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

Integrantes:
- Francisco Javier Jaramillo Castro.
- Juan Carlos Alverca Beltran.
- José Francisco Riofrío Maldonado.

Resumen técnico

Como entorno principal para la programación y depuración del código. El backend está implementado con FastAPI, encargado de procesar la lógica interna y coordinar las operaciones del sistema. Esta herramienta permite una comunicación eficiente entre los distintos módulos, garantizando un rendimiento óptimo. El frontend se construye utilizando React Native, ofreciendo una interfaz adaptable, moderna y centrada en la experiencia del usuario. Su enfoque multiplataforma facilita la compatibilidad con diferentes dispositivos.

Para la gestión de los datos, se integra MariaDB como sistema de base de datos, proporcionando un entorno robusto y confiable para el almacenamiento de la información. Este motor relacional asegura la integridad y disponibilidad de los datos en todo momento. Además, su estructura flexible permite escalar el sistema conforme crezcan las necesidades del proyecto. En conjunto, estas tecnologías brindan una arquitectura sólida, eficiente y alineada con los objetivos de desarrollo del sistema.

Vulnerabilidades del OWASP Top 10 (2021) relevantes para su backend.

1. **A01:2021 Broken Access Control (Pérdida de Control de Acceso)**

Este tipo de vulnerabilidad ocurre cuando un sistema no restringe adecuadamente las acciones que los usuarios pueden realizar, en consecuencias un atacante puede acceder a información o funciones que deberían estar protegidas. Su objetivo es garantizar que cada usuario solo tenga acceso a los recursos que le corresponden según su rol o permisos, para evitarlo se deben aplicar políticas de control de acceso sólidas y verificar la autenticación en cada solicitud.

2. **A03:2021 Injection (Inyección)**

Las vulnerabilidades de inyección surgen cuando un atacante logra insertar código malicioso en una aplicación, aprovechando entradas no válidas. Su función es alertar sobre la necesidad de validar y filtrar correctamente los datos que se reciben del usuario. Los tipos más comunes incluyen inyección SQL, de comandos o de scripts, que pueden comprometer bases de datos o ejecutar acciones no autorizadas. La prevención se logra mediante el uso de consultas preparadas y validaciones estrictas**.**

3. **A07:2021 Identification and Authentication Failures (Fallos de Identificación y Autenticación)**

Se relaciona con errores en los mecanismos que verifican la identidad de los usuarios. Su objetivo es garantizar que solo los usuarios legítimos puedan acceder al sistema. Las causas más comunes incluyen contraseñas débiles, falta de autenticación multifactor o

sesiones mal gestionadas. Cuando estos fallos ocurren, los atacantes pueden suplantar identidades o acceder a información confidencial. Para mitigarlo, es esencial aplicar políticas de contraseñas seguras y autenticación robusta.

4. **A05:2021 Security Misconfiguration (Configuración de Seguridad Incorrecta)**

Esta falla ocurre cuando un sistema, servidor o aplicación tiene una configuración insegura o por defecto. El propósito de esta categoría es fomentar prácticas adecuadas de configuración y mantenimiento. Un entorno mal configurado puede exponer información sensible, permitir accesos indebidos o facilitar otros tipos de ataques. Se recomienda revisar periódicamente las configuraciones, eliminar componentes innecesarios y mantener las actualizaciones de seguridad al día.

5. **A02:2021 Cryptographic Failures (Fallos Criptográficos)**

Se refiere a la incorrecta implementación o ausencia de mecanismos de cifrado en el manejo de datos sensibles. Su propósito principal es proteger la confidencialidad e integridad de la información, especialmente durante la transmisión o el almacenamiento. Cuando se cometen errores en los algoritmos o claves de cifrado, los atacantes pueden interceptar o alterar los datos. Es fundamental emplear protocolos modernos y seguros, así como una gestión adecuada de las claves criptográficas.

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

1. **A01:2021 Broken Access Control (Pérdida de Control de Acceso)**

Descripción del Riesgo: El riesgo principal es que las rutas de la API son públicas y no verifican quién está haciendo la solicitud. Cualquier persona que conozca la URL puede intentar modificar o eliminar datos, necesitando únicamente adivinar el id de un recurso (persona, artículo, cliente, etc.). No hay ninguna validación que confirme que el usuario que quiere borrar un artículo es realmente el dueño de ese artículo.

Impacto en el Sistema:

Destrucción de Datos: Un usuario malintencionado podría crear un script simple para recorrer todos los IDs numéricos y llamar a DELETE **/articulos/api/artículo/delete/{id}.** En cuestión de segundos, podría borrar todos los artículos de la plataforma.

Robo de Información Personal: Un atacante podría usar la ruta GET **/personas/api/persona/{id}** para extraer los datos personales (nombre, apellidos, teléfono, identificación) de todos los usuarios registrados en el sistema.

Suplantación y Fraude: Un usuario podría usar PUT **/articulos/api/artículo/update** para cambiar la descripción o el estado de un artículo que no le pertenece, por ejemplo, marcándolo como "intercambiado" para sabotear un trueque ajeno.

2. **A03:2021 Injection (Inyección)**

Descripción del Riesgo: El riesgo de inyección SQL ocurre cuando un atacante puede manipular la estructura de una consulta a la base de datos enviando código SQL malicioso en un campo de entrada. Afortunadamente, el riesgo actual es bajo porque usamos SQLAlchemy de forma segura, que neutraliza este tipo de ataques. Sin embargo, si en el futuro construimos una consulta manualmente concatenando strings, el riesgo se volvería crítico.

Impacto en el Sistema:

Exposición Total de la Base de Datos: Un atacante podría usar una consulta manipulada en un campo de búsqueda para extraer toda la información de todas las tablas, incluyendo las contraseñas de los clientes, datos personales y todos los artículos.

Eliminación Masiva de Datos: Con una sola consulta inyectada, podría ejecutar un DROP TABLE y eliminar tablas enteras, causando un daño irreparable a la plataforma.

3. **A07:2021 Identification and Authentication Failures (Fallos de Identificación y Autenticación)**

Descripción del Riesgo: Hemos detectado un fallo grave en cómo gestionamos las credenciales en mi módulo Cliente. El principal riesgo es que estamos almacenando las contraseñas de los usuarios en la base de datos sin aplicarles ningún tipo de 'hash' criptográfico. Esto significa que se guardan en texto plano o de una forma fácilmente reversible.

Impacto en el Sistema: El impacto de esto es enorme, principalmente para mis usuarios. Si mi base de datos se viera comprometida, un atacante obtendrá una lista completa de usuarios y sus contraseñas exactas. Dado que muchas personas utilizan las mismas contraseñas, esto les daría acceso a sus correos, redes sociales y otros servicios. Para mi plataforma, una filtración de este tipo significa una pérdida total de la confianza y la reputación.

4. **A05:2021 Security Misconfiguration (Configuración de Seguridad Incorrecta)**

Descripción del Riesgo: Este riesgo proviene de dos frentes: La posibilidad de que la aplicación muestre mensajes de error detallados en un entorno de producción y dos tener datos sensibles, como la contraseña de la base de datos.

Impacto en el Sistema: Los mensajes de error detallados (stack traces) son un mapa para los atacantes. Les informamos sobre la estructura de nuestros archivos, las versiones de las librerías que usamos y cómo funciona nuestro código internamente. Con esta información, pueden buscar vulnerabilidades conocidas para esas versiones específicas y planificar un ataque dirigido.

5. **A02:2021 Cryptographic Failures (Fallos Criptográficos)**

Descripción del Riesgo: El riesgo aquí es la transmisión de datos (sin cifrar). La API, al ejecutarse localmente, funciona sobre HTTP. Si se despliega en producción sin

configurar HTTPS, toda la comunicación entre los usuarios y el servidor es vulnerable a ser espiada.

Impacto en el Sistema: Cuando un usuario se registra o inicia sesión, su usuario y contraseña viajan como texto plano por la red, un atacante en la misma red Wi-Fi puede interceptar este tráfico y capturar las credenciales en tiempo real sin necesidad de atacar la base de datos.

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

Medida Planificada:

- **A01:2021 Broken Access Control (Pérdida de Control de Acceso)**

Implementaremos un sistema de autenticación basado en tokens JWT. Al iniciar sesión, cada usuario recibirá un token único que deberá presentar en las solicitudes a rutas protegidas. En mi backend con FastAPI, cada endpoint crítico, verificará este token para identificar al usuario. Antes de ejecutar la operación, mi código comprobará que el ID del usuario extraído del token coincida con el ID del propietario del recurso, asegurando así que un usuario solo pueda gestionar sus propios datos.

- **A07:2021 Identification and Authentication Failures (Fallos de Identificación y Autenticación)**

Para proteger las credenciales de mis usuarios, implementaremos el hasheo de contraseñas utilizando la biblioteca passlib con el robusto algoritmo bcrypt. Al momento de que un cliente se registre o actualice su contraseña, esta se convertirá en un hash criptográfico irreversible antes de ser almacenada en la base de datos. De esta forma, en el proceso de login, la contraseña que ingrese el usuario se comparará contra el hash almacenado, nunca contra la contraseña en texto plano, mitigando el riesgo de exposición en caso de una brecha de datos.

- **A05:2021 Security Misconfiguration (Configuración de Seguridad Incorrecta)**

Mantendremos toda la información importante, como las contraseñas de la base de datos, fuera del código principal del proyecto. Para esto, usaré un archivo especial donde se guardan estos datos de forma separada, evitando que se publiquen o compartan por error. Además, cuando la aplicación esté funcionando de manera oficial, se desactiva el modo de pruebas para que no se muestre información interna o mensajes de error que puedan revelar detalles del sistema. Esto se aplicará más adelante en el lanzamiento del proyecto.

- **A02:2021 Cryptographic Failures (Fallos Criptográficos)**

Para cuidar la información que viaja entre los usuarios y el sistema, haré que toda la comunicación se realice de forma segura mediante el uso de HTTPS. Esto garantiza que los datos personales, como contraseñas o información privada, estén protegidos y no puedan ser vistos por otras personas mientras se envían. De esta forma, se evita que alguien pueda interceptar o robar la información cuando se conecta desde una red insegura.

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

- **A01:2021 Broken Access Control (Pérdida de Control de Acceso)**

```python
@articulo_router.delete("/api/articulo/delete/{articulo_id}")
def delete_articulo(articulo_id: int):

    existing_articulo = conn.execute(select(articulos).where(articulos.c.id == articulo_id)).fetchone()
    if existing_articulo is None:
        raise HTTPException(status_code=404, detail="Artículo no encontrado")

    conn.execute(articulos.delete().where(articulos.c.id == articulo_id))
    return {"message": "Artículo eliminado correctamente"}

# Seguridad mejorada: solo el propietario puede eliminar su artículo

@articulo_router.delete("/api/articulo/delete/{articulo_id}")
def delete_articulo(articulo_id: int, usuario_actual: dict = Depends(get_usuario_actual)):

    articulo = conn.execute(select(articulos).where(articulos.c.id == articulo_id)).fetchone()

    if not articulo:
        raise HTTPException(status_code=404, detail="Artículo no encontrado")

    if articulo.id_usuario != usuario_actual["id"]:
        raise HTTPException(status_code=403, detail="Acción no permitida. No eres el propietario de este artículo.")

    conn.execute(articulos.delete().where(articulos.c.id == articulo_id))
    return {"message": "Artículo eliminado correctamente"}
```

- **A07:2021 Identification and Authentication Failures (Fallos de Identificación y Autenticación)**

```python
#Toma la contraseña en texto plano que viene del usuario.
#Reemplaza en el diccionario new_cliente por su versión hasheada.
#Guarda ese diccionario en la base de datos. Por lo tanto, lo que se almacena no es la contraseña original, sino el hash.
@cliente_router.post("/api/cliente/create")
def create_cliente(data_cliente: ClienteSchema):
    new_cliente = data_cliente.dict()

    hashed_password = pwd_context.hash(new_cliente["contrasena"])
    new_cliente["contrasena"] = hashed_password

    with engine.begin() as conn:

        conn.execute(clientes.insert().values(new_cliente))
    return {"message": "Cliente creado correctamente"}
```

- **A05:2021 Security Misconfiguration (Configuración de Seguridad Incorrecta)**

```python
import os
from dotenv import load_dotenv
from sqlalchemy import create_engine, MetaData

# Se leen las credenciales de la BD desde variables de entorno para evitar exponerlas en el código.
# La configuración se carga desde un archivo .env usando python-dotenv.

load_dotenv()

DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")
DB_NAME = os.getenv("DB_NAME")
db_url = f"mariadb+mariadbconnector://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

engine = create_engine(db_url)
```

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

Aprendimos que la seguridad es una parte esencial del desarrollo de cualquier sistema, especialmente en el backend. Antes pensaba que bastaba con tener contraseñas o iniciar sesión, pero ahora entiendo que la seguridad implica muchas más cosas, como proteger los datos, validar correctamente la información y evitar configuraciones inseguras.

Comprendimos que los errores pequeños pueden tener grandes consecuencias. Por ejemplo, si una contraseña se guarda sin encriptar o si la API no verifica bien los permisos, un atacante podría acceder a información privada o incluso borrar datos del sistema.

- También la importancia de usar buenas prácticas de seguridad, como:
- Cifrar las contraseñas con algoritmos seguros (bcrypt).
- Usar tokens JWT para controlar quién accede a cada recurso.
- Activar HTTPS para que los datos no viajen sin protección.
- Revisar las configuraciones del servidor para evitar fugas de información.

Para el futuro, se debe mejorar el sistema pensando en la seguridad desde el diseño inicial. No esperaría a que el sistema esté terminado para protegerlo, sino que lo haría parte del proceso de desarrollo.

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.