Vulnerabilidad owasp top 10 en aplicación backend

Título del proyecto: LazyLedger
Integrantes: César López, Alexis Jara, César Ramos
Asignatura: Desarrollo basado en plataformas
Docente: Ing. Edison Coronel
Fecha de entrega: 20 de octubre 2025

Resumen del proyecto

LazyLedger utiliza Java para gestionar [especificar: usuarios, transacciones, inventario, etc. La aplicación implementa un sistema integral de autenticación y autorización, permitiendo a los usuarios acceder a recursos protegidos según sus permisos asignado

Aspecto	Detalle
Lenguaje de programación	Java 21
Framework	Spring Boot
Base de datos	PostgreSQL
Versión de Spring Boot	3.5.6
Autenticación	JWT
Herramienta de Build	Maven
Control de versiones	Git

Introducción

Cuando hablamos de la seguridad es el pilar fundamental al momento de desarrollar una aplicación, por lo que según los reportes de OWASP y diversas organizaciones de ciberseguridad miles de aplicaciones en producción presenta vulnerabilidades que pueden ser explotadas por actores maliciosos esto dando resultados de robo de datos o incluso pérdidas económicas.

Objetivos

1. Identificar y documentar vulnerabilidades presentes en el código base actual de la aplicación, específicamente aquellas relacionadas con el OWASP Top 10 2021
2. Evaluar el impacto potencial de cada vulnerabilidad en la confidencialidad, integridad y disponibilidad de la información del sistema
3. Demostrar las medidas de mitigación implementadas o planificadas con ejemplos de código y configuración

Análisis de vulnerabilidades OWASP Top 10

A04:2021-INSECURE DESIGN (Diseño inseguro)

Archivo principal: ConfiguracionSeguridad.java

1. Líneas de código vulnerable: 31-34, 75-82
2. Archivo de configuración: application.properties (linea 24)

Componente afectado: Arquitectura general del sistema de autenticación y autorización.

Descripción de la localización: El defecto se encuentra en la clase responsable de configurar la seguridad de toda la aplicación Spring Boot. Este es un componente crítico porque sus decisiones afectan a todas las rutas y endpoints de la aplicación

Descripción de la vulnerabilidad: La aplicación tiene un defecto arquitectónico grave: toda la seguridad está deshabilitada por defecto mediante una variable de configuración. Esto significa que si un administrador o desarrollador no activa explícitamente la seguridad, la aplicación se ejecuta completamente sin protección, permitiendo acceso sin autenticación a todos los recursos.

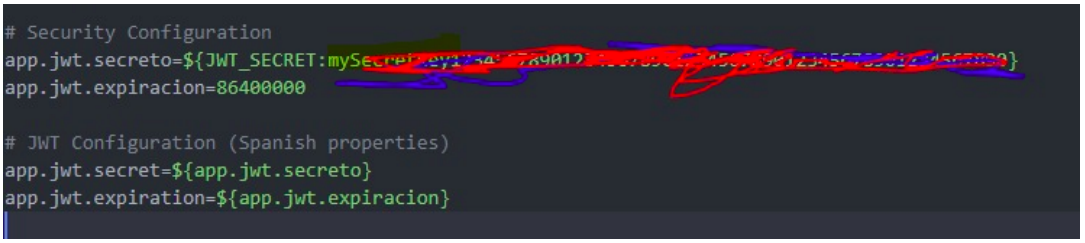
Mecanismo del defecto: La vulnerabilidad funciona mediante la siguiente lógica

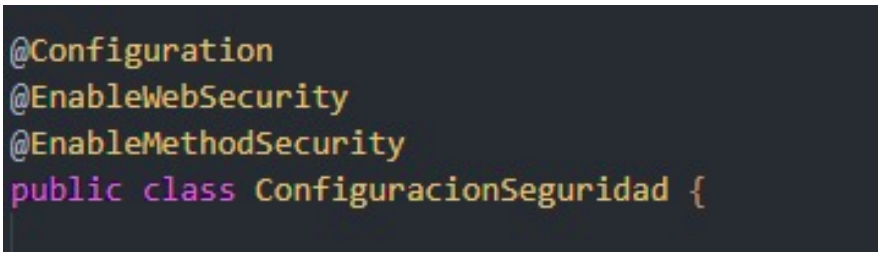
1. Se define una propiedad de configuración llamada app.seguridad.habilitada
2. El valor por defecto de esta propiedad está establecido en false
3. Si está en false, se configura Spring Security para permitir acceso a todos los endpoints sin requerir autenticación
4. No hay restricción de autorización: permitAll() permite acceso a cualquier usuario

Impacto

Recursos potencialmente expuestos	Descripción	Severidad
Listado de usuarios	Acceso a nombres, emails, información de perfil de todos los usuarios	Crítica
Datos de transacciones	Si la app maneja pagos o transacciones, todo es visible	Crítica
Información personal	Direcciones, números de teléfono, documentos de identidad	Crítica
Historial de actividades	Logs de acciones que deberían ser privadas	Alta
Configuraciones del sistema	Versiones, endpoints, estructura interna de la app	Alta

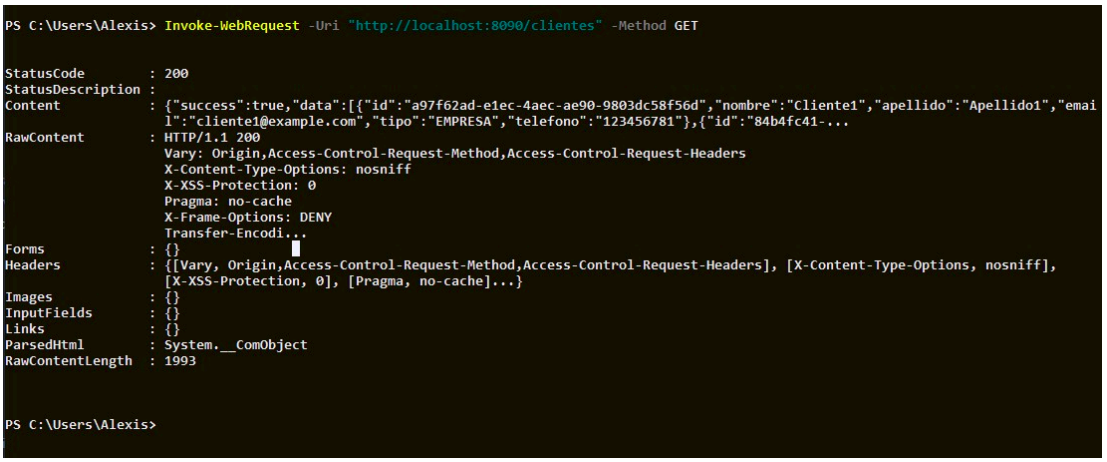
Evidencia de configuración



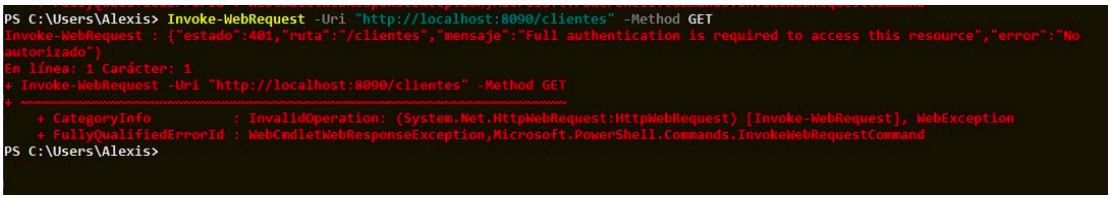


Vulnerabilidad

Antes



Después



A02:2021 - CRYPTOGRAPHIC FAILURES (Fallos Criptográficos)

Archivo principal: JwtUtils.java

Líneas de código vulnerable: 17-24

Archivo de configuración: application.properties (líneas 25-27)

Componente afectado: Sistema de generación y validación de tokens JWT (JSON Web Tokens) para autenticación y autorización de usuarios.

Descripción de la localización: El defecto se encuentra en dos ubicaciones críticas: la clase JwtUtils.java que es responsable de crear y verificar tokens JWT, y el archivo application.properties donde se almacena el secreto criptográfico. Este componente es esencial porque todos los mecanismos de autenticación de la aplicación dependen de la seguridad de estos tokens. Cualquier compromiso en esta capa afecta la confiabilidad de todo el sistema de identificación de usuarios.

Descripción de la vulnerabilidad: La aplicación tiene dos fallos criptográficos graves: primero, el secreto JWT está almacenado en texto plano directamente en el archivo de configuración (application.properties), lo que significa que cualquier persona con acceso al repositorio de código puede verlo. Segundo, el secreto utilizado ("mySecretKey123...") es una clave débil que no fue generada mediante métodos criptográficamente seguros. Esta combinación permite que un atacante que obtenga el secreto pueda forjar tokens JWT válidos, suplantando la identidad de cualquier usuario sin necesidad de conocer sus credenciales**.**

Mecanismo del defecto: La vulnerabilidad funciona mediante la siguiente lógica

1. El secreto JWT está definido en application.properties como: app.jwt.secreto=mySecretKeyxxxxx-xxxxxxxxx-xxxxxxxxxxx
2. La clase JwtUtils.java lee este secreto mediante la anotación @Value("${app.jwt.secreto}")
3. Este secreto se utiliza para firmar digitalmente todos los tokens JWT generados por la aplicación
4. Cuando un usuario inicia sesión, se genera un token JWT firmado con este secreto

Impacto

Recursos potencialmente expuestos	**Descripción**	**Severidad**
Identidad de todos los usuarios	Suplantación total de cualquier cuenta sin credenciales, incluyendo cuentas administrativas	Crítica
Sistema de autenticación completo	Compromiso total: el atacante puede generar tokens válidos para cualquier usuario en cualquier momento	Crítica
Datos personales y sensibles	Acceso no autorizado a información privada de todos los usuarios al autenticarse como ellos	Crítica
Transacciones y operaciones críticas	Capacidad de realizar acciones en nombre de otros usuarios (transferencias, compras, cambios de configuración)	Crítica
Logs y trazabilidad	Imposibilidad de distinguir entre tokens legítimos y	Alta

	falsificados, perdiendo la confiabilidad de las auditorías	
Persistencia del compromiso	Los tokens falsificados permanecen válidos durante 24 horas (86400000 ms), permitiendo acceso prolongado	Alta
Reputación del sistema	Pérdida total de confianza en el sistema de seguridad de la aplicación	Alta

Vulnerabilidad

Antes

```
# application.properties
spring.datasource.username=${DB_USER:postgres}
spring.datasource.password=${DB_PASSWORD:password}
```

```
environment:
  SPRING_DATASOURCE_USERNAME: ${DB_USER:-postgres}
  SPRING_DATASOURCE_PASSWORD: ${DB_PASSWORD:-
password}
```

Después

Configuración de variable de entorno .env

```
JWT_SECRET=your_secure_jwt_secret_here_256_bits_mi
nimum
DB_USER=postgres
DB_PASSWORD=your_secure_password_here
SPRING_PROFILES_ACTIVE=development
```

```
app.jwt.secreto=${JWT_SECRET}

# Database Configuration
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWORD}
```

```
environment:
  JWT_SECRET: ${JWT_SECRET}
  DB_USER: ${DB_USER}
  DB_PASSWORD: ${DB_PASSWORD}
```

Tercero
A09:2021-SECURITY LOGGING AND MONITORING FAILURES (Fallos en Registro y Monitoreo de Seguridad)
Archivo principal: JwtUtils.java
Líneas de código vulnerable: 75-85
Archivos relacionados: GlobalExceptionHandler.java (todo el archivo), ServicioAutenticacion.java
Componente afectado: Sistema de auditoría y monitoreo de eventos de seguridad en toda la aplicación.

Descripción de la localización: El defecto se encuentra distribuido en múltiples componentes críticos de la aplicación: JwtUtils.java que maneja la validación de tokens, ServicioAutenticacion.java que procesa intentos de login, y GlobalExceptionHandler.java que gestiona excepciones. Estos componentes son fundamentales para la seguridad porque registran (o deberían registrar) todos los eventos relacionados con autenticación, autorización y accesos al sistema. La ausencia de logging adecuado afecta la capacidad de detectar, responder y auditar incidentes de seguridad.
Descripción de la vulnerabilidad: La aplicación carece de un sistema adecuado de registro y monitoreo de eventos de seguridad. En lugar de utilizar un sistema profesional de logging, se emplea System.err.println() que solo imprime en consola sin persistir la información. No se registran eventos críticos como intentos de autenticación fallidos, validaciones de tokens inválidos, accesos a recursos sensibles, o modificaciones de permisos. Además, cuando se capturan excepciones de seguridad, no se almacena información contextual vital como dirección IP del atacante, timestamp exacto, usuario afectado, o detalles del intento. Esta ausencia de visibilidad hace imposible detectar ataques en curso, identificar patrones de comportamiento malicioso, o cumplir con requisitos de auditoría.
Mecanismos del defecto: La vulnerabilidad funciona mediante la siguiente lógica

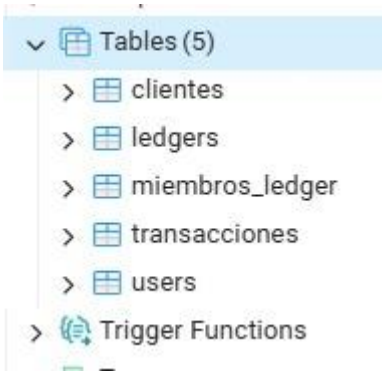
1. cuando ocurre un error de validación de JWT (token malformado, expirado, o con firma inválida), el método validateJwtToken() solo imprime en consola: System.err.println("Invalid JWT token: " + e.getMessage()).
2. Esta información no se persiste en ningún archivo de log, base de datos o sistema de monitoreo.
3. No se registra información crítica como: IP de origen, timestamp, usuario que intentó usar el token, tipo específico de error, o contexto de la petición
4. En el ServicioAutenticacion.java, cuando un usuario intenta autenticarse con credenciales incorrectas, se captura BadCredentialsException pero no se registra el intento fallido.
5. Un atacante puede realizar intentos masivos de fuerza bruta y cada intento fallido queda invisible para el equipo de seguridad.
6. No existe correlación entre eventos: múltiples intentos fallidos desde la misma IP no generan alertas.

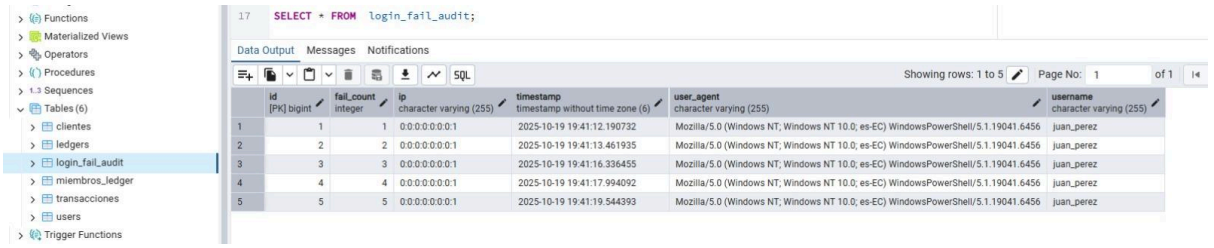
Impacto

Recursos potencialmente expuestos	Descripción	Severidad
Detección de ataques en tiempo real	Imposibilidad de identificar intentos de fuerza bruta, escaneo de vulnerabilidades o accesos no autorizados mientras ocurren	Critica
Respuesta ante incidentes	Sin logs, no se puede determinar el alcance de una brecha de seguridad, qué datos fueron comprometidos o cuándo comenzó el ataque	Critica
Investigación forense	Ausencia de evidencia digital para análisis post-incidente, imposibilitando identificar al atacante o reconstruir la secuencia de eventos	Alta
Cumplimiento normativo	Incumplimiento de GDPR (Art. 33 requiere detección en 72h), PCI-DSS (Req. 10 requiere auditoría), SOC 2, ISO 27001	Alta
Ataques de fuerza bruta	Intentos ilimitados de adivinación de contraseñas sin detección ni bloqueo automático	Alta
Compromiso prolongado	Brechas de seguridad pueden existir durante meses sin ser detectadas, permitiendo exfiltración masiva de datos	Alta
Responsabilidad legal	Sin evidencia de controles de seguridad, exposición a demandas y sanciones regulatorias	Alta

Vulnerabilidad

Antes





Después

Cuarto

A09:2021 - AUSENCIA DE RATE LIMITING (Subcategoría: Fallos de Monitoreo)

Archivo principal: ControladorAutenticacion.java

Líneas de código vulnerable: Endpoint /signin (sin protección)

Componentes afectados: Endpoints de autenticación expuestos sin límite de peticiones.

Descripción de la localización: El defecto se encuentra en el controlador de autenticación, específicamente en el endpoint POST /signin que procesa solicitudes de login. Este endpoint es público y accesible sin restricciones, siendo la puerta de entrada principal al sistema. Al no tener protección de rate limiting, acepta un número ilimitado de peticiones desde cualquier origen.

Descripción de la vulnerabilidad: El endpoint de autenticación no implementa ningún mecanismo de limitación de tasa de peticiones (rate limiting). Esto significa que un atacante puede enviar miles o millones de intentos de autenticación sin ninguna restricción temporal o de cantidad. No existe control por dirección IP, por usuario, ni por ventana temporal.
Esta ausencia permite ataques automatizados de fuerza bruta donde se prueban sistemáticamente combinaciones de usuario/contraseña hasta encontrar credenciales válidas.

Mecanismos de defecto:

1. El endpoint @PostMapping("/signin") está completamente abierto sin middleware de protección
2. Un atacante puede escribir un script automatizado que envíe peticiones HTTP POST al endpoint
3. Cada petición intenta una combinación diferente de usuario y contraseña
4. El servidor procesa cada petición sin restricción, consultando la base de datos cada vez
5. No hay contador de intentos fallidos por IP o por usuario
6. No hay bloqueo temporal después de X intentos fallidos

Impacto

Recursos potencialmente expuestos	Descripción	Severidad
Credenciales de usuarios	Compromiso de cuentas mediante ataques de diccionario o fuerza bruta sin resistencia	Crítica
Cuentas administrativas	Acceso a cuentas privilegiadas por enumeración y ataque masivo	Crítica
Enumeración de usuarios	Posibilidad de descubrir usuarios válidos mediante análisis de tiempos de respuesta	Media
Costos de infraestructura	Consumo excesivo de recursos computacionales procesando ataques automatizados	Media

Vulnerabilidad

Antes

```
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'
Invoke-WebRequest : Error en el servidor remoto: (401) No autorizado.
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand

PS C:\Users\Alexis> _
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
Invoke-WebRequest : Error en el servidor remoto: (401) No autorizado.
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand

PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
Invoke-WebRequest : Error en el servidor remoto: (401) No autorizado.
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand

PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'
Invoke-WebRequest : Error en el servidor remoto: (401) No autorizado.
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand

PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
```

Después

```
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'
Invoke-WebRequest : Login FAIL. Intentos fallidos usuario: 2, IP: 2
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
Invoke-WebRequest : Login FAIL. Intentos fallidos usuario: 3, IP: 3
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
Invoke-WebRequest : Login FAIL. Intentos fallidos usuario: 4, IP: 4
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'_
Invoke-WebRequest : Login FAIL. Intentos fallidos usuario: 5, IP: 5
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
>>   -Method POST `
>>   -Headers @{ "Content-Type" = "application/json" } `
>>   -Body '{"nombreUsuario":"juan_perez","contrasena":"Password!"}'
Invoke-WebRequest : Demasiados intentos fallidos. Bloqueado por seguridad.
En línea: 1 Carácter: 1
+ Invoke-WebRequest -Uri "http://localhost:8090/api/auth/signin" `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\Users\Alexis>
```

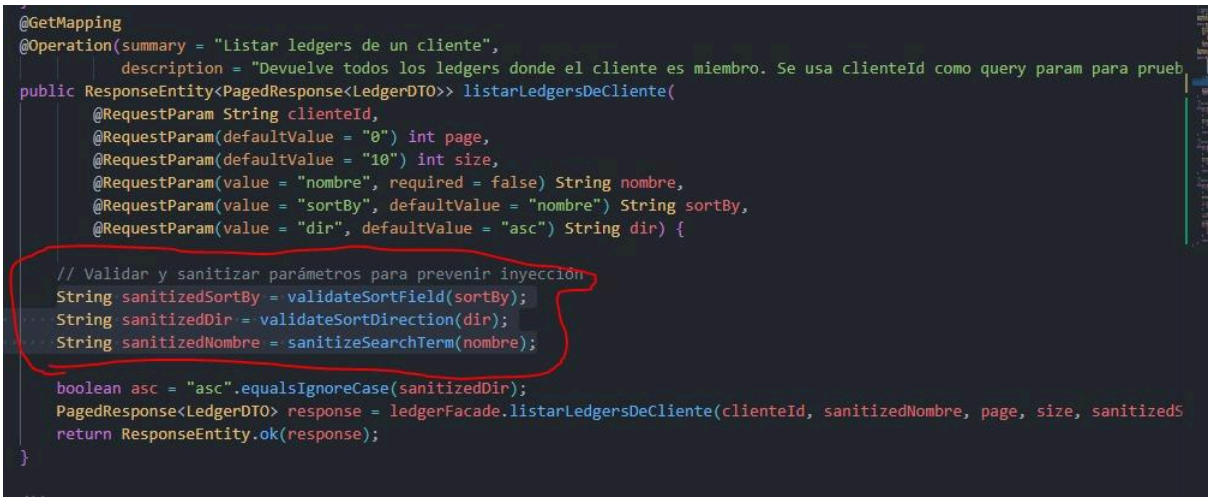
Quinto
A03:2021 - INJECTION (Inyección)
Archivo principal: LedgerController.java
Líneas de código vulnerable: 101 (parámetros de búsqueda dinámica)
Componente afectado: Capa de controladores REST que recibe y procesa parámetros de consulta del usuario para búsquedas y ordenamiento.
Descripción de la localización:
El defecto se encuentra en el controlador LedgerController, específicamente en el método GET que lista ledgers de clientes. Este endpoint acepta múltiples parámetros de consulta incluyendo "nombre" para filtrado y "sortBy" para ordenamiento. Estos parámetros son recibidos directamente desde la URL sin validación ni sanitización antes de ser procesados por la capa de datos. Aunque JPA/Hibernate proporciona cierta protección contra SQL injection básico, la falta de validación de entrada abre múltiples vectores de ataque.
Descripción de la vulnerabilidad: La aplicación no valida ni sanitiza los parámetros de entrada del usuario antes de utilizarlos en operaciones de búsqueda y ordenamiento. El parámetro "sortBy" es particularmente peligroso porque se usa para construir cláusulas ORDER BY dinámicas. Un atacante podría inyectar valores maliciosos como "nombre; DROP TABLE ledgers--" o intentar acceder a columnas no autorizadas. El parámetro "nombre" tampoco es validado, permitiendo potencialmente caracteres especiales que podrían causar comportamientos inesperados. Si en el futuro se agregan consultas nativas (@Query con nativeQuery=true) o se integran bases de datos NoSQL/LDAP, estas vulnerabilidades se magnificarían significativamente.
Mecanismos del defecto

1. El usuario envía una petición GET con parámetros de consulta: /ledgers?clienteId=123&nombre=test&sortBy=nombre
2. El controlador recibe estos parámetros sin ninguna validación: @RequestParam(value = "sortBy", defaultValue = "nombre") String sortBy
3. No existe lista blanca (whitelist) de valores permitidos para sortBy
4. El parámetro sortBy se pasa directamente a la capa de servicio y eventualmente a JPA
5. Las diferencias en tiempos de respuesta o mensajes de error pueden revelar estructura interna de la base de datos
6. Si el parámetro "nombre" contiene caracteres especiales SQL como ' OR '1'='1, podría alterar la lógica de la consulta
7. No hay límites de longitud: nombre puede ser un string de 10MB causando denegación de servicio
8. Si se implementan búsquedas con LIKE dinámico sin escape, abre ataques de wildcard injection

Impacto

Recursos potencialmente expuestos	Descripción	Severidad
SQL Injection futuro	Si se agregan consultas nativas o se modifica a query builder dinámico, vulnerabilidad crítica inmediata	Alta
Exposición de estructura de BD	Enumeración de nombres de columnas, tablas y relaciones mediante inyección en sortBy	Alta
Denegación de servicio	Parámetros de longitud excesiva pueden saturar memoria o causar timeouts de base de datos	Alta
Bypass de control de acceso	Ordenamiento por columnas no autorizadas puede revelar existencia de datos sensibles	Media
Information disclosure	Mensajes de error diferentes según valores inyectados revelan información del sistema	Media

Antes



```
@GetMapping
@Operation(summary = "Listar ledgers de un cliente",
           description = "Devuelve todos los ledgers donde el cliente es miembro. Se usa clienteId como query param para prueb
public ResponseEntity<PagedResponse<LedgerDTO>> listarLedgersDeCliente(
        @RequestParam String clienteId,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "10") int size,
        @RequestParam(value = "nombre", required = false) String nombre,
        @RequestParam(value = "sortBy", defaultValue = "nombre") String sortBy,
        @RequestParam(value = "dir", defaultValue = "asc") String dir) {

    // Validar y sanitizar parámetros para prevenir inyección
    String sanitizedSortBy = validateSortField(sortBy);
    String sanitizedDir = validateSortDirection(dir);
    String sanitizedNombre = sanitizeSearchTerm(nombre);

    boolean asc = "asc".equalsIgnoreCase(sanitizedDir);
    PagedResponse<LedgerDTO> response = ledgerFacade.listarLedgersDeCliente(clienteId, sanitizedNombre, page, size, sanitizedS
    return ResponseEntity.ok(response);
}
```

Después

```
PS C:\Users\Alexis> Invoke-WebRequest -Uri "http://localhost:8090/ledgers?page=0&size=10&nombre=casa&sortBy=nombre&dir=asc" -Method GET -Headers @{ "Authorization" = "Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzb2ZpYV9nb21leiIsImlhdCI6MTc2MDkyMzQwNiwiZXhwIjoxNzYxMDA5ODA2fQ._ZCew6f1rnGmNCb-SobCBgyLYlJkUpnyC6YWVUIcp2c" }


StatusCode        : 200
StatusDescription :
Content           : {"success":true,"data":[],"pagination":{"page":0,"size":10,"totalElements":0,
"totalPages":0,"first":true,"last":true}
                    ,"timestamp":"2025-10-19T20:25:30.932623"}
RawContent        : HTTP/1.1 200
                    Vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
                    X-Content-Type-Options: nosniff
                    X-XSS-Protection: 0
                    Pragma: no-cache
                    X-Frame-Options: DENY
                    Transfer-Encodi...
Forms             : {}
Headers           : {[Vary, Origin,Access-Control-Request-Method,Access-Control-Request-Headers],
 [X-Content-Type-Options, nosniff],
                    [X-XSS-Protection, 0], [Pragma, no-cache]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 159
```

Eliminar los endpoints de debug/prueba es una corrección importante porque:
Elimina backdoors inseguros que permiten acceso sin autenticación
Reduce la superficie de ataque
Previene acceso no autorizado a través de endpoints de prueba
Sin embargo, NO corrige completamente A01:2021 - Broken Access Control porque el problema principal es que los endpoints legítimos NO tienen validación de permisos basada en roles. Por ejemplo:
Un usuario podría acceder a ledgers de otros usuarios
Un miembro podría modificar datos que no le pertenecen
No hay validación de que el usuario autenticado tenga permisos sobre el recurso específico
La eliminación de endpoints de debug es una corrección necesaria pero parcial. Para corregir completamente A01, se necesita implementar @PreAuthorize y validación de permisos basada en roles.
¿El usuario quiere que proceda con la eliminación de endpoints de debug primero, o prefiere que implemente la validación completa de permisos?

1. **En LedgerController.java:**
   - **@GetMapping("/debug/crear-simple") - endpoint de debug inseguro**
2. **En TransaccionController.java:**
   - **@GetMapping("/demo") - endpoint de demo inseguro**
3. **En MiembroLedgerController.java:**
   - **@GetMapping("/demo/invitar")**
   - **@GetMapping("/demo/aceptar")**

- **@GetMapping("/demo/rechazar")**
- **@GetMapping("/demo/pendientes")**

A01:2021 - Broken Access Control (Control de Acceso Roto)

Severidad: CRÍTICA

LedgerController.java

MiembroLedgerController.java

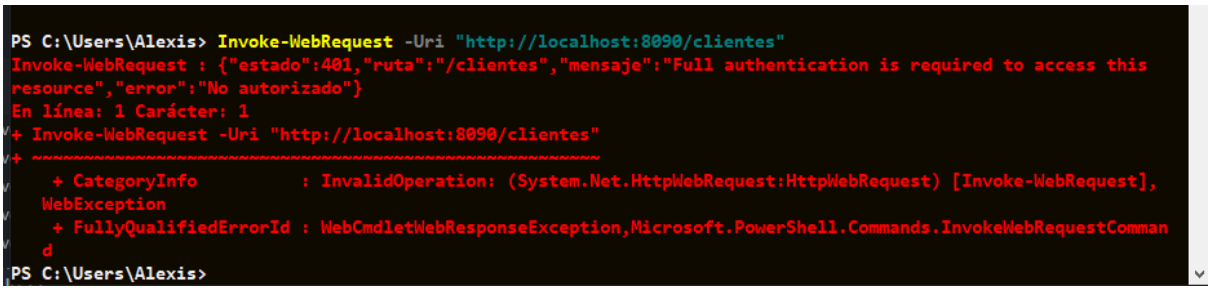
TransaccionController.java

La aplicación NO implementa control de acceso basado en roles ni validación de autorización a nivel de método. Cualquier usuario autenticado puede:

- Acceder a ledgers de otros usuarios
- Modificar datos de ledgers que no le pertenecen
- Invitar miembros sin ser propietario
- Cambiar roles sin validar permisos

```
@RestController
@RequestMapping("/clientes")
@Tag(name = "Clientes", description = "API para gestión de clientes")
@PreAuthorize("hasRole('USER')")
public class ClienteController {

    private final ClienteFacade clienteFacade;
    public ClienteController(ClienteFacade clienteFacade) {
        this.clienteFacade = clienteFacade;
    }
```



Impacto:

- **Acceso no autorizado** a datos financieros sensibles

- **Manipulación de ledgers** de otros usuarios
- **Escalada de privilegios** mediante falsificación de headers

Recomendaciones:

1. Implementar `@PreAuthorize` y `@Secured` en todos los endpoints
2. Obtener el usuario autenticado desde `SecurityContext` en lugar de headers/parámetros
3. Validar que el usuario tiene permisos sobre el recurso solicitado
4. Implementar un sistema de autorización basado en roles (PROPIETARIO, MIEMBRO, **ADMIN)**

```
@PreAuthorize("hasRole('USER')")
@GetMapping("/{id}")
public ResponseEntity<?> getLedger(@PathVariable String id) {
   String currentUserId = SecurityContextHolder.getContext()
      .getAuthentication().getName();
   // Validar que currentUserId tiene acceso al ledger
}
```

Reflexión personal y mejorar futuras

Reflexión personal

1. Comprendí que la seguridad no depende solo de herramientas o configuraciones, sino de la mentalidad con la que se diseña un sistema. Adoptar el principio de *secure by design* significa pensar en la seguridad desde el primer boceto del proyecto. Esta forma de trabajo evita vulnerabilidades que, aunque parezcan pequeñas, pueden comprometer completamente la integridad del sistema.
2. Aprendí que si la seguridad está deshabilitada desde el inicio, se envía el mensaje de que no es prioritaria. En cambio, un sistema que es seguro por defecto demuestra madurez y responsabilidad profesional, ya que protege incluso cuando el entorno o la experiencia del usuario son limitados.
3. Entendí que la seguridad debe estar presente en cada etapa del ciclo de vida del software. Desde la planificación hasta el mantenimiento, cada fase influye en la solidez del sistema. Este enfoque me hizo valorar la importancia del trabajo colaborativo entre diseñadores, desarrolladores y administradores, porque la seguridad es una responsabilidad compartida.

Mejoras a futuro

1. **Implementar Perfiles de Spring diferenciados por entorno**
    - Crear configuraciones específicas para desarrollo, staging y producción
    - Asegurar que cada entorno tenga el nivel apropiado de seguridad
    - Automatizar el switching entre perfiles

2. **Implementar JWT con firma segura**
    - Usar algoritmos fuertes (RS256 o ES256, no HS256)
    - Implementar refresh tokens con espiración corta
    - Validar todos los tokens en cada solicitud

3. **Añadir Validación de Entrada Comprehensiva**
    - Implementar validaciones en todos los endpoints
    - Usar anotaciones como @Valid y @NotBlank
    - Sanitizar entrada para prevenir inyecciones

Conclusiones

La realización de este análisis de vulnerabilidades OWASP Top 10 en nuestra aplicación backend ha sido un ejercicio fundamental para comprender la complejidad y la criticidad de la seguridad en el desarrollo de software. Lo que inicialmente parecía ser una aplicación estándar de Spring Boot reveló vulnerabilidades arquitectónicas significativas que, en un escenario de producción, podrían resultar en compromiso completo del sistema.

Prompt usado: Copilot configurado en modo agente.

Identifica las vulnerabilidades del top 10 owasp de lazy ledger, al menos 6.