


## Coursework cover sheet

## Section A - To be completed by the student

Full Name: Chan Khai Shen	
CU Student ID Number: 11323943	
Semester: Aug 2023	
Lecturer: <b>R.K.KRISHNAMOORTHY</b>	
Module Code and Title: <b>INT6005CEM SECURITY</b>	
Assignment No. / Title: <b>SECURE APPLICATION DEVELOPMENT</b>	50% of Module Mark
Hand out date: <b>5/10/23</b>	Due date: <b>23/11/23</b>
Penalties: No late work will be accepted. If you are unable to submit coursework on time due to extenuating circumstances, you may be eligible for an extension. Please consult the lecturer.	
Declaration: I/we the undersigned confirm that I/we have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I/we confirm that this piece of work is my/our own. I/we consent to appropriate storage of our work for plagiarism checking.	
Signature(s): 	
Group Member 1: _____	
Group Member 2: _____	
Group Member 3: _____	
Group Member 4: _____	

Intended learning outcomes assessed by this work:		
<ol style="list-style-type: none"> <li>1. Develop and evaluate software that addresses the most common and most severe security concerns.</li> <li>2. Critically evaluate the security of an IT ecosystem.</li> </ol>		
Marking scheme	Max	Mark
<b>1. Report</b> <i>Refer to the section on Marking Scheme on detailed breakdown.</i>	<b>60</b>	
<b>2. System Functionality</b> <i>Refer to the section on Marking Scheme on detailed breakdown.</i>	<b>40</b>	
<b>Total</b>	<b>100</b>	
<u>Lecturer's Feedback</u>		
<u>Internal Moderator's Feedback</u>		

# Table of Contents

1.0	Introduction and System Overview .....	2
2.0	Design.....	2
2.1	Authentication and access control.....	2
2.2	Password hashing.....	2
2.3	Password rules .....	4
3.0	Implementation.....	5
3.1	Password hashing.....	5
3.2	Password Rules.....	7
4.0	Discussion.....	9
5.0	Summary.....	10
6.0	References.....	11

## 1.0 Introduction and System Overview

The target system in this project is SoccerDB, a website that allows user to check the details of a soccer event and bookmark the event. The system has administrator which can view and delete normal user. Security is important in SoccerDB because the system should limit the access of administrative features because these features are possibly destructive. Besides that, it is also important to protect the users' password so that these passwords cannot be cracked in case there is a data breach.

## 2.0 Design

This subtopic discusses about the possible security vulnerabilities based on the design of SoccerDB. The authentication and access control of SoccerDB will also be discussed.

### 2.1 Authentication and access control

SoccerDB implements role-based access control for normal user and administrator. The bookmark page can only be accessed by normal user, whereas the admin page can only be accessed by administrator. Figure 2.1 depicts the authentication required when accessing the said pages.

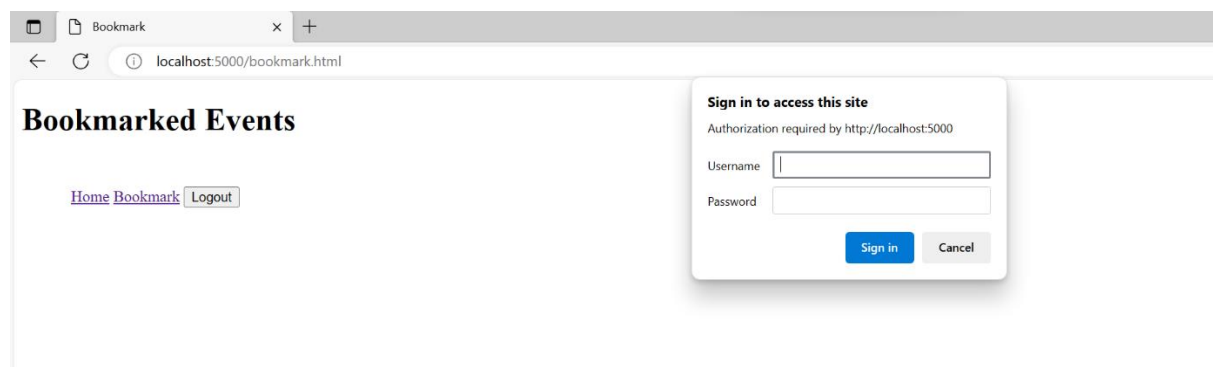


Figure 2.1 Authentication

At normal user level, SoccerDB implements attribute-based access control where every user is only allowed to access their own bookmarked lists.

### 2.2 Password hashing

In SoccerDB, Message Digest algorithm (MD5) hashing has been used to protect the password. Upon registration, the user's password is hashed before it is stored in the database. When the user login, the input password will be hashed and compared with the stored hash. If both hashes match, the user will be granted access.

Hashing is an irreversible method of converting any amount of data into a fixed length hash (Defuse Security, n.d.). One property of hashing is a slight change in the input will completely change the resultant hash. Since the resultant hash is irreversible and in fixed length, hashing prevents the real password from being revealed in the case when there is a data breach.

#### Rainbow table and reversed lookup table attack

However, hashed password is not resistant to hash cracking through rainbow table. In a rainbow table, the attacker can pre-compute the hash from a password dictionary and store it with the corresponding password (Defuse Security, n.d.). If the attacker knows which hash algorithm is used and the rainbow table is sufficient large, it is possible to find out the password.

Besides that, it is also vulnerable to reversed lookup table, i.e. the attacker runs dictionary attack on many passwords at once (Defuse Security, n.d.). This is possibly effective because if two users are using the same password, the resultant hash is also the same, so it is possible to successfully guess more than one user's passwords in one trial.

One solution to the problem is to add salt to the password before hashing. Salt is a random string, which is prepended or appended to the password before hashing, to randomize the resultant hash (Defuse Security, n.d.). When different salts are used for the same password, the resultant hashes will be different, so it is not possible to build one single rainbow table or reversed lookup table for all passwords in the database, so both rainbow table and reversed lookup table will no longer be effective. It is important that different salts are used, because if the same salt is used for all passwords, then the attacker still can use the salt to build a single rainbow table or reversed lookup table for all passwords.

#### Dictionary attack and brute force attack

However, although adding salt to password is able to make rainbow table not effective, but by knowing the salt of each password, it is still possible to launch a dictionary attack on every single password. Besides that, brute force attack is still effective because high-end graphic cards can compute the hashes in high speed (Defuse Security, n.d.).

One technique to make brute force attack less effective is key stretching. Key stretching uses stronger key derivation function, such as PBKDF2, Bcrypt, Scrypt and Argon2 to

iteratively compute the hash to add more computation time and memory usage to the password hashing process (Crypto Book, 2021). Key stretching will make the time needed to compute the hashed password becomes longer. In a normal login process, the computation is only run once, so the slow down effect has less impact. Oppositely, in a brute force attack and a dictionary attack, the computation is run multiple times, so the slow down effect has larger impact.

### Implementation

In conclusion, in this project, randomly generated salt and Password-Based Key Derivation Function 2 (PBKDF2) will be implemented to improve the existing hash function. The implementation is by using crypto library in JavaScript (will be explained in 3.1).

## **2.3 Password rules**

In SoccerDB, the password rule is such that the user is allowed to use any password with length of at least one character. When there is no specific password rule, the user has higher tendency to use a real word in dictionary or a simple variation of real word (Singer, et al., 2013). This is understandable because a real word is easier to remember than a random sequence of alphabets.

### Dictionary attack

However, this situation will impose higher risk of dictionary attack. In a dictionary attack, the attacker can use a list of dictionary real words to guess the password (Singer, et al., 2013). It is sure that, if there is an unknown sequence of alphabets, the number of all possible real words is much lesser than the number of all possible combination. So, the effort of running a dictionary attack is considerably low, but if the password is a real dictionary word, the chance of success guess is high.

One solution is enforcing password rule. Password rule reduces the probability of successful dictionary attack by increasing the password entropy (Singer, et al., 2013). Password rule, like forcing to add special characters and digits, will make the password becomes very different from a real dictionary word, so it will turn a dictionary attack not effective.

### Brute force attack

Generally, the purpose of password rule is to increase password entropy to reduce dictionary attack and brute force attack. Some researcher conceptually defined that password entropy as a measure of the randomness of a password in all possible passwords (Singer, et al., 2013). Other researcher defined password entropy as a measurement of how difficult a password can be guessed by brute force attack (Sheldon, n.d.).

As represented in the equation below, commonly the password entropy (E) is calculated as the logarithm of the total number of possible characters (R) in the power of the length of password (L) (Sheldon, n.d.).

$$E = \log_2 R^L$$

By looking at the formula, increase the number of possible characters (R) can increase the entropy (E). One way of increasing number of possible characters (E) is force the user to include special characters, digits, lower-case and upper-case characters (which is in line with the solution for dictionary attack). One other way of increasing entropy (E) is increasing length of password (L) which is simply enforce minimum length.

### Implementation

In this project, password rules will be enforced during user registration and a simple password strength checker will be implemented on the register page. The password checker displays the required password rules on screen and update the adherence of the rule as the user enters the password.

## **3.0 Implementation**

This subtopic discusses about the implementation that has been done in this project based on the security vulnerabilities discussed in 2.0.

### **3.1 Password hashing**

To overcome rainbow table attack, reversed lookup table attack, dictionary attack and brute force attack, PBKDF2 has been used for password hashing. PBKDF2 takes password, salt, pseudorandom function (PRF), number of iterations (C) and length of derived key (dklen)

as input (Vettivel, 2021). Below are the steps in PBKDF2. (Note: DK: derived key; T: subkey; hlen: length of PRF output; U: output of PRF)

$$DK = \text{PBKDF2}(\text{password}, \text{salt}, \text{PDF}, C, \text{dklen})$$

$$DK = T_1 + T_2 + T_3 + \dots + T_{\text{dklen}/\text{hlen}}$$

$$T_i = F(\text{password}, \text{salt}, c, i) \text{ where } i = 1, 2, 3, \dots, \text{dklen}/\text{hlen}$$

$$F(\text{password}, \text{salt}, c, i) = U_1 \text{ xor } U_2 \text{ xor } U_3 \text{ xor } \dots \text{ xor } U_c$$

$$U_1 = \text{PRF}(\text{password}, \text{salt} + \text{INT\_32\_BE}(i))$$

$$U_j = \text{PRF}(\text{password}, U_{j-1}) \text{ where } j = 2, 3, 4, \dots, c$$

As depicted above, the derived key (DK) is constructed by concatenating a few subkeys (T) produced based on the pseudorandom function (PRF). The number of subkeys is calculated based on the length of derived key divided by the length of PRF output (which is also the length of subkey).

At the subkey level, the password, the salt, the number of iterations and the subkey number (i) are used as the input to compute the subkey. The PRF is run as many times as the specified number of iterations. In the first iteration, the password and the salt concatenate with a big-endian integer representation of i are used as the input of the PRF. In subsequent iterations, the password and the PRF output of the previous iteration are used as the input of the PRF. Finally, exclusive or operation is performed on the output string of every iteration and the resultant string is the subkey.

It is important to note that, in the implementation of PBKDF2, the particular digest algorithm specified in the input will be implemented as the PRF (Geeks for geeks, 2021). This means the PRF can be any cryptography hashing algorithm.

In this project, the crypto library is used to implement the PBKDF2 function. As depicted in Figure 3.1, the PRF specified is Secured Hash Algorithm – 256 (SHA-256) and the number of iterations is fixed as 100. The length of derived key is fixed as 512. As depicted in Figure 3.1, the crypto library is also used to generate a different random 32-byte long salt for every user.



```

const salt = crypto.randomBytes(32).toString('base64');

crypto.pbkdf2(password, salt, 100, 512, 'sha-256', (err, derived)=>{

  if (err != null) {
    console.log('register error: ', err);
    res.status(500).send();
    return;
  }

  const securePassword = derived.toString('base64');

  const UserValue = new userModel({
    username: username,
    password: securePassword,
    salt: salt
  });

  UserValue.save().then(result=>{
    console.log('register: Success save ' + result);
    res.status(200).json({
      message: 'Registered'
    });
  }).catch(error=>{
    console.log('register: Error save ' + error);
    res.status(500).send();
  });
});

```

Figure 3.1 PBKDF2 code

Figure 3.2 depicts the generated password and salt. The original password entered is “My\$Password1234”.

```

register cks3 My$Password1234
register: Success save {
  username: 'cks3',
  password: 'WfGF/YATqB2+8p2b0T09xvy4qt51PeDXsr7yA4TkNgCt500BAiaSiL40Dw81KzIpFMAfnapo/bAiiAeDHgdTvPhauieFZ8e4EMW09GY35eJqbgVnc8B3JM7ZvQcB1N
z0t9PX7oPh021HUP5Pfb7ZpCCwzWuSc9xcRTsrQXTlG0o56ybQmgSjtyHt+Uh2wikcTaQRH6fVhBaYR4rH/wOeolsF8c8q89pcmslvmMZ1cHQvwnqcLQE3ox41/H5CiyvEMnBMMWJ1
MwJ1fYQ7Sg5iCTCvVj87k8L0KhOtNyn7s0wn/dfq1lfAeG2iSMX8XqXiSv3hPvqYoz1jmbBjDr24FZERYfg/W3PF5Q6+e1Vvmwt2B3X49BiYu1QSjPsESxc9js8mAND0+NIiow7KCV9
wERJcw+4spT3X3Tb1q3ciJ9PMMc3abmGps8WYCjTZMPR90qMK8H5C971SM8hz4TKMoGaRL/Yy4puSfzCF0qrdxIzVPa2mrPZH1hE6raVRT6L+JAvQ5oD6Q4AKBhRwCpPcAjIR6K+BQk
1W8VswDzp1ZvIrIF0Q0qDX+ugQAdA80aIMmwnTcLr8nyTsxGs50t61EUrEY=',
  salt: 'g651IY/73V6whaZft4K9ck86BWrnmbljmey0c1j7qgg=',
  _id: new ObjectId("655ddd42b26bfb567724ee85"),
  registerDate: 2023-11-22T10:51:46.694Z,
  __v: 0
}

```

Figure 3.2 PBKDF2 results

Upon successful registration, the password and salt of that user is stored into the database. During login, the salt stored for the user is used to compute the hashed password from the password provided by user. If that hashed password matches the stored password, then login is successful.

## 3.2 Password Rules

To make dictionary attack and brute force attack more difficult, password rules are enforced and password strength checker are implemented. The mandatory requirement of password enforced during registration are listed in Table 3.1.

Table 3.1 Password requirements

Requirement	Attack
Minimum 10 characters	Brute force attack
Has digit	Dictionary attack
Has lower case alphabet	Dictionary attack
Has upper case alphabet	Dictionary attack
Has special character	Dictionary attack

The following is the pseudocode of the password strength checker.

```

R = 0

IF has 10 characters:
    Indicate has 10 characters

IF has digit:
    Indicate has digit
    R = R + 10

IF has lower case alphabet:
    Indicate has lower case alphabet
    R = R + 26

IF has upper case alphabet:
    Indicate has upper case alphabet
    R = R + 26

IF has special character:
    Indicate has special character
    R = R + 32

L = Length of password
E = R ** L

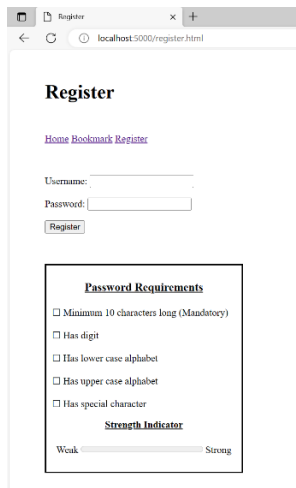
SET value of strength meter = E

IF has 10 characters AND has digit AND has lower case alphabet AND has uppercase alphabet
AND has special character:
    Allow submit registration
ELSE:
    Not allow submit registration
  
```

As depicted in the pseudocode, the password strength checker will check the adherence of the password requirements as the user inputs the password. Initially the requirement is “not ticked”. As the requirement is fulfilled, the requirement will be “ticked”. To ensure the overall strength of all users’ passwords, the user is allowed to submit the registration only when he

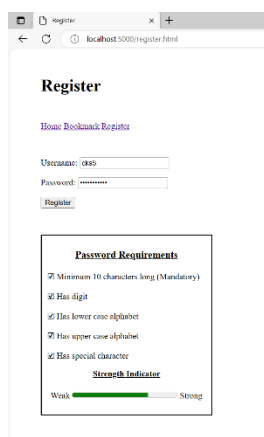
fulfilled all requirements. Besides that, the password strength checker will also calculate the password entropy and reflect the entropy in a meter. The password entropy is calculated based on the formula stated in 2.2.

Figure 3.3 shows the password strength checker in idle condition. Figure 3.4 shows the password strength checker checking a user's password.



The screenshot shows a web browser window with the address bar displaying 'localhost:5000/register.html'. The page title is 'Register'. Below the title, there are links for 'Home', 'Bookmark', and 'Register'. The 'Register' link is highlighted. Below the links, there are input fields for 'Username:' and 'Password:', followed by a 'Register' button. Below the input fields, there is a section titled 'Password Requirements' with a list of requirements: 'Minimum 10 characters long (Mandatory)', 'Has digit', 'Has lower case alphabet', 'Has upper case alphabet', and 'Has special character'. Below the requirements, there is a 'Strength Indicator' section with a meter showing 'Weak' on the left and 'Strong' on the right, with a green bar indicating the current strength level.

Figure 3.3 Password strength checker



The screenshot shows the same web browser window as Figure 3.3, but with the 'Password' field filled with a password. The 'Register' button is still visible. The 'Password Requirements' section now shows all requirements as checked: 'Minimum 10 characters long (Mandatory)', 'Has digit', 'Has lower case alphabet', 'Has upper case alphabet', and 'Has special character'. The 'Strength Indicator' section shows a green bar indicating a strong password, with 'Weak' on the left and 'Strong' on the right.

Figure 3.4 Password strength checker (functioning)

## 4.0 Discussion

In this project, the design of SoccerDB that is related to security is examined. In a nutshell, it can be concluded that SoccerDB has implemented role-based and attribute-based access control with proper authentication feature. There are some pitfalls in SoccerDB, particularly on password hashing and password rules enforcement.

As for password hashing, in SoccerDB, password is directly hashed without adding any salt. This will open to the risk of rainbow table attack and reversed lookup table attack. So, in this project, it is suggested to add salt to password before hashing. Due to the existence of high-end graphic card which can compute hashed password in short time, dictionary attack and brute force attack are still effective. So, key stretching should be implemented to slow down the password hashing process. Finally, in this project, for the password hashing part, PBKDF2 is implemented as it includes both adding salt and key stretching.

As for password rules, in SoccerDB, the only rule is that the password must have at least one character long. This will open to the risk of dictionary attack because when there is no restriction, it is possible to use a real dictionary word as password. So, it is suggested that restrictions, such that must have digits, lower case alphabet, upper case alphabet and special character, must be enforced. Besides that, to make brute force attack more expensive, it is also suggested that a minimum length is enforced. Finally, to guide the user through the password requirements, a password strength checker is implemented on the registration page, which will update the adherence of the password requirement in a list as the user enters the password, and at the same time depicts the password entropy in a meter.

## **5.0 Summary**

As a summary, publicly available websites, like SoccerDB, is open to various types of security attacks, so it is important to continuously strengthen the system in security aspect. As part of future enhancement, user activity logging can be looked into for retaining a record of user's interaction with the system.

## 6.0 References

Crypto Book, 2021. *KDF: Deriving Key from Password*. [Online]  
Available at: <https://cryptobook.nakov.com/mac-and-key-derivation/kdf-deriving-key-from-password>

[Accessed 2 November 2023].

Defuse Security, n.d. *Salted Password Hashing - Doing it Right*. [Online]  
Available at: <https://crackstation.net/hashing-security.htm>

[Accessed 27 October 2023].

Geeks for geeks, 2021. *Node.js crypto.pbkdf2() Method*. [Online]  
Available at: <https://www.geeksforgeeks.org/node-js-crypto-pbkdf2-method/>

[Accessed 15 November 2023].

Sheldon, R., n.d. *Password Entropy*. [Online]  
Available at: <https://www.techtarget.com/whatis/definition/password-entropy>

[Accessed 2 November 2023].

Singer, A., Anderson, W. & Farrow, R., 2013. Rethinking Password Policies.

Vettivel, N., 2021. *PBKDF2 Hashing Algorithm*. [Online]  
Available at: <https://nishothan-17.medium.com/pbkdf2-hashing-algorithm-841d5cc9178d>

[Accessed 14 November 2023].

Paper Title	Uploaded	Grade	Similarity	
Coursework 2 draft Chan Khai Shen.pdf	23 Nov 2023 23:09 +08	--	1%	  