Security testing report

Vulnerability Assessment Report: Cross-Site Request Forgery (CSRF)

Vulnerability	**Cross-Site Request Forgery (CSRF)**
Target Application	DVWA (Damn Vulnerable Web Application)
Affected Endpoint	http://127.0.0.1/vulnerabilities/csrf/
Severity	**Critical**

Executive Summary

A **critical** Cross-Site Request Forgery (CSRF) vulnerability was identified in the "Change your admin password" function of the DVWA. This flaw allows an unauthenticated attacker to create a malicious webpage that, when visited by a logged-in administrator, will force the administrator's browser to perform an unauthorized action—in this case, changing the administrator's password to a value chosen by the attacker.
The root causes of this vulnerability are:

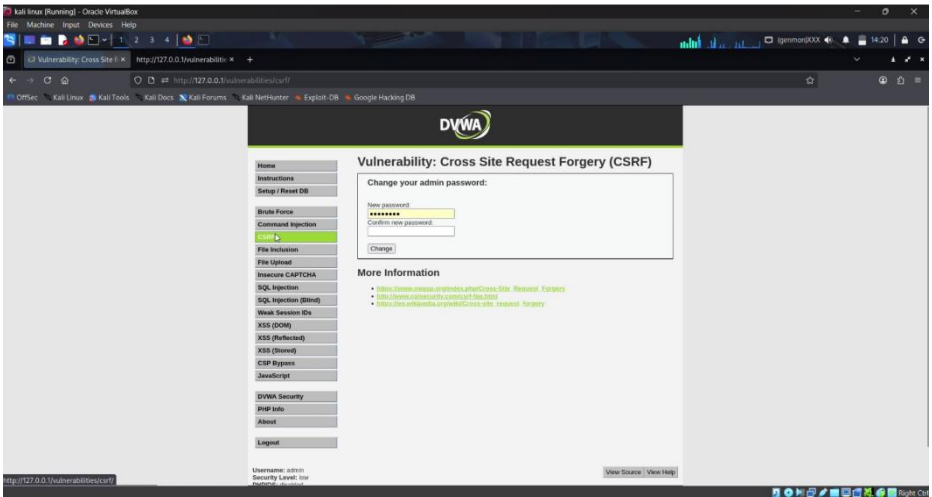
1. **Missing Anti-CSRF Tokens:** The password change form lacks a unique, secret token to verify that the request was intentionally submitted by the user.
2. **Use of `GET` Method for State Change:** The application insecurely uses an HTTP `GET` request to change the password, making the attack trivial to script.

Successful exploitation, as demonstrated in the provided evidence, results in a **full account takeover of the administrator account**, granting the attacker complete control over the application.

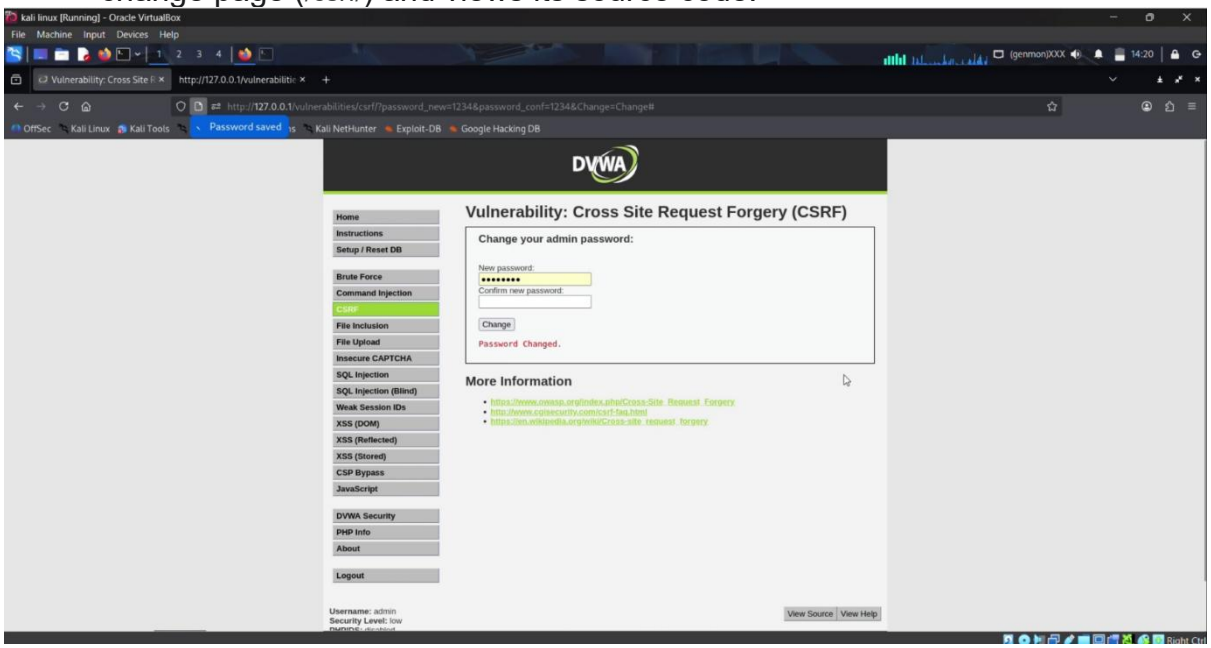
Steps to Reproduce (Attack Narrative)

The attack was performed by tricking an authenticated user into submitting a forged request.

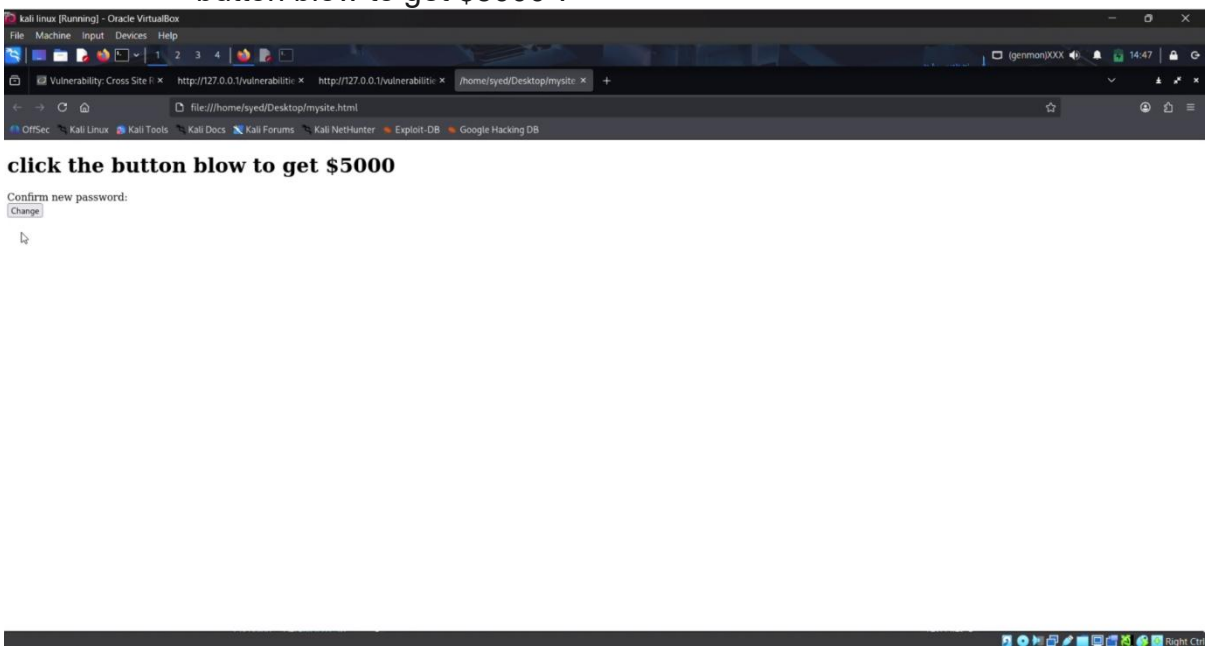
1. **Prerequisite - Victim Authentication:** The victim (in this case, `admin`) logs into the DVWA application. The browser now stores a valid session cookie for `127.0.0.1`.



2. **Vulnerability Identification (Attacker):** The attacker inspects the password change page (/csrf/) and views its source code.

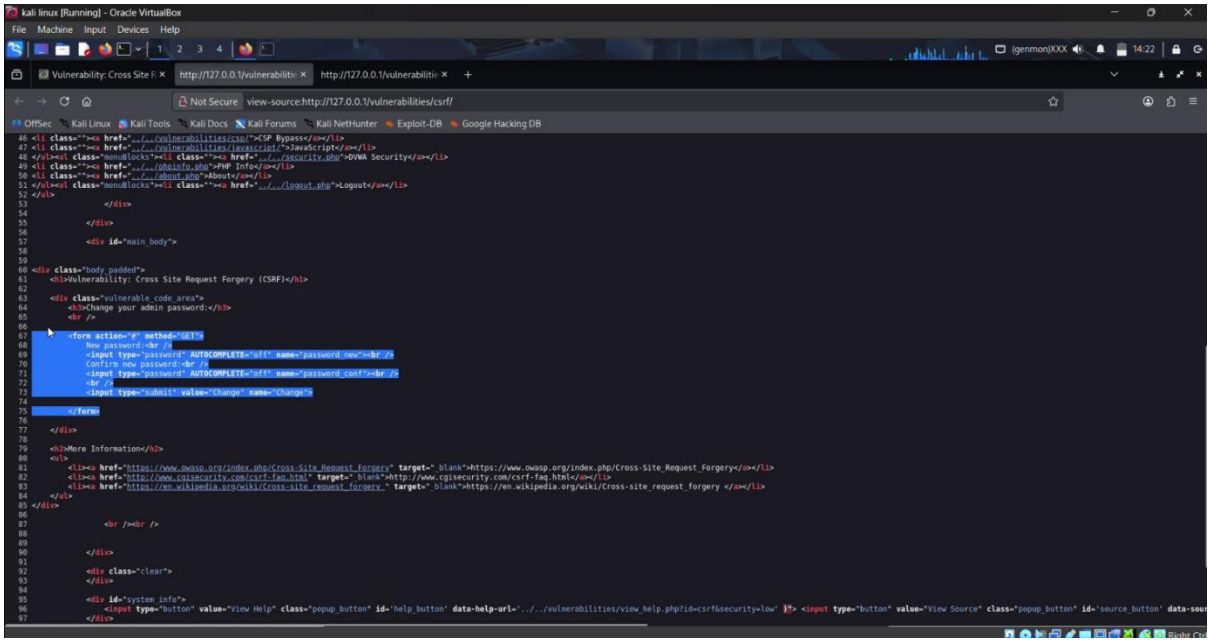


   - The inspection reveals that the form submits its data using an **HTTP GET method**.
   - Crucially, the form contains **no hidden, unique, or random anti-CSRF token**.
3. **Exploit Creation (Attacker):** The attacker creates a simple, malicious HTML page (mysite.html) to host on their own server or send directly to the victim. This page contains an HTML form that mimics the legitimate password change request.
   - **Lure:** The page is disguised with a social engineering trick, "click the button blow to get $5000".

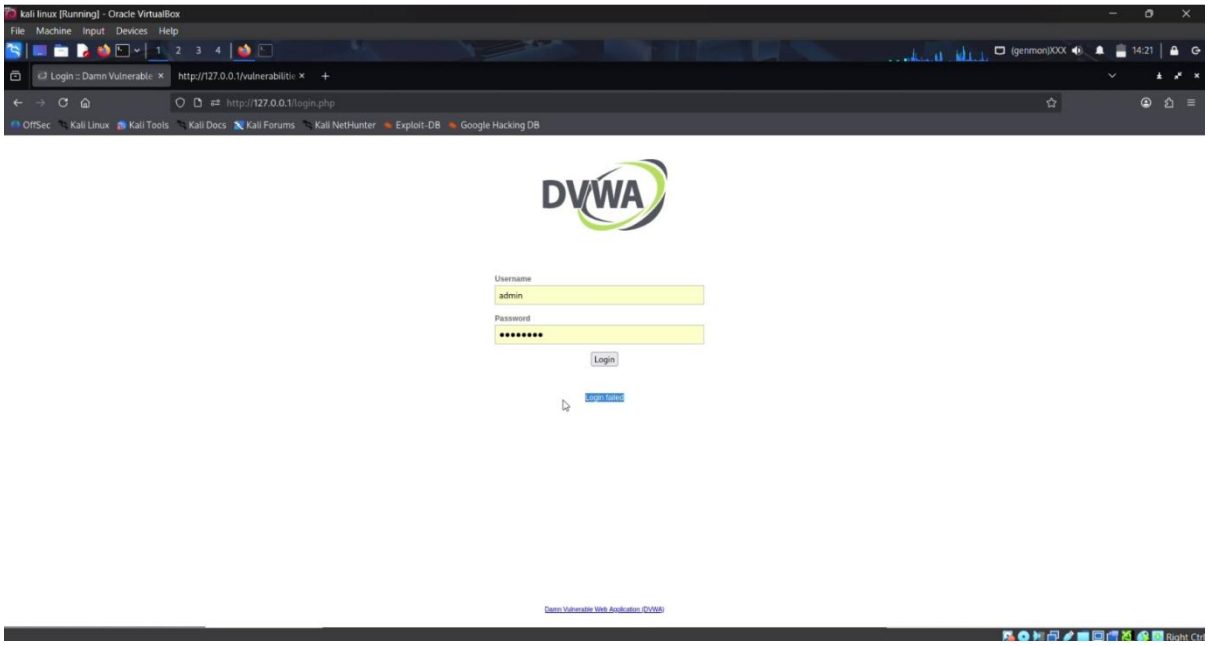


   - **Malicious Form (Hidden from user):** The source code of this page would be:
     HTML
     <html>

```
  <form action="http://127.0.0.1/vulnerabilities/csrf/" method="GET">
    <input type="hidden" name="password_new" value="hacked" />
    <input type="hidden" name="password_conf" value="hacked" />
    <input type="submit" name="Change" value="Click to get $5000" />
  </form>
</html>
```



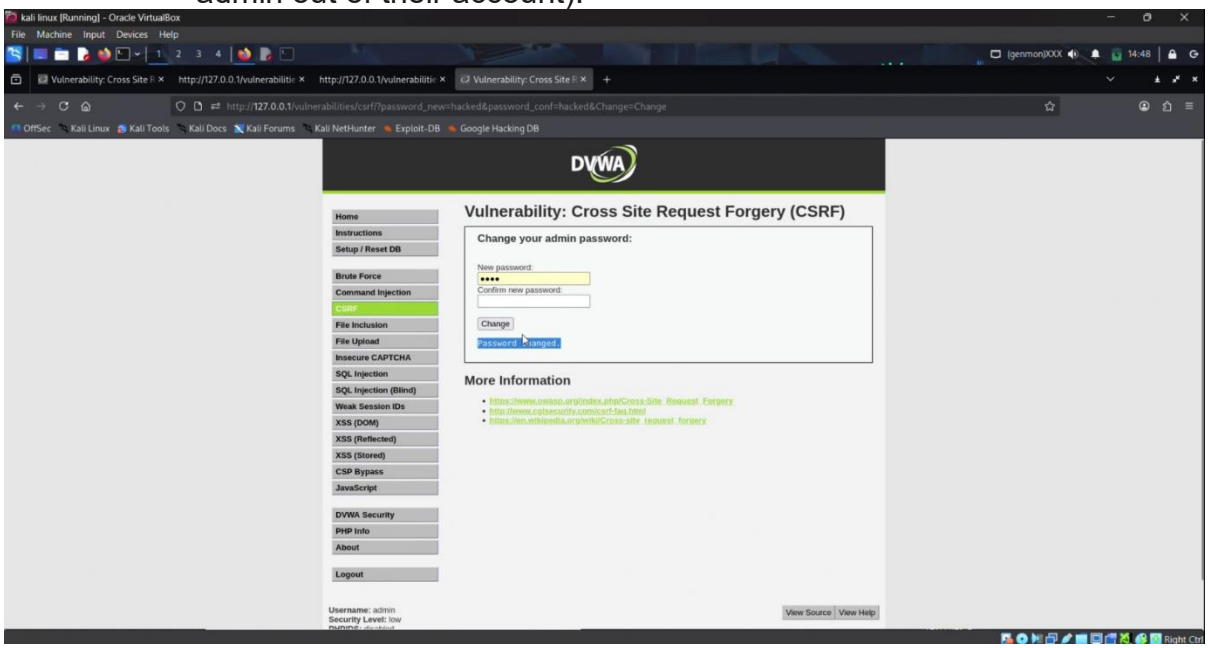
4. **Execution (Attack):** The authenticated admin user is lured (e.g., via a phishing email) to the attacker's malicious page (mysite.html).



5. **Impact (Result):** The victim clicks the "Change" (lure) button.
   - The victim's browser constructs the malicious GET request and sends it to the DVWA server:

     http://127.0.0.1/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&Change=Change

- Because the victim is logged in, the browser **automatically attaches the valid admin session cookie** to this request.
- The server receives the request, checks the valid cookie, and finds no anti-CSRF token to validate. It believes the request is genuine and **changes the admin's password to "hacked"**.
- shows the password was successfully changed, locking the legitimate admin out of their account).



💣 Impact

This vulnerability is **Critical**. It allows a complete, unauthenticated takeover of a privileged user's account. Once the attacker controls the admin account, they have full control over the application, its data, and its users.

🛡️ Remediation and Recommendations

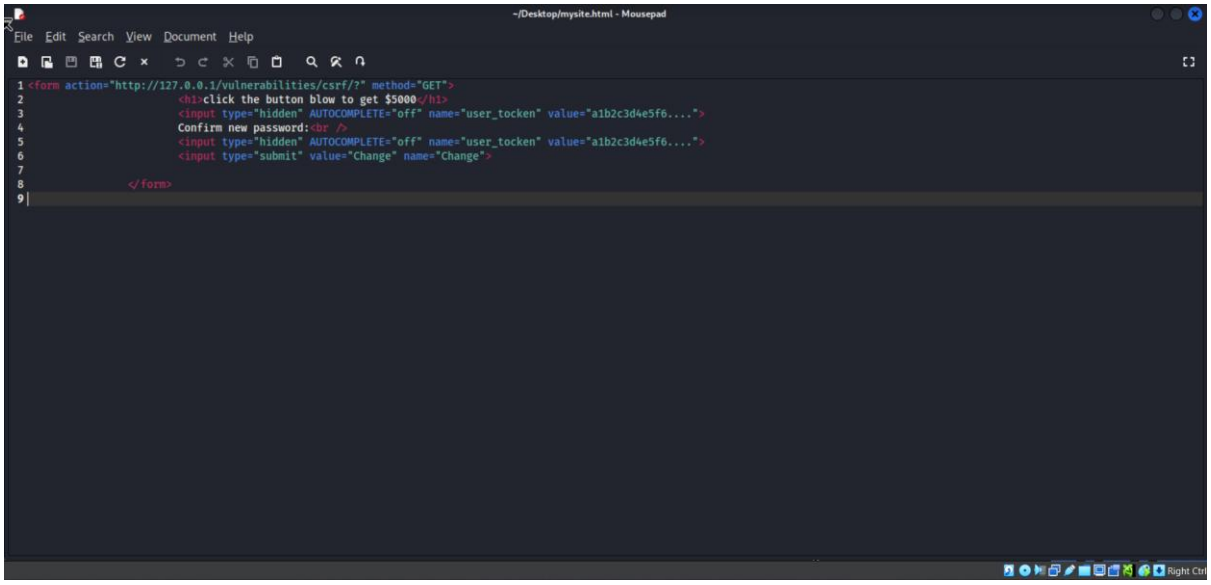
To mitigate this vulnerability, the following controls must be implemented:

1. **Implement Anti-CSRF Tokens (Primary Fix):**
   - **How:** For every state-changing request (like changing a password or submitting a form), the server must generate a unique, secret, and unpredictable token. This token should be embedded as a hidden field in the form.
   - **Validation:** When the user submits the form, the server must verify that the token submitted matches the one it generated for that user's session. If the tokens do not match, or if the token is missing, the request **must be rejected**.

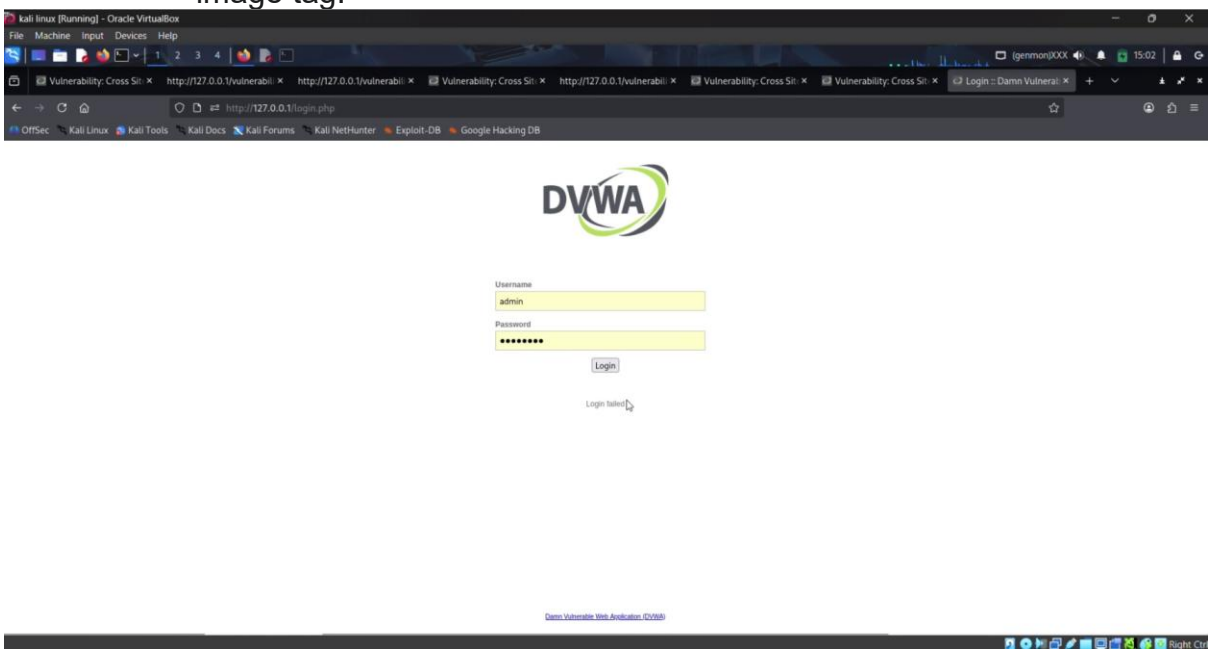
   Secure Code Example (PHP):

   PHP
   ```
   // On the form page
   $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
   <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
   // On the processing page
   ```

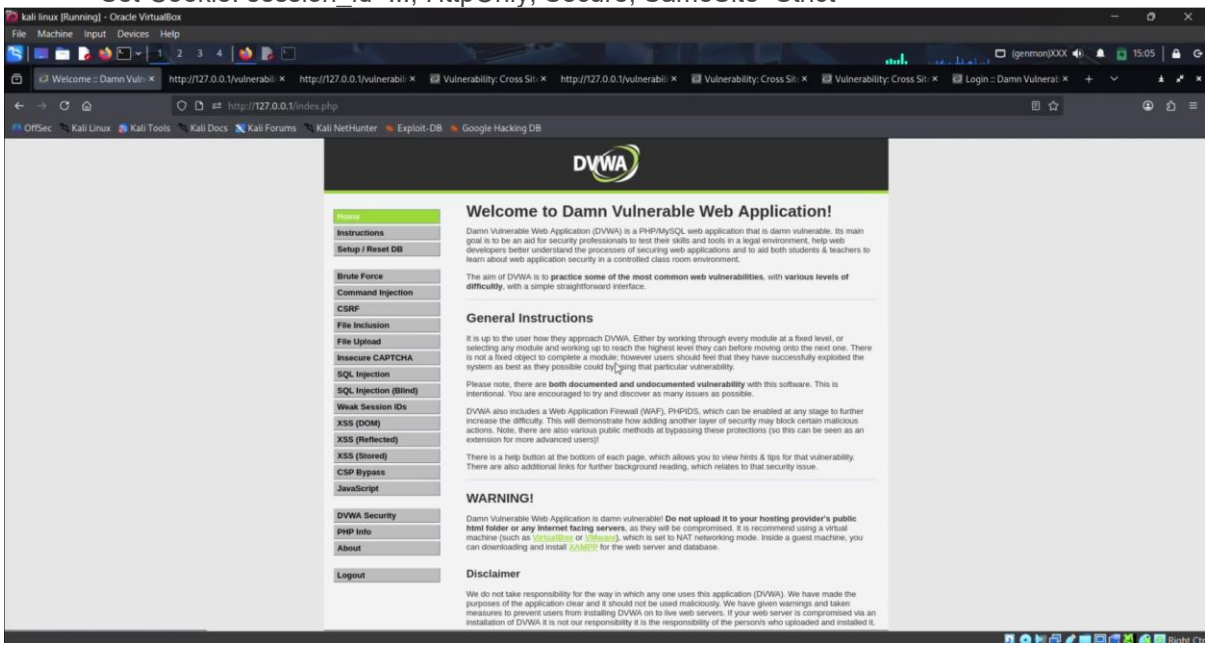
```
if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {

die("Invalid CSRF token!");

}
```



2. **Use POST for State-Changing Requests:**
   - **Why:** All actions that change data (create, update, delete) **must** use the HTTP POST method. GET requests should be idempotent (safe to repeat) and used for data retrieval only. While POST alone does not prevent CSRF, it makes it more difficult than a simple malicious link or image tag.



3. **Implement SameSite Cookie Attribute:**
   - **How:** As a strong, modern defense-in-depth, set the application's session cookie with the SameSite=Strict or SameSite=Lax attribute.

- **Effect:** This instructs the browser **not** to send the cookie with requests originating from a different domain (like the attacker's mysite.html). This effectively breaks the CSRF attack chain at the browser level.

Set-Cookie: session_id=...; HttpOnly; Secure; SameSite=Strict