Caso - Clasificación de repositorios

Contexto

La exposición de información sensible en el código de las aplicaciones presenta una vulnerabilidad significativa y un riesgo potencial de filtración de datos en diversas organizaciones. Es crucial comprender las razones detrás de esta problemática:

1. Exposición involuntaria: Con el acceso al código fuente por parte de desarrolladores, revisores de código y aquellos con privilegios para acceder al repositorio, cualquier información sensible contenida directamente en el código puede estar expuesta sin requerir un esfuerzo adicional para su obtención.

2. Rastreo de código y herramientas automatizadas: Las herramientas automatizadas, como los escáneres de seguridad de código estático, pueden detectar patrones conocidos de información sensible durante el análisis del código fuente. Si dicha información está presente directamente en el código, estas herramientas pueden identificar fácilmente estas vulnerabilidades y advertir sobre los riesgos potenciales.

3. Cambios de contexto y mantenimiento: A medida que el código evoluciona y se realizan cambios en el proyecto, es posible que la información sensible necesite ser modificada o eliminada. Si no se gestiona adecuadamente, la información sensible puede ser olvidada y dejarse expuesta, lo que aumenta el riesgo de compromiso de datos.

4. Dificultad para aplicar prácticas de seguridad: Cuando se maneja la información sensible correctamente, es posible implementar medidas de seguridad adicionales para protegerla, como el cifrado o el almacenamiento seguro. Sin embargo, si la información sensible se encuentra directamente en el código, estas medidas de seguridad son difíciles de aplicar de manera eficiente.

Es fundamental abordar esta problemática para garantizar la seguridad de los datos confidenciales. Por esta razón, muchas organizaciones emplean soluciones y prácticas para proteger la información sensible y minimizar los posibles riesgos asociados a su exposición. Estas soluciones suelen incluir el uso de variables de entorno, archivos de configuración seguros o herramientas de gestión de secretos para almacenar y proteger adecuadamente la información sensible.

Objetivo

Desarrollar una API REST en **Golang** que permita identificar si un repositorio de código contiene o no información sensible. Para ello se espera que la api reciba la URL de un repositorio de Github y analice todos sus archivos de manera iterativa. La herramienta deberá clasificar los archivos según si contienen o no información sensible y generar un resultado donde se pueda detallar si se encontró información sensible y donde.

Para el desarrollo de este proyecto solo se requiere analizar **la rama principal del proyecto** y **se considerará información sensible sólo si se detectan los siguientes tipos de datos:**

- **IP**
- **CREDIT CARD**
- **EMAIL**

Debe apoyarse en el uso de expresiones regulares y/o validaciones extras que considere convenientes para asegurarse de la detección.

Solo se deben analizar máximo 100 archivos por repositorio. En caso de que el repositorio tenga más de 100 archivos, el orden de escogencia de cuáles deben ser esos 100 archivos es indiferente para el desarrollo de este proyecto. Por cada archivo el máximo que se debe analizar son 200.000 caracteres.

Se deberá contar con una base de datos **PostgreSQL** ó **MySQL** (según lo que se considere mejor para persistir la información del resultado de los escaneos). En esta base de datos debe poder verse: configuraciones, historial de escaneos, y la clasificación actualizada a partir del último escaneo de cada repositorio.

La tabla de clasificación debería contener los siguientes campos:

TABLE: CLASSIFICATION

REPOSITORY_URL	**FILE**	**INFORMATION_TYPE**	**DETECTION_DATE**	**JOB_ID**	REPOSITORY_OWNER	REPOSITORY_EMAIL	AMOUNT_DETECTIONS
http://github.com/ejemplo	main.go	N/A	2024/02/01	1234214	ahab	ahab@test.com	0
http://github.com/ejemplo	users.csv	CREDIT_CARD	2024/02/01	243242	ahab	ahab@test.com	20

Los campos que no están en negrilla son opcionales, de igual forma siéntase libre en caso de querer adicionar más campos que considere que pueden mejorar el objetivo final posterior a la detección, que es la mitigación de la exposición. Imagine que esta data va a ser usada para un reporte y debe ser claro para el usuario donde debe corregir.

El **JOB_ID** debe ser el identificador del job inicial programado para ejecutar el escaneo sobre el repositorio. La intención funcionalmente es que la API reciba la URL del repositorio y luego ejecute asíncronamente el escaneo. Los resultados del escaneo deben ser posibles de validarse posteriormente.

Para cumplir con las anteriores especificaciones, la API deberá contar mínimamente con los siguientes endpoints para cubrir la funcionalidad solicitada:

POST /api/v1/scan
Recibir los datos de la url del repositorio que debe escanear.

Body:

Nombre	Tipo	Descripción
url	string	Indica la url del repositorio a escanear

Respuesta:

- En caso de éxito:
  - Status code: 201.
  - Body: el id del job de escaneo creado en la base de datos.
- En caso de error:
  - Status code: el que corresponda al tipo de error.
  - Body: un mensaje descriptivo del error.

GET /api/v1/result/:job_id

Recibir el id del job del cuál se quiere ver sus resultados. Si el job aún no ha finalizado debe poder verse en la respuesta.

Parámetros:

Nombre	Tipo	Descripción
job_id	integer	Id del job de escaneo

Respuesta:

- En caso de éxito:
    - Status code: 200.
    - Body: JSON con la estructura de los resultados solo si el job está listo.
- En caso de que el job aún no haya finalizado:
    - Status code: 202.
    - Body: mensaje descriptivo.
- En caso de error:
    - Status code: el que corresponda al tipo de error.
    - Body: un mensaje descriptivo del error.

Entregables

- Repositorio en github del código fuente ó un .zip con el código fuente.
- Dockerfile correspondiente para ejecutar la aplicación.
- Script SQL para crear la base de datos de la aplicación.
- Documentación de la estrategia y solución.

Bonus

- Logging.
- Testing.
- Autenticación en la api.

- Uso de concurrencia.
- Endpoint para obtener el resumen del resultado de un escaneo del resultado renderizado en HTML, con métricas de interés en relación a los tipos de datos encontrados, cantidad de archivos analizados, etc.

Consideraciones generales

- No se pueden utilizar herramientas open-source o comerciales que ya resuelvan este problema.
- Se podrán crear todas las funciones complementarias que se consideren necesarias para un correcto funcionamiento de la aplicación.
- Se recomienda modularizar y aplicar buenas prácticas de programación para un mejor entendimiento del código.
- Toda decisión asumida en cuanto a los requerimientos o desarrollo, deberá ser debidamente documentada.