

Authentication and authorization in modern JavaScript web applications

How hard can it be?

Brock Allen
brockallen@gmail.com
<http://brockallen.com>
@BrockLAllen



Outline

- Constraints with JavaScript web applications
 - Affects how we implement security
- OpenID Connect
 - Authentication for JavaScript web applications
 - Authentication and authorization to APIs
- Application considerations
 - Token validation
 - Token management

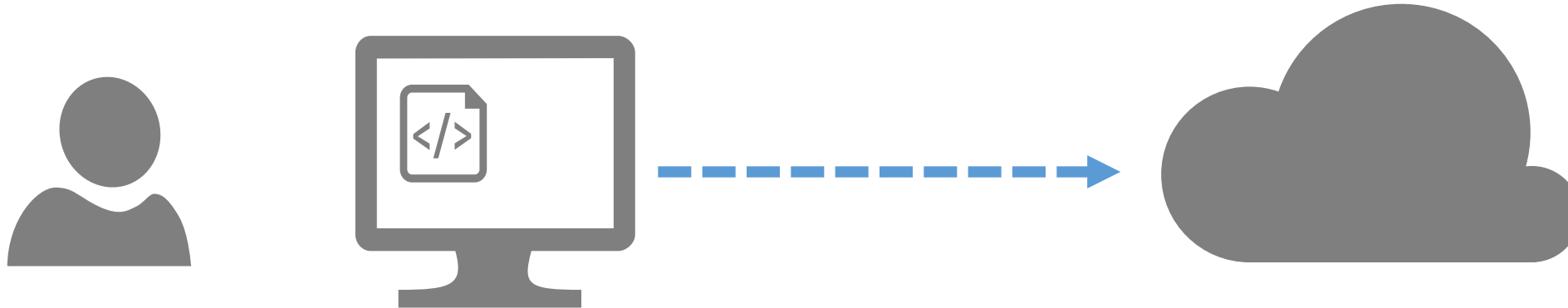
Modern/Pure JavaScript apps

- Client

- Browser-based
- Entirely JavaScript (SPA)
- Dynamic rendering all client side

- Server

- Thin server
- Static content (HTML, JS, CSS, etc.)
- Ajax endpoints (HTTP APIs)



Securing modern JavaScript apps

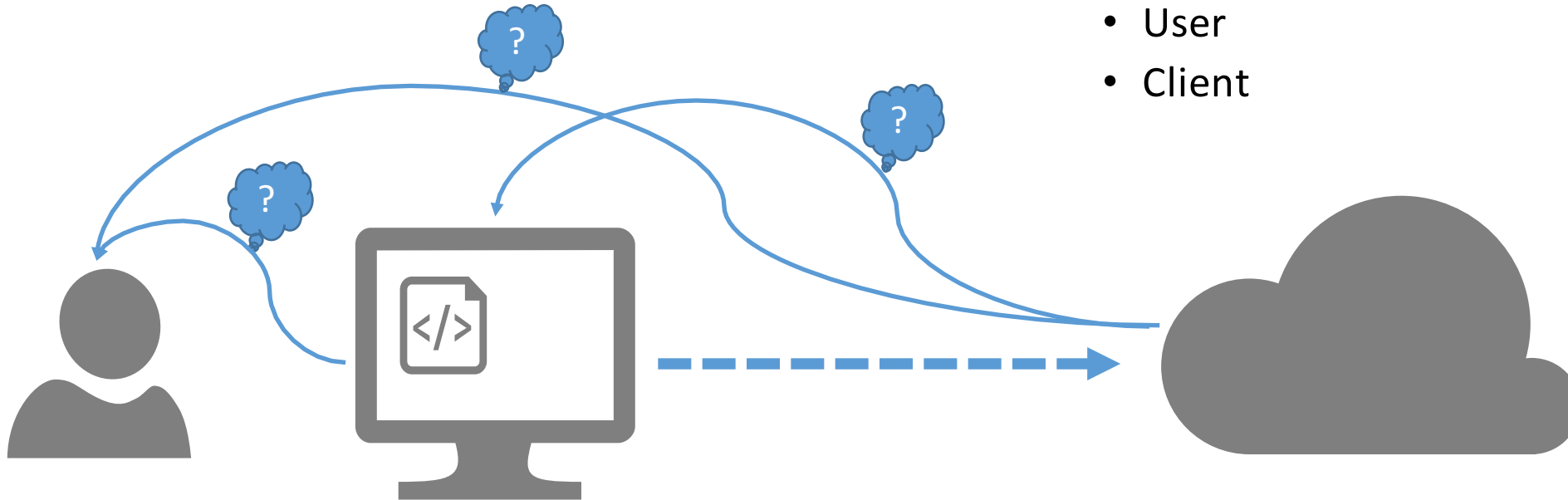
- Client

- Who is the user

- Server

- Who is the caller

- User
 - Client



No more cookies for security

- Cookies are the typical approach for server-side applications
 - But not appropriate for modern JavaScript apps
- Modern apps don't have/use server-side HTML framework
 - SPAs (or mobile apps) are doing the UI client-side
- APIs can't use cookies
 - API might be cross-domain
 - Cookies don't make sense for non-browser clients
 - Cross-site request forgery (XSRF) security issues

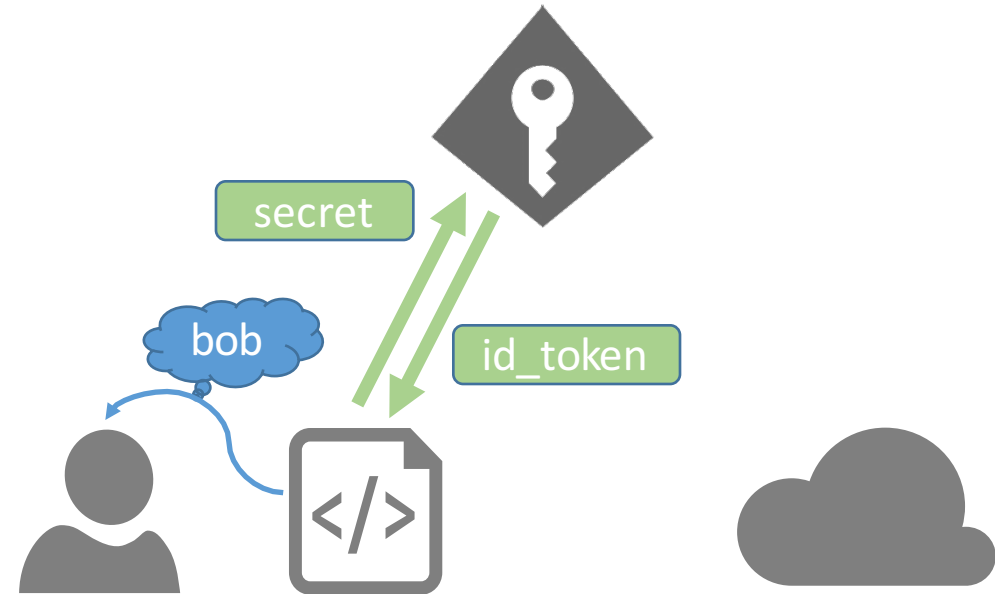
OpenID Connect for security

- OpenID Connect (OIDC) modern security protocol
 - Designed for modern application types (client-side, server-side, and mobile)
- Allows for authentication to client application
 - With *id_token*
- Allows for securing server APIs
 - With *access_token*



Authentication in JS-based apps

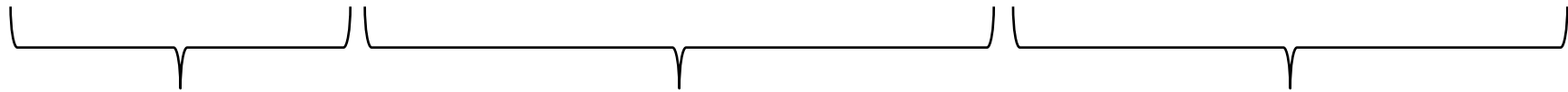
- OpenId Provider (OP)
 - Issues tokens
- 1) Client makes request to OP
 - User authenticates
 - User consents (optional)
- 2) OP returns to client
 - Accept id token
 - Client validates id token



Id tokens

- Format is JSON web token (JWT)

eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOiJ0eMD.4MTkzODAsDQogImh0dHA6Ly9leGFt



Header

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "x5t": "mj399j..."  
}
```

Claims

```
{  
  "iss": "https://idsrv3",  
  "exp": 1340819380,  
  "aud": "app1",  
  "nonce": "289347898934823",  
  
  "sub": "182jmm199",  
  "email": "alice@alice.com",  
  "email_verified": true,  
  "amr": "password",  
  "auth_time": 12340819300  
}
```

Signature

Validating id tokens

- Steps to validate:
 1. Verify **state** is same as sent in request (prevents XSRF/replay)
 2. Base64Url decode **id_token** and parse into JSON (formatting step)
 3. Verify **nonce** is same as sent in request (prevents XSRF/replay)
 4. Validate **signature** on token (establishes trust [requires crypto])
 5. Validate **iss** same as issuer of OIDC OP (establishes trust)
 6. Validate **aud** same as this client's identifier (prevents privilege escalation)
 7. Validate **exp** is still valid (prevents stale tokens)

OidcClient

- JavaScript helper class that implements OIDC protocol
 - Includes **id_token** validation
 - Including crypto implementation
 - Heavy use of promises
- **<http://github.com/IdentityModel/oidc-client-js>**
 - Also available via npm



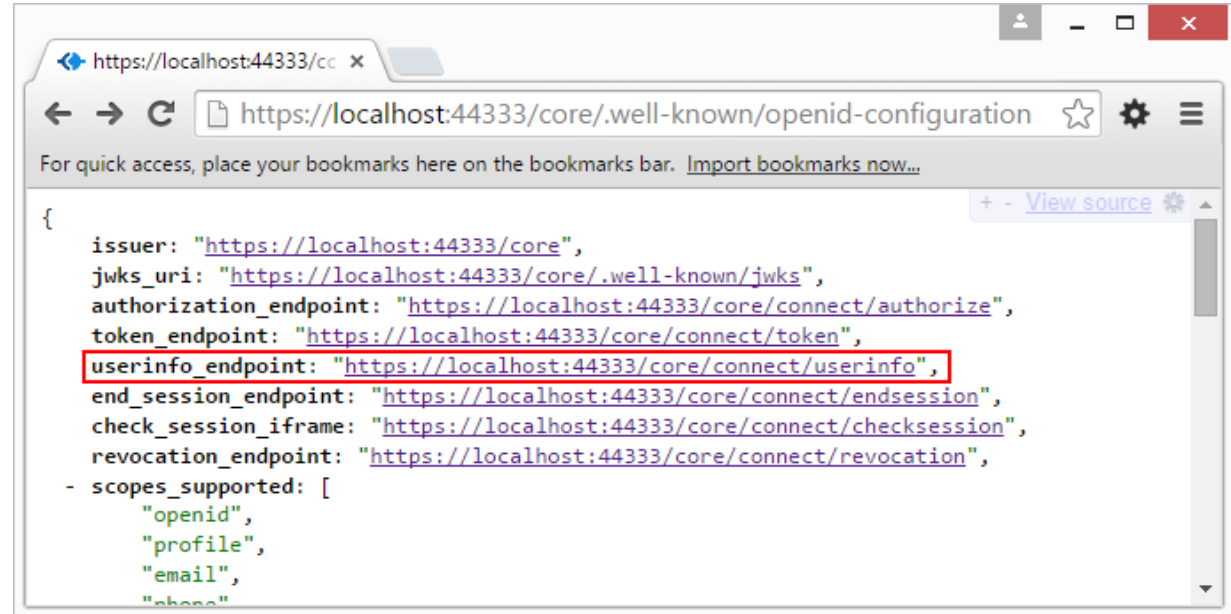
More identity data

- Might need more than ***sub*** (subject) claim
- ***scope*** used to ask for more identity data

Scope	Claims
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated_at
email	email, email_verified
address	address
phone	phone_number, phone_number_verified
offline_access	requests refresh token

More identity data with user profile

- Id token might become too large
 - Needs to fit into URL
- OIDC defines user info endpoint
 - Ajax call to load user profile
 - Requires authorization with an access token obtained in OIDC request

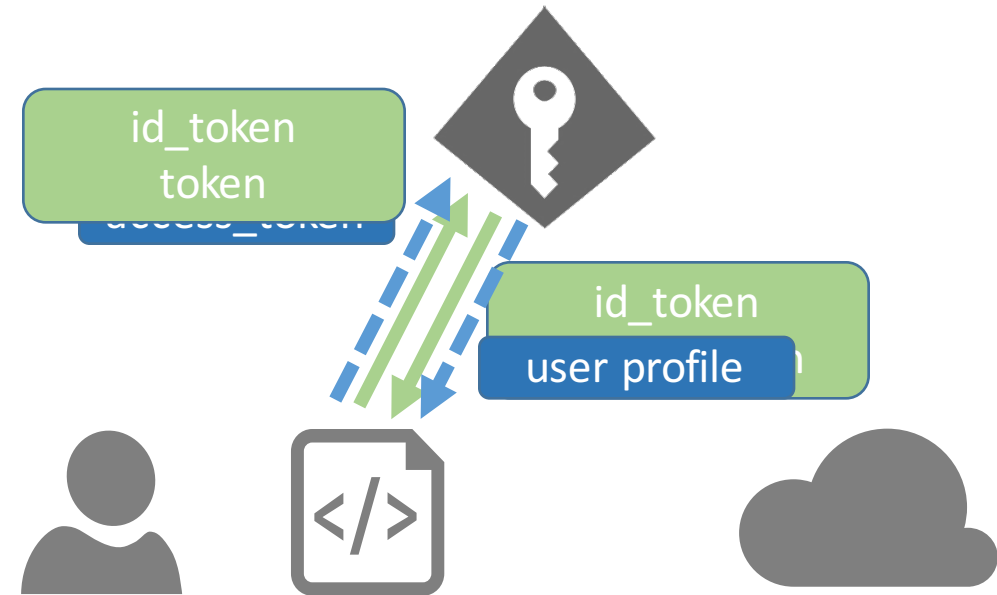


The screenshot shows a web browser window with the address bar displaying `https://localhost:44333/cc`. The page title is `https://localhost:44333/core/.well-known/openid-configuration`. The page content is a JSON object representing the OpenID Connect configuration. The `userinfo_endpoint` is highlighted with a red box.

```
{
  issuer: "https://localhost:44333/core",
  jwks_uri: "https://localhost:44333/core/.well-known/jwks",
  authorization_endpoint: "https://localhost:44333/core/connect/authorize",
  token_endpoint: "https://localhost:44333/core/connect/token",
  userinfo_endpoint: "https://localhost:44333/core/connect/userinfo",
  end_session_endpoint: "https://localhost:44333/core/connect/endsession",
  check_session_iframe: "https://localhost:44333/core/connect/checksession",
  revocation_endpoint: "https://localhost:44333/core/connect/revocation",
  - scopes_supported: [
    "openid",
    "profile",
    "email",
    "phone"
  ]
}
```

Requesting access token

- Add "token" to ***response_type*** parameter to authorization endpoint
- More validation required (same as before, plus):
 - Hash access token and compare left half to ***at_hash*** in id token (ensures id token is paired with access token)



Using access token to call user profile

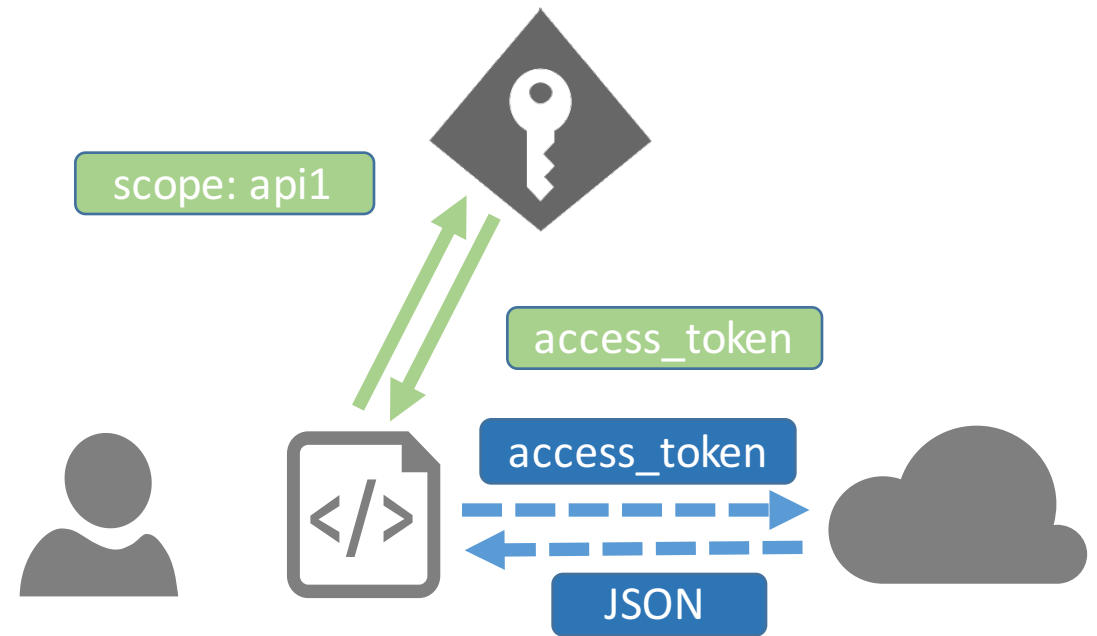
- Access token passed as ***Authorization*** HTTP request header
- Response is JSON of user profile based upon requested scopes

```
var xhr = new XMLHttpRequest();
xhr.onload = function () {
    var user_profile = JSON.parse(xhr.response);
}

xhr.open("GET", user_profile_endpoint);
xhr.setRequestHeader("Authorization", "Bearer " + access_token);
xhr.send();
```

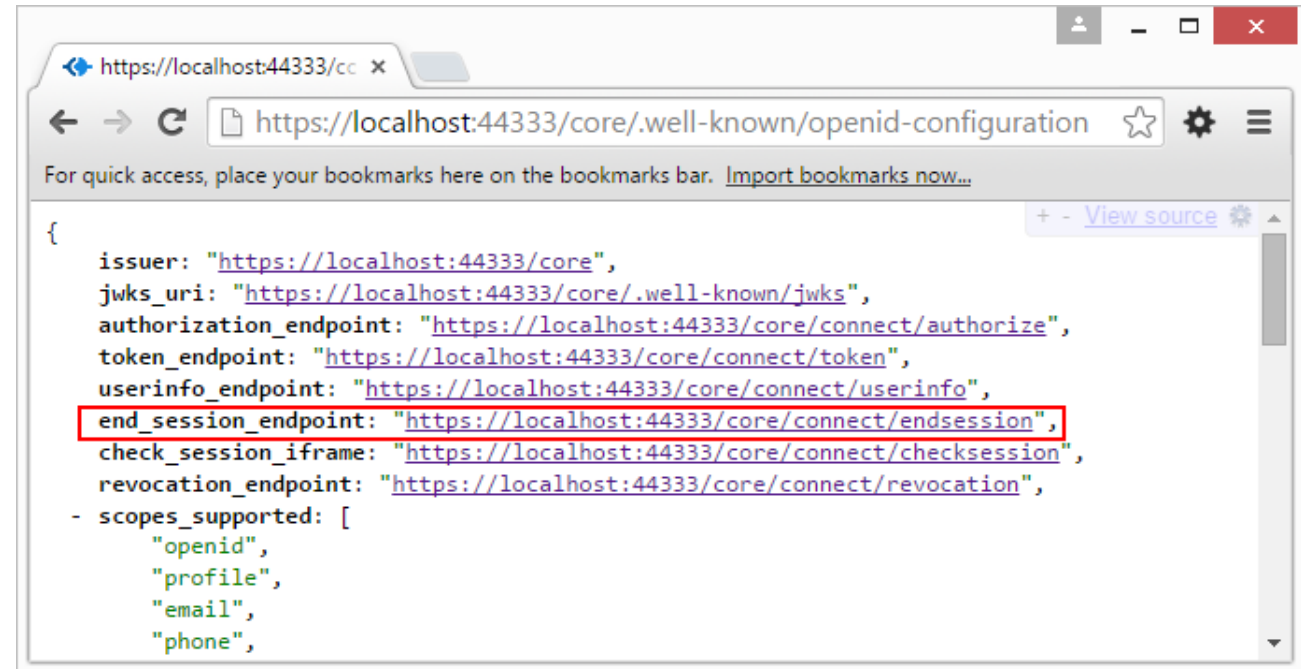
Calling other web APIs

- APIs use access token from same OIDC OP
- Just need to request more scopes



Logout

- Throw away tokens in client
- Signing out of OIDC OP
 - Must make request to OP
- Post logout redirect
 - Must pass redirect URL as ***post_logout_redirect_uri***
 - Must pass original id token as ***id_token_hint***



```
{
  issuer: "https://localhost:44333/core",
  jwks_uri: "https://localhost:44333/core/.well-known/jwks",
  authorization_endpoint: "https://localhost:44333/core/connect/authorize",
  token_endpoint: "https://localhost:44333/core/connect/token",
  userinfo_endpoint: "https://localhost:44333/core/connect/userinfo",
  end_session_endpoint: "https://localhost:44333/core/connect/endsession",
  check_session_iframe: "https://localhost:44333/core/connect/checksession",
  revocation_endpoint: "https://localhost:44333/core/connect/revocation",
  - scopes_supported: [
    "openid",
    "profile",
    "email",
    "phone",
  ]
}
```


Token management

- Token storage
 - localStorage
 - sessionStorage
 - indexedDb
- Token expiration
 - Access tokens expire (1h, 10h, 1d, 30d, whatever)
 - Need a way to manage this lifetime
 - Wait for 401 from API
 - Renew prior to expiration

UserManager

- JavaScript helper class to manage tokens, expirations, and renewals
 - Implemented in terms of ***OidcClient***
- Part of oidc-client-js library



Renewing access tokens

- Unlike cookies, access tokens don't slide
 - Must return to OIDC OP to obtain new access token
- Start from scratch
 - Almost same as starting all over
 - Don't want to lose the state in the app
- Popup window
 - Better than starting over
 - Somewhat intrusive
- Hidden iframe
 - Nice tradeoff for usability

Summary

- Cookies aren't appropriate for modern JavaScript apps
 - XSRF issues
- OpenID Connect is the one protocol to rule them all
 - Allows for authentication and authorization
- Client-side applications have non-trivial work to do (depending on requirements)
 - Token validation
 - Token management