AgroSenso Lite

Sistema de gestión agrícola con demostración de vulnerabilidades OWASP Top 10 2021

Tabla de Contenidos

1. Descripción del Proyecto
2. Requisitos del Sistema
3. Instalación
4. Vulnerabilidades Implementadas
5. Estructura del Proyecto
6. Uso y Demostración
7. Créditos

Descripción del Proyecto

AgroSenso Lite es un sistema de gestión agrícola desarrollado en Laravel que permite:

- Dashboard con métricas agrícolas
- Gestión de parcelas (CRUD completo)
- Catálogo de productos agrícolas
- Registro de lecturas de sensores (simuladas)
- Sistema de autenticación

Este proyecto fue creado con **propósitos educativos** para demostrar las **6 vulnerabilidades más críticas** del OWASP Top 10 2021 en una aplicación web real.

Requisitos del Sistema

Antes de instalar, asegúrate de tener:

- **PHP** >= 8.2
- **Composer** >= 2.6
- **MySQL** >= 8.0
- **Node.js** >= 18 (opcional, para assets)
- **Git**

Verificar instalaciones:

```
php --version
composer --version
mysql --version
```

Instalación

1 Clonar el repositorio

```
git clone https://github.com/AndrewMontero/Proyecto-Seguridad-TI.git
cd Proyecto-Seguridad-TI
```

2 Instalar dependencias de PHP

```
composer install
```

3 Configurar variables de entorno

Copia el archivo `.env.example` y renómbralo a `.env`:

```
cp .env.example .env
```

O en Windows:

```
copy .env.example .env
```

Edita el archivo `.env` con tus credenciales:

```
APP_NAME=AgroSenso
APP_ENV=local
APP_DEBUG=true
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=agrosenso_db
DB_USERNAME=root
DB_PASSWORD=tu_password
```

4 Generar clave de aplicación

```
php artisan key:generate
```

5 Crear base de datos

Accede a MySQL:

```
mysql -u root -p
```

Ejecuta:

```
CREATE DATABASE agrosenso_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
EXIT;
```

6 Ejecutar migraciones

```
php artisan migrate
```

7 Poblar base de datos con datos de prueba

```
php artisan db:seed --class=DemoSeeder
```

Esto creará:

- ☑ 3 usuarios de prueba
- ☑ 6 parcelas
- ☑ 6 productos

8 Iniciar servidor de desarrollo

```
php artisan serve
```

Accede a: **http://127.0.0.1:8000**

Credenciales de Acceso

Usuarios de prueba:

Email	Password	Rol
josue@gmail.com	admin123	Usuario
maria@gmail.com	admin123	Usuario

Email	Password	Rol
carlos@gmail.com	admin123	Usuario

Vulnerabilidades Implementadas

Este proyecto incluye **6 vulnerabilidades críticas** del OWASP Top 10 2021:

---

1. **SQL Injection (SQLi)** - A03:2021

Ubicación: `app/Http/Controllers/ProductoController.php` - Línea 15

Descripción:
La búsqueda de productos concatena directamente la entrada del usuario en la consulta SQL sin usar parámetros preparados.

Código vulnerable:

```
$busqueda = $request->input('buscar', '');
$productos = DB::select("SELECT * FROM productos WHERE nombre LIKE
'%$busqueda%'");
```

Impacto:

- ⚠ Acceso no autorizado a toda la base de datos
- ⚠ Extracción de información sensible
- ⚠ Modificación o eliminación de datos
- ⚠ En casos extremos: compromiso total del servidor

Cómo explotarlo:

1. Ve a `/productos`
2. En el campo de búsqueda, ingresa: `' OR '1'='1`
3. Resultado: Muestra todos los productos sin filtro

Payload de prueba:

```
' OR '1'='1
' UNION SELECT 1,2,3,4,5--
'; DROP TABLE productos; --
```

Solución correcta:

```
// Usar parámetros preparados
$productos = DB::select("SELECT * FROM productos WHERE nombre LIKE ?",
["%$busqueda%"]);
```

```
// O mejor aún, usar Eloquent:
$productos = Producto::where('nombre', 'like', "%$busqueda%")->get();
```

---

2. **Cross-Site Scripting (XSS)** - A03:2021

Ubicación: `app/Http/Controllers/ProductoController.php` - Línea 28

Descripción:
Los datos ingresados no se validan ni sanitizan antes de almacenarse, y se renderizan sin escapar en las vistas.

Código vulnerable:

```
DB::insert("INSERT INTO productos (nombre, descripcion, precio, created_at,
updated_at)
            VALUES (?, ?, ?, NOW(), NOW())",
            [$request->nombre, $request->descripcion, $request->precio]
);
```

Vista vulnerable en `resources/views/productos/index.blade.php`:

```
{!! $producto->descripcion !!}  <!--  NO escapa HTML -->
```

Impacto:

- Robo de cookies y sesiones
- Redireccionamiento a sitios maliciosos
- Modificación del contenido de la página
- Captura de credenciales (keylogging)

Cómo explotarlo:

1. Ve a `/productos/crear`
2. En el campo **Descripción**, ingresa:

```
<script>alert('XSS Vulnerable!')</script>
```

3. Guarda el producto
4. Al ver la lista de productos, el script se ejecuta

Payloads de prueba:

```
<script>alert(document.cookie)</script>
<img src=x onerror="alert('XSS')">
```

```
<svg onload="alert('XSS')">
<iframe src="javascript:alert('XSS')">
```

Solución correcta:

```
// En la vista, usar {{ }} en lugar de {!! !!}
{{ $producto->descripcion }}  <!-- ☑ Escapa automáticamente -->
```

---

3. **Broken Access Control (IDOR)** - A01:2021

Ubicación: `app/Http/Controllers/ParcelController.php` - Líneas 26 y 39

Descripción:
Los métodos `edit()` y `update()` NO verifican que el usuario sea el propietario de la parcela antes de permitir acceso.

Código vulnerable:

```
public function edit($id)
{
    $parcel = Parcel::findOrFail($id);
    //  VULNERABLE: NO verifica ownership
    return view('parcels.edit', compact('parcel'));
}

public function update(Request $request, $id)
{
    $parcel = Parcel::findOrFail($id);
    //  VULNERABLE: NO hay verificación de propietario ni autorización
    $parcel->name = $request->input('name');
    // ... actualiza sin verificar
}
```

Impacto:

- Un usuario puede editar parcelas de otros usuarios
- Acceso no autorizado a información privada
- Modificación o eliminación de datos ajenos
- Violación de la privacidad

Cómo explotarlo:

1. Inicia sesión como `josue@gmail.com`
2. Crea o identifica una parcela (ej: ID=1)
3. Cierra sesión e inicia con `maria@gmail.com`
4. Cambia la URL manualmente a: `/parcels/1/edit`

5. ¡Puedes editar la parcela de Josue sin ser el dueño!

Solución correcta:

```
public function edit($id)
{
    $parcel = Parcel::findOrFail($id);

    //  Verificar propiedad
    if ($parcel->user_id !== Auth::id()) {
        abort(403, 'No autorizado');
    }

    return view('parcels.edit', compact('parcel'));
}
```

---

4. **Broken Authentication** - A07:2021

Ubicación: `routes/web.php` - Líneas 158-170

Descripción:
El sistema de autenticación permite:

- ✗ Contraseñas débiles sin requisitos de complejidad
- ✗ Tokens de reset predecibles
- ✗ Sin límite de intentos de login (fuerza bruta)
- ✗ Mensajes que revelan si un usuario existe

Código vulnerable:

```
Route::post('/demo-login', function (Request $r) {
    $email = $r->post('email');
    $password = $r->post('password');

    $user = \App\Models\User::where('email', $email)->first();

    //  Acepta contraseña débil "admin123" para cualquier usuario
    if ($user && $password === 'admin123') {
        Auth::login($user);
        return redirect('/dashboard');
    }

    return back()->with('error', 'Invalid demo credentials');
});
```

Impacto:

- Ataques de fuerza bruta sin restricción

- Enumeración de usuarios válidos
- Adivinación de tokens de reset
- Cuentas vulnerables a takeover

Cómo explotarlo:

Ataque 1: Contraseña débil universal

- Cualquier usuario puede acceder con `admin123`
- No hay requisitos de complejidad de contraseña

Ataque 2: Token predecible

```
// El token se genera como: md5(email + floor(time() / 600))
// Ruta: /demo-reset-request
POST email=test@test.com
// Resultado: Token predecible que puede ser adivinado
```

Ataque 3: Fuerza bruta sin límite

```
# Intentos ilimitados sin bloqueo
curl -X POST http://127.0.0.1:8000/demo-login -d
"email=josue@gmail.com&password=test1"
curl -X POST http://127.0.0.1:8000/demo-login -d
"email=josue@gmail.com&password=test2"
# ... sin límite de intentos
```

Solución correcta:

```
// 1. Validar complejidad de contraseña
'password' => 'required|min:8|regex:/[A-Z]/|regex:/[0-9]/'

// 2. Usar Hash::make() para contraseñas
Hash::make($password)

// 3. Rate limiting
RateLimiter::hit('login:' . $request->ip(), 60);

// 4. Mensajes genéricos
return back()->with('error', 'Credenciales incorrectas');
```

---

5. **Server-Side Request Forgery (SSRF)** - A10:2021

Ubicación: `routes/web.php` - Líneas 130-145

Descripción:
Endpoint `/fetch?url=` que permite hacer peticiones HTTP a cualquier URL sin validación.

Código vulnerable:

```
Route::get('/fetch', function (Request $r) {
    $url = $r->query('url', '');

    //  VULNERABLE: sin validación ni lista blanca
    $content = @file_get_contents($url, false, $context);
    return response($content, 200);
});
```

Impacto:

- Escaneo de red interna
- Acceso a servicios internos (AWS metadata, Redis, etc.)
- Bypass de firewalls
- Robo de información sensible
- Lectura de archivos locales del servidor

Cómo explotarlo:

Ataque 1: Leer archivos locales

```
http://127.0.0.1:8000/fetch?url=file:///etc/passwd
```

Ataque 2: AWS Metadata (en servidores cloud)

```
http://127.0.0.1:8000/fetch?url=http://169.254.169.254/latest/meta-data/
```

Ataque 3: Escaneo de red interna

```
http://127.0.0.1:8000/fetch?url=http://192.168.1.1:8080
http://127.0.0.1:8000/fetch?url=http://localhost:3306
```

Ataque 4: Acceder a servicios internos

```
http://127.0.0.1:8000/fetch?url=http://localhost:6379  # Redis
http://127.0.0.1:8000/fetch?url=http://localhost:9200  # Elasticsearch
```

Solución correcta:

```php
// Whitelist de dominios permitidos
$allowed_domains = ['api.example.com', 'cdn.example.com'];
$host = parse_url($url, PHP_URL_HOST);

if (!in_array($host, $allowed_domains)) {
    abort(403, 'Domain not allowed');
}

// Validar protocolo
$scheme = parse_url($url, PHP_URL_SCHEME);
if (!in_array($scheme, ['http', 'https'])) {
    abort(403, 'Invalid protocol');
}
```

---

6. **Security Logging and Monitoring Failures** - A09:2021 NUEVA

Ubicación: `app/Http/Controllers/ParcelController.php` - Líneas 40, 60, 72

Descripción:
El sistema NO registra eventos críticos de seguridad:

- ✗ No registra quién accede a qué parcelas
- ✗ No registra modificaciones (valores anteriores vs nuevos)
- ✗ No registra eliminaciones de datos
- ✗ No hay timestamps ni IP del usuario
- ✗ Sin alertas de actividad sospechosa

Código vulnerable:

```php
public function update(Request $request, $id)
{
    $parcel = Parcel::findOrFail($id);

    // Actualiza sin registrar quién, qué, cuándo
    $parcel->name = $request->input('name');
    $parcel->save();

    //  VULNERABLE: NO hay log de auditoría
    return redirect()->route('parcels.index');
}

public function destroy($id)
{
    $parcel = Parcel::findOrFail($id);
    $parcel->delete();

    //  VULNERABLE: Eliminación sin rastro
    // No hay forma de saber quién lo eliminó ni recuperar los datos
```

```
        return redirect()->route('parcels.index');
    }
```

Impacto:

- Atacantes actúan sin ser detectados
- Imposible rastrear cambios maliciosos
- No se pueden recuperar datos eliminados
- Sin evidencia para investigaciones forenses
- Incumplimiento de regulaciones (GDPR, SOC2)
- Brechas de seguridad se descubren tarde

Cómo explotarlo:

Escenario de ataque:

1. Login como `josue@gmail.com` → password: `admin123`
2. Crear parcela "Parcela Importante"
3. Anotar el ID (ejemplo: ID=5)
4. Logout
5. Login como `maria@gmail.com` → password: `admin123`
6. Ir a `/parcels/5/edit` (IDOR - A01)
7. Cambiar nombre a "Parcela Hackeada" o eliminarla
8. **Resultado:** ✗ No hay registro de quién lo hizo, cuándo, ni valores anteriores

Verificar falta de logs:

```
php artisan tinker
```

```
>>> DB::getSchemaBuilder()->getTables();
// No existe tabla "audit_logs" o "activity_log"

>>> Parcel::withTrashed()->find(5);
// Si fue eliminada: null
// Sin soft deletes, el dato se pierde para siempre

>>> // No hay forma de saber:
// - Quién eliminó la parcela
// - Cuándo se eliminó
// - Qué datos contenía
// - Por qué se eliminó
```

Demostración visual:

```
# 1. Ver logs del sistema
tail -f storage/logs/laravel.log
```

```
# 2. Eliminar una parcela desde la interfaz

# 3. Observar el log:
# - NO aparece registro de eliminación
# - NO hay user_id del actor
# - NO hay IP address
# - NO hay valores eliminados
```

Solución correcta:

```
// 1. Crear tabla de auditoría (migration)
Schema::create('audit_logs', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->string('action'); // 'create', 'update', 'delete', 'view'
    $table->string('model'); // 'Parcel', 'Producto', etc.
    $table->unsignedBigInteger('model_id');
    $table->json('old_values')->nullable();
    $table->json('new_values')->nullable();
    $table->ipAddress('ip_address');
    $table->string('user_agent')->nullable();
    $table->timestamps();
});

// 2. Registrar cada acción importante
use Illuminate\Support\Facades\Log;

public function update(Request $request, $id)
{
    $parcel = Parcel::findOrFail($id);
    $oldValues = $parcel->toArray();

    $parcel->update($request->all());

    // ☑ Registrar cambio
    AuditLog::create([
        'user_id' => auth()->id(),
        'action' => 'update',
        'model' => 'Parcel',
        'model_id' => $id,
        'old_values' => $oldValues,
        'new_values' => $parcel->fresh()->toArray(),
        'ip_address' => $request->ip(),
        'user_agent' => $request->userAgent(),
    ]);

    Log::info('Parcel updated', [
        'user_id' => auth()->id(),
        'parcel_id' => $id,
        'ip' => $request->ip(),
        'changes' => $parcel->getChanges(),
```

```
    ]);
}

// 3. Usar Soft Deletes para recuperación
use Illuminate\Database\Eloquent\SoftDeletes;

class Parcel extends Model
{
    use SoftDeletes; // Permite recuperar datos "eliminados"
}

// 4. Monitorear actividad sospechosa
$suspiciousDeletes = AuditLog::where('created_at', '>', now()->subHour())
    ->where('action', 'delete')
    ->count();

if ($suspiciousDeletes > 10) {
    // Alertar administradores
    Mail::to('admin@example.com')->send(new SecurityAlert());
}
```

---

7. **Security Misconfiguration** - A05:2021

Ubicación: `routes/web.php` - Líneas 111-126 y archivo `.env`

Descripción:
Múltiples configuraciones inseguras:

- ✗ `APP_DEBUG=true` en "producción"
- ✗ Endpoint `/demo/leak-env` expone el archivo `.env`
- ✗ Stack traces detallados revelan estructura del sistema
- ✗ Sin cabeceras de seguridad HTTP

Código vulnerable:

```
Route::get('/demo/leak-env', function (Request $request) {
    //  Expone archivo .env si APP_DEBUG=true
    if (config('app.debug') !== true) {
        abort(404);
    }

    $envPath = base_path('.env');
    $content = file_get_contents($envPath);
    return response($content, 200);
});
```

Impacto:

- Exposición de credenciales (DB, API keys)

- Revelación de estructura del sistema
- Información sobre versiones de software
- Facilita otros ataques

Cómo explotarlo:

Ataque 1: Leer .env

```
http://127.0.0.1:8000/demo/leak-env
```

Resultado: Expone credenciales de base de datos, API keys, etc.

Ataque 2: Forzar error para ver stack trace

```
http://127.0.0.1:8000/parcels/999999999
```

Resultado: Stack trace completo con:

- Rutas absolutas del servidor
- Versiones de Laravel, PHP, MySQL
- Queries SQL ejecutadas
- Estructura de tablas

Ataque 3: Detectar tecnologías

```
curl -I http://127.0.0.1:8000
# Headers revelan:
# X-Powered-By: PHP/8.2
# Server: Apache/2.4.52
```

Solución correcta:

```
# En producción
APP_DEBUG=false
APP_ENV=production

# Eliminar endpoints de debug
# - /demo/leak-env
# - /phpinfo
# - /debug

# Agregar cabeceras de seguridad
# En middleware:
$response->headers->set('X-Frame-Options', 'SAMEORIGIN');
$response->headers->set('X-Content-Type-Options', 'nosniff');
$response->headers->set('X-XSS-Protection', '1; mode=block');
```

Estructura del Proyecto

```
Proyecto-Seguridad-TI/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   │       ├── ParcelController.php      #  IDOR + A09 Logging Failure
│   │       ├── ProductoController.php    #  SQLi + XSS vulnerable
│   │       └── PruebaController.php
│   └── Models/
│       ├── Parcel.php
│       ├── Producto.php
│       └── User.php
├── database/
│   ├── migrations/
│   │   ├── 2025_10_24_023752_create_parcels_table.php
│   │   └── 2025_10_24_024343_create_productos_table.php
│   └── seeders/
│       └── DemoSeeder.php
├── resources/
│   └── views/
│       ├── layouts/
│       │   └── app.blade.php             # Layout principal
│       ├── demo/
│       │   ├── login.blade.php           #  Auth débil
│       │   └── register.blade.php
│       ├── parcels/
│       │   ├── index.blade.php           # Con botón eliminar (A09)
│       │   ├── create.blade.php
│       │   ├── edit.blade.php            #  IDOR
│       │   └── show.blade.php
│       ├── productos/
│       │   ├── index.blade.php           #  XSS
│       │   └── create.blade.php
│       └── dashboard.blade.php
├── routes/
│   └── web.php                           #  Todas las rutas vulnerables
├── .env                                  #  Misconfig
└── README.md
```

🎮 Uso y Demostración

Rutas principales:

Ruta	Descripción	Vulnerabilidad
`/login`	Página de login	A07 - Auth débil

Ruta	Descripción	Vulnerabilidad
`/dashboard`	Dashboard principal	-
`/parcels`	Listado de parcelas	-
`/parcels/create`	Crear nueva parcela	-
`/parcels/{id}/edit`	Editar parcela	A01 - IDOR
`/parcels/{id}` (DELETE)	Eliminar parcela	A09 - No logs
`/productos`	Listado de productos	A03 - SQLi
`/productos/crear`	Crear producto	A03 - XSS
`/fetch?url=`	Endpoint SSRF	A10 - SSRF
`/demo/leak-env`	Expone .env	A05 - Misconfig
`/demo-login`	Login vulnerable	A07 - Auth
`/demo-reset-request`	Reset token	A07 - Token predecible

Demostración paso a paso:

1. SQL Injection (A03)

```
# 1. Accede a http://127.0.0.1:8000/productos
# 2. En la búsqueda, ingresa: ' OR '1'='1
# 3. Resultado: Muestra todos los productos sin filtro
# 4. Prueba también: '; DROP TABLE productos; --
```

2. XSS (A03)

```
# 1. Accede a /productos/crear
# 2. Nombre: Producto Test
# 3. Descripción: <script>alert('XSS Vulnerable!')</script>
# 4. Precio: 100
# 5. Guarda y ve la lista
# 6. Resultado: El script se ejecuta mostrando la alerta
```

3. IDOR - Broken Access Control (A01)

```
# 1. Login como josue@gmail.com / admin123
# 2. Crear parcela "Mi Parcela Privada" (nota el ID, ej: 7)
# 3. Logout
# 4. Login como maria@gmail.com / admin123
```

```
# 5. Accede manualmente a: /parcels/7/edit
# 6. Resultado: Puedes ver y editar la parcela de Josue
```

4. Broken Authentication (A07)

```
# 1. Intenta login con cualquier email registrado
# 2. Password: admin123
# 3. Resultado: Acceso concedido sin validación real
#
# 4. Para token predecible:
# POST /demo-reset-request
# body: email=test@test.com
# Resultado: Muestra token predecible basado en md5(email+time)
```

5. SSRF (A10)

```
# 1. Accede a: /fetch?url=http://example.com
# 2. Prueba: /fetch?url=file:///etc/passwd
# 3. Prueba: /fetch?url=http://localhost:3306
# Resultado: Puede hacer requests a cualquier URL
```

6. Security Logging Failure (A09)

```
# 1. Login como josue@gmail.com / admin123
# 2. Crear parcela "Parcela Importante"
# 3. Logout, login como maria@gmail.com / admin123
# 4. Ir a /parcels/X/edit (IDOR)
# 5. Eliminar la parcela
# 6. Verificar: php artisan tinker
#    >>> DB::table('parcels')->where('id', X)->first()
#    >>> // null - sin forma de recuperar
#    >>> // Sin logs de quién la eliminó
```

7. Security Misconfiguration (A05)

```
# Con APP_DEBUG=true:
# 1. Accede a: /demo/leak-env
# Resultado: Muestra todo el contenido del archivo .env
#
# 2. Fuerza un error: /parcels/999999999
# Resultado: Stack trace completo con rutas, queries, versiones
```

Advertencias

- **ESTE PROYECTO ES SOLO PARA FINES EDUCATIVOS**
- **NO USAR EN PRODUCCIÓN**
- **NO DESPLEGAR EN SERVIDORES PÚBLICOS**
- Las vulnerabilidades son intencionales para demostración académica

---

Recursos Adicionales

- OWASP Top 10 2021
- Laravel Security Best Practices
- PHP Security Cheat Sheet
- SQL Injection Prevention
- XSS Prevention

Resumen de Vulnerabilidades

# OWASP	Vulnerabilidad	Implementada	Ubicación	Impacto
A01:2021	Broken Access Control	☑	ParcelController.php	Alto
A03:2021	Injection (SQLi + XSS)	☑	ProductoController.php	Crítico
A05:2021	Security Misconfiguration	☑	.env + routes/web.php	Medio
A07:2021	Authentication Failures	☑	routes/web.php	Alto
A09:2021	Logging Failures	☑	ParcelController.php	Medio
A10:2021	SSRF	☑	routes/web.php	Alto

Total: 6 vulnerabilidades críticas implementadas

Herramientas de Testing

Para probar las vulnerabilidades:

```
# SQL Injection
sqlmap -u "http://127.0.0.1:8000/productos?buscar=test" --dbs

# SSRF
curl "http://127.0.0.1:8000/fetch?url=file:///etc/passwd"

# XSS
# Usar Burp Suite o ZAP Proxy

# Auditoría de composer
composer audit
```

```
# Análisis de código estático
./vendor/bin/phpstan analyse app
```

Créditos

Proyecto: Seguridad en Tecnologías de la Información
Institución: Universidad Tecnica Nacional **Año:** 2025

Autores:

- Andrew Montero - @AndrewMontero
- Deivis Jimenez - [@DeivisJm]

Agradecimientos:

- OWASP Foundation por la documentación del Top 10
- Comunidad de Laravel por el framework
- Profesores y compañeros del curso de Seguridad TI

Licencia

Este proyecto es de uso educativo exclusivamente.

MIT License - Ver archivo `LICENSE` para más detalles.

Contribuciones

Las contribuciones educativas son bienvenidas. Por favor:

1. Fork el proyecto
2. Crea una rama (`git checkout -b feature/NuevaVulnerabilidad`)
3. Commit tus cambios (`git commit -m 'Agrega nueva vulnerabilidad educativa'`)
4. Push a la rama (`git push origin feature/NuevaVulnerabilidad`)
5. Abre un Pull Request

Directrices:

- Documenta claramente cada vulnerabilidad
- Incluye ejemplos de explotación
- Proporciona la solución correcta
- Añade referencias a OWASP

FAQ (Preguntas Frecuentes)

¿Es seguro ejecutar este proyecto?

Solo en entorno local para aprendizaje. **NUNCA** en producción o servidores públicos.

¿Cómo resuelvo el error "419 Page Expired"?

```
php artisan config:clear
php artisan cache:clear
# Borra cookies del navegador
# Reinicia php artisan serve
```

¿Las vulnerabilidades son reales?

Sí, todas están basadas en vulnerabilidades reales documentadas por OWASP. Son intencionalmente inseguras para fines educativos.

¿Puedo usar este proyecto como referencia?

Sí, pero **solo** como ejemplo de qué NO hacer. Úsalo para aprender a identificar y corregir vulnerabilidades.

¿Dónde puedo aprender más?

- OWASP Academy
- PortSwigger Web Security Academy
- HackTheBox

¿Preguntas sobre el proyecto?

- **GitHub Issues**: Abrir issue
- **Repositorio**: https://github.com/AndrewMontero/Proyecto-Seguridad-TI

Objetivos de Aprendizaje

Al completar este proyecto, habrás aprendido:

- ☑ Identificar las 6 vulnerabilidades más críticas del OWASP Top 10
- ☑ Entender cómo los atacantes explotan cada vulnerabilidad
- ☑ Implementar código vulnerable de forma controlada
- ☑ Aplicar las soluciones correctas para mitigar riesgos
- ☑ Usar herramientas de testing de seguridad
- ☑ Documentar vulnerabilidades de forma clara

Changelog

v1.0.0 (2025-01-31)

- ☑ Implementación inicial de 6 vulnerabilidades OWASP Top 10
- ☑ Sistema completo de gestión agrícola

- ☑ Dashboard interactivo
- ☑ Documentación completa
- ☑ Datos de prueba (seeders)

💻 Desarrollado con propósitos educativos | 🔓 Aprende Seguridad de forma práctica

---