Topics for Today

Flask Authentication

Flask Authorization

Authentication

The process of verifying a user’s identity.

Examples: Username and password, two-factor authentication, biometric data.

Importance: Protects against unauthorized access.

Authorization

The process of determining what a user can do after authentication.

Examples: Admin, user, guest roles with different permissions.

Importance: Ensures data privacy and security.

Flask Authentication Methods

Basic Auth: Simple but insecure.

Database-based Auth: Storing user credentials in a database.

Token-Based Auth: JWT, OAuth, etc.

Basic Authentication

Clients send username and password with each request.

Pros: Simple to implement.

Cons:

- Insecure
- Passwords are sent in plain text
- Limited to HTTP authentication.

When to use: For very simple, low-security applications or as a proof-of-concept.

Database-Based Authentication

User credentials are stored in a database, verified upon login.

Pros:

- More control over user management
- Can store additional user information.

Cons:

- Requires database setup and management
- Potential security risks if not handled properly.

When to use: When you need to store additional user data or have complex user management requirements.

Token-Based Authentication

Server issues a token after successful authentication, client sends token with subsequent requests.

Pros:

- More secure than Basic Auth
- Stateless
- Can be used with different protocols (HTTP, WebSocket).

Cons: Requires additional infrastructure for token management.

Types: JSON Web Token (JWT), OAuth

When to use: Most modern web applications, APIs, and mobile apps.

Password Hashing with bcrypt

A strong one-way password hashing function.

Creates unique hash for each password that is computationally infeasible to reverse engineer.

Password Storage: Bcrypt is used to hash and store user passwords in the database.

Password Verification: When a user logs in, the provided password is hashed and compared to the stored hash.

Used in conjunction with other authentication mechanisms.

Sessions

Server-side mechanism to maintain user state across multiple requests.
Stores user info (user ID, preferences) on the server.

Implemented using cookies.

Session

The user sends login request

The user sends new request (with a cookie)

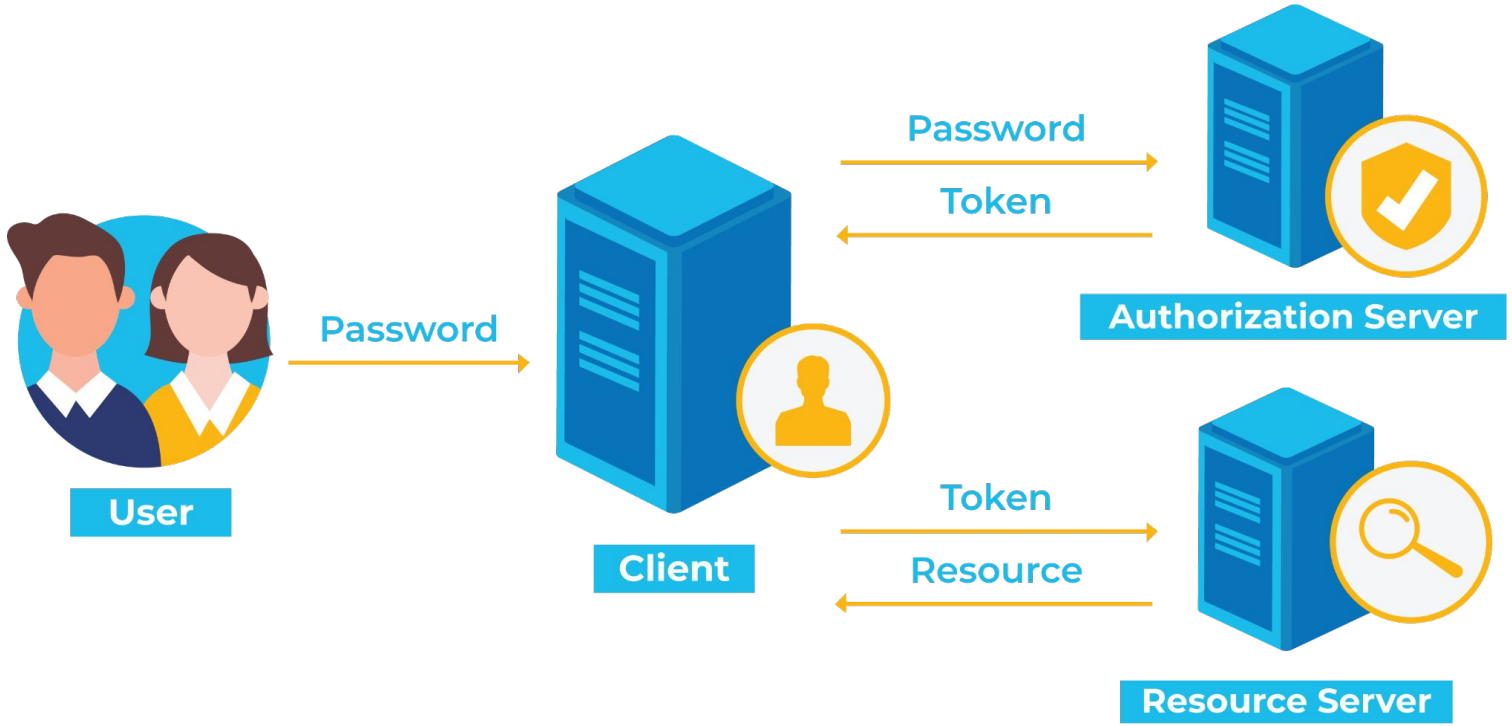
Token

Server doesn’t save the token

Tokens cannot be tampered with

Server checks whether the token is valid or not

Token

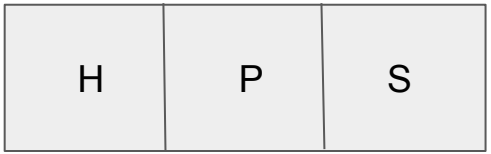

Password
Token
Authorization Server
Password
User
Client
Token
Resource
Resource Server

okta

JWT (JSON Web Token)

Standard for creating source and self-contained JSON objects that can be used to securely transmit information between parties.

Contains 3 parts: Header, payload, and signature.



Header (token metadata)
Payload (Use info)

Header and Payload are encoded in Base64URL format.

Signature is created by hashing header and payload with a secret key.

Examples: https://jwt.io/

JWT (JSON Web Token)

The server securely stores a secret key (or public/private key pair) used to sign the JWT.

JWT stored in client-side only as it is stateless.

Can be stored in 3 locations:

Local - Convenient but less secure
Session - Less Persistent but more secure than Local storage
HTTP-only cookies - More secure, prevents client-side JavaScript access.