**token\_encrypt.py**  
  
El script token\_encrypt.py forma parte de un sistema seguro para la gestión de credenciales dentro del pipeline. Su función principal es **encriptar un token de GitHub** utilizando el algoritmo **Fernet** de la librería cryptography, asegurando que el token nunca quede expuesto en texto plano.

El proceso sigue estos pasos:

1. **Generación de clave de encriptación:**
   * Si no existe un archivo secret.key, se genera una nueva clave y se guarda en dicho archivo.
   * Si la clave ya existe, se carga para reutilizarla.
2. **Cifrado del token de GitHub:**
   * El usuario debe pegar manualmente su token de GitHub en el script.
   * Se cifra el token con la clave generada/cargada.
3. **Almacenamiento seguro:**
   * El token cifrado se guarda en un archivo llamado github\_token.enc, impidiendo que se almacene en texto plano.

**Decisiones de Diseño para Seguridad y Criptografía**

* **Encriptación con Fernet:**
  + Se utilizó **Fernet** porque ofrece cifrado simétrico con claves de 256 bits y permite tanto la encriptación como la desencriptación segura del token.
  + Esto garantiza la integridad y confidencialidad del token de GitHub.
* **Gestión segura de claves:**
  + La clave de cifrado se almacena en secret.key en el sistema de archivos, lo que permite su reutilización sin necesidad de generar nuevas claves en cada ejecución.
* **Prevención de exposición del token:**
  + El token nunca se almacena en texto plano ni se imprime en la consola, reduciendo riesgos de filtración.
  + Se recomienda que github\_token.enc y secret.key sean añadidos a .gitignore para evitar su exposición en repositorios de código.

Este script asegura que el pipeline pueda utilizar credenciales de GitHub sin comprometer la seguridad del token, cumpliendo con buenas prácticas en el manejo de información sensible.

**data\_pipeline.py**

El archivo data\_pipeline.py implementa una clase DataPipeline que automatiza la descarga, procesamiento y almacenamiento de datos en GitHub. Sigue los siguientes pasos:

1. **Autenticación y acceso a datos:**
   * Se conecta con la API de Kaggle para descargar datasets.
   * Se verifica el rol del usuario a través del archivo roles.json antes de ejecutar el pipeline.
   * Se carga y desencripta el token de GitHub desde github\_token.enc para interactuar con el repositorio.
2. **Procesamiento de datos:**
   * Se cargan los datos desde archivos CSV descargados.
   * Se eliminan valores nulos y la columna session\_id si está presente.
   * Se encriptan los valores de session\_id para preservar la privacidad.
   * Se convierten variables categóricas (protocol\_type, encryption\_used, browser\_type) en valores numéricos usando LabelEncoder.
3. **Almacenamiento de datos:**
   * Se suben los datos originales y los datos procesados a un repositorio de GitHub.
   * Se maneja la actualización de archivos en GitHub verificando su existencia y actualizándolos si es necesario.

**Decisiones de Diseño para Seguridad y Criptografía**

* **Autenticación segura:**
  + Se usa un sistema de roles (roles.json) para restringir el acceso.
  + Se autentica en Kaggle y GitHub con tokens cifrados.
* **Manejo seguro de credenciales:**
  + Se encripta el token de GitHub con Fernet, evitando que quede expuesto en texto plano.
  + El token desencriptado solo se usa en memoria durante la ejecución.
* **Protección de datos sensibles:**
  + Se encripta la columna session\_id para evitar la exposición de identificadores de sesión.
  + Se asegura que los archivos con credenciales (github\_token.enc, secret.key) no sean expuestos en repositorios de GitHub.

Este pipeline proporciona un flujo de trabajo automatizado y seguro para el manejo de datos en entornos de ciberseguridad y aprendizaje automático.

**ml\_model.ipynb**

El modelo de Machine Learning sigue una arquitectura basada en aprendizaje supervisado con clasificadores para la detección de intrusiones en ciberseguridad. Utiliza datasets descargados desde GitHub y preprocesados para mejorar la calidad de los datos antes de entrenar distintos modelos de clasificación.

**Flujo de Trabajo:**

1. **Carga de Datos:** Se obtienen los datasets desde un repositorio de GitHub.
2. **Preprocesamiento:** Normalización de datos con MinMaxScaler, eliminación de valores nulos y transformación de variables categóricas.
3. **División del Conjunto de Datos:** Separación en conjuntos de entrenamiento y prueba con train\_test\_split.
4. **Entrenamiento de Modelos:** Se prueban diferentes clasificadores como:
   * Support Vector Machine (SVM)
   * Decision Tree Classifier
   * Multilayer Perceptron (MLP)
5. **Optimización con Algoritmos Genéticos:** Se emplea GASearchCV para encontrar los mejores hiperparámetros.
6. **Evaluación del Modelo:** Se calcula la matriz de confusión y la precisión para medir el desempeño.
7. **Exportación del Modelo:** Se guarda el modelo entrenado usando joblib.

**Decisiones de Diseño**

**Seguridad y Criptografía**

* **Acceso a Datos Seguro:** Los datos son obtenidos directamente desde GitHub usando enlaces raw, evitando almacenamiento local inseguro.
* **Procesamiento Restringido:** Solo se utilizan los datos necesarios, eliminando atributos innecesarios.

**Limpieza y Preprocesamiento de Datos**

* **Normalización:** Se utiliza MinMaxScaler para estandarizar valores.
* **Codificación de Datos Categóricos:** Se transforma la información categórica en numérica para facilitar el entrenamiento.
* **Manejo de Datos Faltantes:** Se eliminan registros con valores nulos para evitar sesgos en el entrenamiento.

Este enfoque garantiza que el modelo de Machine Learning sea eficiente, seguro y reproducible, con una optimización basada en algoritmos genéticos para mejorar su desempeño.

**app.py**

app.py es una aplicación web desarrollada en Streamlit que proporciona una interfaz gráfica para ejecutar el pipeline de datos, visualizar los datasets y ejecutar el modelo de Machine Learning entrenado. Se conecta con GitHub para cargar datasets procesados y mostrar resultados gráficos de las predicciones del modelo.

**Arquitectura del Data Pipeline**

El app.py interactúa con DataPipeline, una clase encargada de manejar la descarga, limpieza y almacenamiento de datos. El pipeline sigue estos pasos:

1. **Ejecución del Data Pipeline**: Descarga los datos desde Kaggle y los almacena en GitHub.
2. **Visualización del Dataset**: Recupera y muestra datasets desde GitHub.
3. **Ejecución del Modelo**: Carga el modelo y el scaler para realizar predicciones sobre los datos procesados.

**Integración con el Modelo de IA**

* Se carga el modelo best\_model.pkl y el scaler.pkl.
* Se preprocesan los datos escalándolos antes de realizar las predicciones.
* Se generan y visualizan las predicciones, incluyendo una matriz de confusión para evaluar el desempeño del modelo.

**Decisiones de Diseño en Seguridad y Criptografía**

1. **Gestón de Tokens**: Se utilizan claves encriptadas (github\_token.enc) para proteger credenciales sensibles al interactuar con GitHub.
2. **Protección de Datos Sensibles**: Se utiliza cryptography.fernet en DataPipeline para encriptar ciertos datos del dataset.
3. **Control de Acceso**: El pipeline verifica los permisos del usuario mediante roles.json, permitiendo el acceso solo a usuarios autorizados.

**Flujo de Trabajo**

1. El usuario ejecuta el pipeline mediante un botón en la interfaz.
2. Se cargan los datasets desde GitHub y se muestran en la aplicación.
3. Se ejecuta el modelo sobre los datos procesados.
4. Se visualizan resultados de predicciones y la matriz de confusión.

Este diseño modular facilita la integración con diferentes modelos y datasets, asegurando escalabilidad y seguridad en el proceso.