# APDS7311 POE PART 1

ST10085360

Mikail VH
VCNMB

# TABLE OF CONTENTS

# Security Measures for the Login and Registration Process

## 1. Https Requests And Traffic Security

HTTP is a generic protocol used for communication between user agents and proxies/gateways to other protocols such as SMTP, FTP, etc. (Acunetix, 2019). It works over TCP(Transmission Control Protocol), which provides the delivery of streams of data. With it being message-based(request, response), a request would start with a request line that contains a *GET* method. This essentially *gets* the contents of a URL. Through all this communication, lies the risk of information being intercepted by anyone with access to the network traffic, therefore, the I will be implementing HTTPS(HTTP Secure) Encryption to ensure that all communication is encrypted. This will keep the data confidential while in transit (Acunetix, 2019).

## 2. Input Validation

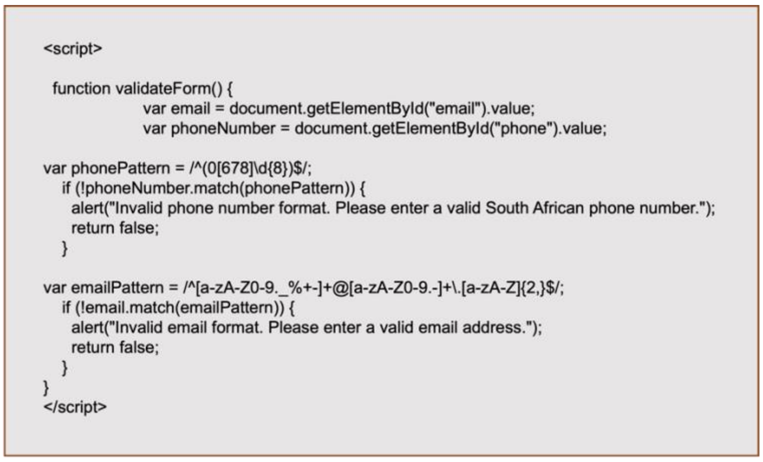
Input validation is an extremely important part of ensuring that user credentials are secure and robust enough not to be guessed. This is the process of disallowing unsuitable inputs. This also prevents attackers from inputting data that could cause harm to the system. The inputted data must meet certain criteria for it to be read. Two main forms of attacks that are associated with an absence of input validation are:

1. Buffer Overflow
    - This occurs when an attacker floods the system with information. Without input validation, the attacker can add as much information as needed to cause a system failure and in turn – loss of data (Nesbo, 2023).
2. SQL Injection
    - This form of attack occurs when SQL queries are added to input fields with the hopes of tricking the system to execute the queries and then gain access to sensitive data should there be no input validation  (Nesbo, 2023).

To prevent these attacks, the following rules will be implemented:

- The field cannot be null.
- Whitelisted characters will be those that aren't associated with special characters used in SQL queries.
- I will be blacklisting characters that aren't allowed such as: "*", "- ", "\", "/" (Used in SQL queries)
- Email fields have to be in a certain format.
- There will also be a rule in place that limits the number of characters that can be added.

Here is an example of how the JavaScript code would look like when validating cellphone number and password fields:



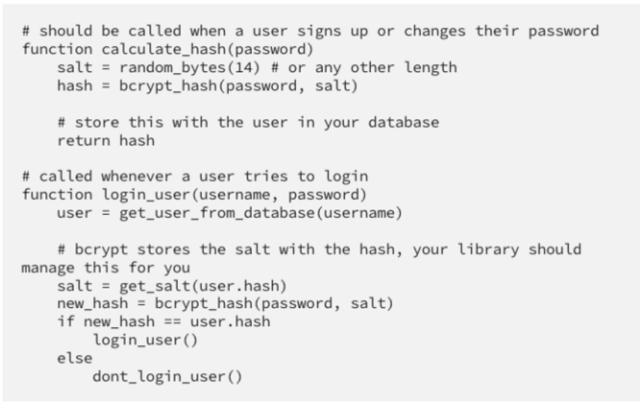
```
<script>

  function validateForm() {
                    var email = document.getElementById("email").value;
                    var phoneNumber = document.getElementById("phone").value;

var phonePattern = /^(0[678]\d{8})$/;
    if (!phoneNumber.match(phonePattern)) {
      alert("Invalid phone number format. Please enter a valid South African phone number.");
      return false;
    }

var emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
    if (!email.match(emailPattern)) {
      alert("Invalid email format. Please enter a valid email address.");
      return false;
    }
}
</script>
```

When the system puts these rules in place, threats such as SQL injection won't be able to affect the system.

## 3. Storing And Hashing Of Passwords

Hashing a password is a vital part of implementing security in a log in system. This involves using a strong algorithm that encrypts a password. This means that these password hashes are difficult to be reverse-engineered (Mistele, 2020). I will hash all of the passwords with a unique salt which will add additional security in the case that an attacker use a rainbow table. A random salt will be generated and stored with the password hash and then combined when the user tries to log in. Due to this extra layer of security, hackers wont be able to use rainbow tables to crack passwords. The attacker word need to crack not only the hash, but the salt as well.

The below image demonstrates how salt will be used at add another layer of security to a hashed password:



```
# should be called when a user signs up or changes their password
function calculate_hash(password)
    salt = random_bytes(14) # or any other length
    hash = bcrypt_hash(password, salt)

    # store this with the user in your database
    return hash

# called whenever a user tries to login
function login_user(username, password)
    user = get_user_from_database(username)

    # bcrypt stores the salt with the hash, your library should
manage this for you
    salt = get_salt(user.hash)
    new_hash = bcrypt_hash(password, salt)
    if new_hash == user.hash
        login_user()
    else
        dont_login_user()
```

Image taken from (Mistele, 2020)

## 4. Maintaining Authentication State

There are 2 main ways that I can ensure that users are authenticated and that their sessions aren't compromised. The first is by implementing session timeouts. This will automatically log users out after a certain amount of time of inactivity. This will reduce the risk of unauthorized access. The second way is through using random session tokens that will associate a user session with their login credentials (Gorgam, 2023).

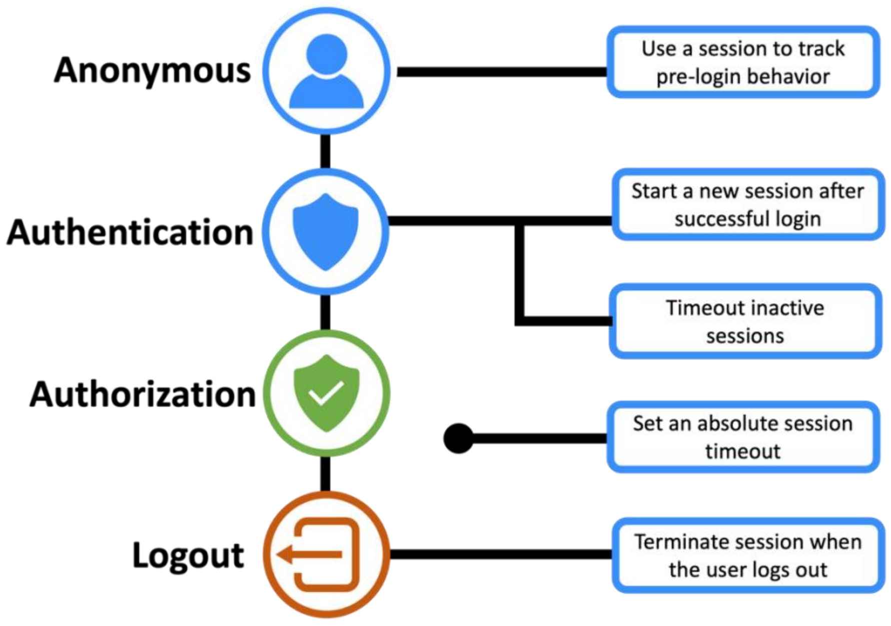
## 5. Credential Security

Credential security highlights the policies put in place by a system to ensure that a user's password isn't weak enough to be guessed or reused for too long to be compromised. Attackers can keep trying to force their way into a system if passwords aren't changed on a regular basis. There are two ways to avoid this form of attack:

1. Password Policy
   - A strong password policy forces a user to enter or create a password that needs to meet criteria such as the use of special characters, capitals, numbers and letters. This ensures that simple passwords such "123456", or "abc123" can't be used, hence protecting the user's account. Alongside that requirement, having a database of common passwords such as "Password1" and ensuring that they aren't being used, will also ensure strong security for the user's credentials (Manico & Detlefsen, 2015).
2. Password manager
   - Some companies offer a password manager service that automatically creates a long, complicated password that are nearly impossible to guess. Allowing this service in your application provides an extra layer of security.

## 6. Overall Flow Of The Login Process

The login flow for my application will be simple, while ensuring that the user's session is secure. The user will login with their credentials and after a successful and authenticated login, a new session will be created. At this point, the user is authorized and able to use the application. After a certain time of inactivity, the session will timeout (based on an absolute timeout period). Should the user choose to logout, the session will be terminated.



Anonymous
Use a session to track pre-login behavior
Authentication
Start a new session after successful login
Timeout inactive sessions
Authorization
Set an absolute session timeout
Logout
Terminate session when the user logs out

Diagram adapted from (Manico & Detlefsen, 2015)

## 7. Username Harvesting

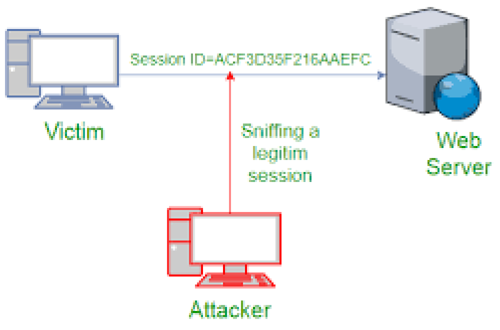
Username harvesting, or credential harvesting, is a type of cyberattack in which criminals gather masses of user credentials and use them to access systems and gather sensitive information. These login credentials are also put up for sale on the dark web (Bergmans, 2023). The strategy I will use to protect the application against this from of attack is limiting rates on the API endpoints. This will allow me to limit the number of requests that a specific IP can make within a given time frame. The method I can use to accomplish this is called the Leaky-Bucket method. This entails a first come, first serve basis whereby items are queued and then processed at a regular rate. This prevents a flood of data from being processed at the same time.

## 8. Brute Force Attacks

A brute-force attack is a trial-and-error form of cyberattack in which the attacker attempts to use a large number of password combinations to gain unauthorized access to the victims account (Descalso, 2022). Although this has accounted for millions of intrusions across user accounts, there is a simple way in which I will protect the application against this attack. When the incorrect password has been entered more than 5 times, I will have the account locked. This way, the attacker won't be able to force their way into the user's account.

## 9. Session Jacking

Session jacking involves the attacker literally hijacking a user's session, with them losing control and having their personal information available to the assailant (Baig, 2021). An example of when this attack occurs quite often is when a user engages in online banking. The protective measure I'll take to prevent this attack is Identity Verification (Baig, 2021). By validating IP addresses associated with the active session, I can invalidate the session if an IP is attempting to access it from a different location.



## 10. Session Fixation

Session fixation is the process of an attacker setting the user's session identifier to a value that is known to the attacker himself. This way, both the user and attacker have the same session cookie in their browser (Selenius, 2021). The moment the user logs in, the attacker will also have access to those credentials. The protection strategy I will be using to prevent this kind of attack is known as session regeneration. This is the process of creating a new session identifier upon login. This allows the application to grant a user with a new, authenticated session identifier.

# Bibliography

Overby, S., 2022. *7 Ways to Protect Against Credential Theft.* [Online]
Available at: https://www.mimecast.com/blog/7-ways-to-protect-against-credential-theft/
[Accessed 12 September 2023].

Acunetix, 2019. *HTTP Security: A Security-Focused Introduction to HTTP.* [Online]
Available at: https://www.acunetix.com/blog/web-security-zone/http-security/
[Accessed 12 September 2023].

Shar, L. K. & Tan, H. B. K., 2012. Predicting Common Web Application Vulnerabilities from. pp. 310-313.

Mistele, K., 2020. *How to securely hash and store passwords in your next application.* [Online]
Available at: https://medium.com/codelighthouse/how-to-securely-hash-and-store-passwords-in-your-next-application-edf15873a432
[Accessed 12 September 2023].

Gorgam, L., 2023. *JWTs vs. sessions: which authentication approach is right for you?.* [Online]
Available at: https://stytch.com/blog/jwts-vs-sessions-which-is-right-for-you/
[Accessed 12 September 2023].

Descalso, A., 2022. *How to Prevent Brute Force Attacks in 8 Easy Steps.* [Online]
Available at: https://www.itsasap.com/blog/how-to-prevent-brute-force-attacks
[Accessed 12 September 2023].

Bergmans, B., 2023. *WHAT IS CREDENTIAL HARVESTING?.* [Online]
Available at: https://www.crowdstrike.com/cybersecurity-101/cyberattacks/credential-harvesting/
[Accessed 12 September 2023].

Baig, A., 2021. *What is Session Hijacking and How Do You Prevent It?.* [Online]
Available at: https://www.globalsign.com/en/blog/session-hijacking-and-how-to-prevent-it
[Accessed 12 September 2023].

Selenius, T., 2021. *Session Fixation Attacks and Prevention.* [Online]
Available at: https://www.appsecmonkey.com/blog/session-fixation
[Accessed 12 September 2023].