How to Prevent Cross Site Request Forgery (CSRF) Attacks in PHP Using Synchronizer Token Pattern

This document teaches you how to prevent a Cross Site Request Forgery (CSRF) Attack in a PHP web application by including a CSRF token with each request header cookie or using a hidden form field of CSRF token each form body. A Cross Site Request Forgery (CSRF) Attack exploits a web application vulnerability wherein the victim unintentionally runs a script in their browser that takes advantage of their logged in session to a particular site. CSRF attacks can be performed over GET or POST requests.

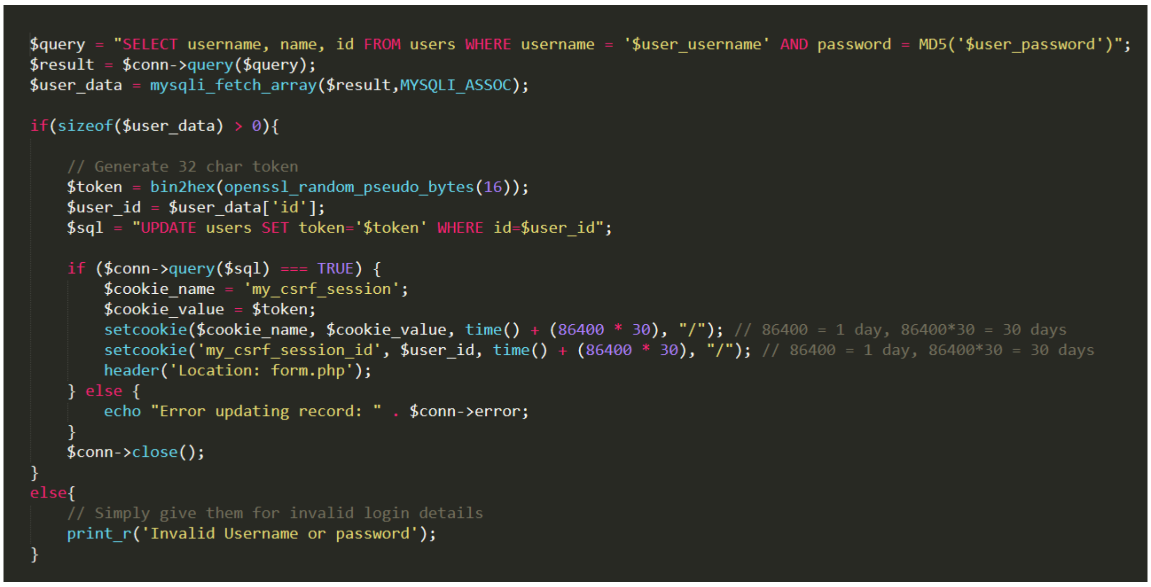
1. Synchronizer Token Pattern

(Including a CSRF token with each request and save in storage.)

generated for each session. We generate the token and then include it in every form as a hidden input. The system then checks if the form is valid by comparing the token with the one stored in the user's session variable(on server database). An attacker will be unable to generate a request without knowing the token value.

1.1 Generate the CSRF token on Login

- Create the check_login.php file.
- Check user existent with database.
- If valid user generate token and set in cookie and also store in database for session validation on any request.



```php
$query = "SELECT username, name, id FROM users WHERE username = '$user_username' AND password = MD5('$user_password')";
$result = $conn->query($query);
$user_data = mysqli_fetch_array($result,MYSQLI_ASSOC);

if(sizeof($user_data) > 0){

    // Generate 32 char token
    $token = bin2hex(openssl_random_pseudo_bytes(16));
    $user_id = $user_data['id'];
    $sql = "UPDATE users SET token='$token' WHERE id=$user_id";

    if ($conn->query($sql) === TRUE) {
        $cookie_name = 'my_csrf_session';
        $cookie_value = $token;
        setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day, 86400*30 = 30 days
        setcookie('my_csrf_session_id', $user_id, time() + (86400 * 30), "/"); // 86400 = 1 day, 86400*30 = 30 days
        header('Location: form.php');
    } else {
        echo "Error updating record: " . $conn->error;
    }
    $conn->close();
}
else{
    // Simply give them for invalid login details
    print_r('Invalid Username or password');
}
```

Here we check username and password with one store in database and if user login successfully then we generate 32 bit auth token and store both place in database with user session and in client side cookie.

If username and password not match with to database record then give error to enter correct details

Also we store user id in cookie for the reference of user record.

Token Generation Logic

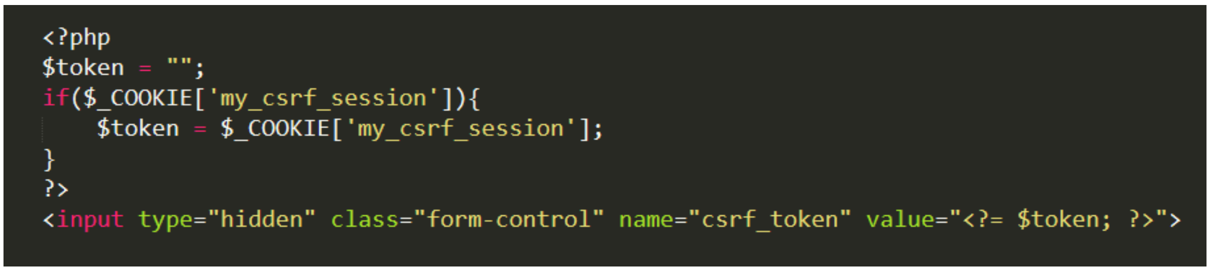
- For creating strong 32 bit chat token we use below syntax to generate CSRF token.

  bin2hex(openssl_random_pseudo_bytes(16))

  bin2hex — Convert binary data into hexadecimal representation. Returns an ASCII string containing the hexadecimal representation of str. The conversion is done byte-wise with the high-nibble first. For more details check here.

  openssl_random_pseudo_bytes — Generate a pseudo-random string of bytes. Generates a string of pseudo-random bytes, with the number of bytes determined by the length parameter. For more details check here.

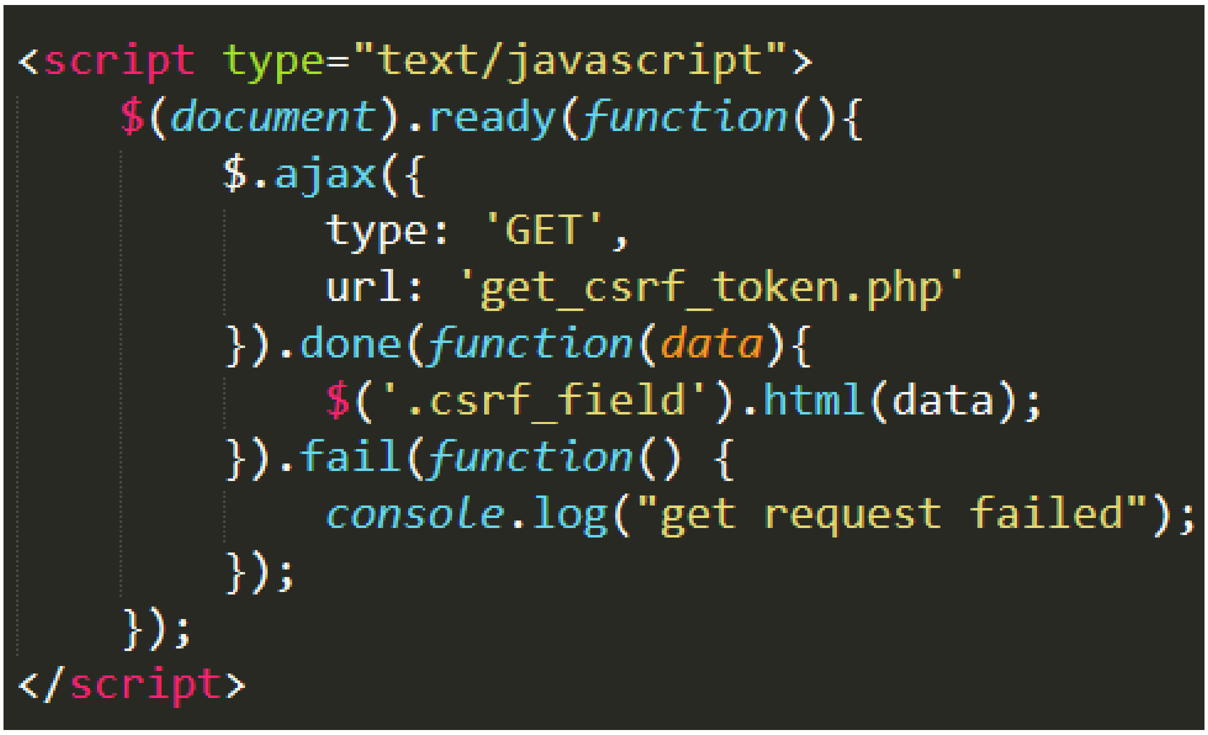
1.2 Create get_csrf_token.php file



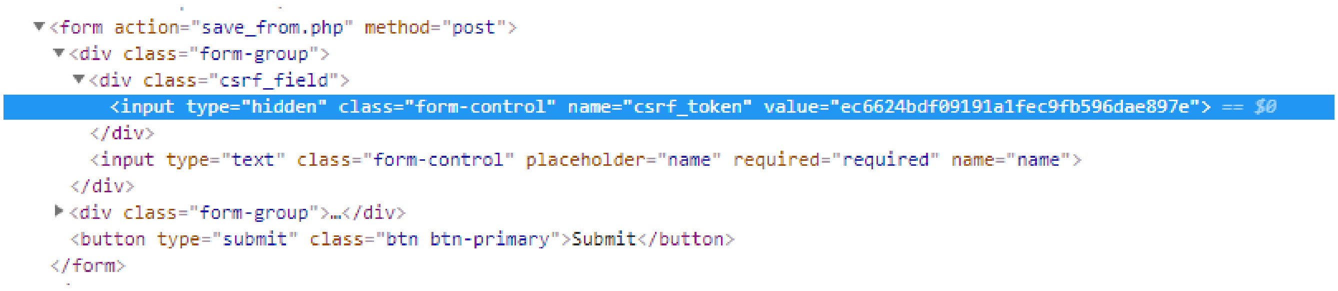
```php
<?php
$token = "";
if($_COOKIE['my_csrf_session']){
    $token = $_COOKIE['my_csrf_session'];
}
?>
<input type="hidden" class="form-control" name="csrf_token" value="<?= $token; ?>">
```

- Here we call ajax and get token store in session cookie to add in form body.
- This response field with token is add in every form submission as hidden filed type.

1.3 Write JQuery to get and update form body with CSRF Token

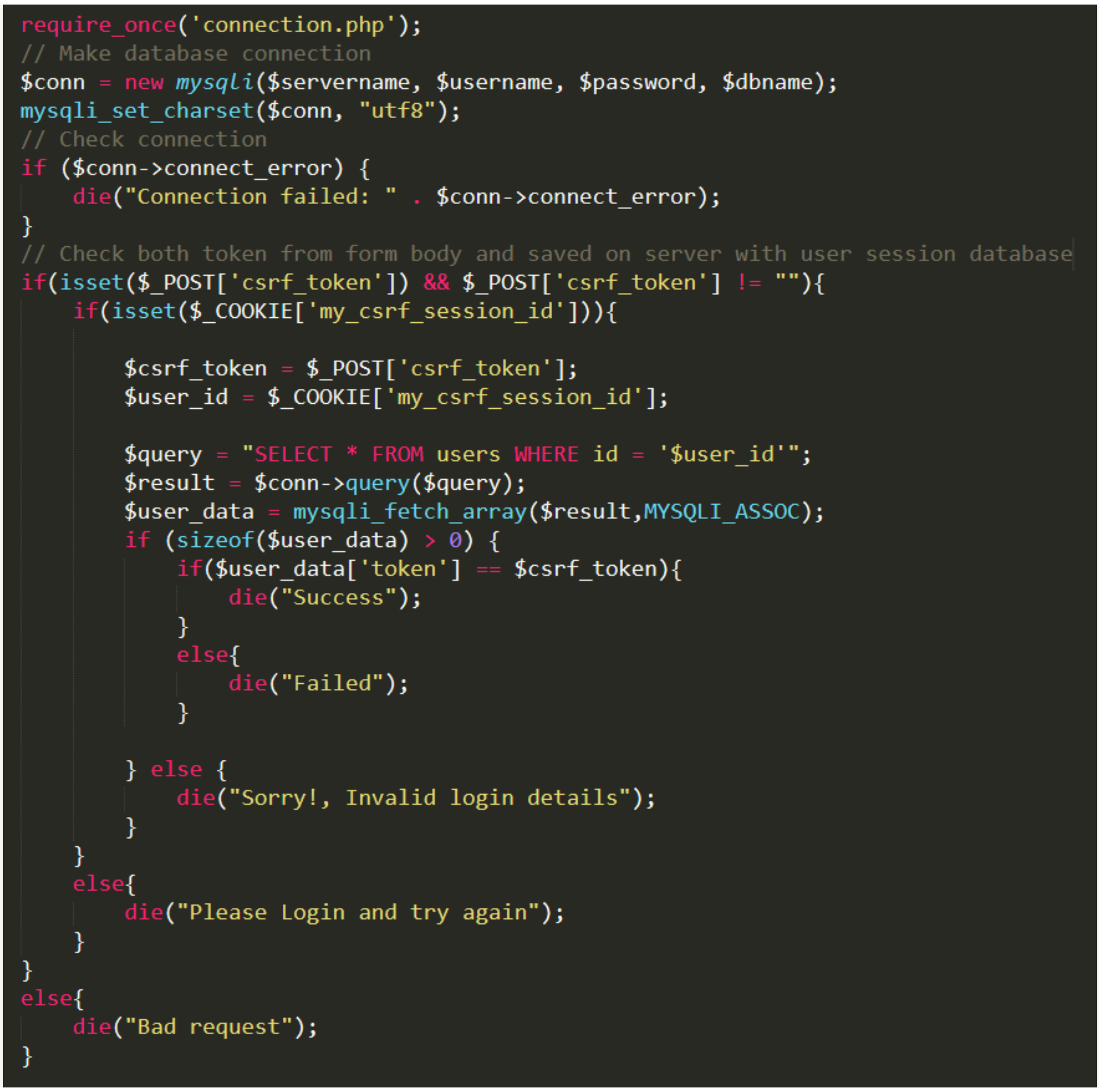


```
<script type="text/javascript">
    $(document).ready(function(){
        $.ajax({
            type: 'GET',
            url: 'get_csrf_token.php'
        }).done(function(data){
            $('.csrf_field').html(data);
        }).fail(function() {
            console.log("get request failed");
        });
    });
</script>
```



```
▼<form action="save_from.php" method="post">
  ▼<div class="form-group">
    ▼<div class="csrf_field">
        <input type="hidden" class="form-control" name="csrf_token" value="ec6624bdf09191a1fec9fb596dae897e"> == $0
      </div>
      <input type="text" class="form-control" placeholder="name" required="required" name="name">
    </div>
  ►<div class="form-group">…</div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
```

1.4 Create file save_form.php which check for valid request



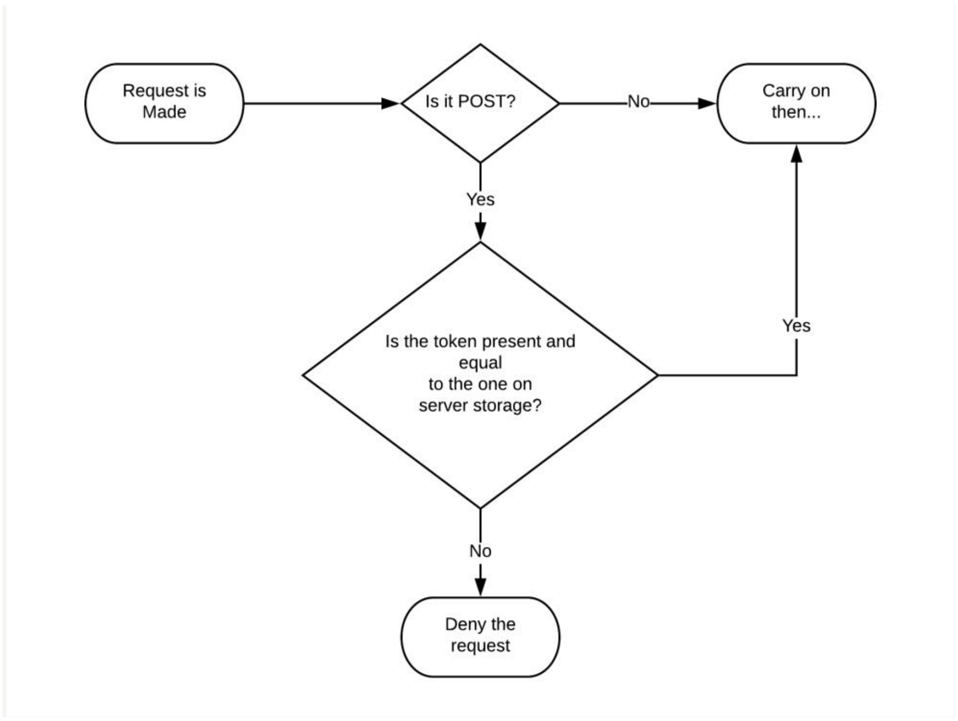
```php
require_once('connection.php');
// Make database connection
$conn = new mysqli($servername, $username, $password, $dbname);
mysqli_set_charset($conn, "utf8");
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
// Check both token from form body and saved on server with user session database
if(isset($_POST['csrf_token']) && $_POST['csrf_token'] != ""){
    if(isset($_COOKIE['my_csrf_session_id'])){

        $csrf_token = $_POST['csrf_token'];
        $user_id = $_COOKIE['my_csrf_session_id'];

        $query = "SELECT * FROM users WHERE id = '$user_id'";
        $result = $conn->query($query);
        $user_data = mysqli_fetch_array($result,MYSQLI_ASSOC);
        if (sizeof($user_data) > 0) {
            if($user_data['token'] == $csrf_token){
                die("Success");
            }
            else{
                die("Failed");
            }

        } else {
            die("Sorry!, Invalid login details");
        }
    }
    else{
        die("Please Login and try again");
    }
}
else{
    die("Bad request");
}
```

- Check request is valid or not.
- First we make connection with database to query token store in database.
- Create a query to database by user id. Get user id from cookie that we store at the time of login request (Point 2.1).
- On getting user data compare the server token with the token that we send in form body as a hidden field.
- If both are same its mean valid request and if not valid then tell the user this is unauthorized request.



Conclusion

Including a CSRF token with each request and save in storage is a second method to validate every request coming to server. In this method we store token server side with user session and check it with the token that include it in every form as a hidden input. It’s also call server side request validation.