1v0 Ship v1.0 Faster

How to Build a Secure File Upload API in Laravel

How to Build a Secure File Upload API in Laravel

File uploads are a common attack vector. A secure API must validate file size and MIME type, store files outside the public web root, generate safe filenames, optionally virus-scan, and expose only signed URLs for downloads. In this guide you'll create a hardened upload API using Laravel's validation, Storage, policies, and (optionally) a queue-powered malware scan.

1 – Configure Storage & App Limits

We'll store files on the `local` disk (outside `public/`) and expose them via signed routes. Also ensure PHP upload limits are sane for your use case.

```
php artisan storage:link   # only if you plan to serve some files via 'public' diskCode language: Bash (bash)
```

`storage:link` is not required for the `local` disk. If you later move to the `public` disk (e.g., for images), the symlink lets the web server reach `storage/app/public`. For private downloads we'll stream files via a controller instead of direct access.

.env (review these) UPLOAD_MAX_FILESIZE=5M POST_MAX_SIZE=6M FILESYSTEM_DISK=local

Match `UPLOAD_MAX_FILESIZE` and `POST_MAX_SIZE` to your needs (and your server's `php.ini`). Using the `local` disk keeps files private by default.

2 – Migration & Model for Uploaded Files

We'll track uploaded files in a table with original name, stored path, size, detected MIME, and an owner. We'll also keep a SHA-256 hash for deduplication and security audits.

```
// database/migrations/2025_08_27_000000_create_uploads_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('uploads', function (Blueprint $table) {
            $table->id();
$table->foreignId('user_id')->constrained()->cascadeOnDelete();
            $table->string('original_name');
            $table->string('disk')->default('local');
            $table->string('path');                // e.g.
uploads/2025/08/xyz.pdf
            $table->string('mime', 190);           // detected server-
side
            $table->unsignedBigInteger('size');    // bytes
            $table->string('sha256', 64)->index(); // content hash
            $table->boolean('is_safe')->default(true); // set false
during scan if suspect
            $table->timestamps();
        });
    }
    public function down(): void {
        Schema::dropIfExists('uploads');
    }
};Code language: PHP (php)
```

This schema captures essential metadata for each upload. The `sha256` allows duplicate detection and forensic checks. `is_safe` can be toggled by a scanner job to quarantine suspicious files.

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

use Illuminate\Database\Eloquent\Model;

class Upload extends Model
{
    protected $fillable = [
'user_id','original_name','disk','path','mime','size','sha256','is_saf
e'
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}Code language: PHP (php)
```

The model is straightforward, linking each upload to its owner. We'll use this for authorization and listing endpoints later.

3 – Validation Rules for Secure Uploads

Validate both size and type. Prefer `mimetypes` for server-side MIME detection and restrict to a small allow-list.

```
/**
 * Example rules for PDFs and images only (5 MB max).
 */
```

```
$rules = [
    'file' => [
        'required',
        'file',
        'max:5120', // KB => 5 MB
        'mimetypes:application/pdf,image/jpeg,image/png'
    ],
];Code language: PHP (php)
```

`max` is in kilobytes. Using `mimetypes` ensures the server-inspected MIME matches your allow-list, which is safer than extension-only checks. Adjust the list to your needs.

4 – Upload Controller (Store Privately + Hash)

This controller validates the upload, stores it with a safe path, computes a hash, and records metadata. We'll protect the route with `auth:sanctum`.

```
// app/Http/Controllers/UploadApiController.php
namespace App\Http\Controllers;

use App\Models\Upload;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Str;

class UploadApiController extends Controller
{
    public function store(Request $request)
    {
        $data = $request->validate([
            'file' =>
['required','file','max:5120','mimetypes:application/pdf,image/jpeg,im
age/png'],
```

```
        ]);

        $file = $data['file'];

        // Generate a safe path (no user-supplied filename)
        $folder = 'uploads/'.now()->format('Y/m');
        $filename = Str::uuid().'.'.$file->guessExtension(); //
guessExtension() based on MIME
        $path = $file->storeAs($folder, $filename, disk: 'local'); //
private by default

        // Read contents for hashing (small/medium files). For very
large files, stream hash.
        $sha256 = hash_file('sha256',
Storage::disk('local')->path($path));

        $upload = Upload::create([
            'user_id'       => $request->user()->id,
            'original_name' => $file->getClientOriginalName(),
            'disk'          => 'local',
            'path'          => $path,
            'mime'          => $file->getMimeType(),
            'size'          => $file->getSize(),
            'sha256'        => $sha256,
            'is_safe'       => true, // or false until a scanner job
verifies
        ]);

        return response()->json([
            'id' => $upload->id,
            'message' => 'Uploaded successfully',
        ], 201);
    }
}Code language: PHP (php)
```

Files are stored under `storage/app/uploads/YYYY/MM` with a UUID filename to avoid collisions and path traversal issues. The DB row ties the file to the user and includes a content hash for later integrity checks or deduplication.

5 – Secure Download via Signed Route

Serve private files by streaming them from storage only if the signed URL is valid and the user is authorized.

```
// app/Http/Controllers/DownloadController.php
namespace App\Http\Controllers;

use App\Models\Upload;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Gate;
use Illuminate\Support\Facades\Storage;

class DownloadController extends Controller
{
    public function show(Request $request, Upload $upload)
    {
        if (! $request->hasValidSignature()) {
            abort(403);
        }

        // Optional: owner-only access (or replace with a policy)
        if ($request->user()?->id !== $upload->user_id) {
            abort(403);
        }

        if (! $upload->is_safe) {
            abort(423, 'File is quarantined.');
        }

        return Storage::disk($upload->disk)->download($upload->path,
$upload->original_name);
```

```
    }
}Code language: PHP (php)
```

`hasValidSignature()` ensures the URL wasn't tampered with. We also restrict downloads to the owner and block quarantined files. The Storage `download()` helper streams the file with correct headers.

```
// Generate a temporary signed URL (e.g., in a controller or resource)
use Illuminate\Support\Facades\URL;

$signedUrl = URL::temporarySignedRoute(
    'uploads.show',
    now()->addMinutes(10),
    ['upload' => $upload->id]
);Code language: PHP (php)
```

A temporary signed URL expires after 10 minutes, reducing link leakage risks. Return this from an API that lists a user's files when they request a download.

6 – Routes (Protected Upload, Signed Download)

Separate API (token-protected) for uploads from signed web routes for downloads. You can also expose a "list my files" endpoint.

```
// routes/api.php
use App\Http\Controllers\UploadApiController;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:sanctum')->group(function () {
    Route::post('/uploads', [UploadApiController::class,
'store'])->name('api.uploads.store');
});Code language: PHP (php)
```

The upload route is protected by Sanctum. Only authenticated clients can POST files. Add rate limiting via the `throttle` middleware if needed for abuse control.

```
// routes/web.php
use App\Http\Controllers\DownloadController;
use Illuminate\Support\Facades\Route;

Route::get('/uploads/{upload}', [DownloadController::class, 'show'])
    ->middleware(['signed','auth'])
    ->name('uploads.show');Code language: PHP (php)
```

The download route requires a valid signature and an authenticated session. If you need token-based downloads instead, put it under `routes/api.php` with `auth:sanctum` and still use signed URLs.

7 – Optional: Antivirus Scan with a Queue Job

For higher security, queue a malware scan (e.g., ClamAV) right after upload and quarantine the file until it's cleared.

```
// app/Jobs/ScanUpload.php
namespace App\Jobs;

use App\Models\Upload;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Support\Facades\Storage;

class ScanUpload implements ShouldQueue
{
    use Queueable;

    public function __construct(public int $uploadId) {}
```

```
    public function handle(): void
    {
        $upload = Upload::find($this->uploadId);
        if (! $upload) return;

        $path = Storage::disk($upload->disk)->path($upload->path);

        // Pseudocode: replace with actual scanner integration
        // $clean = ClamAV::scan($path);
        $clean = true;

        $upload->update(['is_safe' => (bool) $clean]);
    }
}Code language: PHP (php)
```

This job loads the file from disk and runs a scan. If flagged, set `is_safe` to false so downloads are blocked. Wire it in after the upload is created: `dispatch(new ScanUpload($upload->id));`.

8 – UI: Minimal Upload Form with Progress

Here's a tiny Blade page that uploads to the API using Axios, shows a progress bar, and prints the result. Paste a Bearer token from your Sanctum login first.

```
<!-- resources/views/uploads/test.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>Secure Upload Test</h1>

  <div class="mb-3">
    <label class="form-label">Bearer Token</label>
```

```
    <input id="token" type="text" class="form-control"
placeholder="Paste your token">
  </div>

  <div class="mb-3">
    <input id="file" type="file" class="form-control">
  </div>

  <div class="progress mb-3" style="height: 8px;">
    <div id="bar" class="progress-bar" role="progressbar"
style="width: 0%;"></div>
  </div>

  <button class="btn btn-theme" onclick="upload()">Upload</button>

  <pre id="result" class="mt-3"></pre>
</div>

<script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
function upload() {
  const token = document.getElementById('token').value;
  const file  = document.getElementById('file').files[0];
  if (!file) { alert('Choose a file'); return; }

  const form = new FormData();
  form.append('file', file);

  axios.post('/api/uploads', form, {
    headers: { Authorization: `Bearer ${token}` },
    onUploadProgress: (evt) => {
      if (evt.total) {
        const percent = Math.round((evt.loaded / evt.total) * 100);
        document.getElementById('bar').style.width = percent + '%';
      }
    }
  }).then(res => {
```

```
    document.getElementById('result').textContent =
JSON.stringify(res.data,null,2);
  }).catch(err => {
    const msg = err.response ?
JSON.stringify(err.response.data,null,2) : err.message;
    document.getElementById('result').textContent = msg;
  });
}
</script>
@endsectionCode language: HTML, XML (xml)
```

The UI posts the file to `/api/uploads` with a Bearer token. The progress bar updates as the browser streams the file. On success you'll get the upload ID to fetch a signed download link later.

Wrapping Up

You built a secure upload pipeline: strict validation, private storage, hashed contents, owner-based authorization, signed downloads, and an optional antivirus scan with a queue job. This approach prevents unsafe files from being served publicly and gives you full control over who can access which file and for how long.

What's Next

- Using Laravel Passport for Advanced API Authentication
- How to Add JWT Authentication to Laravel APIs
- Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)