Descripción de la Arquitectura Utilizada

1. Introducción

La solución desarrollada consta de dos componentes principales: una API RESTful construida con Flask y una aplicación de visualización de datos construida con Dash. Ambos componentes interactúan con una base de datos PostgreSQL para gestionar y visualizar datos de vulnerabilidades de ciberseguridad.

2. Componentes de la Arquitectura

a. API RESTful con Flask

La API RESTful proporciona endpoints para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las vulnerabilidades de ciberseguridad almacenadas en la base de datos.

- **Framework**: Flask
- **ORM**: SQLAlchemy
- **Base de datos**: PostgreSQL
- **Características de seguridad**: Las credenciales de la base de datos están encriptadas y almacenadas en archivos separados (config.json y key.key).

Endpoints:

1. **POST /vulnerabilidades**: Agregar una nueva vulnerabilidad.
2. **GET /vulnerabilidades**: Obtener todas las vulnerabilidades.
3. **PUT /vulnerabilidades/<int:id>**: Actualizar una vulnerabilidad existente.
4. **DELETE /vulnerabilidades/<int:id>**: Eliminar una vulnerabilidad existente.

b. Aplicación de Visualización con Dash

El dashboard permite visualizar, analizar y gestionar las vulnerabilidades de ciberseguridad a través de diversas gráficas y filtros interactivos.

- **Framework**: Dash (basado en Flask)
- **Bibliotecas de visualización**: Plotly
- **Bibliotecas de procesamiento de datos**: Pandas
- **Modelo de aprendizaje automático**: Transformers para generación de resúmenes y recomendaciones

Funcionalidades del Dashboard:

1. **Análisis**: Gráficas para analizar la severidad, tendencias en el tiempo, correlaciones de CVSS, distribución por proveedores y proyectos, etc.
2. **Lista de Vulnerabilidades**: Visualización paginada de las vulnerabilidades con capacidades de búsqueda y filtrado.
3. **Storytelling y Recomendaciones**: Generación de historias basadas en patrones de datos y recomendaciones automatizadas utilizando modelos de lenguaje.

proyecto/

├── app.py # Código de la API RESTful con Flask

├── dashboard.py # Código del Dashboard con Dash

├── config.json # Archivo de configuración encriptado con credenciales de la base de datos

├── key.key # Archivo con la clave de encriptación

├── requirements.txt # Lista de dependencias del proyecto

└── README.md # Instrucciones y documentación del proyecto

Detalles de Implementación

a. Manejo de Credenciales

Las credenciales de la base de datos están encriptadas para asegurar que no se exponen directamente en el código fuente. Se utilizan los archivos config.json y key.key para almacenar la configuración encriptada y la clave de encriptación respectivamente.

b. Modelo de Base de Datos

La base de datos PostgreSQL almacena los datos de las vulnerabilidades con la siguiente estructura:

```
class Vulnerabilidad(db.Model):
    __tablename__ = 'vulnerabilidades'
    id = db.Column(db.Integer, primary_key=True)
    cve_id = db.Column(db.String(255))
    vendor_project = db.Column(db.String(255))
    product = db.Column(db.String(255))
    vulnerability_name = db.Column(db.String(1024))
    date_added = db.Column(db.Date)
    short_description = db.Column(db.Text)
    required_action = db.Column(db.Text)
    due_date = db.Column(db.Date)
    notes = db.Column(db.Text)
    grp = db.Column(db.Integer)
    pub_date = db.Column(db.Date)
    cvss = db.Column(db.Float)
```

cwe = db.Column(db.String(255))

vector = db.Column(db.String(255))

complexity = db.Column(db.String(255))

severity = db.Column(db.String(255))

c. Funciones del Dashboard

El dashboard utiliza Dash para generar una interfaz interactiva que permite a los usuarios visualizar y analizar los datos de las vulnerabilidades. Utiliza gráficos de Plotly y funciones de Pandas para manipular y presentar los datos.

Autor: Ing. Maximiliano Fochi

Fecha: 26-06-2024

Versión: V1