Grupo 5: Eberson Guayllas – Wilmer Pardo y Santiago Tamayo

Tema

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

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.

Actividad

1. **Resumen técnico del proyecto (entorno, lenguaje, framework).**

Característica	**Detalle**
Nombre del Proyecto	**Codium** (Plataforma de gamificación para retos de programación).
Arquitectura	Cliente-Servidor (Análisis C4 previo).
Backend	Python, Flask, CORS.
Base de Datos	MySQL (codium_db).
Conexión DB	mysql.connector. Credenciales almacenadas directamente en Backend/src/database/db.py.
Control de Versiones	Gitflow (ramas main, DEVELOPE, fronted).
Frontend	React (Vite).

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

OWASP ID	**Nombre de la Vulnerabilidad**	**Riesgo en Codium**
A02:2021	Cryptographic Failures (Fallas Criptográficas)	Las contraseñas se encuentras guardadas en texto plano, no cuentan con encriptación hash
A03:2021	Injection (Inyección)	Uso potencial de consultas SQL no parametrizadas, exponiendo el sistema a ataques de Inyección SQL.
A04:2021	Insecure Design (Diseño Inseguro)	La configuración de la base de datos (credenciales y contraseña) está almacenada directamente en el código fuente (Backend/src/database/db.py).
A05:2021	Security Misconfiguration (Fallo de Configuración de Seguridad)	Uso de CORS abierto (CORS(app)) en Flask, permitiendo a cualquier dominio realizar peticiones al backend.

A08:2021	Software and Data Integrity Failures (Fallo de Integridad de Software y Datos)	Falta de validación y control de la data recibida en el *backend*, permitiendo la inyección de datos inesperados.

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

A02:2021 – Si las contraseñas se almacenan en un texto plano, una filtración de datos podría revelar todas las contraseñas de los usuarios. Además, el manejo de token_refresco en texto plano en la BD puede ser un riesgo.

A03:2021 –Si las consultas SQL no usaran la sustitución de variables segura del conector de MySQL (es decir, concatenando directamente las entradas del usuario a la *query*), un atacante podría enviar comandos SQL maliciosos a través de las rutas API. Un atacante podría obtener información sensible de la tabla 'persona'.

A04:2021 - El uso directo de credenciales de base de datos (DB_CONFIG) dentro del archivo de código fuente es una práctica de diseño insegura. Si el código fuente se filtrara, las credenciales de acceso total a la BD quedarían expuestas. Además, que faltan patrones de diseños seguros.

A05:2021 – El uso de CORS(app) sin restringir dominios en Backend/app.py permite que cualquier sitio web acceda al API. Esto puede permitir que un sitio malicioso pueda enviar solicitudes en nombre del usuario (si existieran sesiones activas), provocando posibles ataques CSRF.

A08:2021 – En el modelo persona, especialmente al realizar un post, si no se controla los campos a actualizar, un atacante podría intentar campos críticos. Por lo que puede existir un cambio de roles o implantación de identidad.

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

A02:2021

Medida Planificada: Uso de hashing seguro

Explicación: La vulnerabilidad de almacenar contraseñas en texto plano se resolvió implementando el *hashing* seguro de contraseñas mediante la librería werkzeug.security de Flask.

A03:2021

Medida planificada: Uso exclusivo de consultas parametrizadas

Explicación: Se ha implementado en personaModel todas las queries que utilizan el formado '%s' y la tupla de datos separada en el método cursor.execute(query, (data,)). Esto garantiza que los datos del usuario se envíen por separado del comando SQL, mitigando la Inyección SQL.

A04:2021

Medida Planificada: Incorporar seguridad desde el diseño (roles).

Explicación: Seguir el principio de menor privilegio para el usuario DB app_user. Se limitaron los permisos del usuario que se conecta desde el backend, este se encuentra al final del archivo schema.sql

A05:2021

Medida Planificada: Restricción estricta de CORS.

Explicación: Cuando la aplicación se despliegue se restringirá para que solo la acceda la IP especifica.

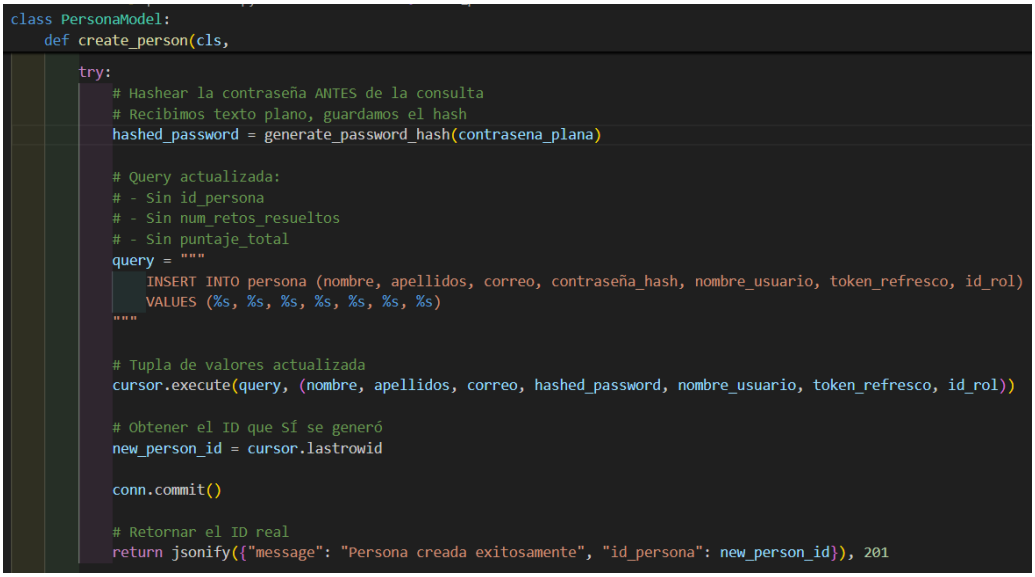
A08:2021

Medida Planificada: Control de campos actualizables

Explicación: En personaModel se utiliza una lista blanca. Solo los campos de esta lista se utilizan para construir la query UPDATE.

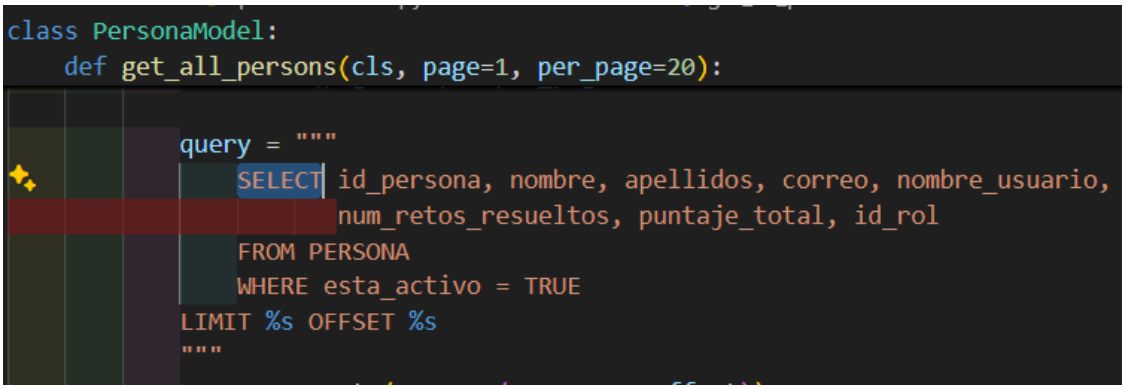
5. Evidencias de verificación: capturas de pruebas, fragmentos de código o resultados de herramientas de análisis.

A02:2021



```
class PersonaModel:
    def create_person(cls,
        try:
            # Hashear la contraseña ANTES de la consulta
            # Recibimos texto plano, guardamos el hash
            hashed_password = generate_password_hash(contrasena_plana)

            # Query actualizada:
            # - Sin id_persona
            # - Sin num_retos_resueltos
            # - Sin puntaje_total
            query = """
                INSERT INTO persona (nombre, apellidos, correo, contraseña_hash, nombre_usuario, token_refresco, id_rol)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """

            # Tupla de valores actualizada
            cursor.execute(query, (nombre, apellidos, correo, hashed_password, nombre_usuario, token_refresco, id_rol))

            # Obtener el ID que SÍ se generó
            new_person_id = cursor.lastrowid

            conn.commit()

            # Retornar el ID real
            return jsonify({"message": "Persona creada exitosamente", "id_persona": new_person_id}), 201
```

A03:2021



```
class PersonaModel:
    def get_all_persons(cls, page=1, per_page=20):

            query = """
                SELECT id_persona, nombre, apellidos, correo, nombre_usuario,
                       num_retos_resueltos, puntaje_total, id_rol
                FROM PERSONA
                WHERE esta_activo = TRUE
            LIMIT %s OFFSET %s
            """
```

```
class PersonaModel:
    def delete_person(cls, id_persona):
        query = "UPDATE persona SET esta_activo = FALSE WHERE id_persona = %s"
        cursor.execute(query, (id_persona,))
```

A04:2021

```
schema.sql
236  CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'S@ntiagoñ2002';
237  GRANT SELECT, INSERT, UPDATE, DELETE ON codium_db.* TO 'app_user'@'localhost';
238  FLUSH PRIVILEGES;
239
240
```

A08:2021

```
Backend > src > models > personaModel.py > PersonaModel > update_person
  6 class PersonaModel:
134     def update_person(cls, id_persona, update_data):
139         # --- INICIO DE CORRECCIÓN DE SEGURIDAD ---
140         # Lista blanca de campos permitidos para actualización
141         allowed_fields = ['nombre', 'apellidos', 'nombre_usuario', 'token_refresco']
142
143         fields_to_update = []
144         values = []
145
146         for key, value in update_data.items():
147             if key in allowed_fields:
148                 fields_to_update.append(f"{key} = %s")
149                 values.append(value)
150             else:
151                 # Opcional: registrar intento de actualizar campo no permitido
152                 print(f"ADVERTENCIA: Intento de actualizar campo no permitido: {key}")
153
154         if not fields_to_update:
155             return jsonify({"error": "No se proporcionaron datos válidos para actualizar"}), 400
156         # --- FIN DE CORRECCIÓN DE SEGURIDAD ---
```

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

La seguridad no es algo opcional o que se puede pasar por alto, sino una responsabilidad que debe abordarse desde las primeras etapas de diseño. El uso de patrones y herramientas de frameworks ayudan a resolver automáticamente algunas vulnerabilidades, pero esto puede generar una falsa sensación de seguridad, en fallos como la falta de control de acceso (A01) son simplemente errores por parte del desarrollador al no proteger las rutas de la API, actualmente en el proyecto Codium es el mayor riesgo ya que se puede manipular totalmente los datos del usuario sin autenticación

En un rediseño futuro se plantea integrar un sistema completo de gestión de usuarios con roles y permisos, aplicando autenticación basada en tokens y cifrado seguro de contraseñas. Otro aspecto relevante seria la arquitectura del backend fortaleciéndola mediante principios de diseño seguro, como separación de responsabilidades, validación en cada capa y el uso de librerías actualizadas.

Preguntas orientadoras

1. ¿Qué vulnerabilidad del OWASP Top 10 representa el mayor riesgo para tu proyecto y por qué?

La vulnerabilidad A04:2021 – Diseño inseguro, almacenar las credenciales de la base de datos (app_user y su contraseña) en texto plano dentro del código fuente genera riesgos si el repositorio del código se vuelve público un atacante puede obtener acceso a las credenciales del usuario de la BD. Aunque el usuario app_user tiene permisos limitados, el atacante podría usar estas credenciales para acceder directamente a la BD.

2. ¿Cómo validas que las medidas implementadas realmente mitigan la vulnerabilidad detectada?

La validación se realiza mediante pruebas estáticas (código) y pruebas dinámicas (ejecución)

3. ¿Qué relación encuentras entre las vulnerabilidades OWASP y los componentes del modelo C4 (backend, base de datos, API Gateway)?

El backend es un punto crítico en la seguridad, es por ello que vulnerabilidades como A02, A03 y A04 se originan aquí (en la lógica del negocio). Por ello en el bakend se deben implementar las mitigaciones.

En la base de datos siendo esta una capa sensible, es objetivo de ataques A03 ya que almacena contraseñas y tokens de refresco, para evitar esto se aplica métodos directamente en el backend que protege la BD.

El proyecto no tiene un API Gateway pero esta capa debería manejar protección de ataques relacionados con el tráfico de red.