d2.md

Authentication Learning Objectives

Define the term authentication

Authentication = Who

Authentication is verifying a user is who they say they are. This involves the user having a *secret* or *credential* of some kind and supplying that to the server and having the server verify that secret.

This differs from Authorization.

Authorization = What

Authorization is *what* Role or Permissions the user has.

Types of Secrets

- **password** - Usually entered directly by the user
- **token** - Like a password but generated by the server and stored on the client so that the client can continue to make requests on the user's behalf. Examples of this include OAUTH tokens and JWT Tokens.
- **session id** - Similar to a token, this is used by the browser to keep track of a user's session with the server, thus verifying that the user is the correct user. (This isn't enough on it's own to authenticate, you must use a token or password first). But it still should be protected just like a password or token.
- **two factor authentication code** - A code a user enters in addition to their password, can be sent via SMS, Email or generated by some software on the user's device.

Describe the difference between asymmetric and symmetric cryptographic algorithms

- **Symmetric Encryption** - A single *private* key is used to encrypt some data using a lossless algorithm. You can use the *private* key to decrypt and restore the original data

  Examples:

  - 1Password or LastPass encryption
  - Encrypting your hard drive with Filevault or Bitlocker.
  - Using GNUPG to encrypt a file.

- **Asymmetric Encryption** - A system with two keys, a *private* key and a *public* key. If Bob wants to send an encrypted message to Alice then Bob uses his *private* key and Alice's *public* key to encrypt the message. Now not even Bob can decrypt the message. The only way to decrypt the message is for Alice to use her *private* key anb Bob's *public* key.

  Examples:

  - Web Servers use SSL (Secure Socker Layer) to create an encrypted communication channel between the client's *private* key and the web server's *public* key.
  - GNUPG can be used to send encrypted Emails to other users as long as you have that user's public key.
  - SSH (Secure Shell) creates an encrypted connection between a client and server to provide remote access (github also uses this for SSH git access)

Identify "strong" vs. "broken" hash functions

Hash functions are algorithms that generate a lossy *hash* of a value such that the original value cannot be recovered if someone has the *hash*

A Salt is a random value added to a password *before* hashing and stored alongside the password in the database. This makes rainbow tables of pre-computed hashes less useful for figuring out the original password.

Broken hash functions

Currently these hash functions are "broken", meaning they should not be used to store user passwords in a database at rest. (Either because the algorithm has been solved, or the algorithm has been shown to have a flaw)

1. MD5
2. SHA-1

Strong hash functions

These are currently "strong" hash functions:

1. PBKDF2
2. bcrypt
3. Argon2

Neither of these are an exhaustive list, but these are common ones available for use with NodeJS libraries and covered in this course.

It's still important to protect the hashes in your database because if they are leaked, they can be attacked in several different ways.

- *Brute force* - This is just where you try to hash every possible combination of characters until you get a matching hash. This is slow.
- *Rainbow Attack* - You use a list of pre-computed hashes (A Rainbow Table) and compare the hash you want to crack against it. This is still slow but faster than the brute force attack

Implement session-based authentication in an Express application

Session based authentication is where you use a session-id, stored in a cookie to authenticate a user once they have first authenticated with a password.

This is the package we use for express session authentication:

```
npm install express-session
```

We configure it as express middleware like so.

```
app.use(session({
  secret: 'a5d63fc5-17a5-459c-b3ba-6d81792158fc', // Secret used to verify the session-id
  re`sa`ve: false, // Recommended by the express-session package
  saveUninitialized: false, // Recommended by the express-session package
}));
```

> The secret should probably be stored in an *environment variable* in your `.env` file instead of hard coded like this.

The middleware creates a `.session` property on the express `req` (Request) object. You can use this to store information you would like to persist for the user, such as the user's `id`, or other information that is relevant for your application.

By default `express-session` stores the session data in memory. Which means if you restart your express server you are going to lose your session data (all users will suddenly be logged out of the sytem).

To remedy this you can configure a different mechanism for the session store.

We can use the `connect-pg-simple` module to connect to postgreSQL and store the session in a table in our database.

```
const store = require('connect-pg-simple');
```

```
app.use(session({
  store: new (store(session))(),
  secret: 'a5d63fc5-17a5-459c-b3ba-6d81792158fc',
  resave: false,
  saveUninitialized: false,
}));
```

It will need an environment variable named `DATABASE_URL` to know which postgreSQL database to connect to.

The postgreSQL url should look something like this:

```
DATABASE_URL=postgresql://<username>:<password>@<host>:<port>/<database name>
```

Implement a strong hash function to securely store passwords

We can use `bcrypt` to store a password hash in the database instead of storing the plain text password, which is insecure.

Installing bcrypt

```
npm install bcryptjs
```

Generating a hashedPassword

Then wherever you register a new user you can add a line like this to generate a hash to store in the `user` table in the database

```
const hashedPassword = await bcrypt.hash(password, 10);
```

Verifying a password against a hashed password

Whenever we need to login a user in, we just check the password they give us against the hashed password from the database like so:

```
const passwordMatch = await bcrypt.compare(password, user.hashedPassword.toString());
```

passwordMatch will be a boolean.

Describe and use the different security options for cookies

- **httpOnly** - Makes the cookie only be able to be read and written by the browser and *not* by JavaScript.
- **secure** - The browser will only send this cookie if the connection is HTTPS.
- **domain** - If this isn't set it will assume the cookie should be the same as the site's domain.
- **expires** - The date and time when the cookie expires
- **maxAge** - The number of milliseconds before the cookie expires (usually easier to use than expires)
- **path** - Defaults to '/' but it's a prefix for the path in the URL that the cookie is valid for (seldom used)