Crypto Analysis Tool

Crypto Analysis Tool is a tool for analyzing the transaction histories of cryptocurrencies, specifically Ethereum, to identify illicit activities such as phishing, money laundering, and other suspicious behaviors. This project encompasses the development of machine learning models for anomaly detection and a web application to make these insights accessible to users.

Github: <https://github.com/RuaidhriBrown/CryptoAnalysis/tree/final-changes?tab=readme-ov-file#money-laundering-detection>

# Table of Contents

1. Project Overview

2. Local Development

- Installing Requirements

3. Machine Learning Model Training

- Phishing Detection

- Money Laundering Detection

4. Web Application

- Overview

- Key Features

- Running the Web Application

- Web Application Configuration

5. University of Buckingham

# Project Overview

The Crypto Analysis Tool aims to provide a comprehensive, automated solution for detecting illicit activities within the Ethereum blockchain. The tool includes machine learning models trained on real transaction data to identify patterns indicative of fraudulent activities such as phishing and money laundering. The models are integrated into a web application that allows users to perform in-depth analysis of blockchain transactions.

# Local Development

To set up a local development environment, use the provided Docker Compose configuration. This setup ensures all dependencies are correctly installed and the environment mirrors the production setting.

Note: We highly recommend using Docker Compose for setting up the project, as it simplifies the process and ensures that the environment is consistent across different systems. Docker Compose automatically handles all configurations, including environment variables, database settings, and static files.

You will still need to ensure the trained models are added to the models directory

- Phishing Detection

- Money Laundering Detection

## Why Use Docker Compose?

Simplifies Configuration: Docker Compose manages all dependencies and configurations in a single file, eliminating the need for manual setup in settings.py.

Ensures Consistency: The Docker environment mirrors the production environment, reducing the risk of inconsistencies or errors.

Quick Setup: Set up the entire project environment with a single command.

Note: the static files may not work in the docker-compose method

## Steps to Start Local Development Using Docker Compose:

1. Clone the repository:

```powershell  
git clone https://github.com/RuaidhriBrown/CryptoAnalysis.git  
cd 'CryptoAnalysis\Application Code\src\Host Applications\Crypto.Tracker.Web.UI\'  
```

2. Run Docker Compose to build and start the services:

Run the following:   
```powershell  
docker-compose up --build  
```

This command will build the Docker images and start the containers for the web application and the database. All configurations, including environment variables and database settings, are automatically managed by Docker Compose.

3. Access the Web Application:

Once the Docker containers are up and running, you can access the web application in your browser at http://localhost:8000.

4. Login to the user:

- Username: admin  
- password: X1B2#WXYZ123a

## Avoid Manual Configuration

If you choose to use Docker Compose, you don't need to manually configure the settings.py file for environment-specific settings like SECRET\_KEY, DEBUG, ALLOWED\_HOSTS, or DATABASES. Docker Compose handles these settings through environment variables defined in the docker-compose.yml file.

However, if you still prefer manual setup, please refer to the Web Application Configuration section below.

## Installing Requirements

Before running the project, you need to install all the required Python packages for both the machine learning models and the web application.

### Installing Requirements for Machine Learning Model Training

1. Navigate to the project directory:

```powershell  
cd 'CryptoAnalysis\Application Code\practice\Project - Testing\'  
```

2. Install the Python dependencies:

Make sure you have Python and pip installed. If you're using a virtual environment (recommended), activate it first.

Run the following command to install the required packages for the machine learning models:

```powershell  
pip install -r requirements.txt  
```

3. Installing Requirements for the Web Application:

Navigate to the web application directory: 'cd CryptoAnalysis/Application Code/src/Host Applications/Crypto.Tracker.Web.UI'

```powershell  
pip install -r requirements.txt  
```

Ensure that the requirements.txt file, which contains all necessary dependencies for running the Django web application, is located in the web application directory.

Note: If you encounter any issues during installation, check that your Python version and pip are up to date. You may also need to install additional system dependencies if required by specific Python packages.

# Machine Learning Model Training

## Overview

The machine learning component of the Crypto Analysis Tool consists of several models designed to detect illicit activities on the Ethereum blockchain, specifically targeting phishing and money laundering behaviors. The models use Random Forest and Graph Neural Networks (GNNs) to analyze transaction data and identify suspicious patterns.

### Phishing Detection

Phishing detection utilizes three datasets:

- Aggregated Ethereum Wallets  
- Ethereum Transaction Data  
- ERC20 Token Transaction Data

The phishing dataset is based on data from [Kaggle](https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset/data). This dataset provides a list of wallets from which transactions and ERC20 transactions were collected using the Etherscan API. The resulting data was used to create an aggregated Ethereum Wallets dataset, representing known phishing accounts. Note that the transactions involve recognized phishing accounts but may not necessarily represent genuine phishing activities.

### Money Laundering Detection

Money laundering detection focuses on datasets derived from incidents like the Upbit hack, including:

- Aggregated Ethereum Wallets  
- Ethereum Transaction Data  
- ERC20 Token Transaction Data

These datasets were created using the script located at:  
`CryptoAnalysis/Application Code/practice/Project - Testing/moneyLaundering/fetch\_data/getMoneyLaunderingData1Layer.py`.

#### Developing the Upbit-Hack Dataset for Money Laundering Detection

The Upbit hack dataset was developed to train models for detecting money laundering activities on the Ethereum blockchain. The dataset construction involved several key steps:

1. API Configuration and Setup: The Etherscan API was used to retrieve Ethereum transaction data. To comply with API rate limits, a maximum of 5 API calls per second was enforced, and retries were handled for failed requests.

2. Data Collection: Transaction data was collected using the `get\_normal\_transactions` and `get\_erc20\_transactions` functions. These functions fetch normal Ether transactions and ERC20 token transactions associated with a given Ethereum address, filtering the results to include only relevant transactions that meet specific criteria (e.g., minimum Ether sent and a specific start time).

3. Address Analysis: Each Ethereum address (wallet) is analyzed for potential involvement in money laundering activities using the `analyze\_wallet` function. Key metrics such as Fast-In Fast-Out (FIFO) ratio, small-volume transaction ratio, zero-out ratio, and network density are calculated to identify suspicious behavior.

4. Detection of Money Laundering: The `detect\_money\_laundering` function applies predefined thresholds to determine if a wallet is suspicious. The thresholds include FIFO ratio, small-volume ratio, zero-out ratio, network density, and minimum balance. If a wallet meets or exceeds the thresholds, it is flagged as suspicious.

5. Interacting Wallets: The `analyze\_interacted\_wallets` function identifies wallets that have interacted with the initial suspicious address. It recursively analyzes these wallets to uncover potential layers of money laundering activity, expanding the dataset to include a broader network of transactions.

6. Data Compilation: Data is accumulated in global dataframes for all analyzed wallets, normal transactions, and ERC20 transactions. These dataframes are then saved to CSV files (`all\_wallet\_summaries.csv`, `all\_normal\_transactions.csv`, and `all\_erc20\_transactions.csv`) for further analysis and model training.

7. Create "Ethereum Transaction" and "ERC20 Token Transaction" Datasets: Combine normal and ERC20 transactions from both suspicious and legitimate wallets.

8. Create "Aggregated Ethereum Wallets" Dataset: Aggregate the "Ethereum Transaction Data" and "ERC20 Token Transaction Data" to derive features.

## Training the Models

To train the machine learning models locally:

### Phishing model code:

First ensure the three following datasets are present in the directory "CryptoAnalysis\Application Code\practice\Project - Testing\datasets\ethereum\phishing\":

1. "combined\_er20.csv"  
2. "combined\_transactions.csv"  
3. "transaction\_dataset\_even.csv"

In the directory: "CryptoAnalysis\Application Code\practice\Project - Testing\Phishing-ML\" there are three Python scripts:

"LR-machine learning er20.py"  
"LR-machine learning transactions.py"  
"LR-machine learning wallets v3.py"

Running these will each create a trained model and the related features PKL file in the "CryptoAnalysis\Application Code\practice\Project - Testing\results\" directory.

### Money Laundering model code:

First ensure the three following datasets are present in the directory "CryptoAnalysis\Application Code\practice\Project - Testing\datasets\ethereum\MoneyLaundering\":

1. "combined\_er20\_transaction\_data.csv"  
2. "combined\_normal\_transactions\_data.csv"  
3. "combined\_transaction\_data.csv"

In the directory: "CryptoAnalysis\Application Code\practice\Project - Testing\moneyLaundering\" there are three Python scripts:

"LR-machine learning er20.py"  
"LR-machine learning transactions.py"  
"LR-machine learning wallets.py"

Running these will each create a trained model and the related features PKL file in the "CryptoAnalysis\Application Code\practice\Project - Testing\results\" directory.

# Web Application

## Overview

The web application is developed using Django and serves as the user interface for interacting with the trained models and analyzing Ethereum blockchain data. It allows users to upload transaction data, run model analyses, and view detailed visualizations of the results.

## Key Features

- \*\*Data Upload and Analysis\*\*: Users can download Ethereum transaction data from etherscan and run analyses using trained machine learning models.  
- \*\*Visualizations and Reporting\*\*: The app provides graphical representations of transaction patterns and allows for exporting analysis results.  
- \*\*User Interface web app\*\*: Facilitates easy navigation, model execution, and result interpretation, supporting digital forensic investigations.

## Running the Web Application

Required:  
6 models trained in the previous section saved into the respective directories:

"CryptoAnalysis\Application Code\src\Host Applications\Crypto.Tracker.Web.UI\models\ethereum\MoneyLaundering" have the following models:

1. "random\_forest\_moneyLaundering\_detector\_aggregated.pkl"  
2. "random\_forest\_moneyLaundering\_detector\_aggregated\_features.pkl"  
3. "random\_forest\_moneyLaundering\_detector\_er20.pkl"  
4. "random\_forest\_moneyLaundering\_detector\_er20\_features.pkl"  
5. "random\_forest\_moneyLaundering\_detector\_transactions.pkl"  
6. "random\_forest\_moneyLaundering\_detector\_transactions\_features.pkl"

and  
"CryptoAnalysis\Application Code\src\Host Applications\Crypto.Tracker.Web.UI\models\ethereum\phishing" have the following files:

1. "random\_forest\_phishing\_detector\_aggregated.pkl"  
2. "random\_forest\_phishing\_detector\_aggregated\_features.pkl"  
3. "random\_forest\_phishing\_detector\_er20.pkl"  
4. "random\_forest\_phishing\_detector\_er20\_features.pkl"  
5. "random\_forest\_phishing\_detector\_transactions.pkl"  
6. "random\_forest\_phishing\_detector\_transactions\_features.pkl"

## Web Application Configuration

To properly set up the Django web application for the Crypto Analysis Tool, users must adjust several settings in the `settings.py` file and configure their environment accordingly.

### Required Configuration Steps

1. \*\*Secret Key\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
SECRET\_KEY = 'b1be7575-abe1-4fe1-9c95-dce1e276f171'  
```

- \*\*What to do\*\*:  
Set the `ETHERSCAN\_API\_KEY` environment variable in your shell or environment manager:  
- \*\*For PowerShell\*\*:  
```powershell  
$env:ETHERSCAN\_API\_KEY = "<Your\_API\_Key>"  
```  
- \*\*For Linux/macOS Bash\*\*:  
```bash  
export ETHERSCAN\_API\_KEY="<Your\_API\_Key>"  
```  
- \*\*For Windows Command Prompt\*\*:  
```cmd  
set ETHERSCAN\_API\_KEY=<Your\_API\_Key>  
```

2. \*\*Debug Mode\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
DEBUG = True  
```

- \*\*What to do\*\*:  
Set `DEBUG = False` for production environments to prevent sensitive data from being exposed. In a development environment, keep it set to `True` to enable detailed error messages.

3. \*\*Allowed Hosts\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
ALLOWED\_HOSTS = []  
```

- \*\*What to do\*\*:  
Add the domain names or IP addresses of your deployment environment to the `ALLOWED\_HOSTS` list to allow Django to serve your app on those hosts. For example:  
```python  
ALLOWED\_HOSTS = ['localhost', '127.0.0.1', 'yourdomain.com']  
```

4. \*\*Database Configuration\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.postgresql',  
 'NAME': 'CryptoTracker',  
 'USER': 'crypto-postgres',  
 'PASSWORD': 'X1B2#WXYZ123a',  
 'HOST': 'localhost',  
 'PORT': '5432',  
 'OPTIONS': {  
 'options': '-c search\_path=dev\_crypto\_tracker,public'  
 }  
 }  
}  
```

- \*\*What to do\*\*:  
Ensure that PostgreSQL is installed and configured correctly on your system. You may need to modify the following database settings to match your environment:  
- `NAME`: Your PostgreSQL database name.  
- `USER`: Your PostgreSQL username.  
- `PASSWORD`: Your PostgreSQL password.  
- `HOST`: The address of your PostgreSQL server (use `localhost` for local development).  
- `PORT`: The port number for your PostgreSQL server (default is `5432`).

5. \*\*Static and Media Files\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
STATIC\_URL = '/static/'  
STATIC\_ROOT = posixpath.join(\*(BASE\_DIR.split(os.path.sep) + ['static']))  
MEDIA\_URL = '/media/'  
MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')  
```

- \*\*What to do\*\*:  
Ensure `STATIC\_ROOT` and `MEDIA\_ROOT` directories are correctly set for your deployment environment. These should exist and be writable by the web server.  
In production, run `python manage.py collectstatic` to gather all static files into the `STATIC\_ROOT` directory.

6. \*\*Environment Variables\*\*

- \*\*ETHERSCAN API Key\*\*:  
The application requires an Etherscan API key to access blockchain data. This key should be stored as an environment variable.  
- \*\*Location in `settings.py`\*\*:  
```python  
ETHERSCAN\_API\_KEY = os.getenv("ETHERSCAN\_API\_KEY", "your-default-api-key")  
```

- \*\*What to do\*\*:  
Set the `ETHERSCAN\_API\_KEY` environment variable in your shell or environment manager:  
- \*\*For PowerShell\*\*:  
```powershell  
$env:ETHERSCAN\_API\_KEY = "<Your\_API\_Key>"  
```  
- \*\*For Linux/macOS Bash\*\*:  
```bash  
export ETHERSCAN\_API\_KEY="<Your\_API\_Key>"  
```  
- \*\*For Windows Command Prompt\*\*:  
```cmd  
set ETHERSCAN\_API\_KEY=<Your\_API\_Key>  
```

7. \*\*Middleware and Installed Apps\*\*

- \*\*Middleware\*\*: The default middleware is set up for standard security and session management. Customize further as needed for your environment.  
- \*\*Installed Apps\*\*: Ensure all necessary apps, including Django default apps and `webview`, are listed under `INSTALLED\_APPS` in `settings.py`.

8. \*\*Timezone and Localization\*\*

- \*\*Location in `settings.py`\*\*:  
```python  
LANGUAGE\_CODE = 'en-us'  
TIME\_ZONE = 'UTC'  
```

- \*\*What to do\*\*: Adjust `LANGUAGE\_CODE` and `TIME\_ZONE` according to your localization needs.

After completing these configurations, your Django application should be properly set up to run in your local or production environment.

The web application will be accessible at `http://localhost:8000`.

### Creating an Admin User

To manage your Django application through the admin interface, you need to create an admin user. Follow these steps using Visual Studio:

1. \*\*Open the Django Project in Visual Studio\*\*:  
- Ensure you have your Django project open in Visual Studio.

2. \*\*Right-click on the Django Project\*\* in the Solution Explorer pane.

3. \*\*Navigate to Python > Django\*\*:  
- Hover over \*\*Python\*\* to expand the menu.  
- Then hover over \*\*Django\*\* to see more options.

4. \*\*Select 'Create Superuser'\*\*:  
- Click on \*\*Create Superuser\*\* from the dropdown menu.

5. \*\*Follow the Prompts in the Output Window\*\*:  
- Visual Studio will prompt you in the Output window to enter the details for your superuser.  
- Provide a username, email, and password when prompted.

6. \*\*Access the Django Admin Interface\*\*:  
- Open your web browser and go to `http://localhost:8000/admin`.  
- Log in using the admin credentials you just created.

# University of Buckingham

This repository was developed as part of Ruaidhri Brown's MSc in Innovative Computing at the University of Buckingham. The project addresses critical challenges in digital forensics and cybersecurity by providing tools to detect and analyze illicit activities on the Ethereum blockchain, thereby aiding law enforcement and research in blockchain security.

# Web Application

## 0. Login as Admin

A white rectangular object with a black border

Description automatically generated

## 1. Download Wallet Details

**Description**: The operator can search for an Ethereum address using the Etherscan API to download normal transactions and ERC20 transactions data for the wallet. The web app saves this data to the database and creates an analysis profile for the wallet. This profile is used to track the wallet and its analyses, allowing easy viewing in the dataset. <http://localhost:8080/>

A screenshot of a computer

Description automatically generated

## 2. Further Wallet Data Download

**Description**: The operator can open a wallet analysis and select other wallets it has interacted with or download the data for all the wallets the current wallet has interacted with. This data can be analysed, or the trained models can be applied to these wallets.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/>

select the ‘Update Data’ tab:

A screenshot of a computer

Description automatically generated

Two options are possible, one for updating the current dataset, this ensures that the most up to date transactions and er20 transactions are in the database. The other option Download Data for Analysis downloads all the data for all the linked wallets.

## 3. Run Trained Models Against the Wallet

**Description**: From the wallet analysis page, the operator can run trained models and see the results. Upon completion, a table will be populated with the results, and a note will be added with the results, serving as an audit trail.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/>

select the ‘Activity’ tab, and then the ‘completed Analyses’ tab:

A screenshot of a computer

Description automatically generated

From here the models and analysis can be run against the wallet\_analyses. There is also the option to ‘View Detailed Transaction Analysis’.

<http://localhost:8080/wallet/188129/transaction-analysis/> the number is the wallet\_analysis wallet\_id:

A screenshot of a computer

Description automatically generated

From here the user can filter to ‘Phishing detected’ and ‘Money Laundering Detected’. The user can move to another wallet after seeing if that wallet is believed to be either Phishing or money laundering.

## 4. Run Trained Models Against All Wallets, Normal Transactions, or ERC20 Transactions

**Description**: The operator can choose to run models against the larger collected data to find potential crime leads, focusing on wallets with a high amount of potentially illicit transactions.

<http://localhost:8080/wallets/> and then click on the ‘Analysis’ tab.

A screenshot of a computer

Description automatically generated

The hope is to have this much analysis option wallets wide:

A screenshot of a computer

Description automatically generated

## 5. Add Notes to Wallet Analysis

**Description**: The operator can add notes to the wallet, allowing them to keep track of findings. System-generated notes are also added when analyses are completed, or transactions are updated.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/> and then click on the ‘Activity’ tab, followed by the ‘Wallet Notes’. This lists the notes:

A screenshot of a computer

Description automatically generated

A screenshot of a chat

Description automatically generated

## 6. Add Tags to Wallet Analysis

**Description**: The operator can use the web app to add tags for tracking the analysis. Tags might include the name believed to be the wallet owner, useful for filtering.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/> and then click on the ‘Activity’ tab, followed by the ‘Wallet Details’. This lists the details for the wallet as set by the User.

A screenshot of a computer

Description automatically generated

## 7. Export Data Collected to CSV

**Description**: The operator can export data from the database, enabling researchers to construct their datasets using the tool.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/>

select the ‘export’ tab:

A screenshot of a computer

Description automatically generated

## 8. View Stats

**Description**: The wallet analysis provides a quick view of the user and their behaviour, displaying statistics such as transaction frequencies, values, balances, and more. <http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/>

A screenshot of a computer

Description automatically generated

## 9. Visualizations

**Description**: The operator can view graphs showing wallet behavior, such as popular transaction times within the day, which help understand the owner's patterns.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/> then click the ‘Activity’ tab followed by the ‘Time Analyses’, this lists the times the user works.

A screenshot of a computer screen

Description automatically generated

## 10. Hash Current Wallet Analysis

**Description**: The operator can run a hash against the current wallet analysis as a precaution to ensure nothing has been modified, using a hash that combines all transactions, details, notes, and completed analysis results.

<http://localhost:8080/wallets/0x00009277775ac7d0d59eaad8fee3d10ac6c805e8/> then click on the ‘Hash Wallet’ tab, from here it will state the current hash and the option to generate a new hash.

A screenshot of a computer

Description automatically generated