Contents

[Session, cookie, JWT, token, SSO, and OAuth 2.0 1](#_Toc158900660)

[Sessions 1](#_Toc158900661)

[How Session Authentication Works 2](#_Toc158900662)

[Best Practices for Session Authentication 3](#_Toc158900663)

[Tokens 3](#_Toc158900664)

[Common Misconception 3](#_Toc158900665)

Session, cookie, JWT, token, SSO, and OAuth 2.0  
  
These terms relate to managing user identity when logging into websites. You declare who you are (identification), your identity is verified (authentication), and you're granted appropriate permissions (authorization). Many solutions exist and continue to emerge.  
  
From simple to more complex:  
  
🔹WWW-Authenticate is very basic. The browser prompts for username and password. It lacks control over the login lifecycle, so is rarely used today.  
  
🔹Session-cookie is prevalent in browsers. Servers maintain session storage, and browsers store session IDs in cookies. Mobile apps can use cookies in web views but often prefer tokens for native functions.  
  
🔹Tokens are encoded data used for validation, allowing clients to avoid sending credentials repeatedly. They ensure data integrity but aren't always encrypted.  
  
🔹JWT provides a standardized format for tokens. They are digitally signed to ensure their authenticity. Because JWTs can hold session or user data in their claims, servers don't need to store this information separately for verification.  
  
🔹SSO (single sign-on) lets you log in once then access multiple sites. Uses central authentication service (CAS) to maintain cross-site info.  
  
🔹OAuth 2.0 authorizes one site to access your info on another site.

# Sessions

Has been the default method for a long time. A session (a small file that stores information about the user including unique session ID, time of login and expirations, and more) is created by the server and stored in the database after you log in. Traditionally, a session ID will be stored on a cookie in your browser. As long as you remain logged in, the cookie will be sent to the server upon subsequent requests.

Upon receiving the cookie, the server compares the session ID it contains against the information stored in its memory. This allows the server to verify your identity and provide a response based on the corresponding state.

### **How Session Authentication Works**

1. You attempt to log in using your credentials.
2. login credentials are verified, and the server creates a session with a session ID for you. This session is stored in the database.
3. session ID is stored in your browser (client) as a cookie.
4. Upon subsequent requests, your cookie is verified against the session ID stored in the server. If it’s a match, the request is considered valid and processed.
5. If you log out of an application, the session ID is destroyed on both the client and server sides.

#### **Limited Scalability**

Since Cookies are stored in your Server’s memory, it becomes inherently difficult to scale, especially where there are too many simultaneous users on the system. This is the opposite with Token-based authentication.

#### **Multiple Domains Challenge**

The use of cookies for authentication can be very problematic where APIs requests are sent to services of different domains. This is because cookies typically work on a [single domain](https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4) or subdomains.

#### **Security**

Cookies are relatively more susceptible to Cross-Site Request Forgery (XSRF or CSRF) attacks and [protective measures](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html) should be employed in your servers.

### **Best Practices for Session Authentication**

1. Keep Session IDs long and [random](https://beaglesecurity.com/blog/article/session-security.html) to prevent brute force attacks. The recommended length is 128 bits.
2. Record Session ID without sensitive or user-specific data. Ideally, the ID should be a random and meaningless string of characters.
3. Enforce mandatory HTTPS communications for all session-based apps.
4. Create Cookies with secure and HTTP-only attributes.
5. Securely manage your sessions. For instance, you could destroy all sessions when you close your browser, where there’s a timeout, or when you log in or log out from different locations.

# Tokens

Lightweight, stored on the client side, need to be validated, can be hijacked.

## **Common Misconception**

While traditionally websites use session authentication with cookies, and mobile apps/SPAs use token authentication with Authorization header, this are not necessarily the case. Authorization Header and Cookies are about the transfer mechanism. Tokens and sessions essentially are about where the authorization state is handled, whether on the server-side or the client-side. For example, a server can issue a JWT token via cookie, or expect a stateful session ID to be provided in the “Authorization” header.