
THE UNIVERSITY
of ADELAIDE
Web and Database Computing
adelaide.edu.au
Server APIs and Authentication: Authentication and Sessions

Sessions and Authentication

Server Sessions & Session Tokens

Recall from the previous lecture that cookies can be used to uniquely identify clients, avoiding the need to send login data with each request.

- The sessions software sets a cookie at the client browser with a cryptographically unique session id, known as a **session token**.
- Whenever the browser sends requests to this website, it sends incldues this data (because cookies)
- The server uses the unique session id to associate that client with data that it stores.
- When the session token expires or the server shuts down the session data is lost.

Therefore, if a user has **already been authenticated** by our web application and we know their session token ID, then **ANY request from that session may also be considered authenticated** for the duration of that session.

Basic Login Workflow

1. User visits website, is given a session token
2. User Logs In
    - If login is successful, record the user's details against their session.
3. For all future requests, check if the request's session has a user attached to it
    - If a user is present, we know they've already logged in, and can allow them to proceed.

To log a user out, simply remove their details from that session.

AJAX Site Authentication Workflow

1. Client sends AJAX request for resource.
2. Authenticate & Authorise Request.
    - Does the user have a valid session?
    - Is the user allowed to access the resource?
3. If okay, send 200 with requested resource (DONE)

   ~ OR ~
4. If NOT okay, send a 401/403
    - Client displays login dialog or redirects to login page
5. Client sends credentials
6. If okay, create session and link to user
    - Client may re-send original AJAX request

AJAX Site Authentication Tips

- Initialise sessions by default
- Use Middleware
  - Separate your routes into public/freely accessible vs user/private data
  - Add middleware to authenticate and authorise all requests to the user routes
- Perform all redirects on client (because AJAX)

Website login & Security Challenges

- Can be difficult to implement well.
  - Very easy to make security mistakes.
- Risk of compromise
  - If compromised users may abandon your service or worse...
- Users commonly have poor security practices
  - Users often use the same password multiple places.
- Users don’t like having to create new accounts for every site they visit

The 1st Rule of Security Programming

Don’t implement your own security

(Get someone who knows what they’re doing to do it)

Introducing OpenID Connect

From openid.net:

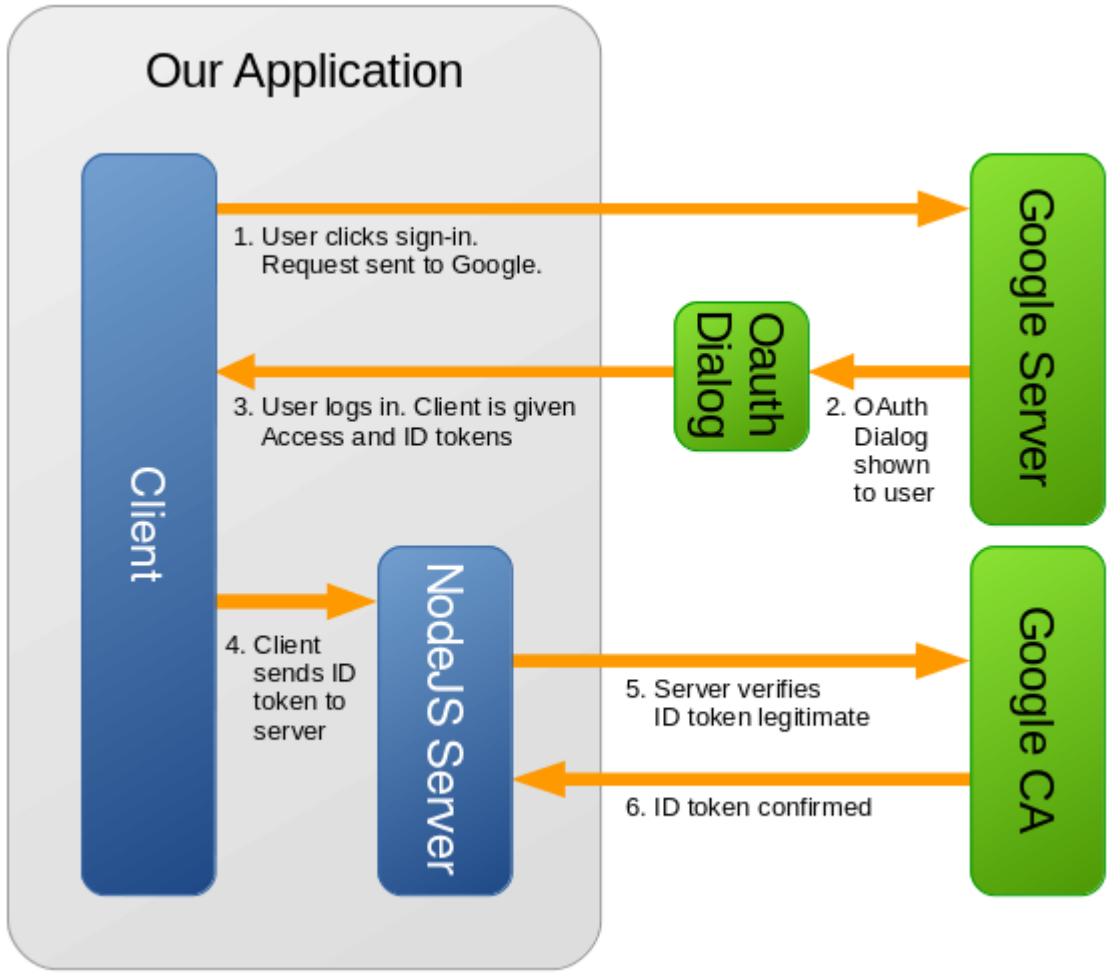
> “OpenID Connect is an interoperable authentication protocol based on the OAuth 2.0 family of specifications.”
>
> “Lets app and site developers authenticate users without taking on the responsibility of storing and managing passwords in the face of an Internet that is well-populated with people trying to compromise your users’ accounts for their own gain.”

In a nutshell:

Use a trusted 3rd Party (Identity Provider) to authenticate users for you

- User clicks OpenID button for their Identity Provider.
- User logs in to Identity Provider's website.
- Identity provider verifies user information and provides verification to our web application.
- Our web application matches identity info to user's account.

More detail


Our Application
Client
NodeJS Server
Oauth Dialog
Google Server
Google CA
1. User clicks sign-in.
Request sent to Google.
2. OAuth
Dialog
shown
to user
3. User logs in. Client is given
Access and ID tokens
4. Client
sends ID
token to
server
5. Server verifies
ID token legitimate
6. ID token confirmed

Understanding OpenID

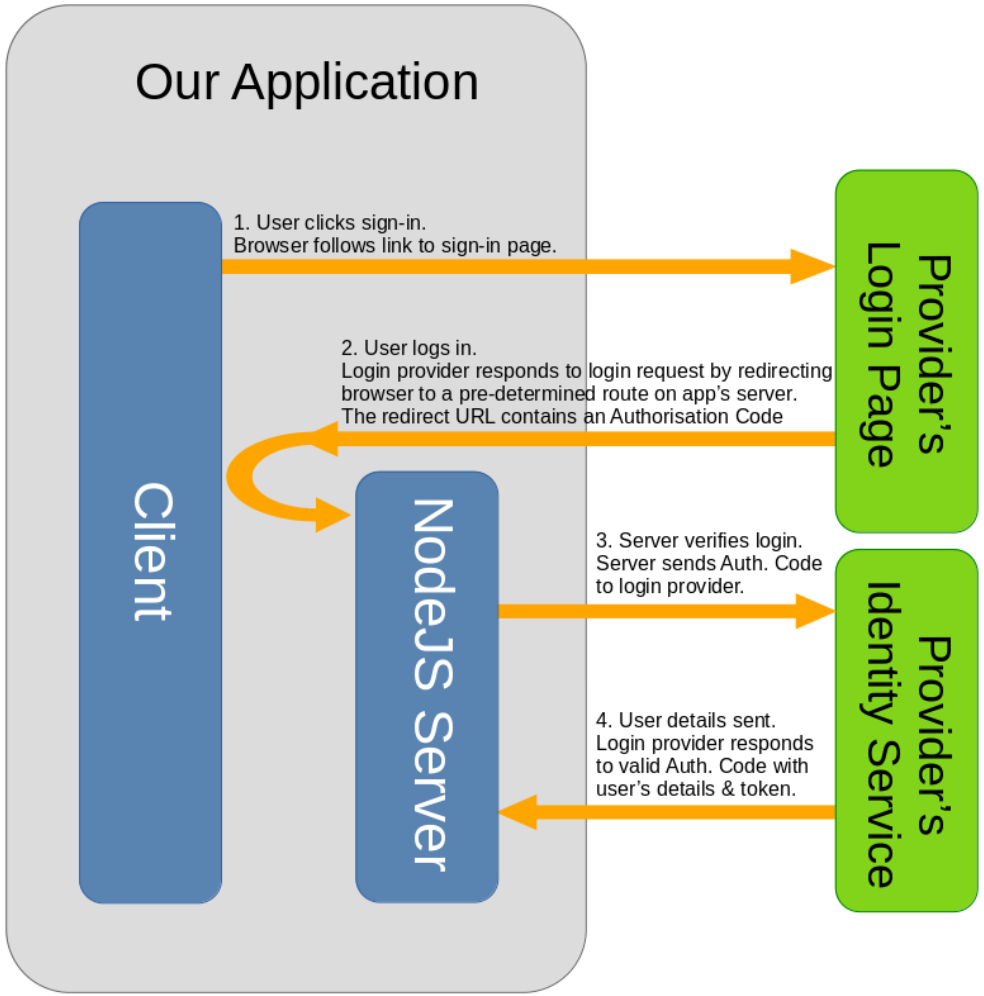
OpenID relies on the client sending our server a token from the Identity Provider.

How do we verify the legitimacy of the Token?

- The token is encrypted using Public Key Cryptography.
  - Encryption is done using the provider's private key.
  - We can decrypt the token with the provider's public key.
  - Decryption will only work if the provider performed the encryption
    i.e. token unmodified
- If the token is verified, then the user data inside is legitimate and we can authenticate the user.

An alternate approach: OAuth with Authorisation Code

An alternate approach used by some service providers that use OAuth is an Authorisation code:



Understanding Authorisation Codes

This alternate approach relies on the browser's ability to redirect to an alternate page on our site.

By including a code in the URL that the browser is directed to, our server consequently recieves the code.

- The code is unique, single-use, limited duration and restricted to only work if sent in a request from our server.
  - Our server can then make a separate request to the login provider,
    using the code to get the user's details.
  - Additional scopes and access to other services from that provider can also be done this way.

You can use any Identity Provider(s) or any Login Libraries for your project

But in our examples we'll use Google's Sign-In Button and/or Passport.js

The user is logged in with their provider; what next?

- Use the identity information provided to match user's info to their user account on the web app.
- If they don't have an account, generate one.
- Perform standard login actions such as grant session token access.

Summary

- We can use sessions to match requests with an authenticated user.
- Use middleware separate the secure/insecure parts of our website.
- OpenID and OAuth provide more secure ways of authenticating users by taking advantage of a trusted 3rd-party.

THE UNIVERSITY
of ADELAIDE