Securing Laravel APIs with Sanctum: Complete Guide

APIs power modern applications — from single-page apps (SPAs) to mobile apps and even IoT devices. But APIs also open doors to attackers if not secured properly. That's where **Laravel Sanctum** comes in. It's a lightweight package that makes it easy to protect your Laravel APIs with tokens, cookies, and middleware.

In this tutorial, you'll learn how to **secure your Laravel 12 APIs with Sanctum**. We'll go step by step: installing Sanctum, configuring it, issuing tokens, protecting routes, and using it with SPAs and mobile clients. Along the way, we'll explain key concepts so you understand not just how, but why it works.

1 – What is Laravel Sanctum?

Sanctum is Laravel's recommended way to authenticate APIs. It supports two main use cases:

- **API tokens:** Ideal for mobile apps or external services. Each user can create and manage personal access tokens.
- **SPA authentication:** Uses Laravel's session cookies with CSRF protection to secure single-page apps built with React, Vue, etc.

Unlike Passport (Laravel's OAuth2 package), Sanctum is simple, lightweight, and perfect for most apps that don't need full OAuth2 complexity.

2 – Install and Configure Sanctum

First, install Sanctum via Composer:

```
composer require laravel/sanctumCode language: Bash (bash)
```

Publish the Sanctum configuration and migration files:

```
php artisan vendor:publish --
provider="Laravel\Sanctum\SanctumServiceProvider"
php artisan migrateCode language: Bash (bash)
```

This creates a `personal_access_tokens` table in your database, which will store API tokens for users.

Now, add Sanctum's middleware in `app/Http/Kernel.php` under the `api` group:

```
// app/Http/Kernel.php
protected $middlewareGroups = [
    'api' => [
\Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::cl
ass,
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];Code language: PHP (php)
```

3 – Enable Token Abilities in the User Model

To let users issue and manage tokens, add the `HasApiTokens` trait to your `User` model:

```
// app/Models/User.php
namespace App\Models;
```

```
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens;

    protected $fillable = ['name','email','password'];
}
```
Code language: PHP (php)

Now each user can create personal access tokens with abilities (like `create-posts`, `delete-posts`).

4 – Issue and Use API Tokens

Let’s create an endpoint where a user can log in and receive a token:

```
// routes/api.php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use App\Models\User;

Route::post('/login', function (Request $request) {
    $request->validate([
        'email' => 'required|email',
        'password' => 'required',
    ]);

    $user = User::where('email',$request->email)->first();

    if (! $user || ! Hash::check($request->password, $user->password))
{
        return response()->json(['message' => 'Invalid credentials'],
401);
    }

    $token = $user->createToken('api-token',['create-posts','delete-posts'])->plainTextToken;

    return response()->json(['token' => $token]);
});
```
Code language: PHP (php)

Now, the user can authenticate by including the token in the `Authorization` header:

```
curl -H "Authorization: Bearer {TOKEN}" http://localhost:8000/api/user
```
Code language: Bash (bash)

Tokens can also be scoped. For example, you could give one token `create-posts` and another `read-posts`, letting you restrict what each client can do.

5 - Protect API Routes with Sanctum Middleware

To secure your API endpoints, apply the `auth:sanctum` middleware:

```
// routes/api.php
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});
```
Code language: PHP (php)

Now, only requests with a valid Sanctum token will succeed. Invalid or missing tokens return `401 Unauthorized`.

6 – Sanctum with SPAs

If you're building a Vue, React, or Angular SPA that talks to your Laravel backend, Sanctum provides cookie-based authentication. Here's how it works:

- The SPA makes a login request to `/login` with credentials.
- Laravel responds with a session cookie.
- Subsequent requests include this cookie, protected by CSRF tokens.

To enable this, add your SPA's domain to `SANCTUM_STATEFUL_DOMAINS` in `.env`:

```bash
# .env
SANCTUM_STATEFUL_DOMAINS=localhost:3000,myapp.com
```
Code language: Bash (bash)

This lets Sanctum recognize stateful requests from your frontend, making cookie-based auth work seamlessly.

7 – Common Errors & Fixes

- **401 Unauthorized:** Make sure the token is included in the `Authorization` header or the SPA is listed in `SANCTUM_STATEFUL_DOMAINS`.
- **CSRF mismatch:** When using SPA auth, ensure you include the `X-XSRF-TOKEN` header with requests.
- **Token not found:** Run `php artisan migrate` to create the `personal_access_tokens` table.

Wrapping Up

You’ve learned how to **secure Laravel APIs with Sanctum**. We installed and configured Sanctum, issued API tokens, protected routes, and even integrated it with SPAs. This lightweight solution is perfect for most apps, providing strong security without the complexity of OAuth2.

What’s Next

- How to Add JWT Authentication to Laravel APIs — explore an alternative token-based approach.
- Building a Mobile App Backend with Laravel 12 API — use Sanctum in a mobile environment.
- How to Build a REST API with Laravel 12 & Sanctum — complete implementation guide.