

Grupo 4 - Integrantes: Miguel Luna, Anthony Gutiérrez y Luis Armijos

1. Tema:

Vulnerabilidades comunes en aplicaciones web (OWASP Top 10) y estrategias de mitigación aplicadas en el backend del proyecto.

2. Propósito

El estudiante analiza críticamente las principales vulnerabilidades de seguridad que pueden afectar su aplicación, identifica los riesgos presentes en su propio código e implementa medidas de mitigación documentadas según las buenas prácticas del estándar OWASP Top 10.

Con ello refuerza su capacidad para diseñar sistemas seguros y éticamente responsables.

3. Desarrollo de la actividad

3.1. Resumen técnico del proyecto (entorno, lenguaje, framework).

El proyecto se desarrolla en el entorno del sistema operativo Windows, utilizando Visual Studio Code como entorno de desarrollo integrado (IDE).

El backend está implementado en Java, encargado de la lógica principal del sistema y la gestión de los procesos internos.

El frontend se desarrolla en Python, haciendo uso del framework Flask para la creación de la interfaz web y la conexión con el backend.

Para el diseño de la interfaz de usuario se emplean tecnologías estándar como HTML, CSS y JavaScript, garantizando una presentación dinámica e intuitiva.

En cuanto al almacenamiento de datos, planeamos la integración de una base de datos relacional o no relacional, considerando opciones como MongoDB, MySQL o PostgreSQL, según los requerimientos finales del sistema.

3.2. Identificación de al menos 5 vulnerabilidades del OWASP Top 10 (2021) relevantes para su backend.

1) A01:2021 Broken Access Control (Control de acceso roto)

Esta vulnerabilidad ocurre cuando las restricciones de acceso a recursos o funciones no fueron aplicadas correctamente. En nuestro backend actual, si los endpoints no verifican adecuadamente los roles de usuario (por ejemplo, administrador o cliente), un atacante podría acceder o modificar información sin autorización.

Riesgo: Exposición o alteración de datos restringidos.

2) A02:2021 Cryptographic Failures (Fallos criptográficos)

El sistema almacena datos en archivos JSON sin cifrado, lo cual expone información sensible en caso de acceso no autorizado al servidor. La falta de cifrado durante la transmisión también representa un riesgo.

Riesgo: Pérdida de confidencialidad .

3) A07:2021 Identification and Authentication Failures (Fallas en identificación y autenticación)

Actualmente, el backend no cuenta con doble autenticación ni con mecanismos de gestión de sesión seguros. Esto facilita ataques de fuerza bruta o robo de sesiones.

Riesgo: Acceso no autorizado.

4) A08:2021 Software and Data Integrity Failures (Fallas en la integridad de software y datos)

El backend depende de archivos JSON sin verificación de integridad, un atacante podría manipular dichos archivos para alterar datos o ejecutar código malicioso.

Riesgo: Alteración de la información almacenada.

5) A09:2021 Security Logging and Monitoring Failures (Fallas en registro y monitoreo de seguridad)

La ausencia de un sistema de logs o alertas de seguridad impide detectar intentos de acceso no autorizado o comportamientos anómalos en el backend.

Riesgo: Ataques no detectados

3.3. Descripción del riesgo y del posible impacto de cada vulnerabilidad en su sistema.

- **A01 – Broken Access Control (Control de acceso roto)**

El principal riesgo radica en que usuarios sin privilegios puedan acceder o modificar recursos restringidos, como cuentas, horarios o descuentos si el backend no valida correctamente los roles o la identidad en cada solicitud, cualquier usuario podría realizar acciones administrativas o manipular información ajena, esto afectaría la integridad y confidencialidad de los datos, provocando pérdida de confianza y posibles fraudes dentro del sistema de gestión.

- **A02 – Injection (Inyección de código)**

Si los parámetros recibidos en las peticiones no son validados, podrían inyectarse comandos o estructuras maliciosas en consultas o archivos JSON esto puede alterar o borrar registros importantes, generar errores de ejecución o incluso permitir la ejecución remota de código. Además, en el frontend podría producirse XSS (Cross-Site Scripting), comprometiendo sesiones de usuario o tokens JWT. El impacto sería una manipulación no autorizada de datos y un posible secuestro de cuentas.

- **A03 – Identification and Authentication Failures (Fallos de autenticación e identificación)**

Una gestión deficiente de contraseñas o tokens podría permitir el acceso de atacantes a cuentas legítimas contraseñas almacenadas en texto plano, JWT sin caducidad o la ausencia de controles ante múltiples intentos de inicio de sesión debilitan la seguridad esto puede ocasionar robo de identidad, suplantación de usuarios o acceso total al sistema administrativo, afectando tanto la seguridad de los usuarios como la del servidor.

- **A04 – Cryptographic Failures (Fallos criptográficos)**

El uso inadecuado de claves o secretos de autenticación expone datos sensibles ya que si el JWT o las contraseñas se guardan en el repositorio o sin cifrado, un atacante podría falsificar tokens o descifrar información confidencial y esto produciría que se generará una filtración de datos personales, pérdida de privacidad y compromisos del backend, con graves consecuencias legales y reputacionales.

- **A05 – Security Misconfiguration (Configuración insegura)**

Configurar incorrectamente servicios como Flask, nginx o CORS puede dejar abiertos endpoints de depuración o permitir el acceso desde dominios no confiables. Asimismo, mantener permisos de archivos inseguros o ejecutar contenedores con privilegios elevados puede facilitar ataques internos y el impacto sería una exposición innecesaria de información sensible, riesgo de explotación remota y un aumento del vector de ataque hacia los servicios internos del sistema.

- **A06 – Vulnerable and Outdated Components (Componentes vulnerables u obsoletos)**

Usar librerías o dependencias desactualizadas en Java, Flask o React puede introducir vulnerabilidades conocidas, como inyecciones, XSS o ejecución remota de código si no se actualizan a tiempo, los atacantes podrían aprovechar esas debilidades para comprometer el sistema esto supondría una pérdida de disponibilidad y control del backend, además de posibles filtraciones de información crítica.

3.4. Medidas de mitigación implementadas o planificadas, explicando cómo se aplicaron en el código o la configuración.

Control de acceso - A01 – Broken Access Control

Medida planificada:

Se implementará un sistema de autenticación basado en tokens JWT, JSON Web Token, que valida la identidad y el rol del usuario en cada solicitud al backend.

En el código Java, cada endpoint verifica los permisos del usuario antes de ejecutar la operación, asegurando que solo los usuarios autorizados accedan a recursos específicos.

A03 – Identification and Authentication

Medida planificada:

Para fortalecer el proceso de autenticación y mitigar las vulnerabilidades detectadas se aplicaran las siguientes medidas de seguridad en el módulo inicio de sesión:

- **Cifrado de contraseñas:** Las contraseñas se almacenarán de forma cifrada utilizando BCrypt evitando almacenamiento en texto plano.
- **Uso de tokens JWT :** El servidor genera un JWT firmado con una clave secreta segura almacenada en variables de entorno el cual incluye el rol y la identidad de cada usuario para evitar la reutilización indefinida.
- **Limitación de intentos fallidos:** Se implementará un contador de intentos de inicio de sesión al superar el número determinado de intentos la cuenta se bloqueara temporalmente para prevenir ataques de fuerza bruta y reducir la adivinación de contraseñas

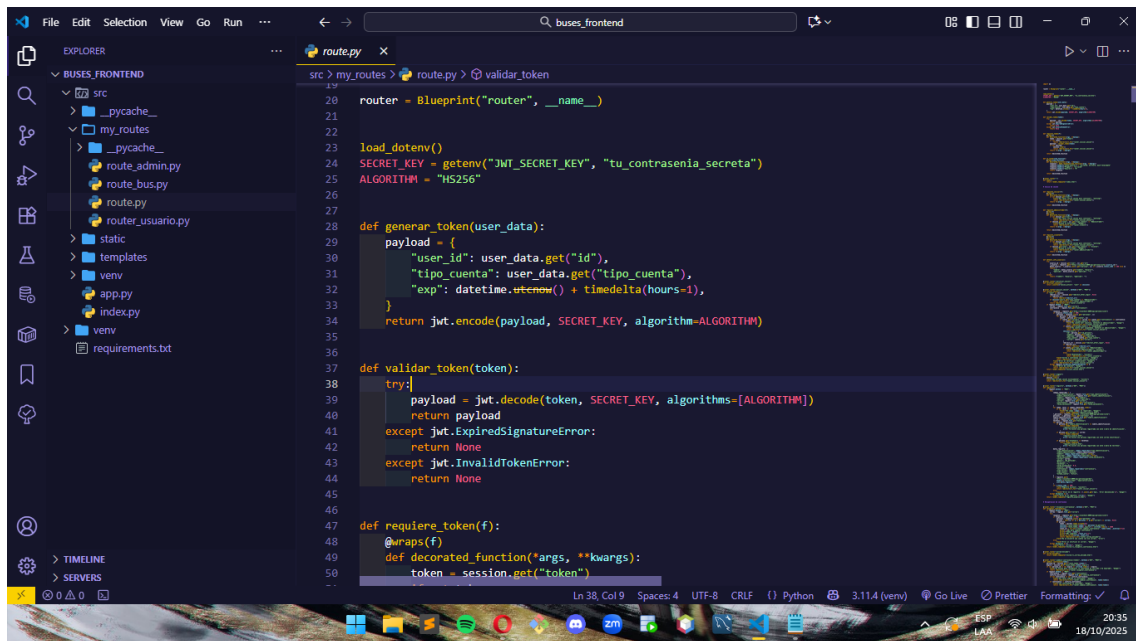
- **Validación y saneamiento de datos de entradas:** Se validan los campos de correo y contraseña en el backend y en la base de datos para evitar inyecciones de código o XSS.

Registro y monitoreo - A09:2021 – Security Logging and Monitoring Failures

Medida planificada:

Se implementará un sistema de registro (logging) y monitoreo de seguridad que permita registrar los eventos relevantes dentro del backend, tales como intentos de acceso no autorizados, errores en la autenticación, cambios de configuración y operaciones críticas realizadas por los usuarios.

3.5.Evidencias de verificación: capturas de pruebas, fragmentos de código o resultados de herramientas de análisis.



```

20 router = Blueprint("router", __name__)
21
22
23 load_dotenv()
24 SECRET_KEY = getenv("JWT_SECRET_KEY", "tu_contraseña_secreta")
25 ALGORITHM = "HS256"
26
27
28 def generar_token(user_data):
29     payload = {
30         "user_id": user_data.get("id"),
31         "tipo_cuenta": user_data.get("tipo_cuenta"),
32         "exp": datetime.utcnow() + timedelta(hours=1),
33     }
34     return jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
35
36
37 def validar_token(token):
38     try:
39         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
40         return payload
41     except jwt.ExpiredSignatureError:
42         return None
43     except jwt.InvalidTokenError:
44         return None
45
46
47 def requiere_token(f):
48     @wraps(f)
49     def decorated_function(*args, **kwargs):
50         token = session.get("token")

```

En esta imagen se muestra el módulo route.py, donde se implementa el control de acceso basado en tokens JWT, JSON Web Token.

El código permite generar, validar y exigir tokens para acceder a rutas protegidas dentro del sistema.

Mediante las funciones generar_token, validar_token y el decorador requiere_token, se garantiza que solo los usuarios autenticados puedan acceder a las funcionalidades restringidas.

Esta medida fortalece la seguridad del backend, evitando accesos no autorizados y mitigando vulnerabilidades como Broken Access Control, OWASP A01.

```

1 package com.buses.rest;
2
3 import controlador.servicios.Controlador_boleto;
4 import javax.ws.rs.core.MediaType;
5 import javax.ws.rs.core.Response;
6 import java.util.logging.Logger;
7 import java.util.logging.Level;
8 import javax.ws.rs.Produces;
9 import java.util.HashMap;
10 import javax.ws.rs.Path;
11 import javax.ws.rs.GET;
12
13
14 @Path("/buses")
15 public class MyResource {
16     private static final Logger logger = Logger.getLogger(MyResource.class.getName());
17
18     @GET
19     @Produces(MediaType.APPLICATION_JSON)
20     public Response getIt() {
21         HashMap<String, Object> response = new HashMap<>();
22         Controlador_boleto controlador = new Controlador_boleto();
23         String aux = "";
24         try {
25             controlador.getBoleto().setNumero_asiento(numero_asiento:1);
26             controlador.save();
27             aux = "Lista vacia: " + controlador.Lista_boletos().isEmpty();
28             logger.info("Guardado. Lista: " + aux);
29         }
30         catch (Exception e) {
31             logger.log(Level.SEVERE, "Error al guardar: " + e.getMessage(), e);
32             e.printStackTrace();
33             response.put(key:"error", e.getMessage());
34             return Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(response).build();
35         }
36         response.put(key:"message", value:"Boleto guardado exitosamente");
37         response.put(key:"Data", "Test: " + aux);
38         return Response.ok(response).build();
39     }
40 }

```

En el código del recurso REST MyResource, se implementó un sistema de registro (logging) utilizando la clase java.util.logging.Logger. Este mecanismo permite registrar eventos importantes del sistema, como operaciones exitosas, advertencias o errores, lo que facilita la detección temprana de fallos, la trazabilidad de eventos y la auditoría de acciones realizadas dentro del servidor.

```

@Path("/login")
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response login(Map<String, Object> body) {
    HashMap<String, Object> response = new HashMap<>();
    try {
        String correo = (String) body.get(key:"correo");
        String contrasenia = (String) body.get(key:"contrasenia");
        if (correo == null || contrasenia == null) {
            response.put(key:"mensaje", value:"correo y contrasenia requeridos");
            return Response.status(Response.Status.BAD_REQUEST).entity(response).build();
        }
        Controlador_cuenta cc = new Controlador_cuenta();
        // buscar cuenta por correo
        controlador.tda.lista.LinkedList<Cuenta> cuentas = cc.lista_cuentas();
        for (int i = 0; i < cuentas.getSize(); i++) {
            Cuenta c = cuentas.get(i);
            if (c.getCorreo().equalsIgnoreCase(correo)) {
                // Lockout policy (configurable via env)
                int maxAttempts = 5;
                long lockSeconds = 300L; // 5 minutes
                String maxAttemptsEnv = System.getenv(name:"AUTH_MAX_ATTEMPTS");
                String lockSecondsEnv = System.getenv(name:"AUTH_LOCK_SECONDS");
                if (maxAttemptsEnv != null) {
                    try { maxAttempts = Integer.parseInt(maxAttemptsEnv); } catch (NumberFormatException ignored) {}
                }
                if (lockSecondsEnv != null) {
                    try { lockSeconds = Long.parseLong(lockSecondsEnv); } catch (NumberFormatException ignored) {}
                }
                long now = System.currentTimeMillis();
                if (c.getLockedUntil() != null && c.getLockedUntil() > now) {
                    long remainingMs = c.getLockedUntil() - now;

```

En el archivo Auth_api.java se implementó toda la lógica del inicio de sesión junto con medidas de seguridad para evitar ataques o accesos no autorizados. Primero, el método login() valida que el usuario haya enviado correctamente el correo y la contraseña, y si falta alguno devuelve un error 400 con un mensaje indicando que los campos son requeridos. Luego, se agregó un control de intentos fallidos, para evitar ataques de fuerza bruta para esto se configuró una política de bloqueo temporal usando variables de entorno llamadas AUTH_MAX_ATTEMPTS y AUTH_LOCK_SECONDS, que determinan el número máximo de intentos permitidos (por defecto 5) y el tiempo de bloqueo en segundos (por defecto 300, es decir 5 minutos). Cada vez que un usuario ingresa una contraseña incorrecta, el sistema incrementa el contador de intentos fallidos (failedAttempts) y, si se supera el límite, se bloquea la cuenta temporalmente asignando un valor a lockedUntil y durante ese tiempo el backend responde con el código HTTP 423 (Locked), impidiendo seguir probando contraseñas hasta que se cumpla el tiempo del bloqueo.

Todos los cambios de estado de la cuenta (como los intentos fallidos o el bloqueo) se guardan usando el Cuenta_dao, así que incluso si se reinicia el servidor, la información se mantiene.

3.6. Reflexión personal: qué aprendió sobre seguridad durante esta unidad y cómo mejoraría el diseño futuro del sistema.

- Durante esta unidad ha quedado demostrada la importancia de incorporar la seguridad desde las primeras etapas del desarrollo y no dejarla como un aspecto secundario. Comprendí que vulnerabilidades como la inyección de código, los fallos de autenticación o las configuraciones inseguras pueden comprometer completamente un sistema, incluso si su lógica funcional es correcta.
- Analizar las vulnerabilidades del OWASP Top 10 (2021) permitió identificar errores comunes que pueden comprometer la integridad, confidencialidad y disponibilidad de la información. También entendí que muchos riesgos no surgen únicamente del código, sino de una falta de planificación o de configuraciones inseguras.
- En el diseño futuro del proyecto, mejoraríamos la arquitectura implementando un modelo más modular y seguro, separando claramente las capas de presentación, lógica y datos. Con ello, el sistema no solo sería funcional, sino también resistente ante ataques y confiable para los usuarios.
- Las vulnerabilidades más comunes como las inyecciones, los fallos de autenticación o las configuraciones inseguras pueden comprometer toda la aplicación, incluso si su lógica es funcional por eso se debe reforzar la arquitectura adoptando principios de defensa en profundidad, validaciones en múltiples capas y un monitoreo constante de los accesos y errores así de esta manera, el sistema no solo ofrecerá un buen rendimiento, sino también confianza, protección en el manejo de la información.

4. Bibliografía

- [1] OWASP Foundation (2023). OWASP Top 10 – Web Application Security Risks.
- [2] Auth0. JWT Handbook. <https://auth0.com/learn/json-web-tokens>
- [3] Spring Security / Django Auth / Express JWT Documentation.
- [4] PlantUML / Mermaid C4 Model Reference.\
- [5] Top 10 most pressing web security challenges: OWASP TOP TEN. [Top 10 Most Pressing Web Security Challenges](#)