Informe de Auditoría de Seguridad – API de Gestión de Clientes

1. Diagnóstico general del entorno

Tras revisar el código fuente y la configuración de la aplicación, se observa que la API de gestión de clientes presenta una superficie de ataque amplia debido a:

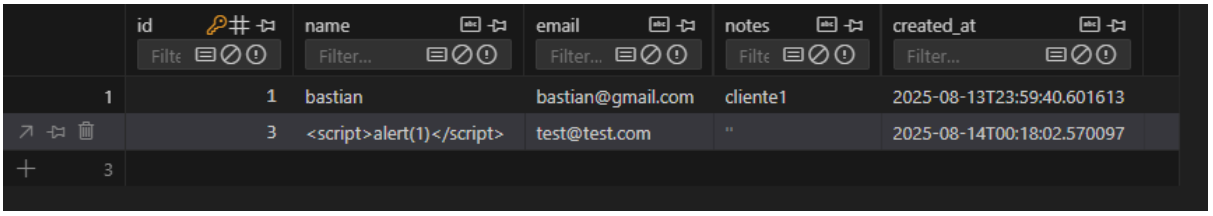
- Configuración **CORS** excesivamente permisiva (`allow_origins=["*"]` con `allow_credentials=True`).
- Falta de mecanismos de autenticación/autorización en todos los endpoints del CRUD.
- Uso de SQLite local (`app/data/lab.db`) sin restricciones de acceso ni cifrado.
- Vulnerabilidades de inyección de código (SQL Injection) por concatenación de strings.
- Renderizado de HTML sin sanitización (XSS almacenado).
- Rutas que permiten cargas cross-origin vía `application/x-www-form-urlencoded` sin validación (riesgo de CSRF).
- Middleware que inyecta errores HTTP 500 de forma aleatoria, exponiendo potencialmente trazas internas.

2. Vulnerabilidades identificadas y justificación técnica

2.1 Cross-Site Scripting (XSS) almacenado

- **Ubicación:** `render_clients_table` y `/clients/render`.
- **Descripción:** Los campos `name` y `notes` se insertan directamente en el HTML sin escape ni sanitización.

```
basti@Blandskron MINGW64 ~
$  curl -X POST http://localhost:8000/clients \
  -H "Content-Type: application/json" \
  -d '{"name":"<script>alert(1)</script>","email":"test@test.com"}'
{"id":3,"name":"<script>alert(1)</script>","email":"test@test.com","notes":null,"created_at":"2025-08-14T00:18:02.570097"}
basti@Blandskron MINGW64 ~
$
```



	id	name	email	notes	created_at
1	1	bastian	bastian@gmail.com	cliente1	2025-08-13T23:59:40.601613
	3	<script>alert(1)</script>	test@test.com	''	2025-08-14T00:18:02.570097
3					

Ejemplo de explotación:

curl -X POST http://localhost:8000/clients \

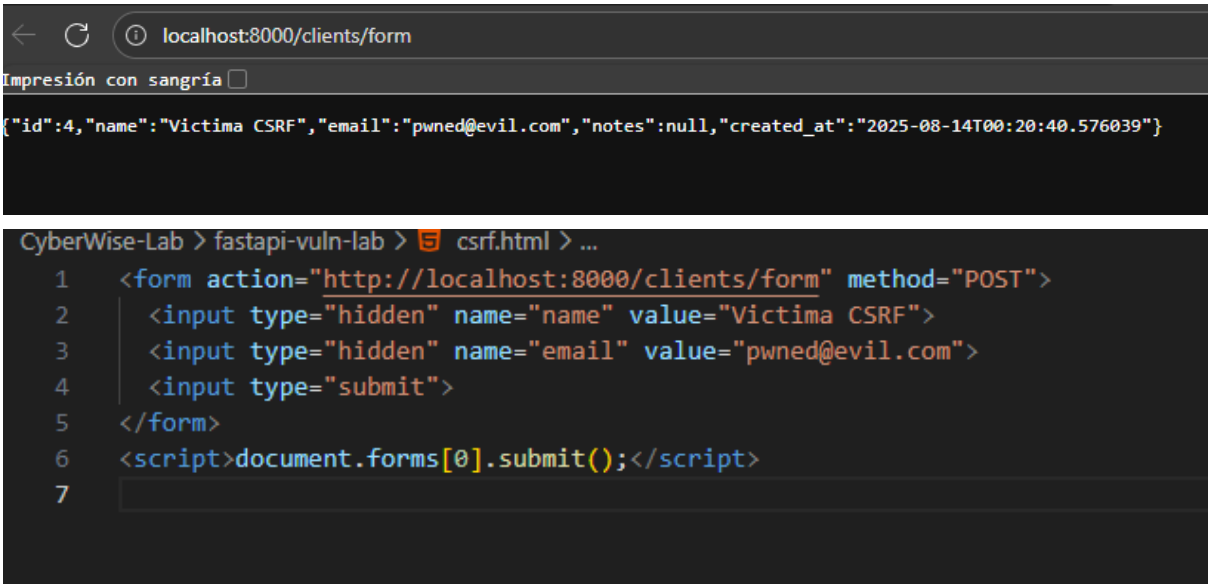
-H "Content-Type: application/json" \

-d '{"name":"<script>alert(1)</script>","email":"test@test.com"}'

- Al visitar `/clients/render`, el script se ejecuta en el navegador.
- **Riesgo:** Un atacante puede ejecutar código JavaScript en el navegador de las víctimas, robar cookies, realizar acciones en su nombre o redirigir a sitios maliciosos.

2.2 Cross-Site Request Forgery (CSRF)

- **Ubicación:** Rutas `/clients/form` y `/clients/{id}/form`.
- **Descripción:** No se valida el origen ni se utiliza token CSRF. Combinado con CORS permisivo, un atacante puede forzar peticiones desde otro dominio.



Ejemplo de explotación (HTML malicioso en otro sitio):

<form action="http://localhost:8000/clients/form" method="POST">

<input type="hidden" name="name" value="Victima CSRF">

<input type="hidden" name="email" value="pwned@evil.com">

<input type="submit">

</form>

<script>document.forms[0].submit();</script>

-
- **Riesgo:** Un usuario autenticado podría ser inducido a ejecutar acciones sin su consentimiento, comprometiendo la integridad de los datos.

2.3 SQL Injection

- **Ubicación:** `client_repository.py` en `get_client`, `create_client`, `update_client`, `delete_client`, `search_clients`.

- **Descripción:** Las consultas SQL se construyen mediante concatenación de strings y variables del usuario.

Ejemplo de explotación:

curl "http://localhost:8000/clients/search?q=%25' OR '1'='1"

- Devuelve todos los registros ignorando filtros.

- **Riesgo:** Un atacante puede manipular consultas, exfiltrar información sensible, modificar o eliminar datos.

2.4 Falta de autenticación robusta

- **Ubicación:** Todos los endpoints del CRUD.

- **Descripción:** No existe autenticación o autorización para crear, modificar o eliminar clientes.

Ejemplo de explotación:

```
curl -X DELETE http://localhost:8000/clients/1
```

- Elimina registros sin credenciales.

- **Riesgo:** Cualquier usuario, incluso no autenticado, puede manipular o borrar datos críticos.

2.5 Manejo inseguro de errores (500 y trazas)

- **Ubicación:** `ChaosMiddleware` y `/debug/random-500`.

- **Descripción:** El middleware devuelve errores HTTP 500 de forma aleatoria. Algunos errores pueden incluir mensajes internos o trazas.

- **Ejemplo de explotación:**

    - Forzar `/debug/random-500` repetidas veces para buscar mensajes internos.

- **Riesgo:** Las trazas pueden revelar rutas de archivos, lógica interna o configuraciones sensibles, facilitando ataques más dirigidos.

3. Conclusión y nivel de riesgo

El sistema presenta múltiples vulnerabilidades de alto riesgo que afectan la **confidencialidad**, **integridad** y **disponibilidad** de la información. Las más críticas son la ausencia de autenticación, la inyección SQL y el XSS, que permiten un compromiso total del entorno.

Nivel de riesgo global: ALTO
Recomendación: Aislar el sistema en un entorno controlado, sin acceso a redes públicas, y aplicar medidas de mitigación antes de cualquier despliegue productivo.