Samiur Rahman (20660512) & Shashank Kotturi (20651362)

31 July 2020

# Design Workflow

## Preflight

Upon landing on the website, the client performs a preflight request to the server. The server consequently evaluates the following:

* Did the client provide a web session ID (cookie = webSessionId)?
  + Yes: perform a query on the web\_session table to determine whether an existing session using the provided ID exists with a future-dated expiry. Did the query return results?
    - Yes: is the client from both a trusted origin and trusted IP address (are the request origin and request IP address equal to those stored in the metadata of the web session stored in the database)?
      * Yes: return success (HTTP code 200 OK) to the client.
      * No: return failure (HTTP code 401 unauthorized) to the client.
    - No: return failure (HTTP code 401 unauthorized) to the client – session expired.
  + No: is the client from both a trusted origin and trusted IP address (are the request origin <http://localhost:8000> and IP address ::1)?
    - Yes: create both a cookie (webSessionId) and a record in the web\_session table using a 32-byte hexadecimal-encoded string unique identifier (sessionId) generated using a PSRNG, with a lifetime of 12 hours. In the metadata field of the database record, store the client origin and client IP address as a JSON string encoded object. Return success (HTTP code 200 OK) to the client.
    - No: return failure (HTTP code 403 forbidden) to the client – client does not have permission.

## Authentication

The login workflow is broken down into a two-stage process: Identify and Login. This procedure implements the challenge-response authentication protocol.

### Identify

In the identification stage, the client disregards the plaintext password and sends only the username to the server. The server consequently evaluates the following:

* Query the user\_login table with the username. Is a salt returned?
  + Yes: generate a 64-byte hexadecimal-encoded challenge using a PSRNG. Store the challenge in the corresponding record with a lifetime of 10 seconds. Return the salt and the challenge to the client. Return success (HTTP code 201 created) to the client.
  + No: an account with that username does not exist. Return failure (HTTP code 404 not found) to the client.

### Login

In the login stage, the client will perform the following:

* Generate a 16-byte hexadecimal-encoded initialization vector using a PSRNG.
* Concatenate the password with the salt and hash the result.
* Encrypt (AES-256-CBC) the challenge using the hashed and salted password, as well as the initialization vector to produce a ciphertext.
* The payload of the request to the server will consist of the username, ciphertext, and initialization vector.

The server consequently performs the following:

* Retrieve the hashed and salted password as well as the challenge from the joined user x user\_login table using the username.
* Encrypt (AES-256-CBC) the challenge using the hashed and salted password, as well as the initialization vector supplied by the client to produce a ciphertext.
* Compare the client-generated ciphertext with the server-generated ciphertext. Are they equivalent? And was the comparison performed before the challenge expired?
  + Yes: is there an existing record in the user\_session table with the corresponding username?
    - Yes: update the existing record’s by extending the expiry by 15 minutes from the current datetime. Create a cookie using the existing user session ID. Return success (HTTP code 200 OK) to the client.
    - No: create both a cookie (userSessionId) and a record in the user\_session table using a 32-byte hexadecimal-encoded string unique identifier generated using a PSRNG, with a lifetime of 15 minutes. Return success (HTTP code 200 OK) to the client.
  + No: return failure (HTTP code 401 unauthorized) to the client denoting that the username or password was incorrect.

Once authenticated, the client will store the plaintext password (called master key) in the browser’s session storage.

# Security Vulnerabilities

## SQL Injection

Using PHP prepared statements protect against SQL injection by providing a parameterized and reusable SQL querying mechanism that forces developers to write the desired SQL command and the user-provided data separately. Using bind variables, the database engine compiles the query using placeholders, and the user-supplied data is added later.

## Cross-Site Request Forgery (CSRF)

Identifying the request source origin from the HTTP request header is a common defense mechanism used to mitigate the risks associated with CSRF. If the request origin is not trusted by the server, the server can block the request regardless of the attacker having possession of data that would otherwise facilitate their access to the server.

## Cross-Site Scripting (XSS)

By encoding the data entered by the client we can prevent the injection of malicious JavaScript code or redirection to third-party sites that consequently gain access to all the data that the webpage’s JavaScript code has access to, such as cookies. Characters such as &, <, >, ", ' are encoded such that this vulnerability is made difficult to exploit.