Learning Objectives: Basic Authentication

Learners will be able to...

- **verify the user and password on the server side**
- **dispatch the basic credentials from the client**

info

Make Sure You Know

This assignment assumes a basic knowledge of JavaScript, HTML and CSS.

Limitations

We will not explain simple programming concepts, it is assumed that the user is familiar with these concepts.

What is Authentication?

Authentication is the process of verifying the identity of a user or an entity that is accessing a system, application or service.

What are some of the uses of authentication?

1. Restrict access to sensitive data
2. Allow only authorized access to proprietary systems
3. Filter out people who haven't paid for access
4. Monitor usage patterns

The examples we tried out in the previous assignments in this course did not require any authentication, they were open to everyone. Access to government information about the weather doesn't need to be protected but the IRS API would obviously be very protected.

Authentication and Authorization

Authorization occurs after authentication, once the credentials of the user have been established, the system determines what their access rights are.

In the example of an online course such as this one, there are multiple tiers of access
- Instructor access means that the instructor can edit the material, view all student work and modify student grades.
- Teaching assistants have the same access as instructors except that they cannot edit the learning materials.
- Students have access only to the materials and their work and they cannot view any other student work or grades.

According to the Open Web Application Security Project (OWASP), **broken access control** was the top form of security breach in 2021. Access control failures enable users to view information in other people's account that they should not have access to. This document lists the things to look out for to prevent these types of breaches and should be followed when you design a Web API.

Basic Authentication

Basic authentication involves sending user credentials (i.e. username and password) in an API request header. This is not considered a secure approach because credentials are encoded, not encrypted. If the exchange is not conducted over a secure connection, the credentials could be easily compromised. The standards document for this authentication scheme may be found here.

important

Installing the node modules

The first time you run something in each of the assignments in this course you need to make sure that you have all the proper node modules installed. To do that click the **Terminal** tab on the left and run the commands below.

```
cd examples
npm i
```

HTTP Authentication Framework

HTTP provides a basic approach for authentication and access control.

The workflow for this process is as follows:
1. The client makes a request to the server and when authentication is required the server will respond with a `401 Unauthorized` and provide a WWW-Authenticate response header with the authentication challenge.

1. Upon receiving the error, the client will prompt the user for the required credentials (usually user name and password).
2. Once the credentials are supplied, the client can request the resource again and include the authorization header with the required information.

In order for this procedure to be secure, it must occur over a secure connection. Hypertext Transfer Protocol Secure is the communication protocol and Transport Layer Security is the encryption method. It's

referred to as HTTPS/TLS. When using HTTPS the entirety of a request is encrypted before being sent over the network.

Basic Authentication examples

- The example in the file `01-basic-auth.js` requires authentication for any resource request. The method `app.use` adds Express middleware which pre-processes all requests for authentication.

- The example in the file `02-basic-auth-selected.js` requires authentication just for delete requests.

important

Try this out - authentication required for everything

We'll try out the app that requires authentication for everything first: 1. Run the app - type `nodejs 01-basic-auth.js` (you need to be in the examples directory).

2. First try to list out all the pets without the authentication information `curl -i http://localhost:3000/pets`
3. Try it with the authentication information: `curl -s -uyourlogin:yourpassword http://localhost:3000/pets`
4. Try deleting a resource without authentication: `curl -X DELETE http://localhost:3000/pets/5`:
5. Try deleting a resource with authentication: `curl -uyourlogin:yourpassword -X DELETE http://localhost:3000/pets/5`:
6. Check to see that it actually deleted by using the button in step 3 again.

important

Try this out - authentication only needed for `DELETE`

We'll try out the app that requires authentication for everything first:
1. Run the app - type `ctrl-c` to quit the running app, then type `nodejs 02-basic-auth-selected.js` to run the other app.

2. List out all the pets without authentication `curl -s http://localhost:3000/pets`
3. Try deleting a resource without authentication: `curl -X DELETE http://localhost:3000/pets/1`:
4. Try deleting a resource with authentication: `curl -uyourlogin:yourpassword -X DELETE http://localhost:3000/pets/1`:
5. Check to see that it actually deleted by using the button in step 2 again.

Authentication Headers and Authorization

Authentication Headers

- **WWW-Authenticate** - The `WWW-Authenticate` header must contain at least one challenge and can contain multiple challenges. A response can also contain multiple `WWW-Authenticate` headers. The challenge specifies the authentication scheme the server requires.

Here is an example of how you might parse the response header:

```
  if (response.status === 401) {
    const wwwAuthHeader = response.headers.get('WWW-
Authenticate');
    if (wwwAuthHeader) {
      // Extract the authentication scheme and realm from the
header
      const [scheme, realm] = wwwAuthHeader.split(' ');
      console.log(`Authentication required: scheme=${scheme},
realm=${realm}`);
    } else {
      console.error("No WWW-Authenticate header found in the
response.");
    }
  }
```

Authorization

The `Authorization` request header is used to send credentials to the server with a request. For Basic Authentication, the user name and password are combined and separated by a colon as in `username:password` and then encoded using base64, a binary to text encoding scheme.

Example of how to create a Basic Authentication string:

‼ You would not hard code the user name and password in a real application, this is just an example.

important

Try it out

- Click the **Terminal** tab on the left and run the commands below:

```
cd examples
nodejs 02-basic-auth-selected.js
```

- Open a new terminal window (**Tools**>**Terminal**) and paste the following in:

```
cd examples
nodejs 03-client-auth.js
```

`03-client-auth.js` deletes the 3rd pet.

- List out all the pets to see it was deleted `curl -s http://localhost:3000/pets`
- Try commenting out the authentication (line 22, add `//` to beginning of line):
- Run it again and see what happens (you must be in the examples directory):

```
nodejs 03-client-auth.js
```

Authentication types and tokens

Web APIs use a variety of authentication mechanisms, the most commonly used is OAuth 2.0 .

Bearer authentication

Bearer authentication is an HTTP verification scheme that uses tokens to authorize requests. The name “Bearer” refers to the fact that the token gives “access to the bearer” of the key. The cryptographic **bearer token** is generated by the server in response to a login request. The client then uses the token in the Authorization header `Authorization: Bearer keygoeshere` for any requests that require authorization. The token can be valid for a specified duration of time `expires_in`. It is recommended that **bearer authentication** requests be conducted over a secure network (HTTPS).

API Key Authentication

A unique generated value is assigned to a user the first time they use a system. The user sends their API key in the request header in subsequent uses and that is how the server recognizes them. This is not considered a very secure system because the key could be intercepted and used by someone else. This simple system is mostly used for read only APIs.

OAuth

OAuth 2.0 is a popular authentication protocol that allows third-party applications to access resources on behalf of a user. It requires the user to grant permission to the third-party application to access their resources, and it uses access tokens to authenticate requests. If you logged into Codio using Single Sign On (SSO) from another system, that is via the OAuth 2.0 protocol. The standards document for this authentication scheme may be found here.

JSON Web Tokens

JWT tokens are often used with OAuth. The server issues the signed token to the user upon authentication, which contains information about the user and their permissions. The token is then included in each subsequent request, allowing the API to authenticate the user without the need for a server-side state. The contents of the token can also be encrypted. More information about JWT tokens can be found here.

Web Security

Best practices to ensure data security

- Planning for data security should be an integral part of the design process of your Web service.
- Use libraries that are up to date and are created and maintained by a reliable source.
- Maintaining a log of API access events and access errors is helpful in assessing and analyzing security risks. This should include all login events, input validation failures and access control issues to help identify suspicious accounts. Repeated failures should generate an alert for admins.
- Protect the integrity of log data by encoding to prevent attacks on the logging system.

Access control, identification and authentication:

- The default should be to deny access for everything except public resources.
- Create access controls that require record ownership rather for any CRUD activity and use them throughout your application.
- Prevent automated attacks by rate limiting access requests.
- Stateless session identifiers should be short lived and stateful session identifiers should be invalidated on the server after logout.
- Implement multi-factor authentication wherever possible and in particular for users with administration or other high level privileges.
- Do not distribute versions of your API with default credentials especially for users with admin privileges.
- Require secure passwords aligned with National Institute of Standards and Technology guidelines.
- Ensure that your failure error messages do not give away crucial information by keeping them uniform.

Cryptographic storage:

- Identify the sensitivity of the data you are storing with respect to proprietary rights, regulatory requirements and privacy laws.
- Discard sensitive data as soon as it is no longer needed.
- All sensitive data should be stored encrypted.
- Do not allow caching of responses that contain sensitive data.
- Ensure that encryption algorithms are strong and current and that encryption keys are properly managed.
- All data transfers should use secure protocols such as TLS (Transport

Layer Security) and forward security which protects data from being decrypted if the key is compromised at a later time.
- Use secure hashing functions, such as Argon2, to store passwords.
- Randomness is key to cryptographic security.
- Use up to date cryptographic functions and padding schemes.
- More information on cryptographic storage.

▼ What is password hashing?
A hash function turns a password into an indecipherable string. There is no decode key for this string, it is a one way translation. This makes it impossible to “guess” a person’s password.

Injection prevention

Injection attacks are performed by inserting malicious code into an input field. The following practices can help prevent them:

- Validate input code that comes from an external source such as a client. Input should match the expected parameters.
- Limit the number of records that can be requested from storage to reduce the risk of large disclosures as a result of an injection attack.
- More information on injection protection.