



# Cookie and Session

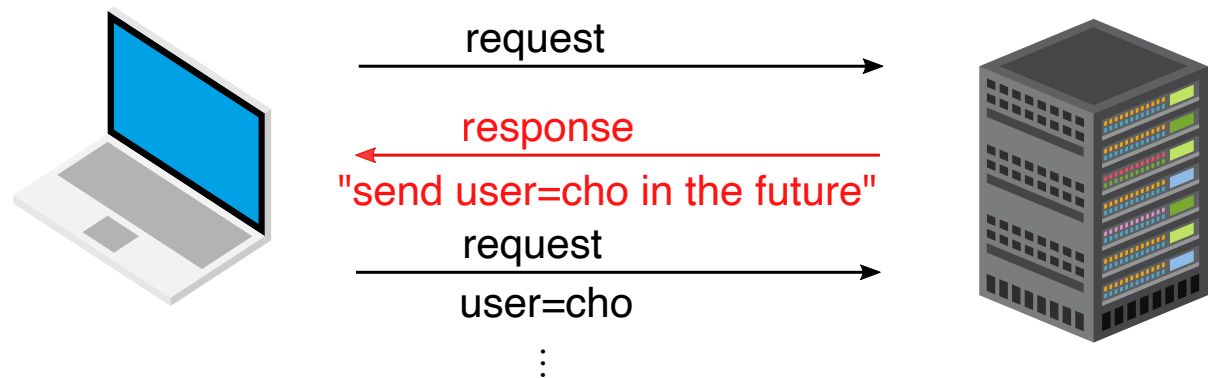
Junghoo Cho  
[cho@cs.ucla.edu](mailto:cho@cs.ucla.edu)

# HTTP: Stateless

- HTTP is a stateless protocol
  - Every request can be handled independently of others
- Q: How does a Web site “remember” a user and customize its
  - Q: How do they know that two requests are from the same user?
- Idea: “Embed” a unique identifier in every request from a user

# Cookie: Key Idea

- *Cookies* allow a server to ask a client to remember "**name=value**" and send them back in all future requests



# Setting Cookie

**Set-Cookie:** username=john; expires=Wed, 21 Oct 2031 07:28:00 GMT;

- Ask client to “set” the cookie **username=john**
- **expires**: expiration time
  - By default, cookie becomes “transient” (= session cookie) and is sent during current browsing session
  - **expires** makes cookie “persistent” until expiration
  - Setting **expires** to past “erases” the cookie
- **path** and **domain**
  - By default, cookie is sent in all requests to the same server
  - This can be adjusted by setting to a specific path and/or domain
  - Example: **path=/cs144/; domain=ucla.edu;**

# Sending Cookie Back

**Cookie:** username=john

- In all future requests to the specified domain and path, client sends **Cookie** header:
  - From the cookie, server knows the requests are all from the same client

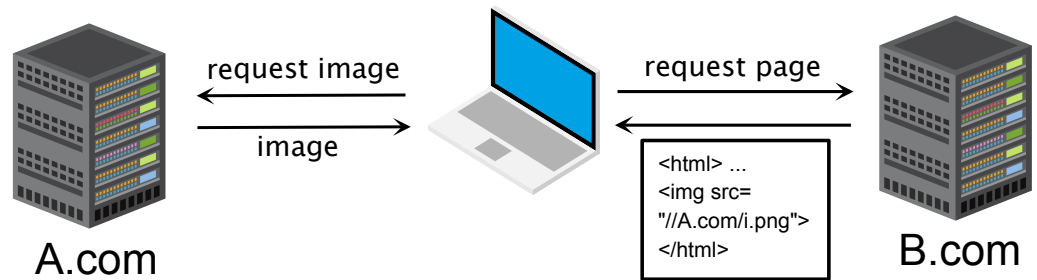
# Same-Origin Policy

- Client sends the cookie only to the domain from which it was received
  - No cross-domain cookie exchange is allowed
- Q: Why same-origin policy?
- Q: Can we use cookie(s) to identify a user across multiple domains?

# Tracking Users Across Domains

- Q: Given same-origin policy, is it possible to track a user's rec across multiple domains?
- Q: But Google/facebook/Amazon... does it. How?

# Third-Party Cookie



- **A** wants to track users across its partner sites **B**'s
- Ask partner **B** to include a (tiny) image from **A** in their pages
  - When browser access a page from **B**, it also sends a request to **A** (to get the image)
  - The request to **A** contains **Referrer** header (with **B**'s URL) and cookies
    - *Third-party cookies*
- **A** knows all URLs that users visited on partner site **B**



# Cookie is Unsafe

- Cookies can be stolen (*cookie theft*)
- Cookies can be tempered with by the client (*cookie poisoning*)
- Be ***very careful*** about what we store in cookie

# Securing Cookies

- `secure;` attribute
  - With `secure;` attribute set, the cookie is sent back *only over https*
  - Protects against cookie theft
- *Signed cookie*
  - Secret-key encrypted *signature* added to the main cookie data
- Attaching expiration date
  - Make sure cookie expires after a while
  - Even if the cookie is stolen, it will be no longer valid after a while

# JSON Web Token (JWT)

- Web standard to encode and exchange client-managed state  
tempering protection
- Format: **header.payload.signature**
  - **header**: information on the token
  - **payload**: “main body” of the token
  - **signature**: encrypted hash value for tempering detection

# JWT Header

- JSON data (encoded into Base64 string)
- Typically has two fields
  - **alg** (hashing algorithm)
  - **typ** (token type)
- Example

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

→ eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

# Payload

- JSON data (encoded to Base64)
- Contains main information
- Example

```
{  
  "iss": "http://oak.cs.ucla.edu",  
  "jti": "3gxhylhd",  
  "exp": 11253352,  
  "user": "junghoo"  
}
```

→ eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9

- “Registered claims (=fields)”
  - **iss** (issuer), **jti** (JWT ID), **iat** (issued at, # seconds since 1970-01-01T00:00:00Z), **exp** (expires at), **sub** (subject), **aud** (audience), ...
- No claim is required. “Unregistered claims” can be used

# Signature

- Secret-key encrypted hash of `header.payload` (encoded into base64)
- Example

```
HMACSHA256(header.payload, "secret password")  
→ eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM
```

- When the JWT is tempered, a hacker cannot generate a correct signature without knowing the correct password
- The creator of JWT can check tempering by comparing attacker's signature with the computed hash value

# Final JWT

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9          // header
.eyJrZXkiOiJ2YWwiLCJpYXQiOiJlMjMDU0NDV9      // payload
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM  // signature
{
  "alg": "HS256",
  "typ": "JWT"
}.{
  "iss": "http://oak.cs.ucla.edu",
  "jti": "3gxhylhd",
  "exp": 11253352,
  "user": "junghoo"
}.signature
```

- JWT is sent to the browser
  - Browser sends it back in future requests

# User Authentication

- Q: How does server authenticate the identity of a user?
- Q: How can we let a user authenticate once, without asking for authentication for every request?
- Q: After authentication, what should we store in the cookie?
- A: Two choices
  - Username
  - "Session ID"



# Session

- When user logs in, the server creates a “session”
  - All session-related “states” reside on the server
  - A unique session ID is associated with a session and set as a cookie
  - Given a session ID in a request, the server obtains session related “state” from its local “session data store”
- Q: What are the pros and cons of using session ID vs username and password?

# What We Learned

- Stateless HTTP protocol
- Cookie
- Same-origin policy
- Cookie theft, cookie poisoning, secure cookie
- JSON Web token (JWT)
- User authentication and session management

# References

- Cookie: [RFC 6265](#)
- [JSON Web Token](#)

