🛡️ Informe Técnico — Mitigación de Exposición del Endpoint `/transcribe`

Severidad

Crítica — El endpoint expone un servicio externo de pago (Gemini) sin autenticación, permitiendo abuso ilimitado, ataques DoS y consumo económico no controlado.

Resumen de la Falla

El endpoint `/transcribe`:

- No requiere autenticación ni login.
- Acepta cualquier archivo de audio enviado por terceros.
- No aplica límites de tamaño ni frecuencia.
- Funciona como proxy público hacia Gemini, permitiendo uso ilimitado y facturación inesperada.

Impacto potencial:

- Consumo masivo de la API de IA a costa del propietario.
- Posible saturación del servidor (DoS) mediante archivos grandes o flood de peticiones.
- Automatización externa con scripts (`curl`, bots) para explotar el servicio.

Objetivo de la Solución

Permitir que el usuario pueda transcribir audio **sin necesidad de login**, pero **evitando abuso** y uso no autorizado de la API de IA.

Solución Propuesta

Se propone un sistema basado en **CAPTCHA + tokens de un solo uso**, implementable **sin Redis**:

1. **Integrar un CAPTCHA invisible** en el frontend (reCAPTCHA, hCaptcha o Cloudflare Turnstile) para evitar bots.
2. Crear un **endpoint** `/session-token` que:
    - Recibe el token del CAPTCHA
    - Valida que sea correcto
    - Genera un **token de sesión efímero de un solo uso**

3. Modificar `/transcribe` para:
    - Verificar el token de sesión
    - Permitir solo un uso por token
    - Validar tamaño y tipo de archivo
    - Expirar tokens automáticamente

Flujo de Uso

1. Usuario entra a la página → resuelve CAPTCHA invisible.
2. Frontend envía token de CAPTCHA a `/session-token`.
3. Backend valida CAPTCHA y devuelve **token de sesión único**.
4. Usuario envía audio a `/transcribe` junto con token.
5. Backend verifica:
    - Token válido
    - Token no usado
    - Token no expirado
6. Backend procesa audio y devuelve la transcripción.
7. Token se marca como **usado**, impidiendo reutilización.

Ejemplo de Implementación (sin Redis)

1. Frontend JS

```
async function getSessionToken() {
    const captcha = await grecaptcha.execute("TU_SITE_KEY", { action:
"transcribe" });
    const res = await fetch("/session-token", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ captcha })
    });
    const data = await res.json();
    return data.token;
}

async function sendAudio(audioBlob) {
    const token = await getSessionToken();
    const form = new FormData();
    form.append("audio_file", audioBlob, "audio.wav");
```

```
    form.append("token", token);
    const res = await fetch("/transcribe", { method: "POST", body: form });
    return await res.json();
}
```

2. Backend FastAPI

```
from fastapi import FastAPI, UploadFile, File, Form, HTTPException
import jwt, uuid, time

app = FastAPI()

SECRET_KEY = "CAMBIA_ESTE_SECRETO"
USED_TOKENS = {}
TOKEN_EXPIRATION = 600  # 10 minutos

@app.post("/session-token")
async def create_session_token(captcha: str = Form(...)):
    # Aquí validar CAPTCHA con proveedor externo
    valid = True  # Simulado
    if not valid:
        raise HTTPException(400, "Invalid CAPTCHA")

    token_id = str(uuid.uuid4())
    payload = {"token_id": token_id, "exp": time.time() + TOKEN_EXPIRATION}
    token = jwt.encode(payload, SECRET_KEY, algorithm="HS256")
    return {"token": token}

@app.post("/transcribe")
async def transcribe(audio_file: UploadFile = File(...), token: str =
Form(...)):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
        token_id = payload["token_id"]
    except:
        raise HTTPException(401, "Invalid or expired token")

    if USED_TOKENS.get(token_id):
        raise HTTPException(403, "Token already used")
    USED_TOKENS[token_id] = True

    contents = await audio_file.read()
    if len(contents) > 10 * 1024 * 1024:
        raise HTTPException(400, "File too large")
    if audio_file.content_type not in ["audio/wav", "audio/mpeg"]:
        raise HTTPException(400, "Invalid audio format")

    # Llamada a Gemini (simulada)
    transcript = "Transcripción simulada"
    return {"status": 200, "message": "Audio transcribed successfully",
```

```
"content": transcript}
```

3. Limpiador de tokens viejos (opcional)

```
import threading

def cleanup_tokens():
    while True:
        now = time.time()
        for token in list(USED_TOKENS.keys()):
            if USED_TOKENS[token]:
                del USED_TOKENS[token]
        time.sleep(600)

threading.Thread(target=cleanup_tokens, daemon=True).start()
```

Beneficios de esta Solución

- Usuarios reales pueden transcribir audio **sin login**.
- Bots y scripts externos (`curl`, Postman) **no pueden** usar `/transcribe`.
- Tokens caducan automáticamente y solo permiten un uso.
- No requiere Redis ni base de datos externa.
- Protege la clave del proveedor de IA (Gemini) y evita uso indebido.
- Permite establecer límites de tamaño de archivo y tipo MIME para seguridad adicional.

Conclusión

La implementación de **CAPTCHA + token de un solo uso** permite:

- Mantener la experiencia de usuario sin login.
- Evitar abuso de la API de transcripción.
- Controlar costos y proteger los recursos del backend.
- Cerrar completamente la vulnerabilidad crítica de exposición del endpoint `/transcribe`

Observación:
Como el frontend actual no está disponible, estas soluciones son propuestas teóricas basadas en la revisión

de la lógica de transcripción observada. La implementación real deberá adaptarse al código y librerías usadas en la aplicación.

> Nota: Todas las recomendaciones deben aplicarse primero en un entorno de prueba antes de pasar a producción para evitar corrupción de datos.