Authentication / Authorization

- Authentication (Auth)
    - Who are you?
        - Think I.D. Card
- Authorization (Authz)
    - What are you allowed to do?
        - Think housekey

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"

Login

Authenticates, possibly authorizes

- Username
- Password

Send both. Per security discussion, server will compare hashed password+salt to stored salt+hash for that username.

But then what?

Beyond Stateless

Web requests are stateless

How do you let the server know a later request is from someone that has already authenticated?

One Option: Passing Data

You could embed any necessary data in a form

- Each form submits info from previous forms

Pro:

- Works

Cons:

- User can change data
    - Security: NEVER TRUST DATA FROM USER
- Only w/forms or generated links

Option Two: Session Id

Store the data on server

- Associated it with unpredictable "key"
- Key secret from others
- Not secret from user

Stored Data = "session"

- Secret key = "session id"

Now sensitive data not changeable by user

Option 3: Signed Auth Token

A value that says user

- is an identity (auth)
- can do something (authz)

"Sign" the value using Public Key encryption

- User sends signed value (string)
    - Much like session id (bearer token)
    - Not secret from user
    - Is secret from public
- Server can validate using a public cert
- We trust the signer/system, not user

Passing the bearer token is annoying

Still sending via form/link

- More effort to generate dynamic HTML

Solution: Cookies!

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 later requests
    - Browser sends a `cookie` header
        - With key=value pair
        - Automatically
    - 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

- Set `HttpOnly' flag
    - Unless using with client JS
- Set `Secure` flag
    - In production
    - Dev might be done in http vs https
- Default to soon-expiring cookies
    - Shared computers are a thing
    - Session ID is EVERYTHING
- Set SameSite option value
    - Normally `Strict`

Removing a Cookie

- Cookie is stored on BROWSER
- Server might have associated data
    - But doesn't know what Browser has
- Server sends a response
    - Includes a `set-cookie` header
        - Removes value
        - Sets expires date to past
    - Server libraries have convenience methods

Session Id and Cookies

When user successfully auths, server will:

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

Later Request

- Browser automatically sends the sid cookie
    - Server can read sid from `req`
    - Server can read 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
- 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 cookie
- Checks the value of cookie to make sure it is valid
- Ensures that user is permitted to do 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
    - Doesn't create false security about 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?