Securing an Angular Application

Security is constantly evolving. You'll need to do your own research on the specific vulnerabilities of your application. However, the following explains fundamental security services provided by Angular.

Before we move on understand the difference between the folowing two terms:

Authentication: Validiating a users credentials to access a system or part of a system

Authorization: Checking if a user has access to a restricted system.

Angular Application Security Checklist

- [ ] Use **HttpOnly** and **Secure** cookies
- [ ] **Sign** the cookies and JWT with a strong secret
- [ ] **Do not** store sensitive data in JWT Payload
- [ ] Ensure JWT lib does not accept **alg: *none***
- [ ] Transport all data over **HTTPS**
- [ ] Use **Content Security Policy** ver. 2
- [ ] Do not allow inline scripts (no **unsafe-inline**)
- [ ] User **integrity** property of external scripts
- [ ] Avoid Angular's ***bypassSecurityTrust*()***
- [ ] Use CSRF protection with **CSRF-Token**
- [ ] Avoid custom auth library implementation
- [ ] Check **all API endpoints** for role-based access
- [ ] Use **AoT compilation** for templates check

Learn more at:
angular-academy.com/security

Json Web Tokens

JWT Tokens are used to store an active session after a user has been authenticated.

Using JWTs there is no need to store the username and password directoly in memory. Everything needed for the server to authenticate and authorize exists in memory, and every token can expire after a set amount of time.

A typical JWT is a long string seperated into 3 parts seperated by the '.' symbol.

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M`
`DIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`

The **Header** contains 2 parameters: The algorithm used to decode the token and the token type, which in our case should always be "JWT"

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Next is the **Payload** of the token, it is encoded in Base64. We can put anything we'd like into here. Never put sensitive information like passwords in here, as it can easily be decoded.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "role": admin
}
```

Finally the **Signature**, which is generated from the payload and header. Note it is encoded first using the header-defined algorithm, then Base64, so decoding it must be done in reverse order.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  { my-256-bit-secret }
)
```

A private key stored on the server side is used to sign the token so we can be sure only our server issued the token.

Server Side JWT Usage

The first step is choosing your method of authentication. Whatever method of authentication you use on the backend is up to you. A few different types of authentication include:

Basic Auth: Comparing plaintext username/passwords directly on the server. Straightforward but remember it is bad practice to hard code any passwords into the code. If using this method I would recommend storing the hashed password in a relational database.

LDAP: Lightwieght Directory Acess Protocol is used for directory services authentication. Connect to an external server which holds credentials. Pretty complicated to set up but very secure when done right.

OAuth : OAuth/OAuth2 is an open standard for securing access to APIs. Connect to a existing third party login provider such as OKTA, Google, Facebook, etc. Some configuration is required and there is a bit of a learning curve, some api documentations are better than others.

On the backend implement the dependecies for whichever auth service you choose, and also any JWT dependencies that are needed. The next steps are as followed:

1. Write a method to generate a Token, choose what fields you want in the payload.
2. Choose method of authentication. RSA encrytption requires a public and private key. It is ok to have a new Key generated every startup

The below is written in Java and I'm not going into specifics on the classes used here because there are already a lot of good guides out there. I into more detail on RSA encryption and signing in my Ktor tutorial

```
private RSAPrivateKey privateKey;
private RSAPublicKey publicKey;

@PostConstruct initKeys() throws NoSuchAlgorithmException {
    KeyPairGenerator generator = KeyParGenerator.getInstance("RSA");
    generator.initialize(2048);
    KeyPair keypair = generator.generateKeyPair();
    privateKey = (RSAPRivateKey) keypair.getPrivate();
    publicKey = (RSAPublicKey) keypair.getPublic();
}

```

```
public String generateToken(String name, String role) {
...
```

Securely Storing the JWT Token

Returning the JWT Token as a HTTP respsonse puts the token in local javascript memory. This puts the application at risk of Cross-Site-Scripting. The production standard way of storing tokens are SSL-encrypted HTTP-only cookies.

Revision #3
Created 16 April 2022 22:53:46 by Elkip
Updated 17 April 2022 01:35:51 by Elkip