Code Fellows

Lab 12
Bearer Authorization

seattle-javascript-401n14

Lab 11 Review

Code Challenge 11 Review

Vocab Review!

What is encryption?

What is encryption?

Encryption is the process of securing data, by using an algorithm to transform it into something else. Encryption typically uses a key; you can convert data to and from a readable version using that key.

What is hashing?

What is hashing?

Hashing is the process of securing data, by using an algorithm to transform it into something else. The difference between hashing and encryption is that hashing is one-way; there is no key that can transform the hash back into the original data. New data can be hashed and checked for equivalence with other existing hashed data.

What is encoding?

What is encoding?

Encoding is a simplistic conversion of stream of data into another format. Note that this does NOT secure the data - this is similar to converting something from English to French - as long as someone knows French there is nothing secretive about the data.

What is a token?

What is a token?

A token is a securely encrypted “key” to gain access to a specific user’s data. Applications use tokens to keep track of authenticated users, and to quickly give clients user data without requiring another username and password login. Clients don’t care about how a token is interpreted or created; the server manages that and the client is only responsible for sending the token in each request.

What is a secret?

What is a secret?

A secret is any string that a server needs in order to decrypt something, verify a client, etc. It is basically a restricted string that either remains in the server, or which the server exposes to the client (like in OAuth) so that the client can verify itself (like a token!).

What is OAuth?

What is OAuth?

OAuth refers to a collection of standards for external applications to share user data. The OAuth standards require a web application to use an authorization code to get a token, and then use the token to get user data.

What we've learned

- Most web applications have a process for **authenticating** a new client, and **authorizing** that client to access certain data
- We can build our own authorization system with **tokens**, or we can use an external system using **OAuth**
- There are many ways to authenticate and authorize, though there are a few standards set in place
- When we store data in our database, we want to protect it using **encryption** or **hashing** (hashing is better!)

Basic Auth

- In our request header, we add the key value pair:

  `Authorization: “Basic ” + base64(username + “:” + password)`

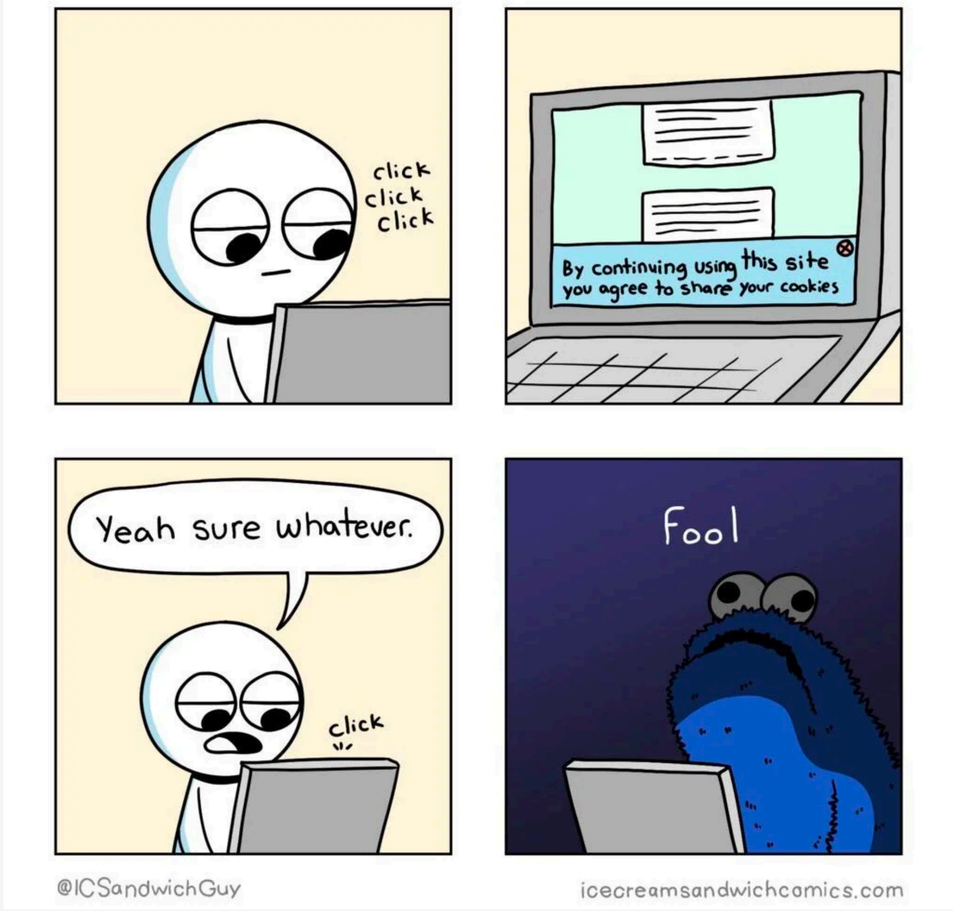
- Uses base 64 encoding, which is not secure (same as a plaintext password!)
- Should only be used once (on initial login)
- This is fine if your website is using HTTPS
    - HTTPS already encrypts client -> server requests, so that no one can see the request headers or request body

Cookies

- Cookies are the server instructing the browser to remember something
- The client makes requests as usual, and the browser loads the cookies on top of each request
- Server sends in the response:

  `Set-Cookie: <cookie data>`
- Browser loads the cookie in client request headers:

  `Cookie: <cookie data>`
- The client doesn’t have to do anything different!



Bearer Auth

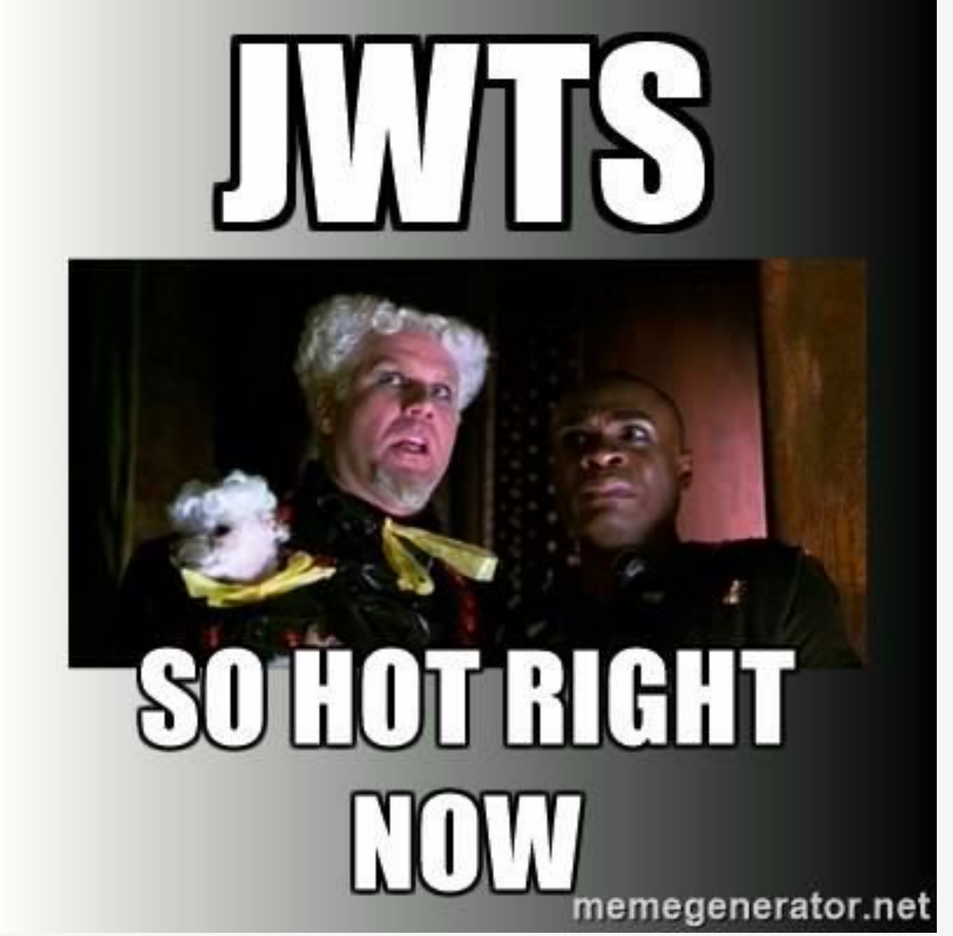
- One of the most common ways to authenticate / authorize
- We've been doing this already with JSON Web Tokens, just not in the most correct way
- In our request header, we add the following key-value pair:

  `Authorization: "Bearer " + token`

  - This can be a JSON Web Token!
  - OAuth has its own kind of tokens, but the logic is the same
- Client has to save the token and use it in future requests

More about Tokens

- Tokens are any string that does not store sensitive information in plaintext
- Anyone who has a token can request sensitive data from the server
- Tokens are usually meaningless to everyone except the server that created it
- We use JSON Web Token because it provides a simple way for us to generate and parse a secure string (just like we use bcrypt to hash for us - we don’t want to spend effort creating a hashing algorithm!)

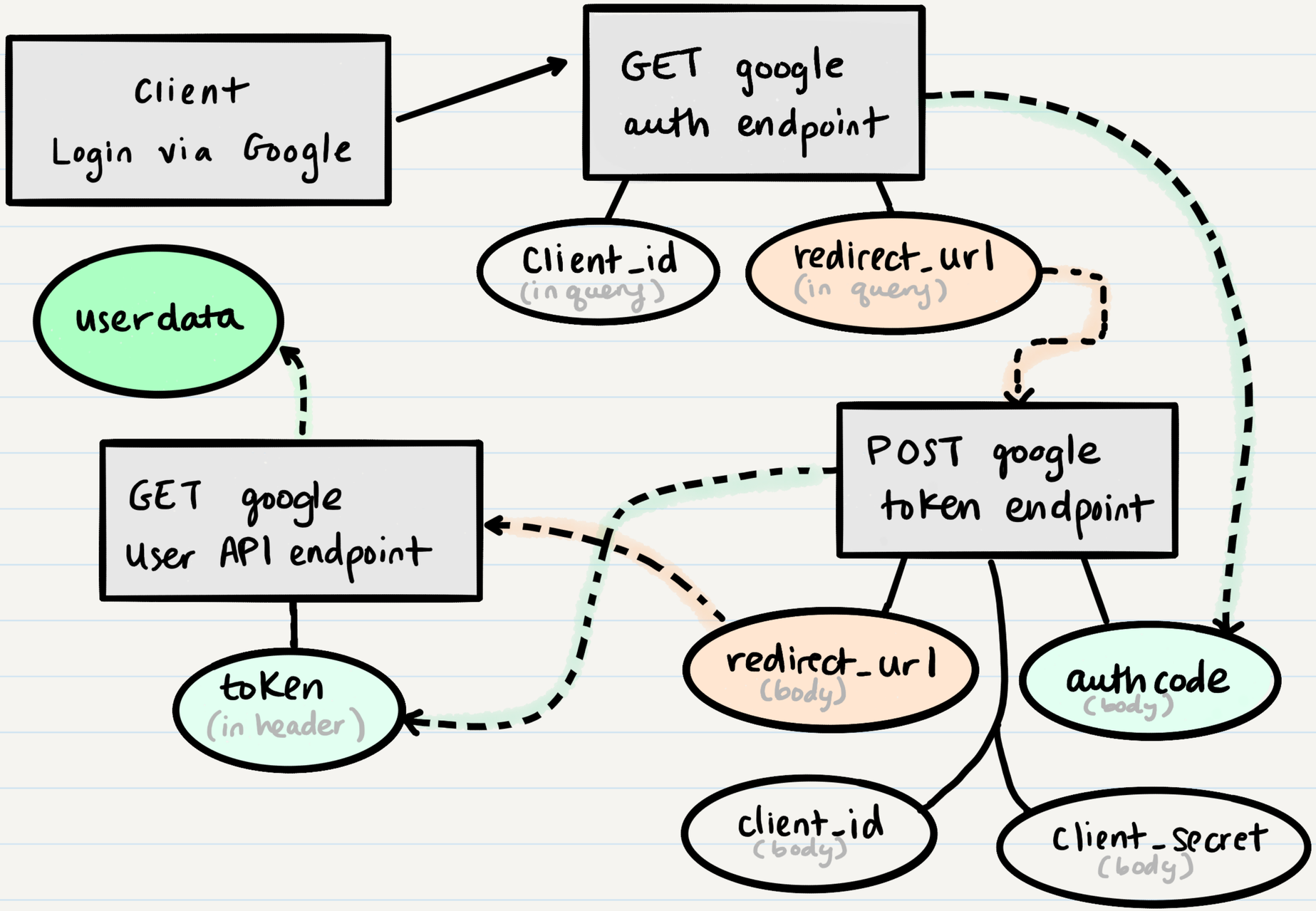


Where does OAuth Fit?

- OAuth uses Bearer Auth when it sends and receives tokens
- When we request something from Google, GitHub, etc, our tokens will be of the format:

  `Bearer <token>`

- Google interprets that in whichever way it needs


Client
Login via Google
GET google
auth endpoint
Client_id
(in query)
redirect_url
(in query)
userdata
GET google
User API endpoint
POST google
token endpoint
token
(in header)
redirect_url
(body)
auth code
(body)
client_id
(body)
Client_secret
(body)

JWT Timeout

- Once we give a token to a client, we probably want to limit how long the token lasts
- Have you ever been on a web app (think bank apps) and been “timed out” of your session?
- We can add a timeout on our JWT easily

```
// one hour timeout
jwt.sign({
    exp: Math.floor(Date.now() / 1000) + (60 * 60)
    data: {id: user._id}
);
```

What's Next:

- NO LAB 12 🎉
- Due by Midnight Wednesday:
  - **Learning Journal Class 12**
- Due by Midnight Thursday:
  - **Code Challenge 12**
- Due by 9am Saturday:
  - **Reading Class 13**
  - **DSA 03 Reading**
- Next Class: **DSA 03 Trees + Class 13 - Access Control (ACL)**

Questions?