KINGSLAND UNIVERSITY

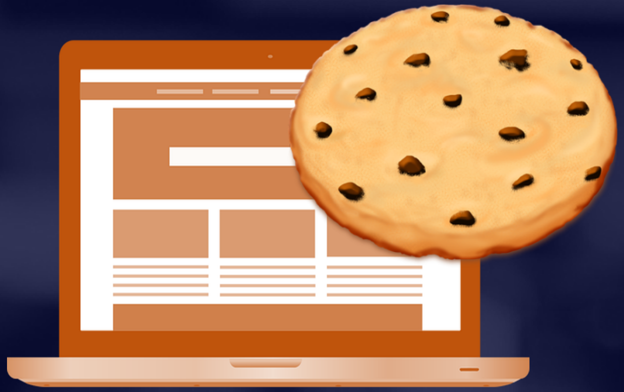
Sessions and Authentication

State Persistence and Application Security

Table of Contents

Have a Question?

#js-web



Cookies and Sessions

Persisting Client State

HTTP Communication

- HTTP is **stateless**
  - The Server and Client **don't remember** each other across requests
- To preserve state, **cookies** are stored on the **Client**
- A **session** exists on the **Server**
  - It can **store information** about a Client
  - Used to **persist state** across requests
  - Matched to a Client by their **cookie**

Session vs Cookie

- **Session** is preferred when you need to store **short-term** information/values
- **Cookies** is preferred when you need to store **long-term** information/values
- Session is **safer** because is stored on the server. Expiration **can not** be set, they will be expired when user close the browser
- Cookies is not very safe. Expiration **can be set**, and they can last for years

Using Cookies

- You can use **cookie-parser** middleware for Express

```
npm install cookie-parser --save --save-exact
```

```
// use in an express app
const cookieParser = require('cookie-parser')
app.use(cookieParser())

app.get('/setCookie', (req, res) => {
  res.cookie("message", "hello")
  res.end('Cookie set')
})

app.get('/readCookie', (req, res) => {
  res.json(req.cookies)
})
```

Using Sessions

- You can use **express-session** middleware for Express

```
npm install express-session --save --save-exact
```

```
// use in an express app
const session = require('express-session')
app.use(session({ secret: 'my secret'},
                { httpOnly: true },
                { secure: true}))

app.get('/setSession', (req, res) => {
  req.session.message = "hello"
  res.end('Session set')
})

app.get('/readSession', (req, res) => {
  res.json(req.session)
})
```

Live Demo

Authentication Concepts

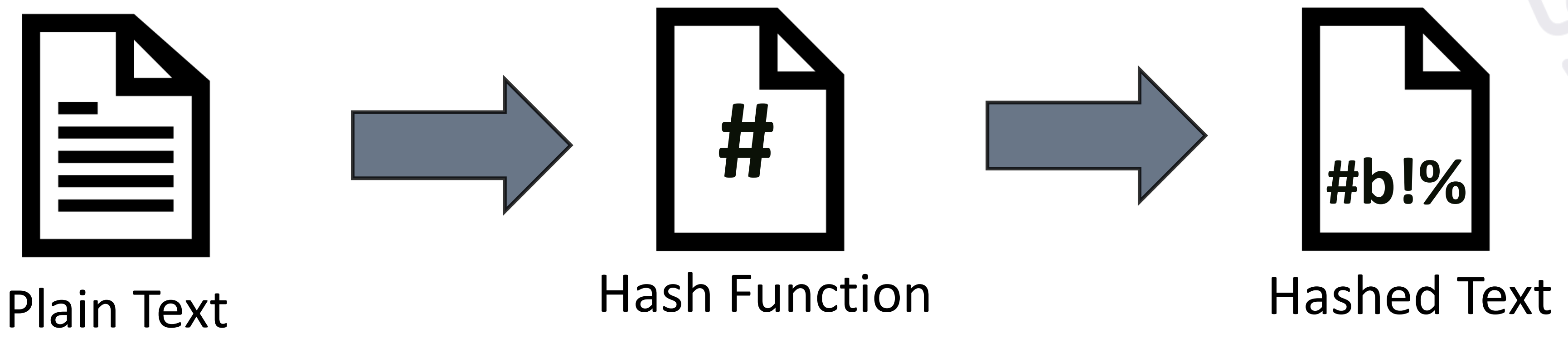
Application Security and User Roles

Application Security

- **Authentication** is an important part of **application security**
- It serves to verify that **clients** can **access** certain resources, depending on their
- It's built on several **layers of abstraction**
  - Cookies → Sessions → Security
- Authentication is **a cross-cutting concern**, best handled away from business logic
  - Request → Authentication → Business Logic → Response

Bcrypt

- **Bcrypt** is a password hashing function
  - Besides incorporating a **salt** to protect against **rainbow table** attacks, **bcrypt** is an adaptive function
    - Over time, the iteration count can be increased to make it slower, so it remains resistant to **brute-force search attacks** even with increasing computation power



Bcrypt

- Installation

```
npm install bcrypt --save
```

- Hash password

```
const bcrypt = require('bcrypt');
const saltRounds = 9;
const myPlainTextPassword = "password123";

bcrypt.genSalt(saltRounds, (err, salt) => {
    bcrypt.hash(myPlainTextPassword, salt, (err, hash) => {
        console.log(hash);
        // $2b$09$pdhUAoT4qE0tmku.ZkXWROeLcJCy.LDRq.1I4IVImjrUTGuUbYQMi
    })});
```

Bcrypt

- Check password

```
const myPlainTextPassword = "password123";
const hash = "$2b$09$pdhUAoT4qE0tmku.ZkXWROeLcJCy.LDRq.1I4IVImjrUTGuUbYQMi";

bcrypt.compare(myPlainTextPassword, hash, (err, res) => {
    console.log(res); // true
});
```

- **Async** way is **recommended** to **hash** and **check** password

Authentication vs. Authorization

- **Authentication**
  - The process of verifying the identity of a user or computer
  - Questions: "**Who are you?**", "**How you prove it?** "
  - Credentials can be password, smart card, external token, etc...
- **Authorization**
  - The process of determining what a user is permitted to do on a computer or network
  - Questions: " **What are you allowed to do?**", "**Can you see this page?**"

Live Demo

JSON Web Token

Authentication for REST APIs

What is JWT?

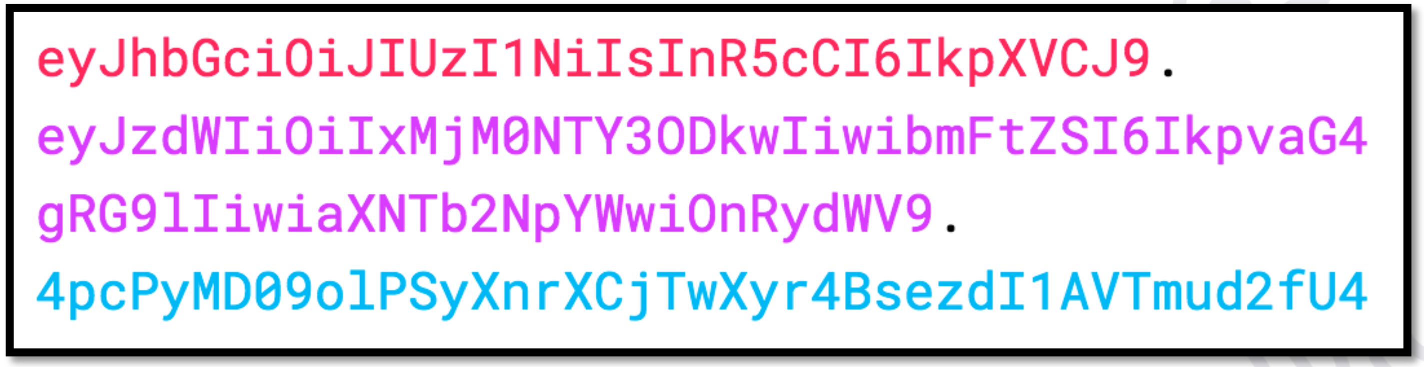
- **JSON Web Token** (**JWT**) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object
- This information can be verified and trusted because it is digitally signed
- JWTs can be signed using a secret or a public/private key pair using **RSA** or **ECDSA**

When Should You Use JWT?

- JSON Web Tokens are useful for
  - **Authorization** (most common scenario) - Once the user is logged in, each subsequent request will include JWT, allowing the user to access routes, services and resources that are permitted with that token
  - **Information Exchange** - JSON Web Tokens are good way of securely transmitting information between parties. Because they are signed digitally

JWT Structure

- In its compact form, JSON Web Tokens consist of three parts separated by dots (.)
  - **Header**
  - **Payload**
  - **Signature**



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

JWT Usage

- Installation

```
npm install jsonwebtoken --save
```

- Encode token

```
const jwt = require(jsonwebtoken');

const payloads = { _id, username };
const options = { expiresIn: '2d'};
const secret = 'MySuperPrivateSecret';
const token = jwt.sign(payload, secret, options);

console.log(token);
//eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXkiOiIxMjM0NTY3ODkwIiwibmFtZSI6
IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.xzK8LJQz0lDkJqsng04BYxcUQzxWngyEBP
```

JWT Usage

- Decode token

```
const token = req.cookies['token'] || sessionStorage.getItem('token');
// Depends where you store the token..

const decodedToken = jwt.verify(token, secretKey);

console.log(decodedToken); // { _id: ..., username: ... }
```

- More about JWT, you can find
  - https://jwt.io/
  - https://www.npmjs.com/package/jsonwebtoken

Live Demo

Summary

- Cookies and Sessions
  - Definitions and Usage
  - Cookies vs Sessions
- Authentication Concepts
  - Application Security with bcrypt
- JSON Web Token
  - What is JWT?
  - Structure and Usage

Questions?

License

KINGSLAND UNIVERSITY

THANK YOU