How to Prevent Cross Site Request Forgery (CSRF) Attacks in PHP using Double Submit Cookies 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. Double Submit Cookies Pattern

(Including a CSRF token with each request and header cookie.)

This is a unique string that is generated for each session. We generate the token and then include it in every form as a hidden input and also sent with header cookie. The system then checks if the form is valid by comparing the token with the one stored in the request cookie. 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 to session validation.

```php
// Get Data from post request
$user_username = $_POST['username'];
$user_password = $_POST['password'];

// PHP Query to check if user with the given username and password exist or not
$query = "SELECT username, name, id FROM users WHERE username = '$user_username' AND password = MD5('$user_password')";
$result = $conn->query($query);
$user_data = mysqli_fetch_array($result,MYSQLI_ASSOC);
if(sizeof($user_data) > 0){
    $token = bin2hex(openssl_random_pseudo_bytes(16));
    $cookie_name = 'my_csrf_session';
    $cookie_value = $token;
    setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
    setcookie('my_csrf_session_id', $user_id, time() + (86400 * 30), "/");
    header('Location: form.php');
    $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 only in client side cookie.

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

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

```javascript
<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>
```

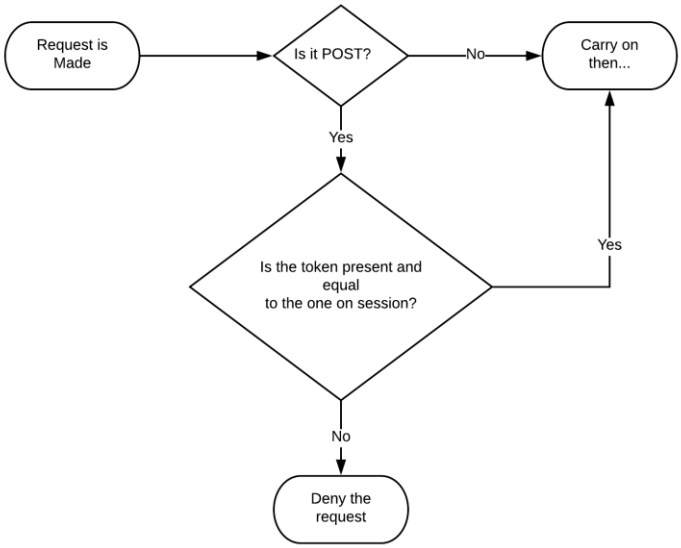
1.4 Create file save_form.php which check for valid request

```php
$headerCookies = explode('; ', getallheaders()['Cookie']);
$cookies = array();
foreach($headerCookies as $itm) {
    list($key, $val) = explode('=', $itm,2);
    $cookies[$key] = $val;
}
$token_cookie = $cookies['my_csrf_session'];

// Check both the token to validate request
if(isset($_POST['csrf_token']) && $_POST['csrf_token'] != ""){
    $csrf_token = $_POST['csrf_token'];

    if ($token_cookie == $csrf_token) {
        die("Success");
    } else {
        die("Failed");
    }
}
else{
    die("Bad request");
}
```

- In this file we check request is valid or not.
- First we get the cookie token variable from request header and then compare it with the token that we get in form body.
- If both are same its mean valid request and make next process and if not valid then go back stop next process and tell the user this is unauthorized request.



Conclusion

Including a CSRF token with each request and header cookie is the first method to validate every request coming to server. Its use browser cookie to store token and check with the token that include it in every form as a hidden input. It’s also call client side request validation.