Session Management

Kameswari Chebrolu

Department of CSE, IIT Bombay

Background: Overall Outline

- ~~What constitutes a webpage?~~
- ~~What goes on inside a Browser?~~
  - ~~What standard security mechanisms implemented?~~
- ~~How do client and server communicate?~~
  - ~~HTTP/HTTPs protocol~~
  - **Session Management via cookies and tokens**
- How does a web server process requests and generate responses?
  - Static vs Dynamic content
- Summary: A walk through of what happens when you visit a website

Outline

- Cookies
- Session Management
- Token based Authentication
  - JSON Web Tokens

HTTP Protocol

- Stateless Protocol → Simple and Scalable
- Stateless: Server does not maintain state between requests
  - Each HTTP request is processed independent of past or future requests
  - Each request must contain all the information needed for server to understand and process
- Growth of web and e-commerce → need to maintain user state
  - E.g. shopping cart, user preferences, login status etc
    - E.g. No state of login status → user has to re-enter credentials with each request!

Solution: Cookies

- Server sends small amount of data (cookie) to store at client
- Every time client contacts the server, browser sends cookie to server
  - As per same origin policy (SOP), cookie only sent to appropriate server!

Example

- Browser to Server

  Post /users/info HTTP/1.1

  [form contents can includes username/password]

- Server to Browser

  HTTP/1.1 200 OK

  Set-Cookie: username=Chotu; expires=Wed, 06 Dec 2023 12:00:00 GMT

  (Server also maintains an entry in backend database)

- Browser to Server

  GET /info.html HTTP/1.1

  Cookie: username=Chotu

[Server accesses backend to retrieve information logged and acts accordingly]

Cookie Structure

Browsers need to support at least 4KB cookie size

- Name of cookie
- Value of cookie
- Expiry of cookie
- Path the cookie is good for
- Domain of the cookie
- secure: cookie will only be sent over secure (HTTPS) connections.
- HttpOnly: If present, the cookie cannot be accessed by client-side scripts (JavaScript). This helps mitigate certain types of attacks.
- SameSite: Helps prevent certain type of attacks. It can have values like "Strict," "Lax," or "None"

```
Set-Cookie: username=Chotu;
expires=Wed, 06 Dec 2023 12:00:00 GMT;
path=/;
domain=.example.com; secure; HttpOnly;
SameSite=Strict
```

If no expiration date, cookie deleted when user exits browser **Nonpersistent Cookie**

- Several Use cases of Cookies:
  - **Session Management:** Store a session identifier as cookie after authentication
    - Server can recognize user across pages without user having to enter login credentials each time
  - Personalization: User preferences and settings stored in cookie provide a more personalized experience
    - E.g. User's language preference or theme

- Tracking and Analytics: Cookies can be used for tracking user behavior and gathering analytics data
  - How users interact with their site, what internal pages they visit, how long they stay etc
- Advertising and Remarketing: Online advertising can use cookies to track users across websites
  - Can violate privacy (Tracking attacks, not covered)

Types of Cookies

- Session/Non-persistent Cookies: Temporary cookies erased when user closes the browser
  - Used for session management.
- Persistent Cookies: Remain on user's device for a specified period or until manually deleted
  - Used for purposes like remembering user preferences
- First-party Cookies: Set by the website the user is visiting
- Third-party Cookies: Set by a domain other than the one user is currently visiting
  - E.g when accessing an (advertising) image from another domain

Things to Note

- Only servers within a domain can set a cookie for that domain
- A subdomain can set a cookie for a higher-level domain, but not vice versa
  - mail.example.com could access cookies set for example.com or mail.example.com
  - example.com cannot access cookies set for mail.example.com
- Normally cannot set cookies for top-level domains such as .edu or .com (enforced at the browser level)

Outline

- ~~Cookies~~
- **Session Management**
- Token based Authentication
  - JSON Web Tokens

Session Management

- Session: period of time during which a user interacts with a website
  - Starts when a user accesses the website or logs in
  - Ends when user closes the browser or remains inactive for a specified period (session timeout)
- Management: Creation, maintenance, and termination of user sessions within a web application
  - Important for sites that require user authentication and personalized experiences!

- Session Creation: user visits a website and authenticates (logs in) → new session created
  - A unique session identifier (ID) generated, and associated with user's session data
  - Session Data: data specific to user's interactions/preferences; saved in database
    - E.g. user ID, roles, preferences, authentication status, User's IP address etc
  - Same ID is also stored as **a cookie** on the user's device

- Session Tracking: As user navigates, session ID is included by browser in each request to server
  - Server uses it to retrieve and update session data
- Session Termination: User logs out or session expires → session data cleared and ID invalidated
  - Timeout settings are configurable and depend on the security and usability requirements
- Web session management has security implications!
  - Many Attacks possible (covered later)

Client

1. User Credentials

3. Session ID

4. Session ID

(another request)

Web Server

2. Session Info

5. Session ID

6. Session Info

Database

Outline

- ~~Cookies~~
- ~~Session Management~~
- **Token based Authentication**
  - **JSON Web Tokens**

Token based Authentication

- Token: string used to represent user credentials or other sensitive data
  - Often used in modern authentication systems like OAuth
  - Can also be **digitally signed** by a server (e.g. JWT token)
    - Cannot be tampered with in this case
- Sent to client and client includes token in headers of each subsequent request
  - Not sent as cookie!

- Steps:
  - User sends a login request (includes username/password) to the server
  - Server authorizes login and sends a signed token to user

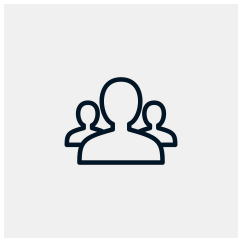
HTTP/1.1 200 OK
Content-Type: **application/json**

{
  "message": "Authentication successful",
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
}

(note: above token is JWT, but it need not be)

- Every request of user carries the token

  GET /api/resource HTTP/1.1
  Host: example.com
  Authorization: Bearer
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM
  0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE
  2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJ
  V_adQssw5c
- Server validates the token and based on it sends requested pages to the user
  - Validation involves checking the token's integrity, verifying its signature, and ensuring that it hasn't expired



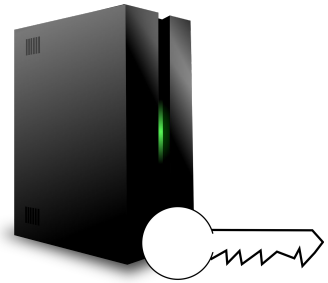
User

Password

Client

Password

Token



Authorization Server

Token

Resource

Resource Server

Cookies vs Tokens

- Storage and Inclusion:
  - Cookies stored in browser and sent by browser with HTTP requests automatically!
  - Tokens stored in various client-side locations (e.g. local storage or session storage)
    - Need to be manually included (by client side javascript) in HTTP requests, typically in the Authorization header (e.g., Bearer <token>).

- Security Considerations:
  - Cookies more susceptible to CSRF attacks
  - Tokens, less susceptible to CSRF but need protection against XSS
- Usage
  - Cookies widely used for traditional session management.
  - Tokens increasingly used in modern web applications, particularly for API-based authentication and authorization
- Size:
  - Tokens generally larger than cookies
    - JWT can contain more data, including claims and metadata.

Session ID as cookie vs Tokens

- When receiving a session ID, server looks its up in a database (stateful)
- When receiving a token, server doesn't have to make any database queries (stateless)
  - Simply authorizes the user's requests based on the validity of token and contents within
- Tokens often used in
  - Server-to-server connections
  - For scalability (no database lookup)
    - Especially in distributed or microservices architectures.

JSON Web Tokens (JWTs)

- A standardized format for sending cryptographically signed JSON data
- Can theoretically contain any kind of data but mostly used for “claims” about users
  - Claims: user id or name or department etc

- Consists of 3 parts: a header, a payload, and a signature
  - Header and payload are base64url-encoded
  - Header contains metadata about the token while payload contains actual "claims" about user
  - Signature is over header and payload
    - Signature can be done using symmetric key (e.g HS256) or using asymmetric key (private; e.g. RS256)
    - Ensures none of the data within the token has been tampered with since issued
    - Impossible for attacker to generate correct signature for a given header or payload.

1 2

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEyOCIsIm5hbWUiOiJQYXBwdSIsImRlcHQiOiJDU0UifQ.griPnqT_NNRXQTD9eSw9PDsLps4Gx7G6l7Z47b2iAzM

3

1 HEADER

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 Payload

```
{
  "id": "128",
  "name": "Chotu",
  "dept": "CSE"
}
```

3 Signature

```
HMACSHA256(
base64UrlEncode(header)
+ "." +
base64UrlEncode(payload),
your-256-bit-secret)
```

```
// Assume you have obtained a JWT after a successful login
const jwtToken = 'your_jwt_here'; // Replace this with relevant code that deals with actual JWT

// Store the JWT in localStorage
localStorage.setItem('jwtToken', jwtToken);

// Retrieve the JWT from localStorage
const storedJwt = localStorage.getItem('jwtToken');

if (storedJwt) {
  console.log('JWT retrieved from localStorage:', storedJwt);

  // Include the JWT in a GET request using the fetch API
  fetch('https://api.example.com/resource', {
        method: 'GET',
        headers: {
        'Authorization': 'Bearer ' + storedJwt,
        'Content-Type': 'application/json'
        },
  })
  .then(response => response.json())
  .then(data => {
        console.log('Response from server:', data);
  })
  .catch(error => {
        console.error('Error:', error);
  });
} else {
  console.log('No JWT found in localStorage.');
}
```

References

- Cookies: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- Token Based Authentication: https://www.okta.com/identity-101/what-is-token-based-authentication/
- JWTs: https://www.geeksforgeeks.org/json-web-token-jwt/

Summary

- Statelessness of HTTP overcome via cookies
- Many use cases of Cookies: Session Management, Personalization, Tracking, Analytics etc
- Session Management vs Token based Authentication
- More recent Json Web Tokens