19 FEBRUARY 2025

ASSIGNMENT
SESSION MANAGEMENT AND WEB SECURITY
/308/MAHMOUD KANI

SUBMITTED BY:
AYILARA BUSARI DARE
IDEAS/24/28133

INTRODUCTION:

Session Management & Web Security - Session Management is a vital aspect of web security that involves handling user interactions with a web application after authentication. It ensures users remain authenticated throughout their session without needing to log in repeatedly. Secure session management involves:

- **Generating unique session IDs**
- **Securely storing and transmitting session tokens**
- **Implementing session timeouts and inactivity logout**
- **Protecting against session hijacking, fixation, and replay attacks**

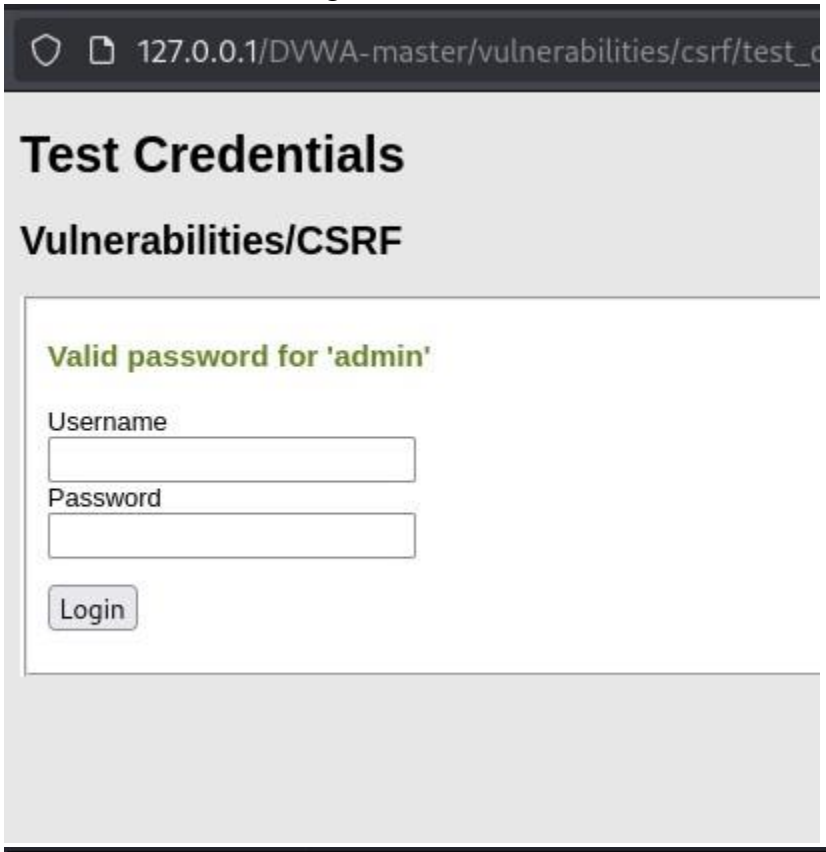
Web Security focuses on protecting web applications from cyber threats like hacking, data breaches, and unauthorized access. Key principles include:

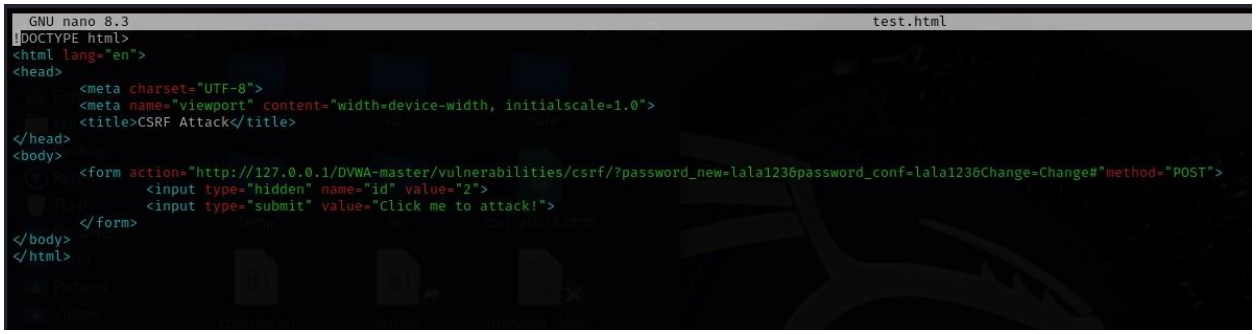
- **Authentication & Authorization (e.g., Multi-Factor Authentication)**
- **Input Validation & Sanitization (to prevent SQL Injection & XSS)**
- **Encryption & Secure Data Transmission (e.g., HTTPS, TLS)**
- **Regular Security Audits & Patch Management**

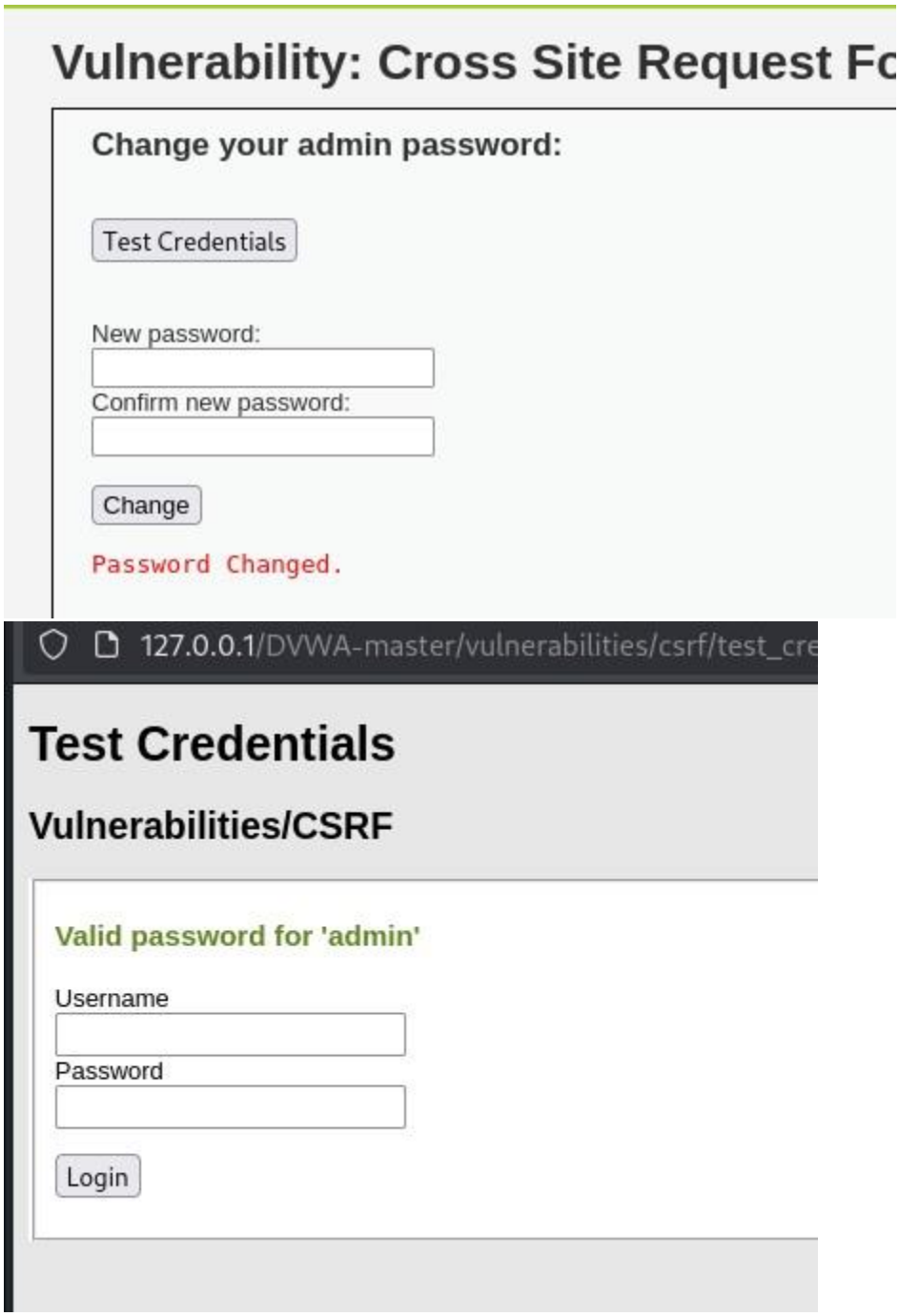
Effective session management and web security work together to ensure safe and uninterrupted user experiences while mitigating cyber threats.

Cross-site Request Forgery (CSRF) Protection.

Exercise 1: Understanding CSRF Attacks







Password has been successfully changed:

This confirms that the **CSRF attack was successful** and that **DVWA is vulnerable to CSRF**. This means an attacker could trick a logged-in user into unknowingly changing their password (or performing other sensitive actions) just by visiting a malicious website or clicking a deceptive link.

Importance of CSRF Protection:

1. **Prevents Unauthorized Actions: CSRF protection ensures that malicious websites or scripts cannot force users to perform actions like changing account settings, transferring money, or deleting data without their consent.**
2. **Safeguards User Trust: By preventing unauthorized actions, CSRF protection helps maintain user trust, ensuring they can interact with web applications without fear of their accounts being misused.**
3. **Maintains Data Integrity: CSRF attacks can manipulate or delete sensitive data. Implementing CSRF protection ensures that only legitimate requests from authenticated users are processed, preserving data integrity.**
4. **Protects Sensitive Transactions: Online banking, e-commerce, and social media platforms often involve sensitive actions. CSRF protection ensures that these transactions cannot be triggered by malicious third parties.**
5. **Compliance with Security Standards: Many security regulations and standards, such as OWASP Top 10 and GDPR, emphasize the importance of preventing CSRF attacks, ensuring organizations meet legal and industry compliance requirements.**

How CSRF Protection Works:

- **CSRF Tokens: Web applications generate unique CSRF tokens for each user session. These tokens are included in HTTP requests, ensuring that only requests with valid tokens are processed.**
- **Same-Site Cookies: Using cookies with the SameSite attribute restricts their transmission to requests originating from the same domain, reducing the risk of CSRF attacks.**
- **User Authentication and Session Management: Proper user authentication and session management help ensure that only authorized users can perform sensitive actions.**
- **CSRF protection is essential for ensuring the security, integrity, and trustworthiness of web applications by preventing unauthorized actions and safeguarding sensitive user data.**

Exercise 2: Implementing CSRF Protection

```php
// Generate a CSRF token
if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

// Include CSRF token in forms

<form action="http://your-dvwa-url/vulnerabilities/csrf/"
method="POST">
        <input type="hidden" name="id" value="2">
        <input type="hidden" name="csrf_token" value="<?php echo
$_SESSION['csrf_token']; ?>">
        <input type="submit" value="Submit">
</form>

session_start();

// Validate the CSRF token
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        if (!hash_equals($_SESSION['csrf_token'],
$_POST['csrf_token'])) {
        die("CSRF token validation failed!");
        }
        // Proceed with the request processing
}
?>
```

How CSRF Tokens Work:

1. **Token Generation:**
    - When a user logs into a web application, the server generates a unique CSRF token for that user's session. ○ The token is usually a long, random string that is difficult to predict. ○ It is stored on the server (linked to the user's session) and sent to the client.
2. **Token Inclusion in Forms:**
    - Every sensitive action that requires user input (e.g., form submissions, account updates, or financial transactions) includes the CSRF token. ○ The token is typically added as a hidden input field within the form: html CopyEdit

```
<form action="/update-profile" method="POST">
  <input type="hidden" name="csrf_token" value="a9f8a7d6c5e4b3a2c1d0" />
```

<input type="text" name="username" />

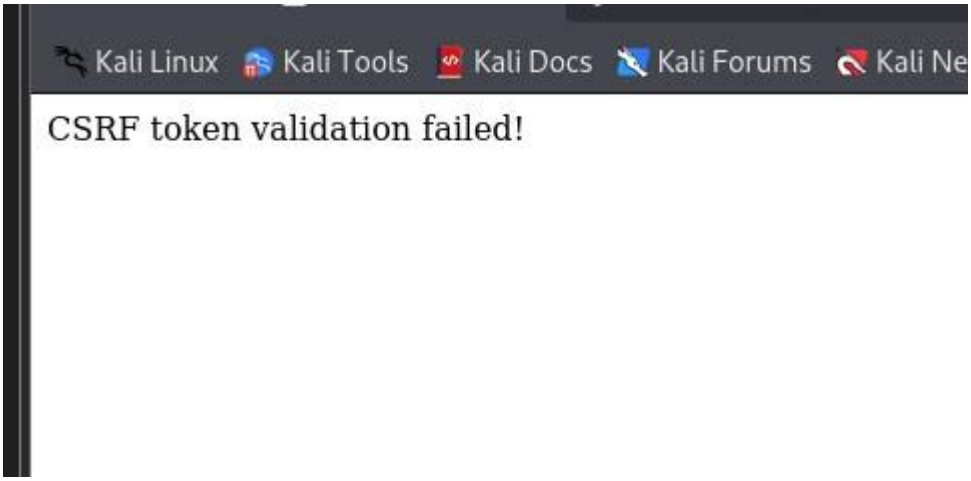
<input type="submit" value="Update" />

</form>

3. **Token Submission with Requests:**
    - When the user submits the form, the CSRF token is sent along with the request. For AJAX requests, the token can be included in the headers or as part of the request body.
4. **Token Validation:**
    - Upon receiving the request, the server compares the token in the request with the one stored in the user's session.
    - If the tokens match, the request is considered legitimate and is processed.
    - If the tokens do not match, the server rejects the request, preventing any unauthorized actions.
5. **Token Expiry and Regeneration:**
    - CSRF tokens often expire after a certain time to enhance security.
    - The server may also regenerate tokens periodically or after certain actions, making it harder for attackers to reuse stolen tokens.

Exercise 3: Testing CSRF Protection 1. Tes

the Protection:



2. Analyze the Response:

CSRF token validation failed: which application is secure against CSRF attacks since it requires a valid token for form submissions.

CSRF Token Removal Test (DOCUMENTATION) Objective:

To determine if the application validates the CSRF token when submitting a form.

Procedure:

1. **Accessed the attack form.**
2. **Removed the CSRF token field from the form.**
3. **Submitted the form without the token. Result:**

The form submission was unsuccessful, indicating that the application rejected

the request due to the missing CSRF token. Evaluation:

The CSRF protection mechanism is effective because the server correctly identified the absence of a valid CSRF token and prevented the form submission. This validation ensures that unauthorized requests from malicious websites cannot be processed, reducing the risk of CSRF attacks. Overall, the CSRF protection is properly implemented and functioning as intended.

REFLECTION:

Potential Limitations of CSRF Tokens

While CSRF tokens are an effective defense mechanism, they have several limitations:

1. **Token Leakage:** If an attacker gains access to a user's CSRF token (e.g., through XSS attacks), they can perform unauthorized actions.
2. **Token Reuse:** Some applications use predictable or static tokens, making it easier for attackers to guess or reuse them.
3. **Token Mismanagement:** Improper token validation, such as only checking for token presence rather than verifying its authenticity, weakens protection.
4. **User Experience Impact:** Frequent token expiration can disrupt user sessions, leading to usability issues.
5. **Third-Party Integrations:** CSRF tokens may not cover requests made through external integrations or APIs if not configured correctly.

How Attackers Might Bypass CSRF Protection

1. **Cross-Site Scripting (XSS):** If an attacker exploits an XSS vulnerability, they can steal a user's CSRF token directly from their browser.
2. **Session Fixation:** Attackers might force a user to use a known session ID, potentially allowing unauthorized requests.
3. **Clickjacking:** By embedding the vulnerable application in an iframe and tricking users into clicking invisible buttons, attackers can bypass visual cues.
4. **Poor Token Implementation:** Weak token generation (predictable or easily guessable tokens) allows attackers to brute force or guess valid tokens.
5. **Browser Vulnerabilities:** Exploiting browser flaws can sometimes expose tokens or bypass same-origin policies.

Additional Security Strategies

1. **SameSite Cookies:** Using cookies with the SameSite=Strict attribute ensures cookies are only sent in same-site requests, mitigating CSRF without relying solely on tokens.
2. **Multi-Factor Authentication (MFA):** Requiring additional verification for sensitive actions reduces the impact of a successful CSRF attack.
3. **User Verification for Critical Actions:** Prompting users to re-enter their password or verify their identity when performing sensitive actions adds an extra layer of protection.
4. **Content Security Policy (CSP):** Implementing CSP reduces the risk of XSS attacks that could steal CSRF tokens.
5. **Secure Token Generation and Validation:** Tokens should be random, unique per session, and validated for both presence and correctness.
6. **Frame-Busting:** Using X-Frame-Options headers prevents clickjacking attacks by blocking the application from being loaded in iframes.
7. **Regular Security Audits:** Conducting penetration tests and code reviews helps identify and fix vulnerabilities that could lead to CSRF token bypass.