Logging in to Websites

Core concepts

- **Authentication** (Auth) vs **Authorization** (Authz)
- **Bearer Tokens** vs **Stateless Web**
- **Sessions** and **Session ID**

Authentication / Authorization

- **Authentication** (Auth)
    - Who are you?
        - Think I.D. Card
        - **Identifies**
            - Doesn't grant permissions
- **Authorization** (Authz)
    - What are you allowed to do?
        - Think housekey
        - **Permission**
            - Doesn't identify

Many situations involve both

Example: Student ID Card

- Identifies (**auth**)
- Grants general student permissions (**authz**)

Still separate concepts

- Even if something involves both auth & authz

Designing an app/site

- Always consider which you need!
    - Auth vs Authz
- Always confirm user actually HAS the auth/authz
    - **Every request handling has to confirm!**
    - Never assume that "previous requests" did it
        - Stateless web = consider just request

Basic Username/Password Login

Authenticates

- Authorizes as a general user
- Specific user may have specific authorizations

What happens AFTER username/password confirmed?

- More details on that confirmation soon

Factors

A way of proving auth/authz

- Something you know
    - passwords, PIN
- Something you have
    - keycards, yubikey, RSA token, cellphone
- Something you are
    - fingerprints, iris, face

2FA is "two factor auth", MFA is "multi-factor (2+) auth"

How does server know login status of a request?

Stateless means only THIS request is considered

- If login was previous request, how do we know THIS request is from a "logged in" user?

There must be something IN the request!

- A "token"
    - A "**bearer token**"
    - Have token = authenticated or authorized

Option 1: Session ID

- Server creates unpredictable and unique "key"
  - Big random string
- Server saves this key
  - Associates with username/user id
- Server sends key in response
  - Later requests that include this **token** are considered authenticated as username

We call this token a **Session ID** (sid)

Option 2: Signed Auth Token

Token is a value that says user

- Is an identity (**auth**)
- And/or can do something (**authz**)

Token is "signed" by a trusted source

- Signed using Public Key Encryption
- Example: Login with Google/Facebook/Github

Any request with this Token

- Server verifies signature from trusted source
- Server considers user auth'ed

How do we send bearer token on every request?

Answer: **Cookies!**

- Server tells browser to set "cookie"
    - Giving a value to a named cookie
- Browser saves this info for this **origin**
- Browser automatically sends cookie name & value
    - For all later requests to that origin

Cookies Managed by Browser

- Server sends a `set-cookie` header on response
    - key=value pair
    - Along with some options
    - Including when it "expires"
- Browser saves this info
- On all later requests (automatically) to this **origin**
    - Browser sends a `cookie` header in request
        - With key=value pair
    - Server can read this cookie

Cookies are just a header

Notice how we didn't change HTTP for this

- Just set a header
- Server treats like a header
- Browser does the extra work

Cookie Security Management

- Browsers store cookie
    - Associate with "origin" and "path"
        - origin = protocol + domain + port
        - path - Don't use this, not worth it
    - Cookies only sent to origin server requests
- Cookies editable by user
    - Generally use for session id only
- Cookies end when browser closed
    - Unless they have an Expiration Date
        - "Remember this computer"

Cookie Best Practices (Server-side)

- Set `HttpOnly` flag if able
  - Prevents client-side JS access
- Set `Secure` flag in production
  - Requires HTTPS (encryption)
- Default to soon-expiring cookies
  - Shared computers are a thing
  - Session ID is EVERYTHING
- Set `SameSite` option value
  - Prevents use as a "3rd-party" cookie
  - Normally `Strict`

Removing a Cookie

- Cookie is stored on Browser
- Server has data using cookie value
- Should remove from both when you can
    - Server sends response header to remove
        - Browser will delete matching cookie
    - Server removes data from server storage

Session Id and Cookies

When user successfully auths, server will:

- Create a big random string (**session id** = sid)
- Connect any auth and authz info with sid
    - Often a DB entry
    - This course: just keep in memory
    - Send cookie with sid in response header

Later Request

- Browser automatically sends the sid cookie
    - Server can read sid from `req`
    - Server reads saved session data using sid
    - Server can read OTHER data w/session data
- Example
    - Session object holds username (by sid)
    - Full user data NOT in Session Data
    - User object holds full user data (by username)
- Session data only lasts between login/logout
    - User data outside of session

Validating Auth of a later request

Server gets a request

- Checks for sid cookie
- Checks the value of sid cookie
    - Is there a sid value?
    - Does sid value match saved data on server?
- Is this user permitted to do this request?

Logout

Two parts to logout

- Clean up sid cookie on browser
    - Server sends `set-cookie` to remove
- Remove session data
    - Example: deleting sid from sessions object

Remember: Most users don't logout

- Stale session data will collect
- Server frameworks may manage
    - But "session" is a general concept

Other tokens

Session Id is a "token"

- With random value

Other tokens may

- Contain usable info directly
- Are "signed" to prove who created them

Example: JWT (JSON Web Token) ("jot")

Still a "bearer token"

- Must keep secret

JSON Web Token - JWT

Signed bit of auth info + expire date

Advantages

- No DB check each time used
- Can be passed to others
    - How many 3rd party login systems work
    - Can pass to disconnected servers

Disadvantages

- Good for their lifetime, even if user "logs out"
- Don't want to store changing info in them

JWT Security

- Don't use if you need fast lockout
- Be sure to validate signatures!
    - Use tested libraries
- Generally use Secure and HTTPOnly cookies
- For server-to-server web calls
    - Expect JWT to be sent as `Auth` header

This course will use sid + cookies

- Most prevalent
- Still informs the server-client exchange

We will NOT use passwords!

- We will check for username "dog"
    - Shows when we check
    - Treat as permission denied (bad password)
    - NOT like "invalid username"
    - Doesn't create false impression of security

Express cookie example

```
// express "middleware", this time as an extra library
const cookieParser = require('cookie-parser');
app.use(cookieParser());

// (skipping over other express stuff)
app.get('/', (req, res) => {
  const store = req.query.store;
  if(store) {
    res.cookie('saved', store);
  }

  const saw = req.cookies.saved;
  res.send(`<p>Request had cookie "saved": ${saw}</p>`);
});
```

Steps

1. Inside new project directory:
    - `npm init -y`
    - `npm install express`
    - `npm install cookie-parser`
2. Create the `server.js` (or whatever you call it) file
3. run `node server.js`
4. go to `localhost:3000` in the browser
5. use `?store=SOMEVAL` at end of url to set the cookie
6. DevTools-Network-Headers to see the `Set-Cookie` in the response and the `Cookie` in the request
7. DevTools-Application-Cookies (left) to see cookies

Changing the cookie example

Do you know how to:

- Store the cookie under a different name
    - not `"saved"`?
- Change the expiration time of the cookie?
- Change the name of the query param you are sending to set the cookie value?
    - instead of `"store"`
- Redirect the user to `'/'` (no query param) after setting the cookie?

What is UUID?

- Universally
- Unique
- IDentifier

(Also known as GUID, for "Globally")

UUID variations

- Some have random-ish
    - Others NOT!
- Often factor in date/time
- Some pull in other info bits
- Generated by algorithm, not a central producer
- Attempt to make collision practical impossibility

session ids want unpredictable in addition to unique

- why?

UUID in node

Used to require a library, but we now have a built in option.

- Node > 14.17.0

```
// No npm install needed, crypto is part of Node
const uuidv4 = require('crypto').randomUUID;

const sid = uuidv4(); // sid common name for "session id"
```

UUID as session id in express

```
app.use(express.urlencoded({ extended: false }));

const sessions = {}; // Created outside a route handler

app.post('/session', (req,res) => {
  const username = req.body.username.trim();
  if (!username) { // Give better errors than this!
    res.status(400).send('username required');
    return; // don't allow redirect attempt
  } else if (username === 'dog') { // Simulates bad password
    res.status(403).send('user account not permitted');
    return;
  }
  const sid = uuidv4();  // from crypto module
  sessions[sid] = { username }; // Do you know why?
  res.cookie('sid', sid);
  res.redirect('/');
});
```

Session Storage

```
// example of sessions
sessions = {
    'asdf-asdf-asdf-asdf': {
        username: 'Jorts',
    },
    'zxcv-zxcv-zxcv-zxcv': {
        username: 'Jean',
    },
};
```

- Same user can have different sid
- But we want the same data for that user
- Store data by username/user id, NOT by sid
- Look up username by sid
    - look up data by username/user id

Checking the SID in express

```
app.get('/users', (req,res) => { // request requires authz
  const sid = req.cookies.sid;
  if(!sid || !isValid(sid)) {
    res.clearCookie('sid');
    res.send(401).sent('invalid login'); // POOR ERROR!
    return;
  }

  const { username } = sessions[sid];
  // Do whatever here
});
```

- `isValid()` is a function/check you have to write
  - Do we know this session?
  - `isValid()` is a concept
    - not a specific requirement

Session only for session-related values!

```
// Outside of route handlers
const sessions = {
    'asdf-asdf-asdf-asdf': { username: 'jorts' },
};
const profiles = {
    jorts: { name: 'Jorts', age: 3, color: 'orange' },
};

// Routes
app.get('/profile', (req,res) => { // request requires authz
  const sid = req.cookies.sid;
  // ... Skipping where it validates sid and username ...
  const username = sessions[sid].username;
  const profileData = profiles[username];
  // ... Do stuff
```

Profile tied to username NOT sid!

- Same username can have multiple sessions
- Profile data survives logout/login

Removing SID to end session

Imagine we have a `/logout` route

- Is this a GET or a POST?
    - When we get to REST, the question changes
- How do we clear the sid cookie?
    - `res.clearCookie('sid');`
    - OR, set cookie to blank value
    - OR, set cookie to immediately expire
- How do we clear the data from the server?
    - Delete this sid from `sessions`

Remember there is data in two places!

`sid` cookie on the browser-side

- `res.clearCookie('sid');` tells browser to delete

`sessions` has the `sid`

- `delete sessions[sid];` will remove that

Deleting in one place will not change the other!

Session Data isn't all your stored data

- Server session data can be deleted by logout
- Data NOT in session can survive logout