Web Security and Auth

Shan-Hung Wu
CS, NTHU

Outline

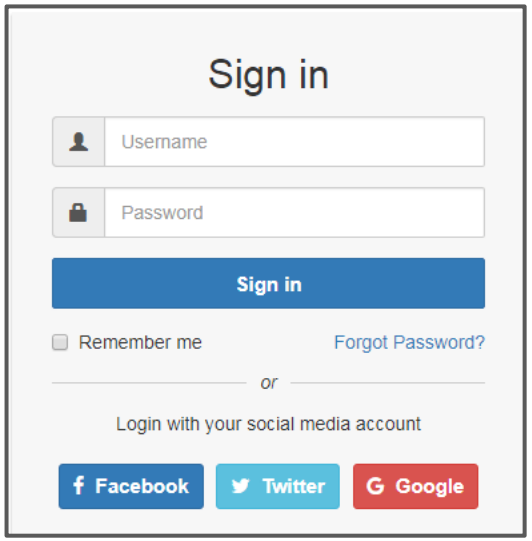
- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

Outline

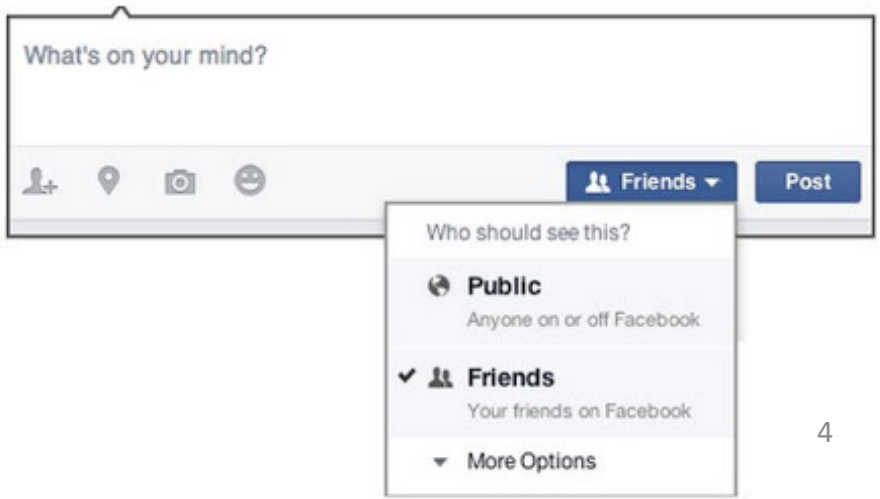
- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

Authentication vs. Authorization

- ***Authentication***: the process to verify you are who you said

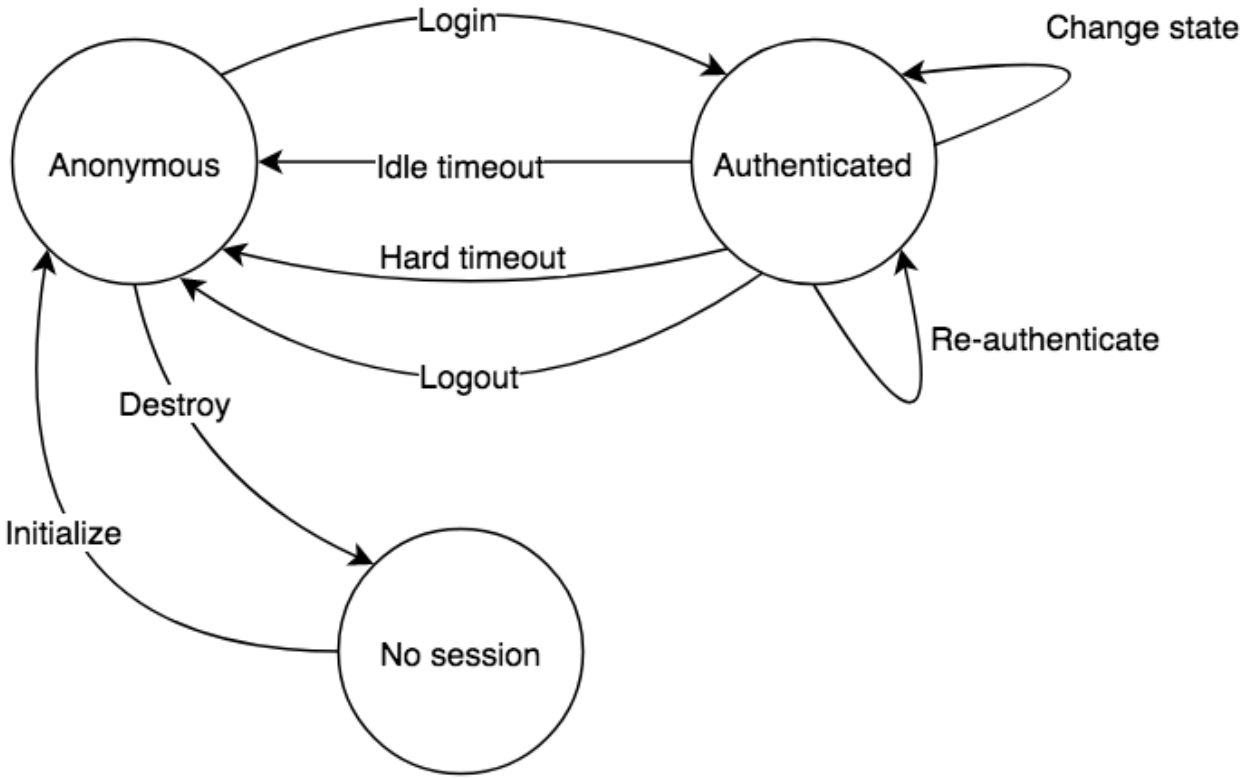


- ***Authorization***: the process to decide if you have permission to access a resource



Session Management

- The process of securely handling multiple requests to a server from a single client (user)



Were to Store Session States?

- Server
  - ***Stateful*** sessions
  - Server processes requests based on the states
- Client
  - ***Stateless*** sessions
  - Server processes requests based on their content

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

Which one should I choose?

Evaluation Criteria

- Complexity
- Reliance on HTTPS
- Reliance on CSRF protection
- Replay and integrity protection
- Session management

- User cases & tips

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

How It Works

- A client attaches clear text password to each request:

```
// Request from client
Authorization: Basic base64(username:password)
```

- Seriously?

Evaluation

- Complexity: Dead simple; tons of libraries
- Reliance on HTTPS: Yes
- Reliance on CSRF protection: Yes
- Replay and integrity protection: Relies on TLS
- Session management: Poor
  - Logout is complicated
- Tips: always use Basic Auth with HTTPS

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

HTTP Digest Auth

- Goal: not to rely on HTTPS/TLS anymore
- Idea: server challenges client
  - No password in every request

- ***Not widely adopted due to complexity!***

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

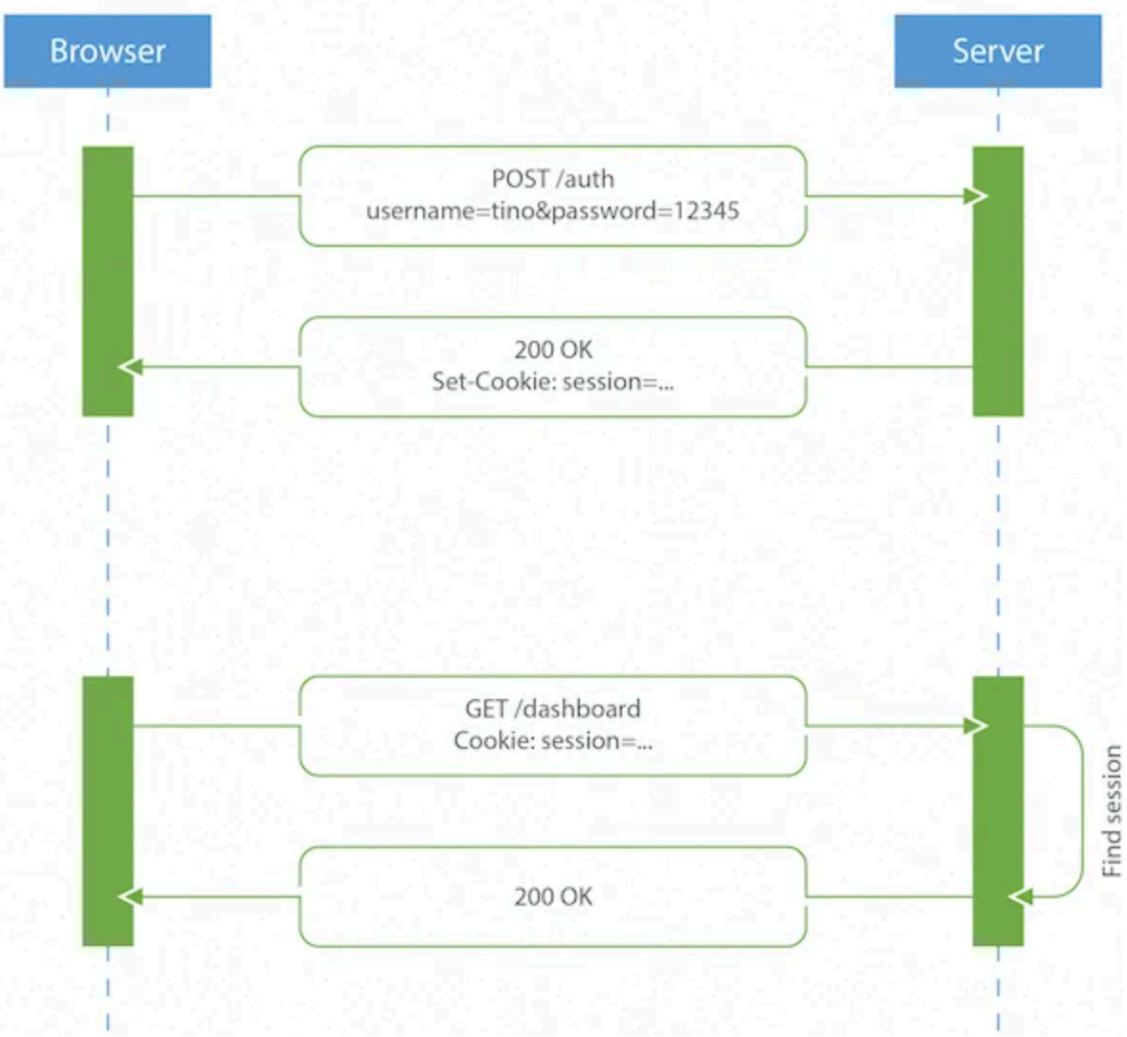
How It Works

```
// Login response from server
Set-Cookie: sessionId=...;
            Domain=.app.com;
            Secure; SameSite; HttpOnly

// Subsequent requests from client
Cookie: sessionId=...
```

- Cookies are managed by browser
  - Sent to server in every subsequent request

Stateful Sessions



Evaluation

- Complexity: simple; tons of libraries
- Reliance on HTTPS: Yes
  - Set the `Secure` flag
- Reliance on CSRF protection: Yes
  - Set the `SameSite` flag
- Replay and integrity protection: Relies on TLS
- Session management: Good
- Tips: Set the `HttpOnly` flag to prevent XSS attacks from stealing it

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

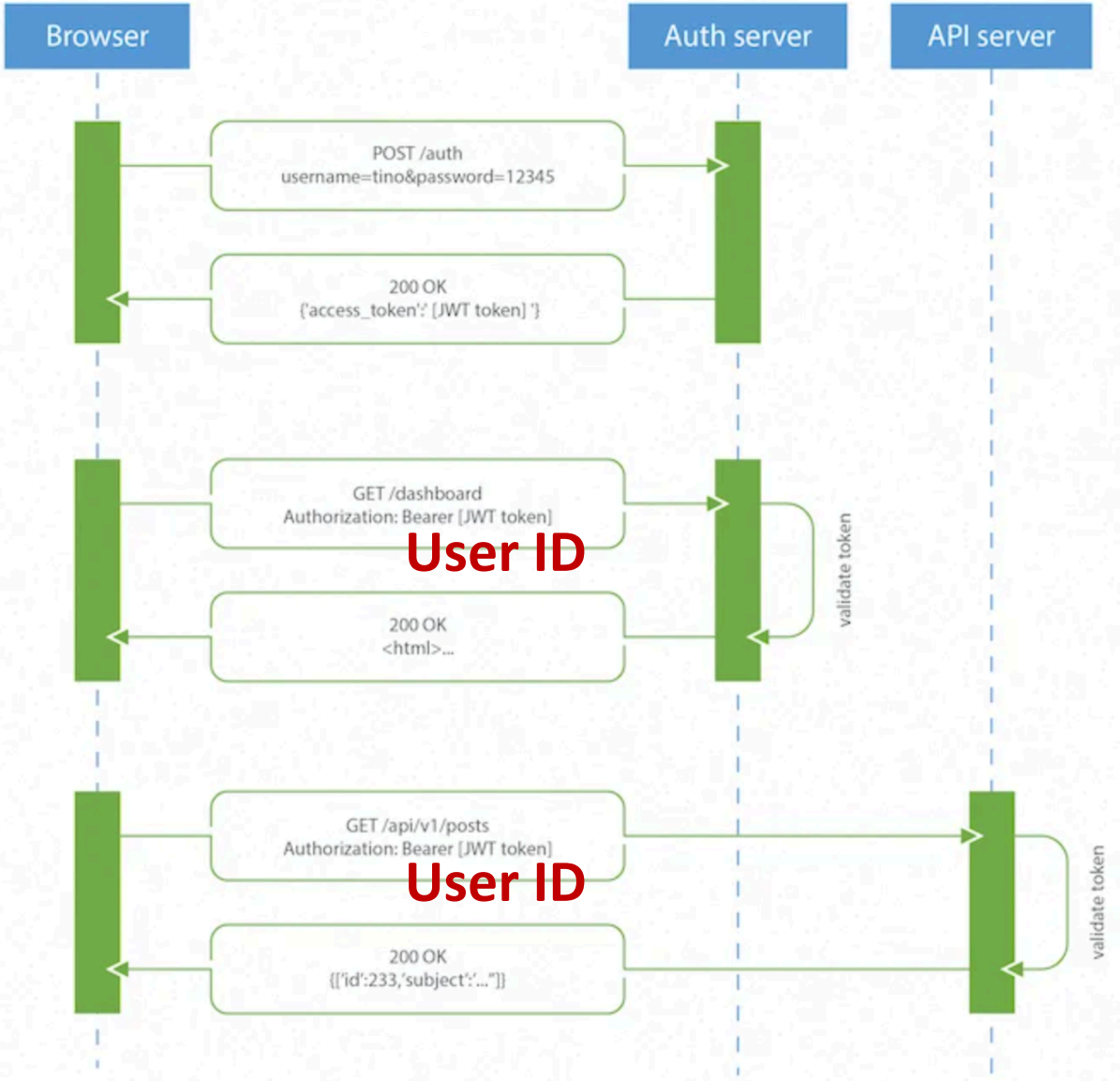
How It Works

```
// Login response from server
{
  token: e2ZahC5b // JWT token
}
// Subsequent request from client
Authorization: Bearer e2ZahC5b // added by JS
```

- A JWT token is ***self-describing*** and ***immutable***
  - Includes user ID, expiration date, etc.

```
(uid, expdate, sha256(uid, expdate, secret))
```

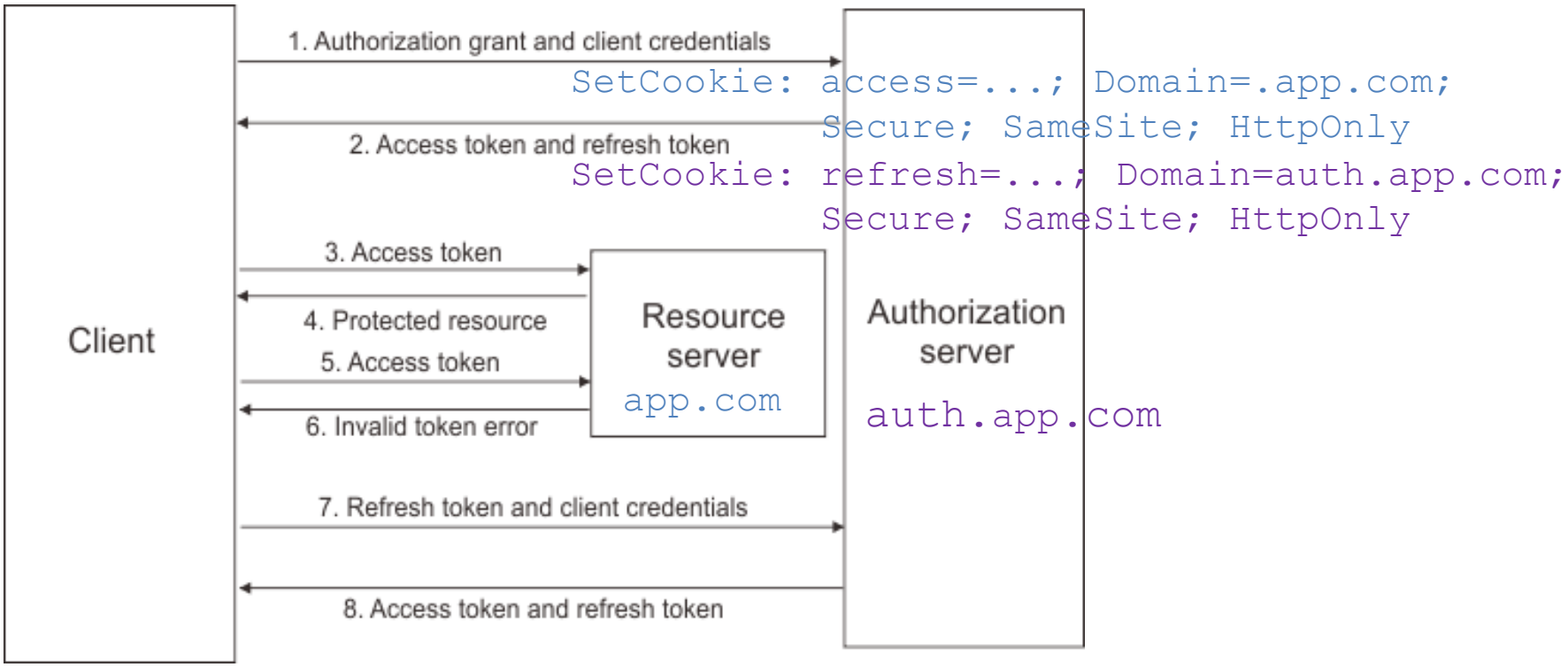
Sateless Sessions



Evaluation

- Complexity: simple with aid from libraries
- Reliance on HTTPS: Yes
- Reliance on CSRF protection: No
- Replay and integrity protection: Relies on TLS
- Session management: Limited
- Tips:
  - Use access and refresh tokens
  - Do ***not*** save tokens in local or session storage

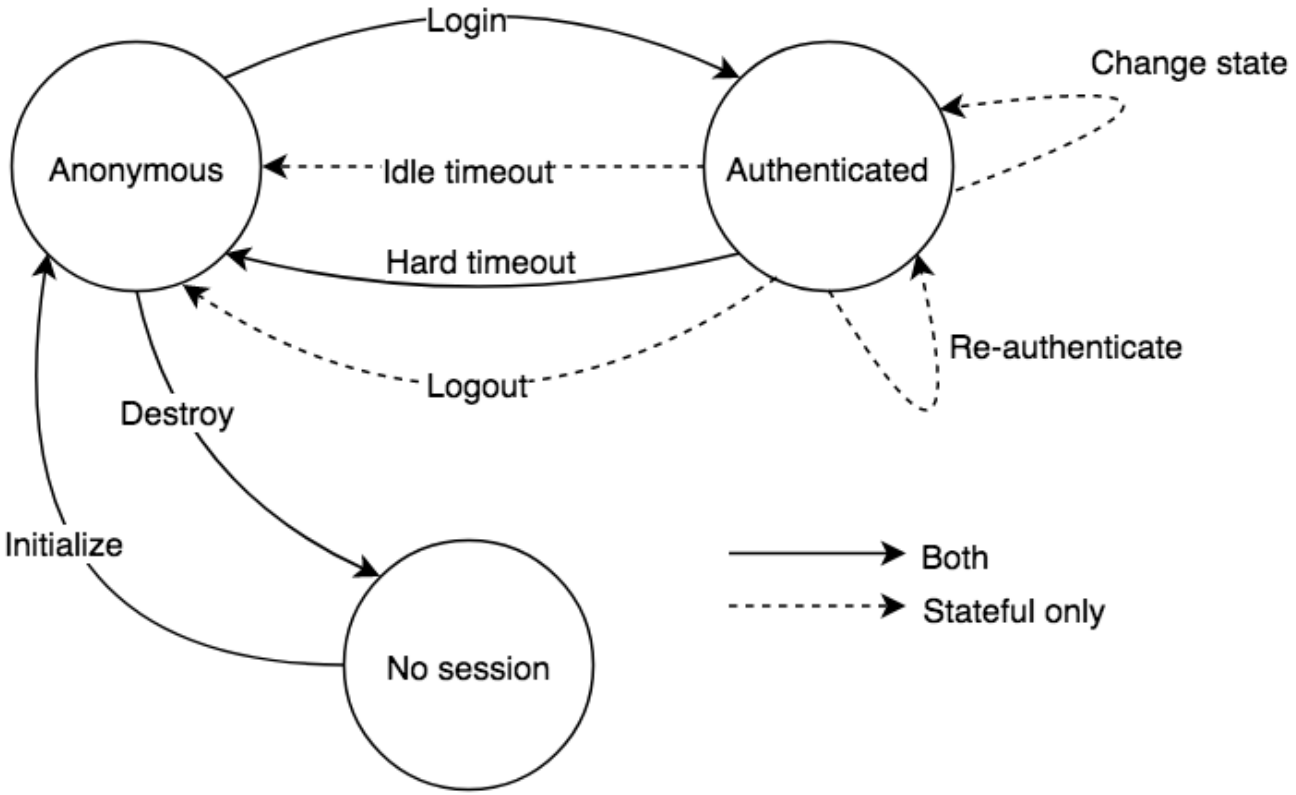
Tips



- Secure → No token stealing
- HttpOnly → No XSS
- SameSite → No CSRF

Statefull or Sateless?

- Stateless: more scalable, but simpler lifecycle
  - Good for single-page sites, APIs, or mobile apps



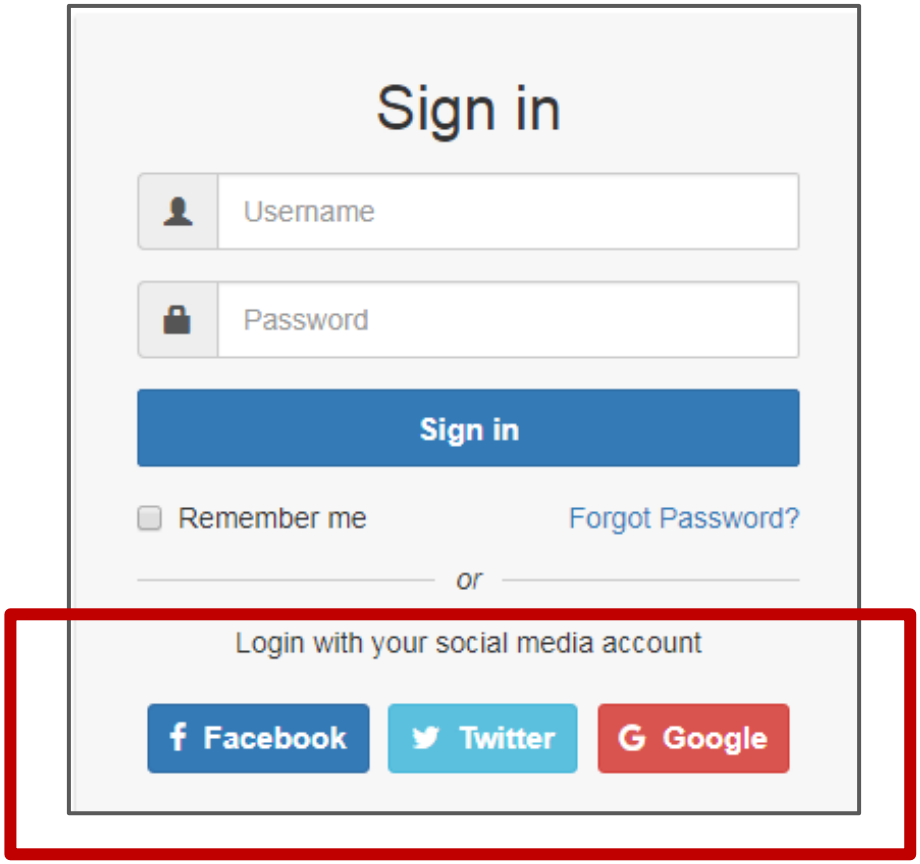
More Authentication Schemes

- For server-to-server communications
  - Based on symmetric/asymmetric key cryptography
- Signature Schemes
  - Idea: to digitally sign every request to prevent request tempering
  - Used by AWS
- TLS Client Certificates
  - Idea: to use TLS certificate to authenticate each other

Outline

- Security risks of web applications
  - Injection, ***broken authentication***, XSS, CSRF, etc.
  - Checklist of 23 Node.js security best practices
- Auth: Authentication, authorization, and session management
  - HTTP Basic auth
  - HTTP Digest auth
  - Cookies for stateful sessions
  - Bearer tokens for stateless sessions
- Single Sign On (SSO)

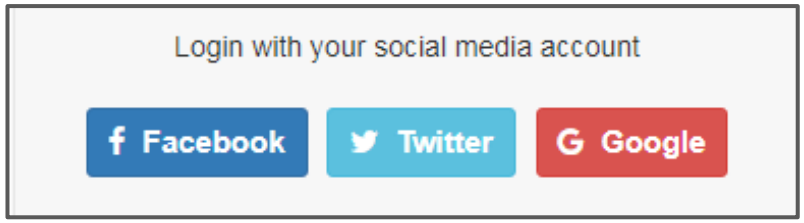
Signgle Sign-On (SSO)

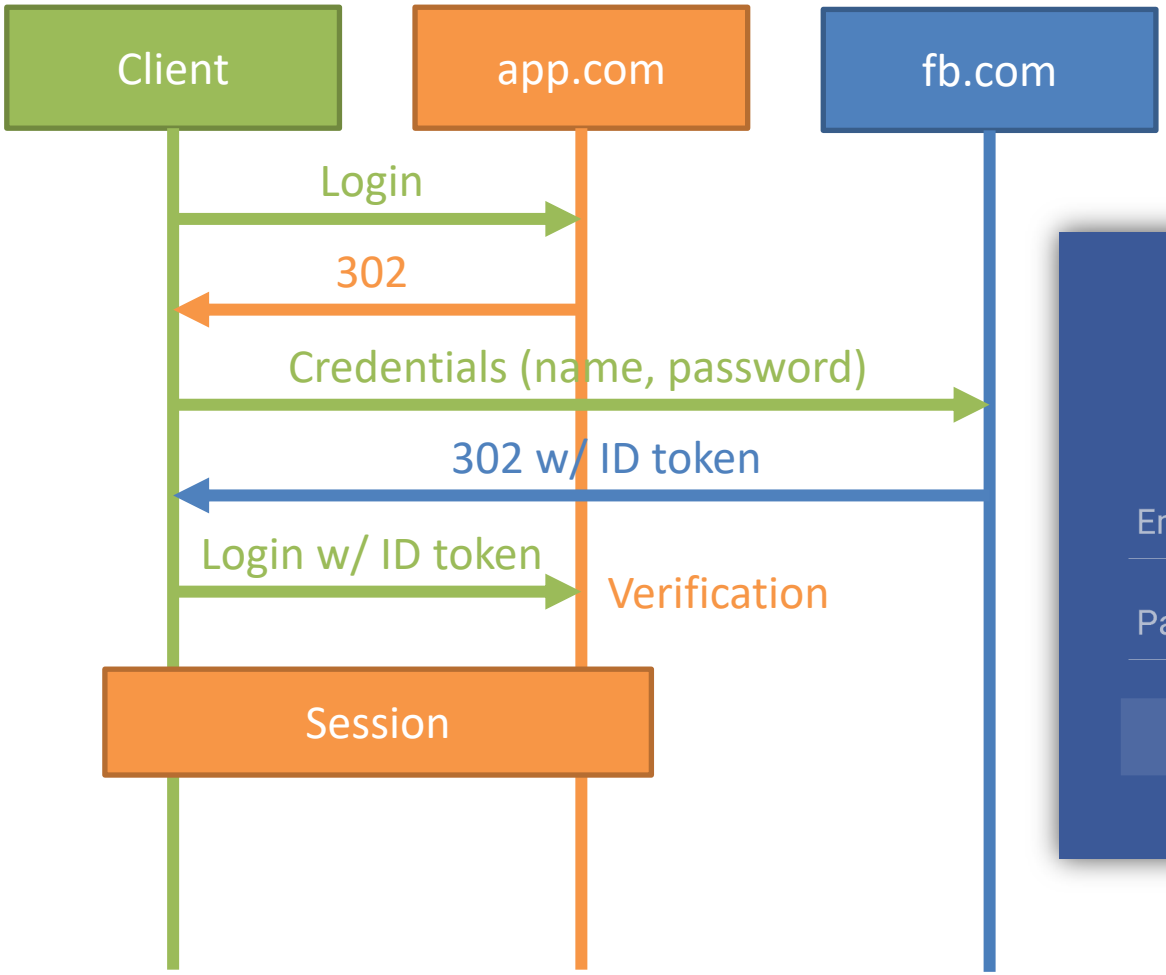


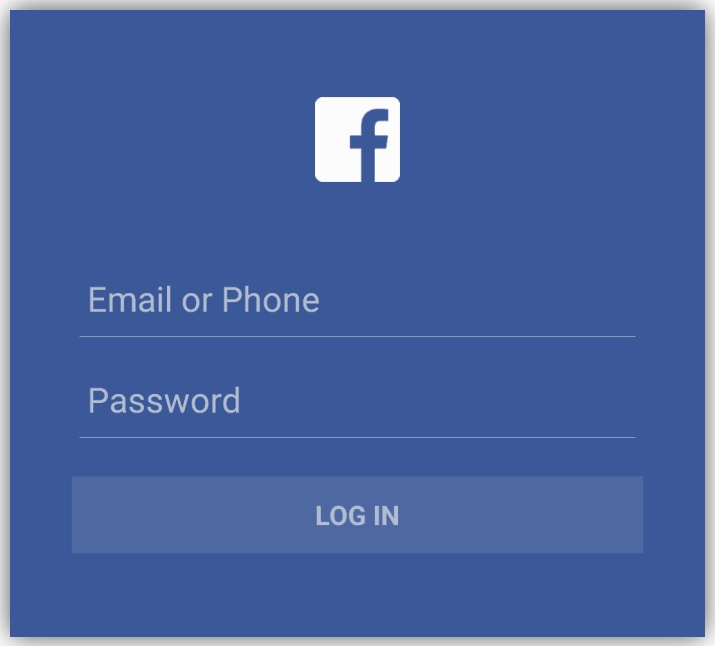
Open ID Connect (OIDC) vs. OAuth

- Authentication
- Authorization

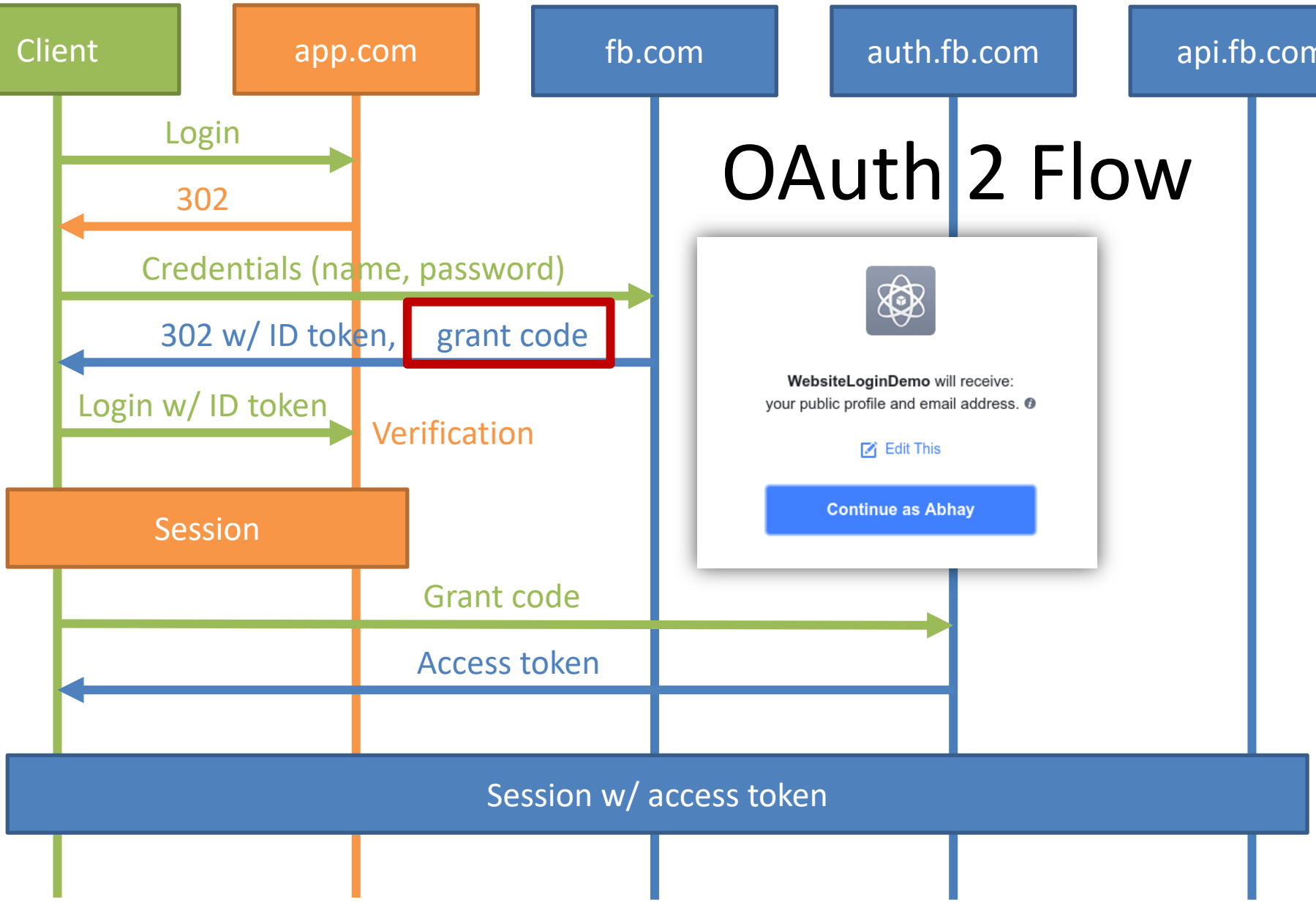
OIDC Flow


Login with your social media account
Facebook
Twitter
Google


Client
app.com
fb.com
Login
302
Credentials (name, password)
302 w/ ID token
Login w/ ID token
Verification
Session


Email or Phone
Password
LOG IN

OAuth 2 Flow


Client
app.com
fb.com
auth.fb.com
api.fb.com
Login
302
Credentials (name, password)
302 w/ ID token,
grant code
Login w/ ID token
Verification
Session
Grant code
Access token
Session w/ access token
WebsiteLoginDemo will receive:
your public profile and email address.
Edit This
Continue as Abhay