2FA Verification

Índice

Introducción

Sistema de autenticación de doble factor biométrico implementado con Flask, que combina reconocimiento facial y de voz con detección de vivacidad para prevenir ataques de suplantación.

Cambios Respecto al Planning Inicial

Framework de Interfaz: Streamlit → Flask + Socket.IO

Nuestra idea inicial era utilizar Streamlit para la aplicación web debido a su sencilla implementación con python. Pero decidimos hacer la migración completa a Flask con Socket.IO debido a:

- **Lag inaceptable**: Streamlit refresca toda la página en cada interacción, causando latencias de 1-3 segundos.
- **Video en tiempo real**: Streamlit no soporta streaming de video fluido, necesario para reconocimiento facial.
- **WebRTC limitado**: Imposibilidad de capturar audio/video del navegador de forma nativa.

En cambio, Flask + Socket.IO permite comunicación bidireccional instantánea y un mayor control sobre el flujo de datos y la UI.

Autenticación de Voz: Enfoque Text-Independent

Primero evaluamos un enfoque text-dependent debido a que en nuestra implementación se generan 6 números aleatorios que el usuario va a tener que decir para evitar la autenticación mediante audios pre-grabados. Entonces, si no depende del texto, se puede hacer utilizar un audio pre-grabado en el que el usuario salga diciendo cualquier cosa. Un ejemplo que planteamos fue DTW sobre secuencias MFCC completas. Nos daba muchos problemas debido a que, si la muestra de iniciar sesión se compara con las 5 secuencias de número, cómo son diferentes combinaciones de números y depende del texto, no va a coincidir con ninguna. Para poder llevarlo a cabo con ese método o se graban todas las posibles combinaciones de números del usuario, lo cual es inviable debido a que hay 2^6 combinaciones, o se genera a partir de esas 5 grabaciones una artificial que coincida con la que dice el usuario al iniciar sesión. Además, DTW es computacionalmente costoso, con una complejidad de $O(n^2)$ en cada comparación.

Es por ello por lo que al final decidimos utilizar text-independent con embeddings basados en estadísticas MFCC. Es mucho más flexible ya que se puede comparar la grabación de iniciar sesión con las 5 registradas sin tener en cuenta la combinación de números. En las pruebas, daba una similitud ~58-65% para mismo usuario, ~25-35% para impostores.

Los componentes del embedding de audio son:

- Media de MFCC base (13 valores)
- Desviación estándar (13 valores)
- Percentiles (10, 25, 50, 75, 90) → 65 valores
- Rango intercuartílico (13 valores)
- Rango total (13 valores)
- Skewness (13 valores)
- Total: 130 dimensiones

Y para calcular la similitud usamos distancia euclidiana normalizada con decaimiento exponencial (factor=22.0). El 22 lo fuimos ajustando empíricamente.

Validación de Contenido por Speech-to-Text

Cómo al final utilizamos un sistema text-independent que solo verifica la identidad del usuario, un usuario podría autenticarse con cualquier grabación de su voz. Entonces, decidimos añadir una capa, previa a la verificación de identidad, de STT para comprobar que el usuario lea los dígitos generados de manera automática en español. Para la transcripción automática utilizamos Google Speech Recognition API y luego comparamos la transcripción con los números generados para que el usuario los diga. Si no coinciden, ya da error en el inicio de sesión previo a la verificación de identidad.

Detección de Vivacidad Facial

El método que implementamos para detectar la vivacidad fue el análisis de movimientos faciales.

- Pestañeo: Relación de aspecto ocular (EAR) < 0.25

- Apertura de boca: Relación de aspecto labial (MAR) > 0.5

EAR y MAR son métricas geométricas usadas en visión por computador para detectar pestañeos y apertura de boca utilizando landmarks.

Umbrales ajustados empíricamente para balance seguridad/usabilidad

Detección de Vivacidad de Voz

Implementamos la Detección de Vivacidad de Voz mediante el análisis de variabilidad acústica con el objetivo de detectar audios sintéticos o pregrabados.

Parámetros:

- Varianza de energía > 0.012 (muy estricto)
- Varianza ZCR > 0.0015 (detección de naturalidad)
- Varianza de pitch > 8 Hz (variabilidad tonal)

Arquitectura Final

Stack tecnólogico

Backend - Servidor Web:

- Flask 3.0+: Framework web minimalista para Python

- Flask-SocketIO 5.3+: Comunicación bidireccional en tiempo real vía WebSockets

- Flask-Session 0.5+: Gestión de sesiones del lado del servidor

- python-socketio 5.10+ + python-engineio 4.8+: Implementación servidor Socket.IO

Backend - Visión por Computadora:

- OpenCV 4.8+ (opencv-python): Captura y procesamiento de frames de video

- dlib 19.24+: Biblioteca C++ con detección facial y landmarks (68 puntos)

- face-recognition 1.3+: Wrapper de dlib con modelo ResNet preentrenado (128D embeddings)

  - Basado en FaceNet de Google

- Precisión ~99.38% en LFW dataset

- Distancia euclidiana para comparación de encodings

Backend - Procesamiento de Audio y Voz:

- librosa 0.10+: Extracción de características acústicas

- MFCC (Mel-Frequency Cepstral Coefficients) con 13 coeficientes base

- Análisis espectral y temporal

- Filtrado pre-énfasis y normalización

- SpeechRecognition 3.10+: Transcripción audio-a-texto

- Integración con Google Speech Recognition API

- Soporte multi-idioma (configurado en es-ES)

- sounddevice 0.4.6+: Captura de audio de alta calidad

- scipy 1.11+: Procesamiento científico (señales, estadísticas)

- Análisis de pitch (autocorrelación)

- Detección zero-crossing rate (ZCR)

- fastdtw 0.3.4: Dynamic Time Warping (no usado en versión final, disponible para testing)

Backend - Datos y Seguridad:

- SQLite 3: Base de datos embebida sin servidor

- Almacena embeddings faciales y vocales serializados (pickle)

- Hashes bcrypt de contraseñas

- Logs de auditoría de intentos de login

- bcrypt 4.0+: Hashing adaptativo con salt automático (factor de trabajo configurable)

- NumPy 1.24+: Operaciones vectoriales y matrices para embeddings

- Pillow 10.0+: Manipulación de imágenes (conversión formatos, redimensionamiento)

Backend - Herramientas de Desarrollo:

- pip-audit 2.6+: Auditoría de vulnerabilidades en dependencias

- pyobjc-framework-LocalAuthentication 9.0+ (macOS): Touch ID nativo (opcional)

Frontend:

- HTML5 + CSS3: Interfaz moderna y responsive

- JavaScript ES6+: Lógica cliente

- Socket.IO Client: Comunicación bidireccional con backend

- MediaRecorder API: Captura de audio/video del navegador

  - getUserMedia() para acceso a dispositivos

  - Codecs: audio/webm o video/webm según navegador

- Canvas API: Renderizado waveform en tiempo real

- Web Audio API: Análisis de frecuencias (visualización)

Deployment:

- Docker: Containerización con multi-stage build

  - Imagen base: python:3.11-slim

  - Build stage: Instalación de dependencias pesadas (dlib, OpenCV)

  - Runtime stage: Copia de artefactos optimizados

  - Tamaño final: ~1.2 GB

- Docker Volumes: Persistencia de base de datos entre reinicios

- Makefile: Automatización de comandos comunes

  - make build, make run, make logs, make clean

- Usuario no-root: Container ejecuta con privilegios limitados

Workflow de autenticación

1. Registro Usuario
    a. Usuario + Contraseña (bcrypt)
2. Registro Facial
    a. Captura video streaming
    b. Validación centrado

    c. Detección pestañeo
    d. Detección apertura boca
    e. Encoding 128D guardado en SQLite
3. Registro Voz
    a. 5 muestras × 6 segundos
    b. Desafíos numéricos aleatorios
    c. Procesamiento: filtrado + normalización + MFCC
    d. Extracción speaker embedding (130D)
    e. Promedio embeddings guardado en SQLite
4. Login
    a. Usuario + Contraseña
    b. Verificación Facial (abrir y cerrar ojos y boca)
    c. Verificación Voz (desafío aleatorio):
        i. STT: Transcripción y validación números
        ii. Biometría: Comparación speaker embeddings
    d. Acceso concedido

Ejecución con docker

1. Clonar repositorio git clone cd 2FA
2. Construir imagen Docker make build

O manualmente: docker build -t 2fa-biometric-app .

3. Ejecutar contenedor con persistencia make run

O manualmente: docker volume create 2fa-data docker run -d --name 2fa-app -p 5001:5001 -v 2fa-data:/app/data 2fa-biometric-app

4. Ver logs make logs
5. Acceder a la aplicación Navegador: http://localhost:5001