Software testing

short assignment.

Task list:

- 1. Create a functional micro-p2p chat app using MERN stack
- 2. Setup Badstore on your machine
- 3. Create 10 test cases for different pages of the BadStore web site using Selenium (JavaScript)
- 4. Identify 5 vulnerabilities in the Badstore web site
- 5. Fix the vulnerabilities in the Badstore web site

Completed tasks: ALL THE FIVE TASKS ARE COMPLETED

The submission should include the following:

- a. maximum 4-minute video demonstrating your work—the functionality and walkthrough of the code. Make sure you are audible.
- b. Link to the GitHub repo complete with a proper readme and a task checklist. Add a report of the vulnerabilities identified and the method of fixing them. Also mention the guides that you followed.

Evaluation criteria:

You'll be judged on the basis of:

- a. Number of tasks completed
- b. Difficulty and rarity of the vulnerabilities
- c. Your speed

vulnerabilities identified:

1. SQL Injection in Search and Login Functions:

 The application is vulnerable to SQL injection attacks in the Search and Login functions. Attackers can manipulate input, such as logging in with a payload like 'joe' OR 1=1 OR 'mary', to exploit SQL injection vulnerabilities and potentially gain unauthorized access.

2. Blind SQL Injection in Supplier Login:

The Supplier Login functionality is susceptible to blind SQL injection attacks. Techniques like single quotes ('), OR 1=1, OR 1=1--, and other SQL commands can be attempted to identify the "magic" combination and exploit the vulnerability.

3. Cross-Site Scripting (XSS) in Guestbook, URL's, and Search:

 The Guestbook, URL fields, and Search functionality are vulnerable to Cross-Site Scripting (XSS) attacks. Injecting scripts, for example, alert('This is an XSS attack!!!')</script>, can lead to the execution of malicious scripts in the context of other users' browsers.

4. Robots.txt Directory Disclosure:

• The robots.txt file is exposed, revealing potential directory and file information that could aid attackers in understanding the application's structure and endpoints.

5. Credential Disclosure via Proxy, XSS, and Brute Force:

 Attackers can disclose credentials using proxy tools to decode Base64-encoded SSOID cookies. Techniques like <script>alert(document.cookie)</script> and brute force attacks can be employed.

6. Command Injection via Parameter Tampering:

 Command injection vulnerabilities exist due to parameter tampering, enabling attackers to execute arbitrary commands on the server.

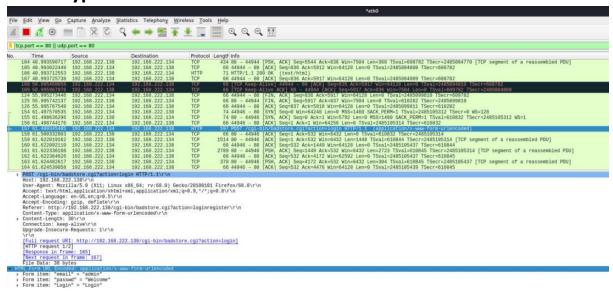
7. Privilege Escalation via Cookie and Hidden Field Tampering:

• Tampering with cookies and hidden fields, particularly the 'Role' parameter, can lead to privilege escalation, allowing attackers to gain unauthorized access.

8. "Secret" Admin Access via URL Parameter:

A secret admin access is accessible via a URL parameter.
Attackers can try accessing the admin functionality using the URL parameter ?action=admin.

9. Unencrypted traffic



Fix Vulnerabilities:

1. SQL Injection:

- Use parameterized queries or prepared statements to handle user inputs securely.
- Implement input validation to ensure that only expected values are accepted.
- Escape special characters in SQL queries.
- Whitelist Input Validation
- Escaping All User Supplied Input

2. Blind SQL Injection in Supplier Login:

- Apply similar measures as mentioned for the SQL Injection in Search and Login Functions.
- Regularly update and patch the application to fix any underlying vulnerabilities in the database interaction.

3. Cross-Site Scripting (XSS) in Guestbook, URL's, and Search:

- Sanitize and validate user input to prevent the execution of malicious scripts.
- Implement Content Security Policy (CSP) headers to mitigate XSS risks.
- Encode output to ensure that user-generated content is displayed safely.
- Perform appropriate validation and escaping on the server-side

4. Robots.txt Directory Disclosure:

- Review the content of the robots.txt file and remove any sensitive information.
- Ensure that the file doesn't expose directory structures or sensitive endpoints.
- Use of META tags rather than entries

 - Adjust the web server's access controls to limit access to sensitive material

5. Credential Disclosure via Proxy, XSS, and Brute Force:

- Avoid storing sensitive information in cookies.
- Use secure protocols (HTTPS) to encrypt communication.
- Implement account lockout mechanisms to prevent brute force attacks.
- Regularly monitor and audit logs for any suspicious activities.

6. Command Injection via Parameter Tampering:

- Validate and sanitize user inputs to prevent command injection.
- Avoid using user inputs directly in system commands.
- Input Validation ("Assume all input is malicious")
- Use a list of acceptable inputs that strictly conform to specifications
- Reject any input that does not strictly conform to specifications
- Use parameterized queries for database interactions to prevent SQL injection, which might lead to command injection.

7. Privilege Escalation via Cookie and Hidden Field Tampering:

- Encrypt and sign cookies to prevent tampering.
- Store sensitive information on the server-side rather than relying solely on client-side data.
- Implement proper access controls to restrict privileges.

8. "Secret" Admin Access via URL Parameter:

 Remove any secret URLs or parameters that provide unauthorized access.

- Implement proper authentication and authorization mechanisms.
- Conduct regular security reviews to identify and eliminate hidden functionalities.

9. SWEET32: 'Birthday attack'

- stop using legacy 64-bit block ciphers
- rekeying the session frequently

Guides:

https://rest.chatengine.io/

https://www.youtube.com/watch?v=SDMs2Pq6w90

https://www.youtube.com/watch?v=HggSXt1Hzfk

nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf

Web Server robots.txt Information Disclosure | Tenable®