# Password Management:

**Q1) Describe how passwords are stored, transmitted and authenticated**

1. The password entered by the user is hashed using SHA-256 algorithm and then stored in the database. The hashed password is stored in the variable $passwordHash in the code. When the user submits their login credentials, the entered password is hashed using the same SHA-256 algorithm, and the resulting hash is compared against the stored hash in the database to authenticate the user.

It is worth noting that the code is using the outdated MySQL extension, which is no longer recommended for use and has been deprecated in PHP 7.0. It is also not using prepared statements or input validation, which can leave the application vulnerable to SQL injection attacks. Additionally, the code is storing the password hash in plaintext cookies, which can be intercepted by attackers and used to impersonate the user. Overall, this code is not secure and should not be used in production environments.

**Q2) Identify and describe two vulnerabilities in the password management system and explain how they can be exploited. (note: your answer should not involve the possibility of “weak passwords” and should not involve cookies)**

1. **Password Hashing Vulnerability**: The code inside register.php and members.php uses a weak hash function to store passwords in the database. The password is hashed using the SHA-256 algorithm in the line **$passwordHash = hash('sha256', $\_POST['password']);**. While SHA-256 is a secure hashing algorithm, it is not sufficient for password storage as it is fast and can be easily cracked by attackers using brute-force attacks or rainbow tables.

To prevent this vulnerability, the code should use a strong password hashing algorithm such as bcrypt or Argon2, which are designed for password storage and provide additional security features such as salt generation and key stretching.

1. **Lack of Input Validation:** The code inside members.php does not validate user input before using it, which could lead to unexpected behavior or errors. For example, the code assumes that $\_POST['username'] and $\_POST['password'] are always set, but this might not be the case.

To fix this vulnerability, validate all user input before using it. Check if the input is empty, has the expected format, or falls within acceptable ranges, which is done by preg\_match('/^[a-zA-Z0-9]+$/')