Authentication, Authorization, and Logging In Cross-Platform Applications

Rakhmad Azhari

20 November 2024

There are only two different types of companies in the world: those that have been breached and know it and those that have been breached and don't know it.

Ted Schlein

Agenda

- Authentication and Authorization
- Access Control
- OAuth and OpenID Connect
- Application Logging

Authentication and Authorization

Authentication and Authorization

- Authentication: Verifying the identity of a user or application.
- Authorization: Granting access rights based on authentication.
- Both are fundamental for application security.

Differences Between Authentication and Authorization

Authentication	Authorization
Confirms user identity	Determines user access
First step in security process	Follows authentication
Example: Login with password	Example: Access control for admin

Access Control

Access Control

- Governs how resources are accessed by users or applications.
- Ensures that only authorized entities perform specific actions on resources.
- Protects sensitive data and prevents unauthorized operations.

Key Components of Access Control

1. Subjects: Entities requesting access (e.g., users, devices, or processes).
2. Objects: Resources being accessed (e.g., files, systems, or services).
3. Actions: Operations performed on objects (e.g., read, write, delete).
4. Policies: Rules defining who can access what, under which conditions.

Types of Access Control

1. Discretionary Access Control (DAC)

- Definition: Access is determined by the resource owner.
- Use Cases: Personal file sharing systems, small businesses.
- Advantages: Simple and flexible.
- Disadvantages: Prone to human error and insider threats.

2. Mandatory Access Control (MAC)

- Definition: Access is based on security classifications set by a central authority.
- Use Cases: Military, government, or highly secure environments.
- Advantages: Highly secure; difficult to bypass.
- Disadvantages: Complex and less flexible.

3. Attribute-Based Access Control (ABAC)

- Definition: Access is granted based on attributes (e.g., role, device type, location, time).
- Use Cases: Cloud applications, dynamic access scenarios.
- Advantages: Fine-grained and context-aware.
- Disadvantages: Complex policy management.

4. Role-Based Access Control (RBAC)

- Definition: Access is based on predefined roles assigned to users.
- Use Cases: Enterprise applications.
- Advantages: Easy to manage; scalable.
- Disadvantages: Less granular than ABAC.

Other Types of Access Control

5. Policy-Based Access Control (PBAC)

- Access decisions are based on high-level policies written in a declarative language.
- Use Cases: Enterprise environments with complex access needs.

Other Types of Access Control

6. Zero Trust Access Control

- Continuous verification of user identity and context; assumes no inherent trust.
- Use Cases: Remote work, modern enterprise environments.

Comparison of Access Control Models

Feature	RBAC	ABAC	MAC	DAC
Flexibility	Medium	High	Low	High
Granularity	Role-level	Attribute-level	Classification	User-level
Ease of Use	Easy	Complex	Strict	Simple
Use Cases	Enterprises	Cloud, IoT	Government	Personal systems

Examples of Access Control

Attribute-Based Access Control (ABAC) Example

- Scenario: A cloud app allows access based on location and time.
- Policy:
  - "Users can access resources from 9 AM to 5 PM from office networks."

Examples of Access Control

Role-Based Access Control (RBAC) Example

- Scenario: A CRM system grants access to managers and employees.
- Roles:
  - Manager: View and edit all customer records.
  - Employee: View assigned customer records.

Example of Access Control

Mandatory Access Control (MAC) Example

- Scenario: A defense organization classifies data as "Top Secret", "Secret", "Confidential", and "Unclassified".
- Implementation:
  - Each document is labeled with classification level.
  - Users (subjects) are granted security clearance levels.
  - Access is gratend only if the user's clearance level matches or exceeds the classification level of resource.

Best Practices for Access Control

1. Principle of Least Privilege (PoLP):
   - Grant only the minimum access necessary for tasks.
2. Separation of Duties:
   - Divide critical tasks among multiple users to reduce risk.
3. Regular Reviews:
   - Audit access policies and logs periodically.
4. Multi-Factor Authentication (MFA):
   - Add an extra layer of identity verification.
5. Dynamic Access:
   - Adjust access based on risk factors (e.g., location, device).

OAuth & OpenID Connect

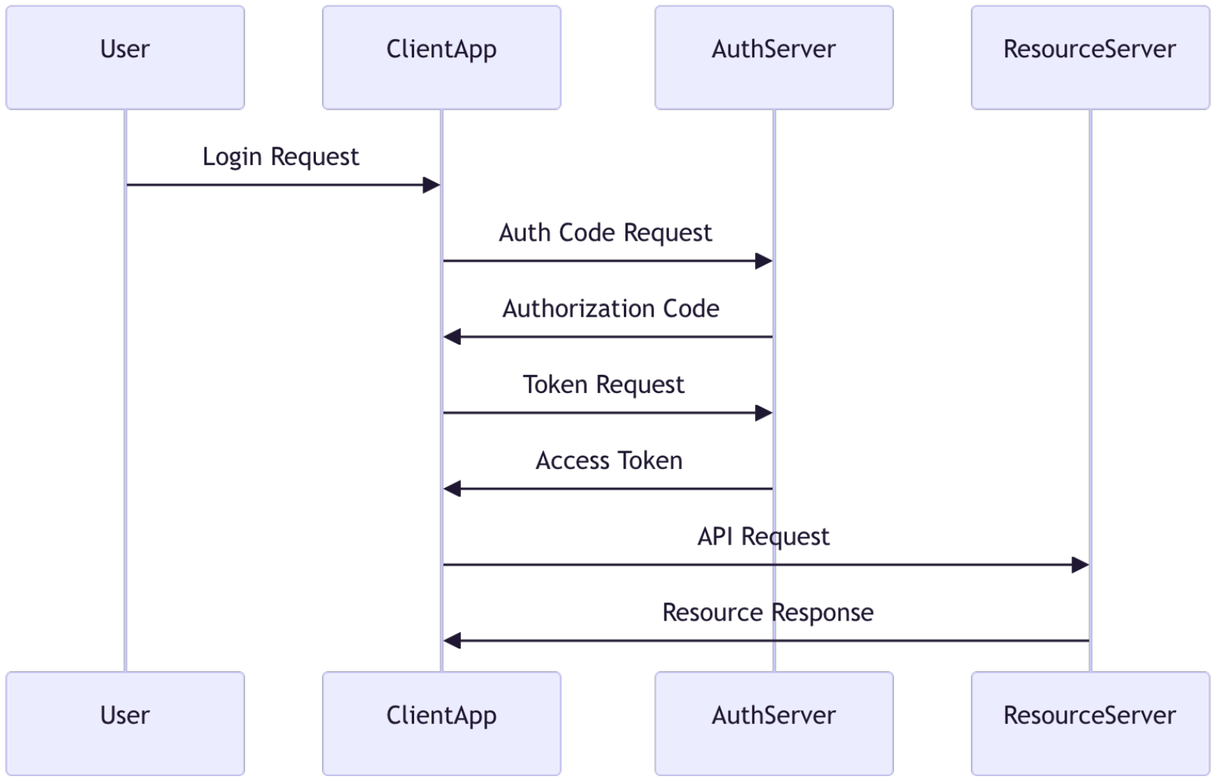
What is OAuth?

- OAuth (Open Authorization) is a standard for token-based authorization.
- Allows users to grant third-party apps limited access to their resources without exposing credentials.
- Example: Logging into a third-party app using your Google account.

OAuth Authorization Code Flow

1. User authenticates via an authorization server.
2. Authorization server provides an authorization code.
3. Application exchanges the authorization code for an access token.
4. Access token is used to access protected resources.

OAuth 2.0 Flow



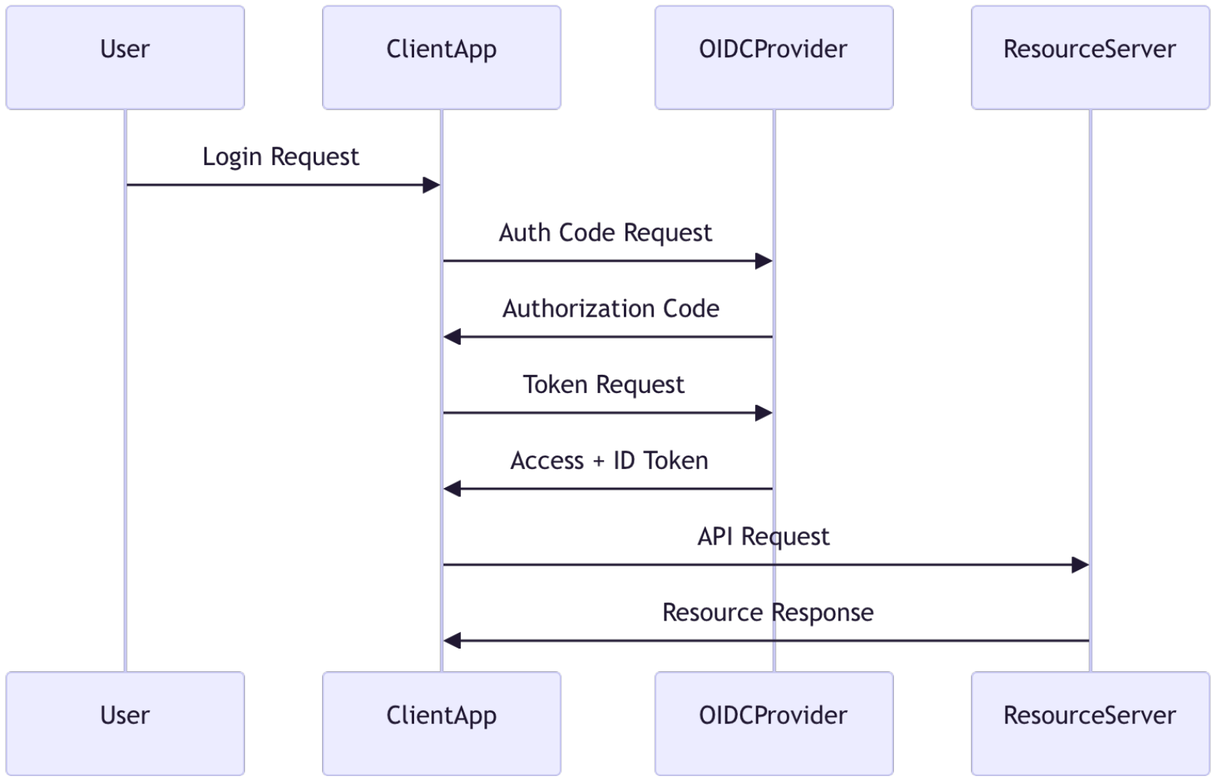
What is OpenID Connect?

- OpenID Connect (OIDC) is a layer on top of OAuth 2.0 for user authentication.
- Provides an ID token alongside the OAuth access token for identity information.
- Ensures both authentication and authorization in one flow.
- Example: Identifying a user logging into a web app.

OpenID Connect Flow

1. User authenticates via an OpenID Connect provider.
2. The provider returns an ID token and access token.
3. The application validates the ID token for user information.
4. The access token is used for resource access.

OpenID Connect Flow



Code Example: OAuth Flow

Python Example (Using Requests Library):

```
import requests

# Step 1: Redirect user to the authorization URL
auth_url = "https://example.com/oauth/authorize"
params = {
    "client_id": "your_client_id",
    "redirect_uri": "https://yourapp.com/callback",
    "response_type": "code",
    "scope": "profile email"
}
print(f"Visit this URL: {auth_url}?{params}")
```

Code Example: OAuth Flow

Python Example (Using Requests Library):

```
# Step 2: Exchange authorization code for access token
token_url = "https://example.com/oauth/token"
data = {
    "client_id": "your_client_id",
    "client_secret": "your_client_secret",
    "code": "authorization_code",
    "redirect_uri": "https://yourapp.com/callback",
    "grant_type": "authorization_code"
}
response = requests.post(token_url, data=data)
access_token = response.json().get("access_token")
```

Code Example: OpenID Connect

Node.js Example (Using Passport.js):

```
const passport = require('passport');
const OpenIDConnectStrategy = require('passport-openidconnect');

// Configure OpenID Connect strategy
passport.use(new OpenIDConnectStrategy({
    issuer: 'https://example.com',
    clientID: 'your_client_id',
    clientSecret: 'your_client_secret',
    callbackURL: 'https://yourapp.com/callback',
    scope: ['openid', 'profile', 'email']
}, (issuer, sub, profile, accessToken, refreshToken, done) => {
    return done(null, profile);
}));
```

Code Example: OpenID Connect

Node.js Example (Using Passport.js):

```
// Authentication route
app.get('/login', passport.authenticate('openidconnect'));

// Callback route
app.get('/callback', passport.authenticate('openidconnect', {
    successRedirect: '/',
    failureRedirect: '/login'
}));
```

Best Practices for Authentication

- Use multi-factor authentication (MFA).
- Encrypt sensitive credentials.
- Implement secure password policies.
- Use OAuth or OpenID for cross-platform applications.

Best Practices for Authorization

- Follow the principle of least privilege.
- Implement role-based access control (RBAC).
- Validate authorization on the server-side.
- Regularly review and audit permissions.

Application Logging

Introduction to Application Logging

- Definition: Logging is the process of recording application events, errors, and operational data.
- Used for debugging, monitoring, auditing, and maintaining application health.
- Logs are critical for understanding application behavior over time.

Best Practices for Logging

- Use a structured logging format (e.g., JSON).
- Avoid logging sensitive data like passwords or tokens.
- Assign appropriate log levels (DEBUG, INFO, WARN, ERROR).
- Encrypt logs during transmission and at rest.
- Implement log rotation and retention policies.

Using Logging for Debugging

- Logs help identify and trace the root cause of issues in code.
- Provides visibility into runtime application states and workflows.
- Enables tracking user actions and API requests for troubleshooting.
- Example: Pinpointing where an exception occurred in a complex workflow.

Common Log Format (CLF)

- CLF is a standardized text format for recording log entries.
- Commonly used by web servers like Apache and Nginx.
- Provides a consistent structure for logging web server activity.
- Helps in debugging, performance monitoring, and analytics.

Structure of CLF

Log Entry Format:

```
<remote_host> <identity> <user> [<timestamp>] "<request>" <status_code> <bytes_sent>
```

Example:

```
127.0.0.1 - - [18/Nov/2024:14:52:30 +0000] "GET /index.html HTTP/1.1" 200 1234
```

CLF Fields Explained

Field	Description
`<remote_host>`	The IP address or hostname of the client.
`<identity>`	The client identity (via identd, often logged as -).
`<user>`	The username of the authenticated client (logged as - if not authenticated).
`[<timestamp>]`	Date and time of the request (e.g., `18/Nov/2024:14:52:30 +0000`).
`"<request>"`	The HTTP request line (e.g., `GET /index.html HTTP/1.1`).
`<status_code>`	The HTTP status code of the server response (e.g., `200`, `404`).
`<bytes_sent>`	Size of the response in bytes (logged as - if not available).

CLF Example Log Entry

```
127.0.0.1 - - [18/Nov/2024:14:52:30 +0000] "GET /index.html HTTP/1.1" 200 1234
```

Breakdown:

- **`127.0.0.1`**: *Client IP address.*
- **`-`**: *No client identity.*
- **`-`**: *No authenticated user.*
- **`[18/Nov/2024:14:52:30 +0000]`**: *Request timestamp in UTC.*
- ****"GET /index.html HTTP/1.1"****: *Requested* **`/index.html`** *using HTTP/1.1.*
- **`200`**: *Server responded with HTTP status 200 (OK).*
- **`1234`**: *Response size in bytes.*

Benefits of CLF

- Simplicity: Easy to read and parse using tools like `awk` and `sed`.
- Compatibility: Supported by most web servers and log analysis tools.
- Standardization: Provides a common format for log entries.

Limitations of CLF

1. Lacks Detail:
   - Does not log referrers, user agents, or HTTP headers by default.
   - Extended Log Format (ELF) is preferred for more detailed logs.
2. Unstructured:
   - Logs are plain text and may require parsing for analytics or visualization.

Examples of Other Log Format Standards (JSON)

```
{
  "timestamp": "2024-11-18T14:52:30Z",
  "level": "INFO",
  "message": "User login successful",
  "userId": "12345"
}
```

Conclusion

Authentication and Authorization:

- Essential for application security.
- OAuth and OpenID Connect simplify cross-platform integrations.
- Follow best practices to mitigate risks.

Conclusion

Logging:

- Crucial for debugging and investigations.
- Ensure logs are secure and structured.
- Regularly review and analyze logs.

Thank You