PHP – CSRF

The acronym "CSRF" stands for Cross-Site Request Forgery. CSRF is an Internet exploit that involves a trusted website user issuing unauthorized commands. Providing adequate protection to a PHP web application against this attack can be achieved by taking the measures explained in this chapter.

By default, the browser uses the "GET" request method to send data. This is commonly used as the exploit point in a CSRF. To inject commands into a specific website, the attacker employs HTML tags like "IMG." For example, the url endpoint of a web application such as "/delete.php?empcode=1234" deletes account as passed from empcode parameter of a GET request. Now, if an authenticated user come across the following script in any other application.

```
<img src="http://example.com/delete.php?empcode=1234"
   width="0" height="0" border="0">
```

Inadvertently causes the data related to empcode=1234 to be deleted.

A common workaround for this problem is the use of CSRF tokens. A CSRF token is a string of random characters embedded into requests so that a web application can trust that a request has been received from an expected source as per the normal workflow.

Steps to Implement CSRF

The steps to implement CSRF token protection in PHP are as follows –

- Begin the script by starting a new session.
- Generate a token of random characters. You can use any of the several built-in function that PHP provides for generation of random string. Let use md5() function to obtain the hash value of uniqueid() function that generates a unique randome string.
- Inside the HTML form to be provided for the user to submit the data, include a hidden file with its value as the random token generated in the above step.
- The token can is then validated by the server against the user session after form submission to eliminate malicious requests.
- You can also add another session variable whose value is the current time, and send an expiry time for the validation purpose.

Example

Here is the PHP code that implements CSRF token verification mechanism. The following script generates a token and embeds in a HTML form.

```
<?php
   session_start();
   if(!isset($_SESSION["csrf_token"])) {

      // No token present, generate a new one
      $token = md5(uniqid(rand(), true));
      $_SESSION["csrf_token"] = $token;

   } else {

      // Reuse the token
      $token = $_SESSION["csrf_token"];
   }
?>
<html>
<body>
   <form method="get" action="test.php">
      <input type="text" name="empcode" placeholder="empcode" />
      <input type="hidden" name="csrf_token" value="<?php echo $token;?>" />
      <input type="submit" />
   </form>
</body>
</html>
```

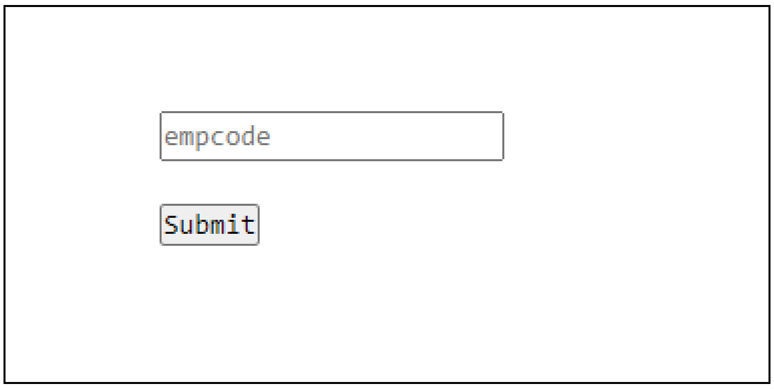
The form is submitted to **"test.php"** script as below –

```
<?php
   session_start();
   echo "hello";
   if ($_GET["csrf_token"] == $_SESSION["csrf_token"]) {

      // Reset token
      echo $_GET["csrf_token"] . "<br>";
      echo $_SESSION["csrf_token"] . "<br>";
      echo "<h3>CSRF token validation successful. Proceed to further action</h3>";
   } else {
      echo "<h3>CSRF token validation failed</h3>";
```

```
    }
);
```

It will produce the following **output** −



To simulate the failure of CSRF validation, open the inspect tool of the browser, edit the value in the hidden field manually and submit the form to see that the tokens don’t match leading to the validation failure.