.use(express.json());

const secretKey = "your_secret_key";

// Login endpoint
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // Validate credentials (e.g., check database)
  if (username === "john_doe" && password === "password123") {
    const payload = { userId: 12345, username: "john_doe", role: "admin" };
    const token = jwt.sign(payload, secretKey, { expiresIn: "1h" });
    res.json({ token });
  } else {
    res.status(401).json({ message: "Invalid credentials" });
  }
});

// Protected endpoint
app.get('/protected', (req, res) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) return res.status(401).json({ message: "No token provided" });

  try {
    const decoded = jwt.verify(token, secretKey);
    res.json({ message: "Access granted", user: decoded });
  } catch (error) {
    res.status(401).json({ message: "Invalid or expired token" });
  }
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```