OCTOBER 4, 2022 / #LARAVEL

# What is Cross-Site Request Forgery (CSRF)? Laravel Web Security Tutorial

Sule-Balogun Olanrewaju Ganiu

In this tutorial, you'll learn about Laravel web security and how to secure your web applications and protect them from Cross-Site Request Forgery, or CSRF attacks.

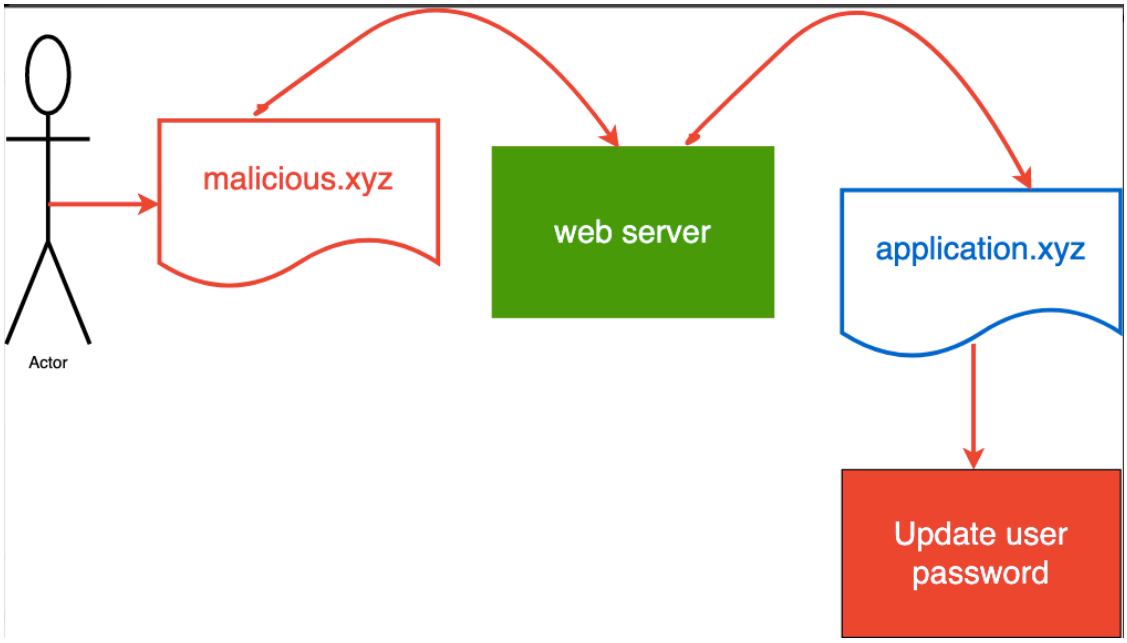
CSRF is a malicious activity that involves an attacker performing actions on behalf of an authenticated user. Fortunately, Laravel provides out-of-the-box measures to prevent this type of vulnerability.

**In this tutorial, you'll learn:**

- What is CSRF?
- How to prevent a CSRF request
- How and where CSRF verification happens

into sending a request through hidden form tags or malicious URLs (images or links) without the user's knowledge.

This attack leads to a change in the state of the user session, data leaks, and attackers can sometimes manipulate end-users data in an application.




CSRF Explainer

The image above illustrates this scenario where an Actor (User) sends a request from **malicious.xyz** through the **webserver** to **application.xyz**. They then realize that their information has been manipulated by **updating their password**.

# How to Prevent CSRF Requests

For each user session, Laravel generates secured tokens that it uses to ensure that the authenticated user is the one requesting the application.

Each time there's a request to modify user information on the server-side (back end) like `POST`, `PUT`, `PATCH`, and `DELETE`, you need to include a `@csrf` in the HTML form request. The `@csrf` is thus a Blade directive used to generate a hidden token validated by the application.

**Blade directive** is the syntax used within the Laravel templating engine called **Blade**. To create a blade file you give it a name – in our case form – followed by the blade extension. This means that the file will have the name `form.blade.php`.

You use the blade file to render views to users on the webpage. There are a couple of default directives or blade shorthand syntaxes you can use. For example, `@if` checks if a condition is met, `@empty` checks if records are not empty, `@auth` checks if a user is authenticated, and so on.

But here we are more interested with the `@csrf` directive. Here's how you use it:

```
<form method="POST" action="{{route('pay')}}">
    @csrf
</form>
```

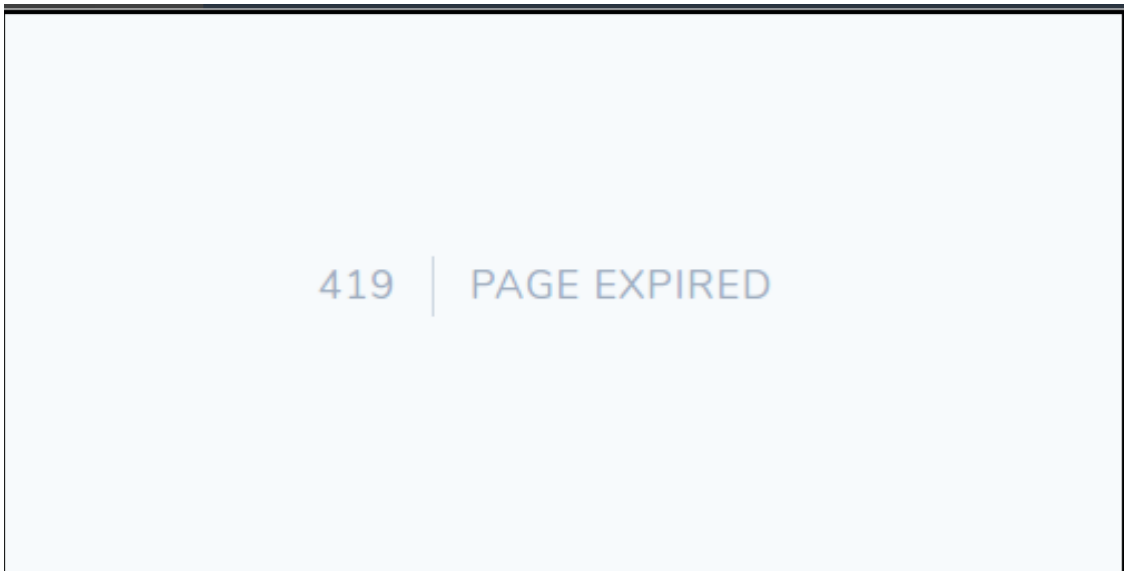
@csrf

Earlier Laravel releases used to look somewhat like this – both work and do the same thing behind the scenes.

```
    <input type="hidden" name="_token" value="{{ csrf_token() }}
</form>
```

hidden csrf_token() in earlier Laravel releases

When the CSRF token is not present in the form request that gets sent or if it appears invalid, Laravel throws an error message "Page Expired" with a status code 419.




Laravel 419 Page Expired

# How and Where CSRF Verification Happens

The `VerifyCsrfToken` middleware handles CSRF verification within the Laravel application. The middleware is registered in the Kernel.php, and found within the application's web route

```
protected $middlewareGroups = [
        'web' => [
            .
            .
            .
            .
            .
             \App\Http\Middleware\VerifyCsrfToken::class,
        ],
    ];
```

The VerifyCsrfToken middleware extends the `Illuminate\Foundation\Http\Middleware\VerifyCsrfToken` class. This means that the CSRF verification is housed within the class.

Let's dive deeper to learn how Laravel handles the CSRF verification.

Within the class, we have the `tokensMatch` function.

```
protected function tokensMatch($request)
    {
        $token = $this->getTokenFromRequest($request);

        return is_string($request->session()->token()) &&
               is_string($token) &&
               hash_equals($request->session()->token(), $token)
    }
```

Determine if the session and input CSRF tokens match.

incoming request attached via a hidden field or the request's header. The token is decrypted and then returned to the token variable.

```
protected function getTokenFromRequest($request)
    {
        $token = $request->input('_token') ?: $request->header('
        if (! $token && $header = $request->header('X-XSRF-TOKEN
            try {
                $token = CookieValuePrefix::remove($this->encryp
            } catch (DecryptException $e) {
                $token = '';
            }
        }
        return $token;
    }
```

Get token from header

2. Cast both request token and session to a string and then use the PHP built-in hash_equals to compare if both strings are equal using the same time. The result of this operation is always a **bool (true) or (false)**.

# Wrapping up

In this article, you have learned about CSRF, how to handle and protect against it, and the behind-the-scenes of how Laravel does the verification.

Happy Coding!

---

**Sule-Balogun Olanrewaju Ganiu**

Experienced software engineer with a passion for developing innovative programs , well versed in technology and writing code to create systems that are reliable and user friendly.

---

Python Absolute Value
%.2f in Python
Java Switch Statement
Git Reset Origin
Change Font with HTML
What is Localhost?
502 Bad Gateway Error
JS Array.includes()
Static Keyword in Java
Compare Dates in JS
String to Number in JS
Python Split String
Python Global Variable
Python enumerate()
Create a Set in Python
What's a Data Analyst Do?
Remove a Dir in Linux
Refresh Page in JavaScript
What is Coding Used For?
Remove Underline from Link
JavaScript String Format
How to Clear an Array in JS
JavaScript String to Date
Capitalize 1st Letter in JS