

Authentication and authorization in modern JavaScript web applications

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



Outline

- Browser-based JavaScript applications and threats
- Approaches to security
- Security protocols
- Application considerations

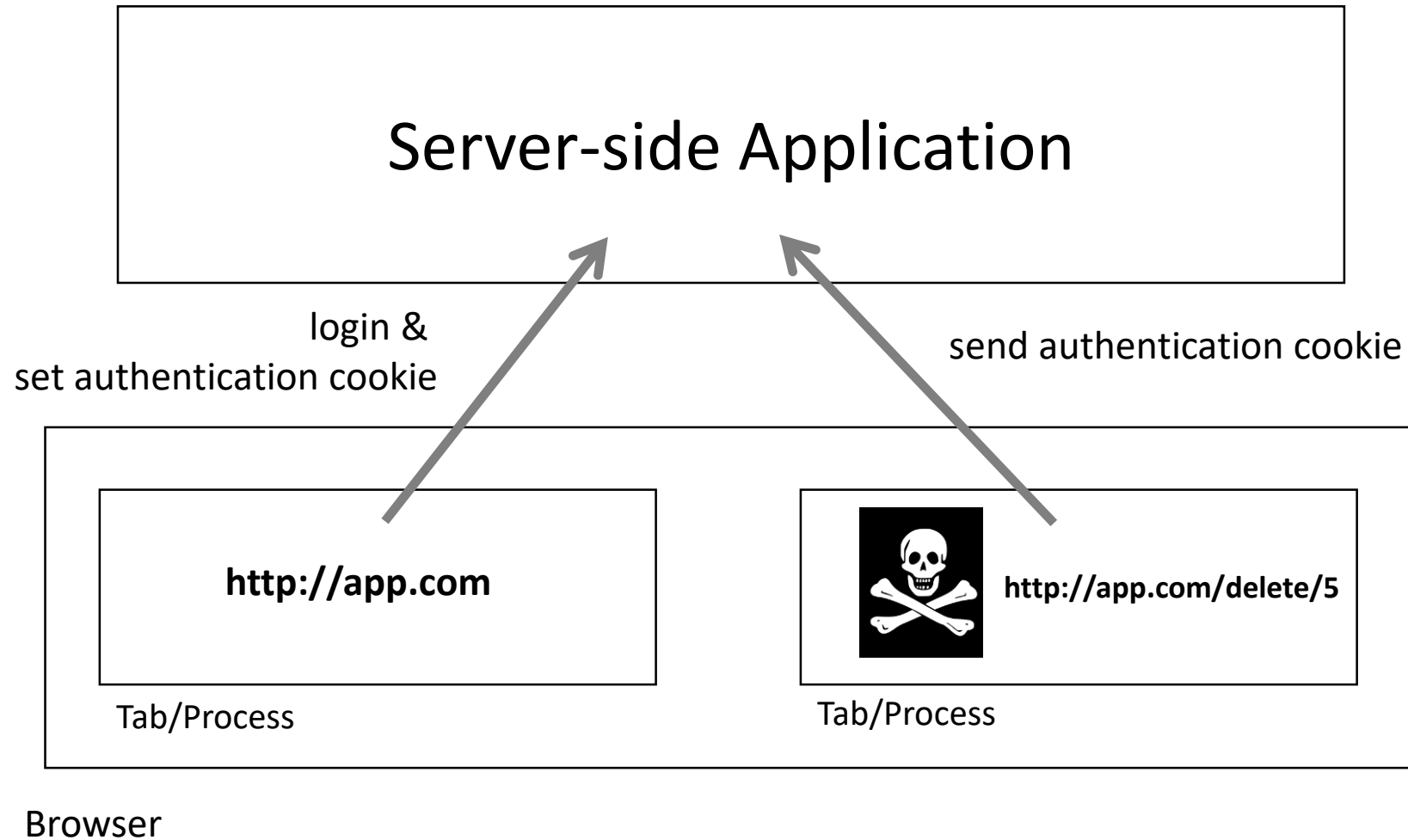
Styles of JavaScript browser-based apps

- Legacy/Mixed
 - Mainly consists of server-side code (ASP.NET, MVC, etc.)
 - Some client-side JavaScript making Ajax calls
 - Using cookies for authentication
- Modern/SPA
 - Server only serves static content
 - All application logic and rendering client-side in JavaScript
 - Calls to server APIs via Ajax
 - Could use cookies or tokens for authentication

Threats against JavaScript apps

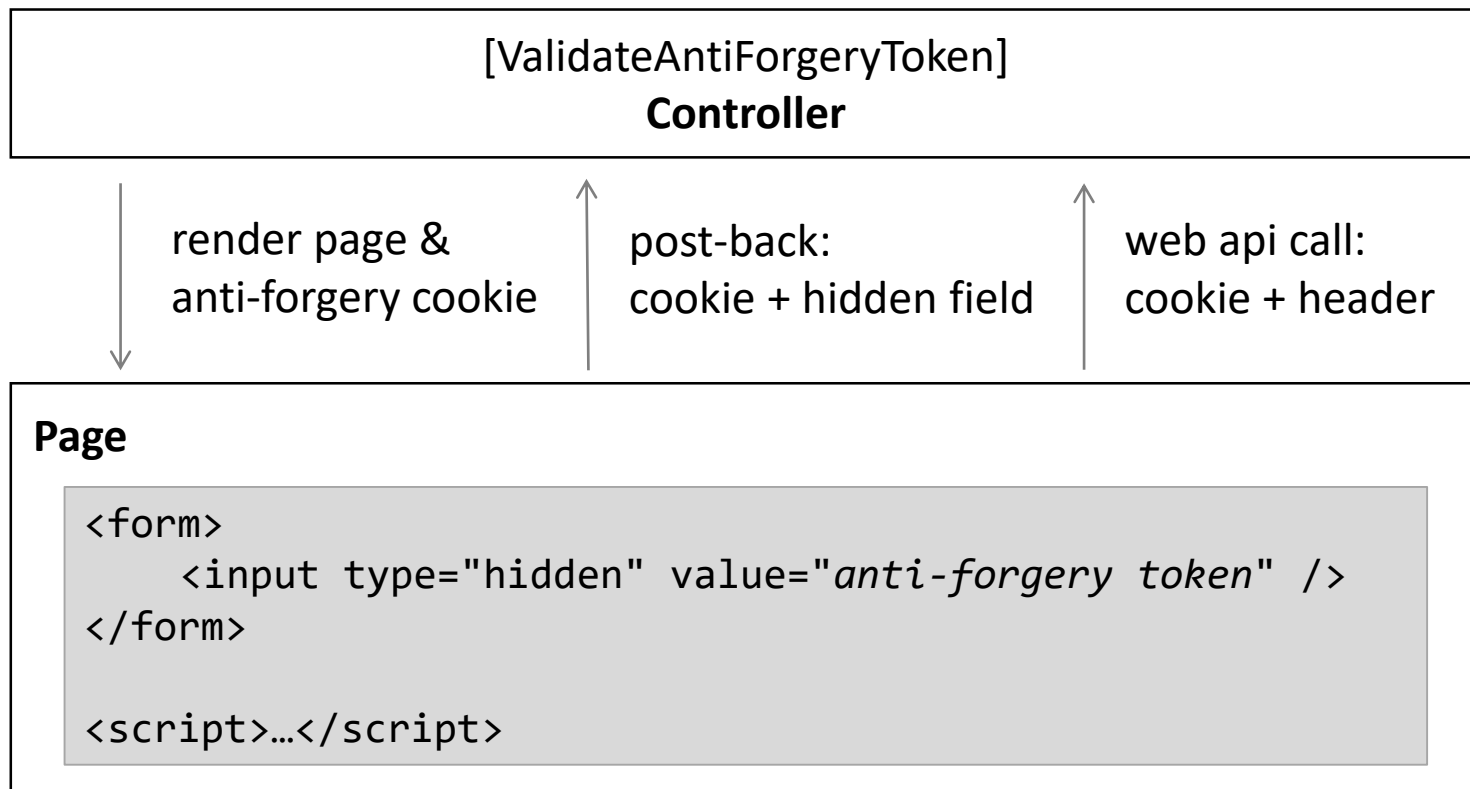
- Cross-site scripting (XSS)
 - Injected JavaScript can control page
- Mitigations
 - HTTP-only cookies prevent exfiltration
 - Content security policy (CSP) restricts sources and behavior of code running on page
- Cross-site request forgery (CSRF)
 - Websites making malicious requests to your server endpoints
- Mitigations
 - Anti-forgery tokens authenticate app making request
 - Easy to add for server-rendered apps
 - Often not done for Ajax endpoints
 - Same-site cookies scopes cookie to calling origin
 - Token based security

CSRF attack



CSRF mitigation with anti-forgery tokens

- Add explicit “credential” on every request
 - Supported with ASP.NET Core’s anti-forgery feature



CSRF mitigation with same-site cookies

- Browser only sends cookie from page from same origin
 - Enabled by default for ASP.NET Core authentication cookies

```
HTTP/1.1 200 OK
Set-Cookie: key=value; HttpOnly; SameSite=strict
```

- Decent browser support (as of 2019)

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *
	12-15	2-59	4-50	3.1-11.1	10-38			
6-10	¹ 16-17	60-65	51-72	12	39-57	3.2-11.4		2.1-4.4.4
^{1 2} 11	18	66	73	12.1	58	12.1	all	67
		67-68	74-76	TP		12.2		

<https://caniuse.com/#search=samesite>

Restrictions using cookies for APIs

- Same-site cookies is still new
 - Some people still on older browsers
- API must have server-side code to issue cookie to browser
 - Easy for legacy/mixed, more work for SPA/modern
- Application must be same origin as API
 - Will your app always be the same domain as the API(s)?
- Browser-based app is only app that can call the API
 - Will you ever want other apps to use the API?

Token based authentication

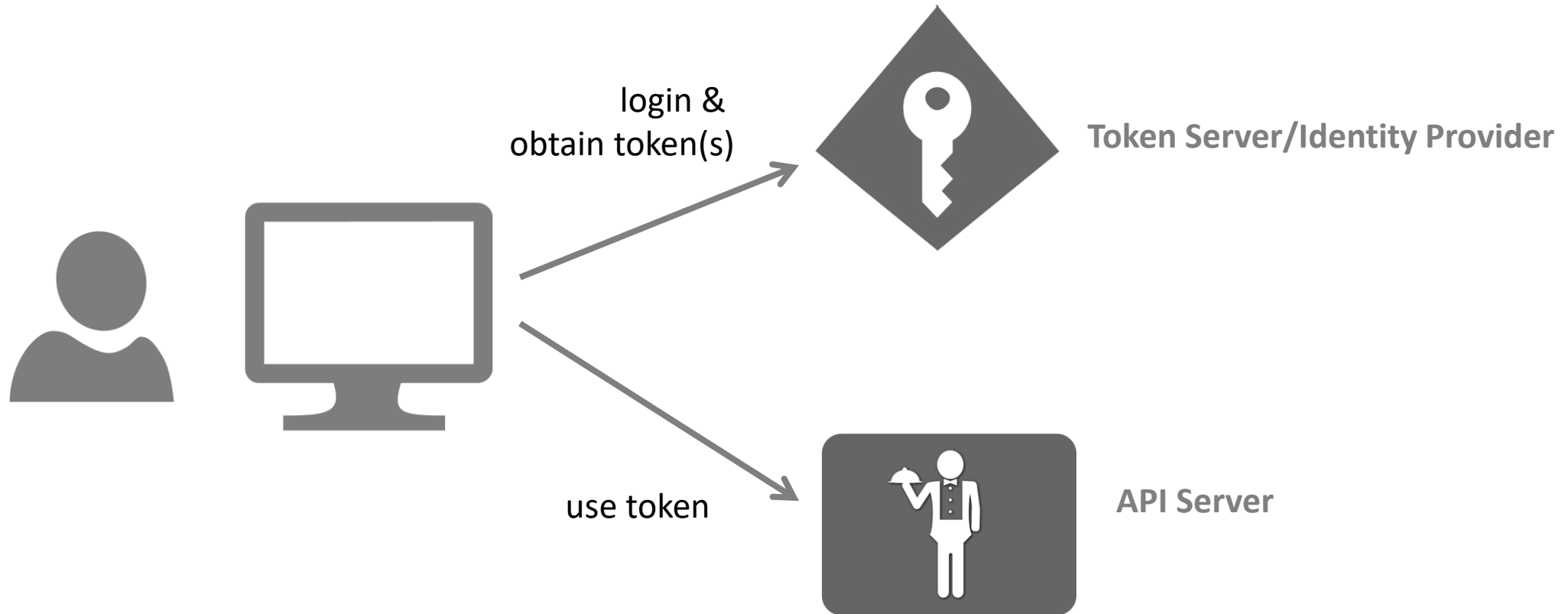
- Tokens are a different form of credential
 - Typically use JSON web token (JWT)
 - Sent as header to authenticate HTTP request
 - Predates same-site cookies as a solution for CSRF
- Tokens help solve the architectural issues
 - More than just browser-based apps can use tokens
 - API is client agnostic
 - No cookie management needed in API
 - Can call APIs cross-domain
 - SSO for users

OpenID Connect and OAuth 2.0

- Protocols for obtaining and using tokens
- Allows for authentication to client application
 - With *id_token*
- Allows for securing server APIs
 - With *access_token*



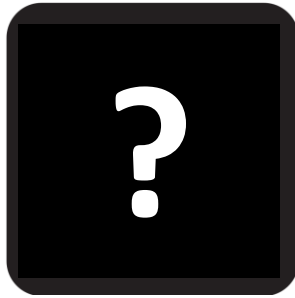
Client, token server, and API server



Protocol flows

- Flows define mechanics to obtain tokens in client
 - Different based on type of app (e.g. server, mobile, SPA)
- Ongoing work in IETF to produce useful guidance
 - <https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps>
- SPA apps use **authorization code flow (with PKCE)**
 - Previous guidance was to use implicit flow

Token server endpoints



**Discovery
Endpoint**



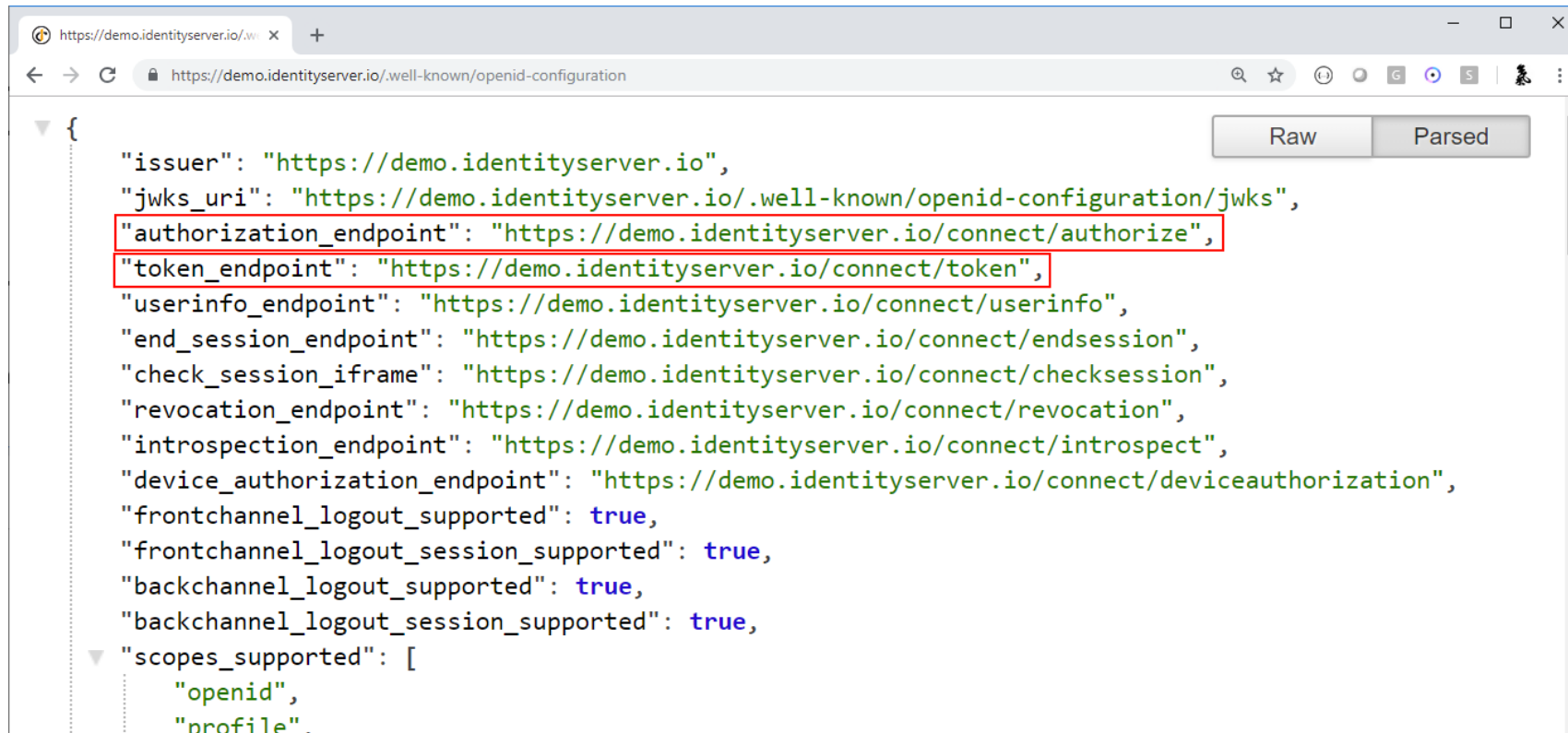
**Authorize
Endpoint**



**Token
Endpoint**

Discovery endpoint

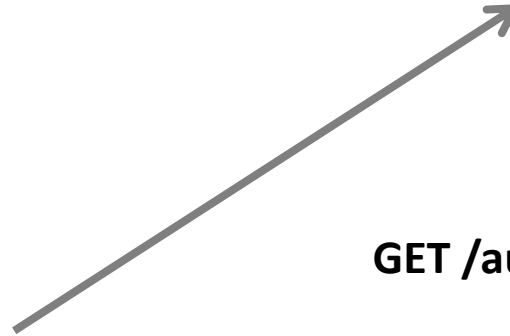
- Contains discovery document JSON
 - Metadata about token server to automate client app configuration



The screenshot shows a web browser window with the address bar displaying `https://demo.identityserver.io/.well-known/openid-configuration`. The page content is a JSON document representing the OpenID configuration. The JSON is displayed in a code editor with a 'Raw' and 'Parsed' toggle. The 'Parsed' view is selected, showing the JSON structure with syntax highlighting. The JSON object contains various endpoints and supported features. The 'authorization_endpoint' and 'token_endpoint' fields are highlighted with red boxes.

```
{
  "issuer": "https://demo.identityserver.io",
  "jwks_uri": "https://demo.identityserver.io/.well-known/openid-configuration/jwks",
  "authorization_endpoint": "https://demo.identityserver.io/connect/authorize",
  "token_endpoint": "https://demo.identityserver.io/connect/token",
  "userinfo_endpoint": "https://demo.identityserver.io/connect/userinfo",
  "end_session_endpoint": "https://demo.identityserver.io/connect/endsession",
  "check_session_iframe": "https://demo.identityserver.io/connect/checksession",
  "revocation_endpoint": "https://demo.identityserver.io/connect/revocation",
  "introspection_endpoint": "https://demo.identityserver.io/connect/introspect",
  "device_authorization_endpoint": "https://demo.identityserver.io/connect/deviceauthorization",
  "frontchannel_logout_supported": true,
  "frontchannel_logout_session_supported": true,
  "backchannel_logout_supported": true,
  "backchannel_logout_session_supported": true,
  "scopes_supported": [
    "openid",
    "profile"
  ]
}
```

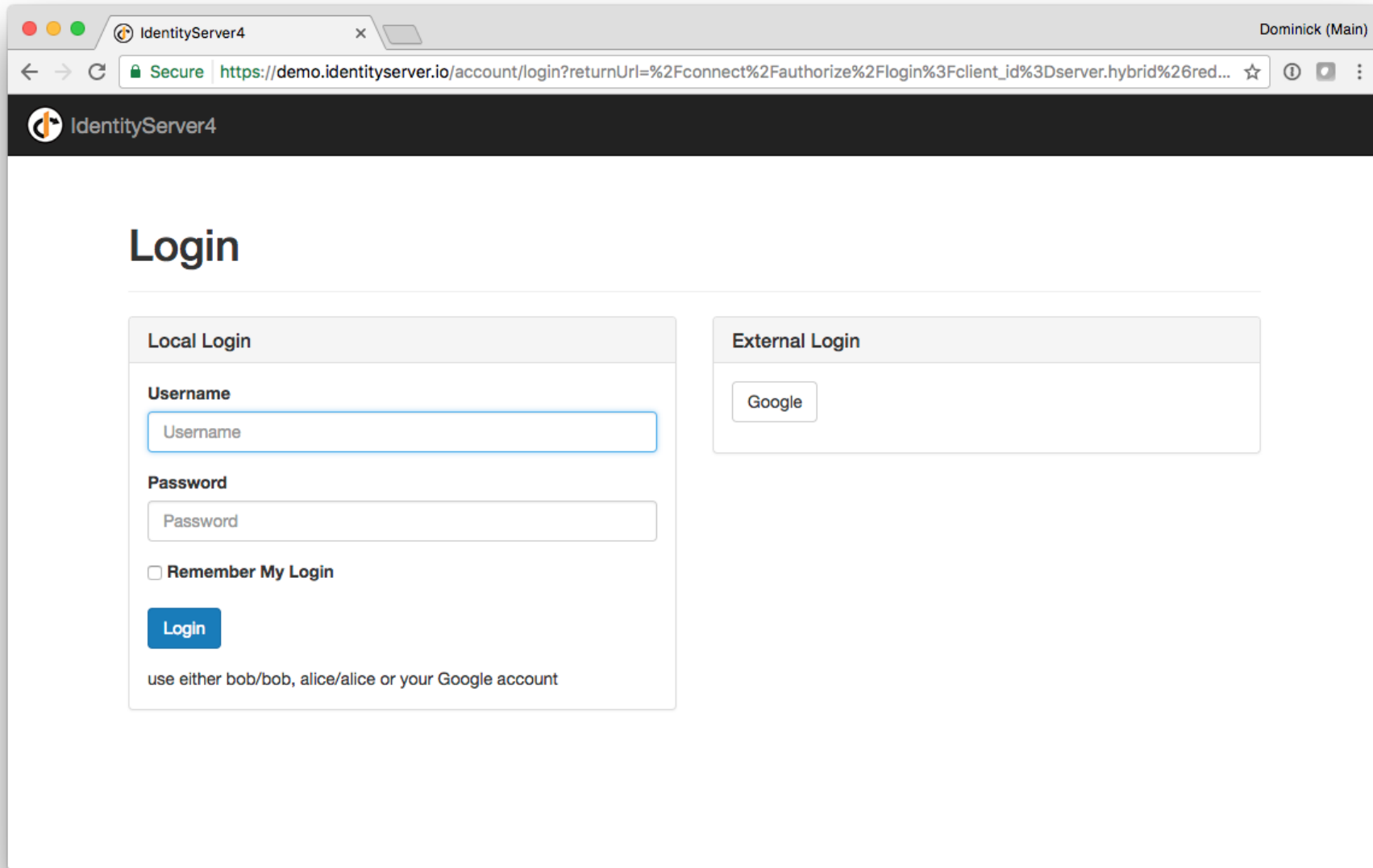
Authorization request



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/cb.html
&response_type=code
&nonce=289...a23
&scope=openid profile email api1 api2
&code_challenge=x929...1921

Authentication



The screenshot shows a web browser window with the title "IdentityServer4" and a user profile "Dominick (Main)" in the top right corner. The address bar shows a secure connection to "https://demo.identityserver.io/account/login?returnUrl=%2Fconnect%2Fauthorize%2Flogin%3Fclient_id%3Dserver.hybrid%26red...". The page header features the IdentityServer4 logo and name. The main content area is titled "Login" and is divided into two sections: "Local Login" and "External Login".

Local Login

Username

Password

☐ Remember My Login

Login

use either bob/bob, alice/alice or your Google account

External Login

Google

Authorization response



set SSO cookie

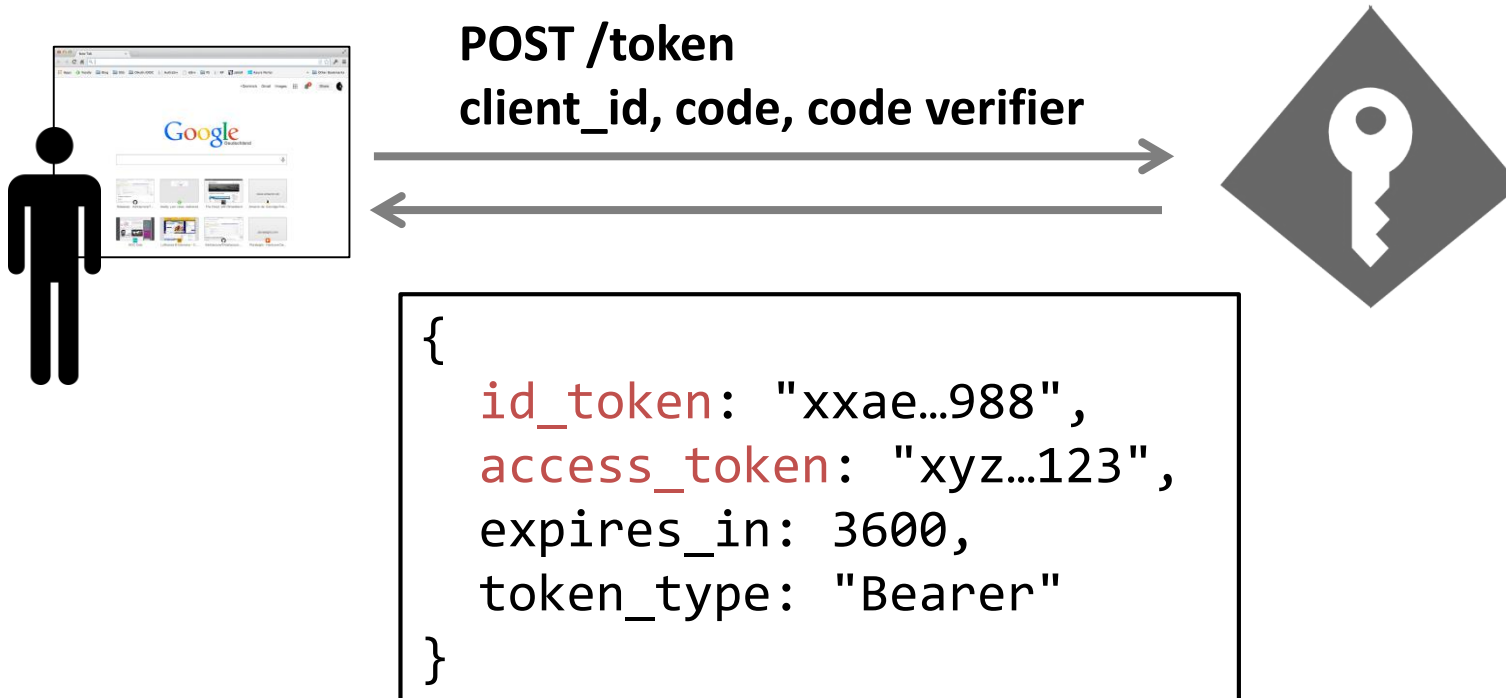


GET <https://app.com/cb.html?code=238...823j>



Token endpoint exchange

- Ajax request made to token endpoint to exchange code for tokens
 - Using client id and code verifier



Id token

- Contains user's claims
 - Format is JSON web token (JWT)

eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header

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

Claims

```
{
  "iss": "https://identityserver.io",
  "exp": 1340819380,
  "aud": "app1",
  "nonce": "289347898934a23",

  "sub": "182jmm199",
  "email": "alice@alice.com",
  "name": "Alice Smith",
  "amr": [ "pwd" ],
  "auth_time": 12340819300
}
```

Signature

Access token

- Application should store access token
 - localStorage
 - sessionStorage
 - indexedDb
- Use access token to call APIs

Using access token to call APIs

- Access token passed as ***Authorization*** HTTP request header
 - Using “Bearer” scheme

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

xhr.open("GET", "https://api.app.com/some_endpoint");
xhr.setRequestHeader("Authorization", "Bearer " + access_token);
xhr.send();
```

Validating access tokens at API

- ASP.NET Core provides JWT bearer authentication handler
 - Populates HttpContext.User with claims from token

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication("Bearer")
        .AddJwtBearer("Bearer", options =>
        {
            options.Authority = "https://identityserver.io";
            options.Audience = "your_api_identifier";
        });
}
```

oidc-client

- JavaScript helper class that implements OIDC and OAuth 2.0 protocols
- <http://github.com/IdentityModel/oidc-client-js>
 - Also available via npm & bower



Access token expiration

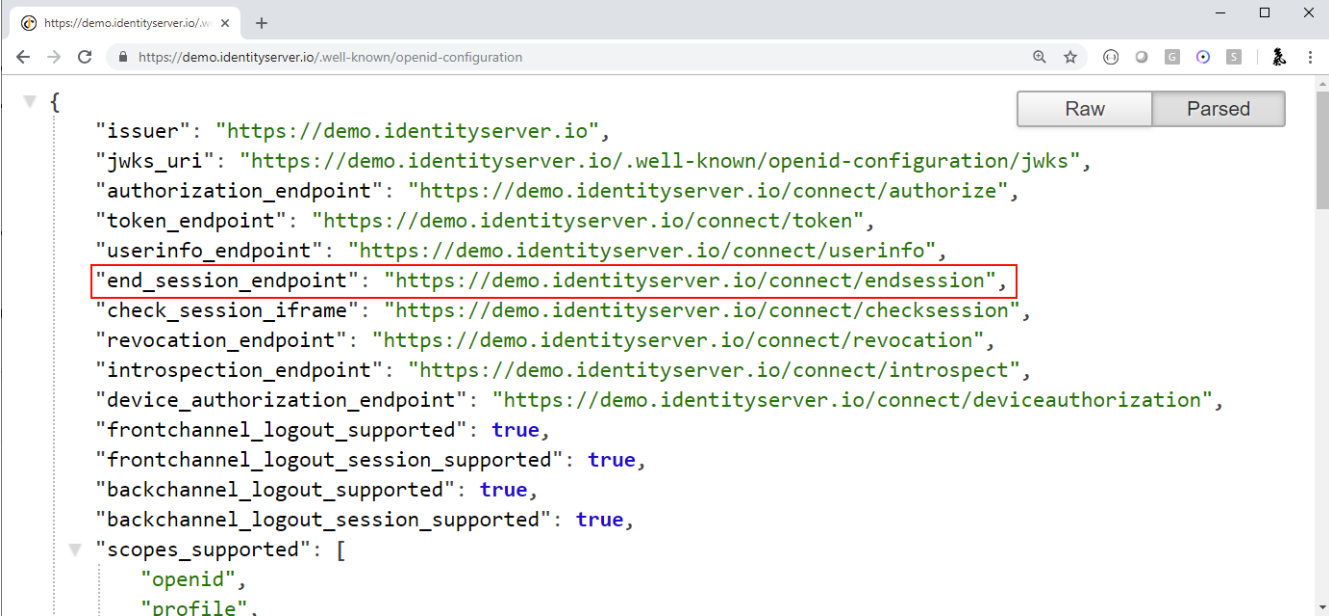
- Access tokens have a fixed lifetime
 - 1h, 10h, 1d, 30d, whatever
- Need a way to manage this lifetime
 - Wait for 401 from API
 - Renew prior to expiration

Renewing access tokens

- Unlike cookies, access tokens don't slide
 - Must return to token server 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

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://demo.identityserver.io",
  "jwks_uri": "https://demo.identityserver.io/.well-known/openid-configuration/jwks",
  "authorization_endpoint": "https://demo.identityserver.io/connect/authorize",
  "token_endpoint": "https://demo.identityserver.io/connect/token",
  "userinfo_endpoint": "https://demo.identityserver.io/connect/userinfo",
  "end_session_endpoint": "https://demo.identityserver.io/connect/endsession",
  "check_session_iframe": "https://demo.identityserver.io/connect/checksession",
  "revocation_endpoint": "https://demo.identityserver.io/connect/revocation",
  "introspection_endpoint": "https://demo.identityserver.io/connect/introspect",
  "device_authorization_endpoint": "https://demo.identityserver.io/connect/deviceauthorization",
  "frontchannel_logout_supported": true,
  "frontchannel_logout_session_supported": true,
  "backchannel_logout_supported": true,
  "backchannel_logout_session_supported": true,
  "scopes_supported": [
    "openid",
    "profile"
  ]
}
```

Summary

- Need XSS and CSRF protection in browser-based JavaScript apps
- Can use same-site cookies for single domain apps and APIs
- Can use token based authentication for more complex scenarios
- Use OpenID Connect and OAuth 2.0 protocols to obtain tokens