

CIS 334 – Protecting PHP Applications with CSRF Tokens

Learning Objectives

By the end of this lesson, you should be able to:

- Explain what a CSRF attack is and how it works
 - Understand why PHP forms and authenticated actions are vulnerable
 - Generate and validate CSRF tokens in PHP 8.3
 - Implement CSRF protection using sessions and hidden form fields
 - Apply best practices for securing form submissions
-

Overview

Cross-Site Request Forgery (CSRF) is an attack where a malicious site tricks a logged-in user's browser into sending unwanted requests to your application.

If your site uses cookies or sessions to track logins, the browser automatically includes those credentials — allowing attackers to perform actions as that user.

Example attack: A user logged into `internship-portal.com` visits a malicious page containing a hidden form that submits to `https://internship-portal.com/delete-account.php`. Because their browser includes the valid session cookie, the server believes it's a legitimate request.

How CSRF Tokens Work

[Slide/Diagram suggestion: Browser → Server → Token → Verify]

A **CSRF token** is a unique, unpredictable value associated with a user's session. When the server generates a form, it includes the token as a hidden input. When the form is submitted, PHP checks whether the submitted token matches the one stored in the session.

If the tokens don't match, the request is rejected — preventing forged submissions from external sites.

Step-by-Step Implementation

1 Generating a Token

Add a helper function in a shared file like `csrf.php`:

```
<?php
declare(strict_types=1);

function generateCsrfToken(): string {
```

```
if (empty($_SESSION['csrfToken'])) {  
    $_SESSION['csrfToken'] = bin2hex(random_bytes(32));  
}  
return $_SESSION['csrfToken'];  
}
```

This creates a random 64-character hex string the first time it's needed and stores it in the session.

2] Embedding the Token in a Form

`delete-selection.php`

```
<?php  
session_start();  
require_once 'csrf.php';  
$token = generateCsrfToken();  
?>  
<form method="post" action="delete-selection.php">  
    <input type="hidden" name="csrf_token" value="<?php echo  
htmlspecialchars($token); ?>">  
    <p>Are you sure you want to delete this selection?</p>  
    <p><input type="submit" value="Confirm Delete"></p>  
</form>
```

Each time the page loads, the user receives a fresh CSRF token stored in their session and embedded in the form.

3] Validating the Token

In the same file, or in a separate handler:

```
<?php  
session_start();  
require_once 'csrf.php';  
  
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    $token = $_POST['csrf_token'] ?? '';  
  
    if (!hash_equals($_SESSION['csrfToken'] ?? '', $token)) {  
        http_response_code(403);  
        exit('<h2>Invalid CSRF token. Request denied.</h2>');  
    }  
  
    // Proceed with the intended action  
    echo "<p>Selection deleted successfully.</p>";  
}
```

`hash_equals()` performs a timing-safe comparison that prevents subtle side-channel leaks.

4 Token Lifecycle and Renewal

To minimize risk:

- Regenerate tokens after sensitive actions or login events.
- Optionally unset the token after validation if the form should only be submitted once:

```
unset($_SESSION['csrfToken']);
```

- Use HTTPS so the token isn't visible in transit.
-

🔗 Integration Tips

- **Sessions are required** because the token must persist between requests.
 - **Hidden form fields** are standard, but you can also include tokens in AJAX requests.
 - **Separate function or middleware**: Larger projects often centralize CSRF protection to apply it automatically to all POST requests.
-

🔒 Best Practices in PHP 8.3

- Use `random_bytes()` or `random_int()` for cryptographically secure randomness.
 - Use **strict typing** (`declare(strict_types=1)`) to avoid loose comparisons.
 - Always escape the token with `htmlspecialchars()` when printing it into HTML.
 - Never reuse tokens across users or sessions.
-

⌚ Quick Review

1. What is a CSRF attack and how does it exploit sessions?
 2. Why is storing the token in the session important?
 3. What PHP function should you use to compare tokens securely?
 4. When should you regenerate or unset a CSRF token?
 5. What would happen if your form omitted the CSRF token?
-

✳️ Practice Exercise

1. Add CSRF token protection to one of your form-handling pages (e.g., canceling or deleting a selection).
 2. Verify that the form works when submitted normally.
 3. Try removing or altering the token manually in the browser's DevTools — confirm that the request is rejected.
 4. Optional: Add token regeneration after a successful action for one-time form submission protection.
-

Key Takeaways

- **CSRF attacks** exploit trusted sessions to perform unauthorized actions.
- **Tokens** are unique, session-bound values that verify a request's origin.
- PHP 8.3 provides secure random functions and safe comparison tools for token handling.
- Always validate tokens for any **state-changing requests** (POST, PUT, DELETE).