Course - Laravel Framework

# Chứng thực CSRF

*Giả mạo request trên nhiều trang web (Cross-site request forgeries - CSRF) là một kiểu khai thác độc hại, theo đó các lệnh trái phép được thực hiện thay mặt cho user đã được xác thực. Rất may, Laravel giúp dễ dàng bảo vệ ứng dụng của bạn khỏi các cuộc tấn công giả mạo request (CSRF).*

Tags: CSRF, csrf protection, laravel

## Giải thích

### Lỗ hổng bảo mật

Trong trường hợp bạn không am hiểu nhiều về giả mạo request (CSRF), thì hãy xem ví dụ về cách mà lỗ hổng này có thể được khai thác. Hãy tưởng tượng ứng dụng của bạn có một route */user/email* chạy với phương thức POST request, mục đích là để thay đổi địa chỉ email của người dùng đã xác thực trước đó. Nếu như vậy, thì trong route này sẽ có một field là email để nhập email chứa địa chỉ email mà người dùng muốn sử dụng.

Nếu không có chứng thực CSRF, một trang web độc hại có thể tạo ra một HTML form, và trỏ đến đường dẫn */user/email* của ứng dụng của bạn, sau đó nó đệ trình địa chỉ email của kẻ xâm hại:

```
<form action="https://your-application.com/user/email" method="POST">
  <input type="email" value="malicious-email@example.com">
</form>

<script>
  document.forms[0].submit();
</script>
```

Nếu trang web độc hại tự động gửi form ở trên khi trang được tải, kẻ xâm hại chỉ cần dụ một user nào đó của bạn chuyển sang truy cập trang web của kẻ xâm hại và địa chỉ email của kẻ xâm hại sẽ được thay đổi vào trong ứng dụng của bạn. Để ngăn chặn lỗ hổng này, chúng tôi sẽ kiểm tra mọi request `POST`, `PUT`, `PATCH` hoặc `DELETE` được gửi đến với một mã SessionID bí mật mà ứng dụng độc hại khác không thể truy cập được.

## Ngăn chặn tấn công CSRF

Laravel tự động tạo mã token CSRF cho user từ lúc khởi động trình duyệt cho đến khi tắt trình duyệt và do ứng dụng quản lý. Mã token này được sử dụng để xác minh user này đã được xác thực và chính xác user này là người đã gửi đến các request cho ứng dụng. Vì mã token này được lưu trữ trong session của user và sẽ thay đổi mỗi khi user vào lại trình duyệt, nên các ứng dụng độc hại sẽ không thể truy cập chính xác vào nó được nữa.

Mã token CSRF hiện tại có thể được truy cập bằng đối tượng session của request, hoặt bằng hàm `csrf_token`:

```php
use Illuminate\Http\Request;

Route::get('/token', function (Request $request) {
  $token = $request->session()->token();

  $token = csrf_token();

  // ...
});
```

Ở bất cứ thời điểm nào, khi bạn tạo form HTML với các phương thức `POST`, `PUT`, `PATCH` hoặc `DELETE` trong ứng dụng của mình, thì bạn cũng nên đưa vào một field CSRF `_token` được ẩn trong HTML form để middleware chứng thực CSRF có thể xác thực được request gửi đến. Để thuận tiện hơn, bạn có thể sử dụng directive `@csrf` của *Blade* để tạo field CSRF token:

```php
<form method="POST" action="/profile">
  @csrf

  <!-- Equivalent to... -->
  <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

Middleware `App\Http\Middleware\VerifyCsrfToken`, được mặc định đưa vào trong nhóm *web middleware group*, sẽ tự động xác minh mã token trong request có khớp với mã token được lưu trữ trong session của ứng dụng. Khi hai mã token này khớp với nhau, thì ứng dụng sẽ biết rằng user đã xác thực này là người gửi đến các request cho ứng dụng.

### Token CSRF với các SPA

Khi bạn xây dựng một SPA và sử dụng Laravel làm chương trình API-backend, thì bạn nên tham khảo tài liệu package Laravel Sanctum để biết thông tin về cách xác thực với các API của bạn và ngăn ngừa các lỗ hổng CSRF.

### Loại bỏ các URI ra khỏi chứng thực CSRF

Có những lúc bạn sẽ muốn loại bỏ một vài URI ra khỏi chứng thực CSRF. Ví dụ: nếu bạn

đang sử dụng Stripe để xử lý thanh toán và đang sử dụng hệ thống webhook của họ, bạn sẽ cần phải loại trừ route xử lý webhook của Stripe ra khỏi chứng thực CSRF vì Stripe sẽ không biết token CSRF nào sẽ được gửi đến cho các route của bạn.

Thường thì bạn nên đặt các kiểu route này bên ngoài nhóm *web middleware group* mà `App\Providers\RouteServiceProvider` đã áp dụng cho tất cả các route trong tập tin route */web.php*. Tuy nhiên, bạn cũng có thể loại trừ các route này bằng cách thêm các URI của chúng vào thuộc tính `$except` của middleware `VerifyCsrfToken`:

```php
<?php

namespace App\Http\Middleware;

use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;

class VerifyCsrfToken extends Middleware
{
  /**
   * The URIs that should be excluded from CSRF verification.
   *
   * @var array
   */
  protected $except = [
    'stripe/*',
    'http://example.com/foo/bar',
    'http://example.com/foo/*',
  ];
}
```

> Để thuận tiện, Laravel sẽ tự động vô hiệu chứng thực CSRF khi chạy thử nghiệm Tham khảo.

## X-CSRF-TOKEN

Ngoài việc kiểm tra mã token CSRF dưới dạng tham số POST, middleware `App\Http\Middleware\VerifyCsrfToken` cũng sẽ kiểm tra tham số header `X-CSRF-TOKEN` trên request. Ví dụ: bạn có thể lưu trữ mã token trong meta tag của HTML như sau:

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

Sau đó, bạn có thể hướng dẫn một thư viện nào đó như jQuery chẳng hạn, tự động thêm mã token vào tất cả các *request header*. Việc này sẽ cung cấp chứng thực CSRF một cách đơn giản và thuận tiện cho các ứng dụng AJAX sử dụng javascript:

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

## X-XSRF-TOKEN

Laravel lưu trữ mã token CSRF hiện tại trong cookie `XSRF-TOKEN` đã được mã hóa và sẽ đi kèm với mỗi response của ứng dụng. Bạn có thể sử dụng giá trị cookie để đặt *request header* `X-XSRF-TOKEN`.

Cookie này chủ yếu được gửi đi để thuận tiện cho một nhà phát triển vì một số framework và thư viện JavaScript, như Angular và Axios, sẽ tự động đặt giá trị của nó vào trong header `X-XSRF-TOKEN` trên các request có cùng nguồn gốc (same-origin - cùng nguồn gốc hoặt cùng host).

> Mặc định, tập tin *resource/js/bootstrap.js* đưa vào thư viện *Axios HTTP* và sẽ tự động gửi header `X-XSRF-TOKEN` cho bạn.