Seguridad e Integridad de Sistemas - BookTracker

Resumen Ejecutivo

Este documento detalla las medidas de seguridad e integridad implementadas en el proyecto BookTracker, una aplicación para seguimiento de lectura. Se describe la arquitectura de seguridad, controles implementados, estrategias de protección de datos y consideraciones para futuras mejoras. El objetivo es proporcionar un sistema robusto que proteja tanto la información de los usuarios como la integridad de la aplicación.

Índice

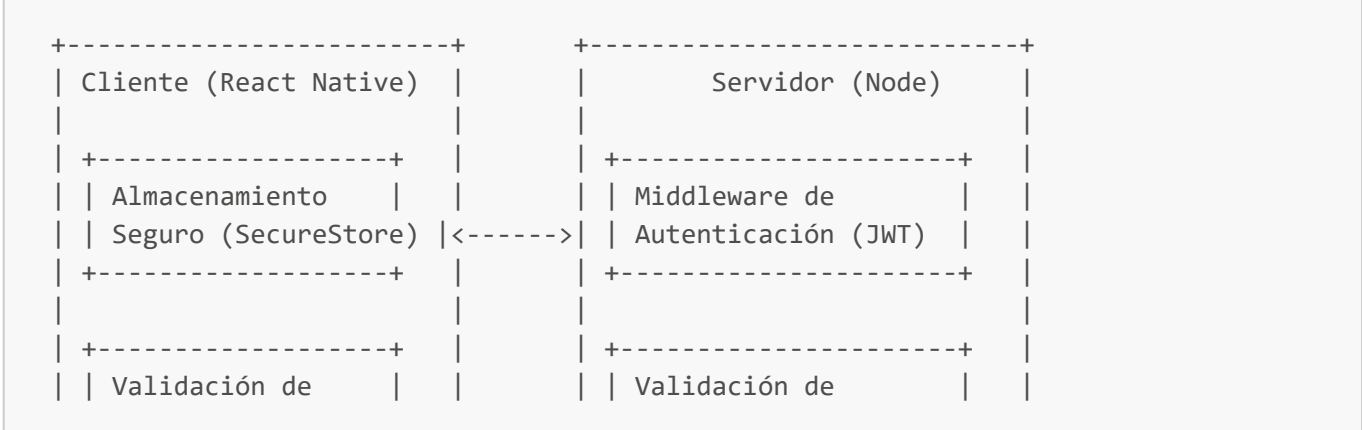
Arquitectura de Seguridad

Principios de Diseño

La arquitectura de seguridad de BookTracker se basa en los siguientes principios:

- **Defensa en profundidad**: Múltiples capas de seguridad que protegen los datos y la lógica de negocio
- **Principio de mínimo privilegio**: Los componentes solo tienen acceso a los recursos necesarios para su función
- **Seguridad por diseño**: Consideraciones de seguridad integradas desde el inicio del desarrollo
- **Protección de datos sensibles**: Almacenamiento seguro de información confidencial

Diagrama de Arquitectura de Seguridad



```
| | Entrada           |   |<------>| | Esquemas (Mongoose) |   |
| +-------------------+   |       | +---------------------+   |
|                         |       |                           |
| +-------------------+   |       | +---------------------+   |
| | Manejo de Errores |   |       | | Cifrado de          |   |
| | Centralizado      |   |       | | Contraseñas (bcrypt)|   |
| +-------------------+   |       | +---------------------+   |
+-------------------------+       +---------------------------+
```

Almacenamiento Seguro de Datos

Cliente (React Native)

- **Implementación de SecureStore**: Utilizado para almacenar datos sensibles como tokens de autenticación
- **Encapsulamiento**: Interfaz unificada para acceder a datos seguros a través del módulo `secureStorage.js`
- **Expiración de tokens**: Control de caducidad de credenciales para minimizar ventanas de ataque
- **Migración desde AsyncStorage**: Estrategia para migrar datos antiguos al nuevo sistema seguro

Servidor (Node.js)

- **Cifrado de contraseñas**: Uso de bcrypt para almacenar contraseñas hasheadas en la base de datos
- **Secretos en variables de entorno**: Configuración separada para distintos entornos
- **Tokens JWT firmados**: Implementación de tokens con payload mínimo y tiempo de expiración

Autenticación y Autorización

Sistema de Tokens

- **JWT (JSON Web Tokens)**: Implementación de tokens firmados para autenticación sin estado
- **Almacenamiento seguro**: Tokens almacenados en SecureStore en el dispositivo cliente
- **Control de expiración**: Verificación automática de validez temporal de tokens

Flujos de Autenticación

- **Registro seguro**: Validación de datos, sanitización y cifrado de contraseñas
- **Inicio de sesión**: Verificación de credenciales con tiempo constante
- **Cierre de sesión**: Eliminación de tokens locales y (opcionalmente) invalidación en servidor
- **Recuperación de contraseña**: Flujo seguro con tokens temporales de un solo uso

Middleware de Autorización

- **Verificación de tokens**: Middleware que valida tokens JWT en cada petición protegida
- **Control de acceso**: Validación de permisos basada en roles (implementación básica)

Protección de API y Comunicaciones

Seguridad en las Peticiones

- **HTTPS**: Todas las comunicaciones cliente-servidor realizadas mediante conexión cifrada
- **Cabeceras de seguridad**: Implementación de cabeceras como `X-Requested-With` para prevenir CSRF
- **Sanitización de parámetros URL**: Limpieza de parámetros en peticiones a APIs externas

Protección contra Ataques

- **Rate Limiting**: Limitación de frecuencia de peticiones (implementación pendiente)
- **Timeout en peticiones**: Control de tiempo máximo para prevenir ataques de agotamiento de recursos
- **Validación de origen**: Verificación de origen de peticiones (CORS configurado en servidor)

Validación y Sanitización de Datos

Estrategias de Validación

- **Validación cliente**: Verificación de formato y contenido antes de enviar datos al servidor
- **Validación servidor**: Segunda capa de validación independiente de la implementación del cliente
- **Esquemas de validación**: Uso de modelos Mongoose para validar estructuras de datos

Sanitización de Entrada/Salida

- **Módulo centralizado**: Implementación de `inputSanitizer.js` para limpieza consistente
- **Protección XSS**: Eliminación de caracteres potencialmente maliciosos en entradas de texto
- **Sanitización de objetos**: Procesamiento recursivo de objetos completos

Manejo de Errores y Excepciones

Estrategia de Manejo de Errores

- **Centralización**: Módulo `errorHandler.js` para procesar todos los errores de forma consistente
- **Mensajes amigables**: Traducción de errores técnicos a mensajes comprensibles para usuarios
- **Separación de entornos**: Logs detallados solo en desarrollo, mensajes genéricos en producción

Logging y Monitoreo

- **Niveles de logging**: Diferenciación entre errores críticos y advertencias
- **Contexto de errores**: Captura de información relevante sin exponer datos sensibles
- **Estructura para telemetría**: Preparación para futuras implementaciones de monitoreo

Protección contra Vulnerabilidades Comunes

Medidas Contra Ataques Comunes

- **Inyección**: Sanitización de entradas y uso de parámetros preparados (prepared statements)
- **XSS**: Escape de caracteres especiales en datos de usuario mostrados en la interfaz
- **CSRF**: Cabeceras específicas y validación de origen en peticiones
- **Broken Authentication**: Implementación de políticas de contraseñas y protección de sesiones

Pruebas de Seguridad

Estrategia de Testing

- **Tests unitarios**: Verificación de funcionamiento correcto de componentes de seguridad
- **Tests de integración**: Validación de flujos completos de autenticación y autorización
- **Validación manual**: Revisión periódica de código y configuraciones

Plan de Respuesta a Incidentes

Procedimiento Básico

1. **Detección**: Identificación de posibles brechas o comportamientos anómalos
2. **Contención**: Aislamiento del problema para minimizar impacto
3. **Erradicación**: Eliminación de la causa raíz
4. **Recuperación**: Restauración de servicios afectados
5. **Lecciones aprendidas**: Documentación y mejora de procesos

Recomendaciones Futuras

- **Implementar Two-Factor Authentication (2FA)**
- **Desarrollar un sistema de auditoría completo**
- **Realizar análisis estático de código de forma automatizada**
- **Implementar análisis de vulnerabilidades en dependencias**
- **Considerar el uso de OAuth 2.0 para autenticación con proveedores externos**

Conclusión

La implementación de estas medidas de seguridad proporciona una base sólida para proteger los datos de los usuarios y la integridad del sistema BookTracker. Este documento representa un compromiso continuo con la seguridad que debe evolucionar junto con la aplicación y las amenazas emergentes.

Este documento fue creado como parte del proyecto BookBox y representa el estado actual de seguridad e integridad del sistema.