CTCQAT Code Documentation

CryptoCurrencyQuery&AnalysisTool

15/09/2025

Purpose

Overview

This Python script fetches and analyzes Ethereum wallet transactions using the Etherscan API. It builds a directed graph where nodes represent wallet addresses and edges represent transactions, then visualizes the number of transactions per address in a bar chart. The script also provides insights into transaction metrics and flags potential illicit activity using basic heuristics (e.g., high-value transactions or high-connectivity addresses).

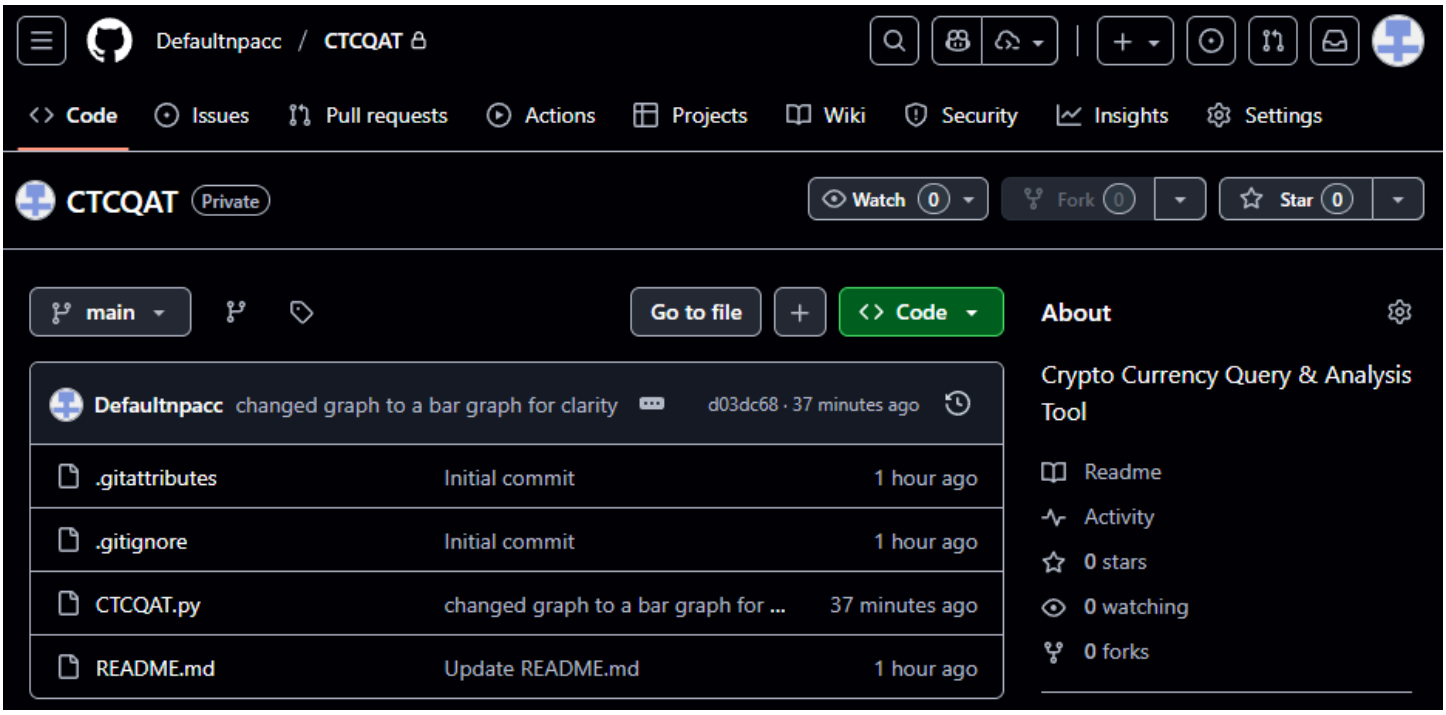

Image of CTCQAT GitHub Repository

Requirements

- **Python**: Version 3.6 or higher.
- **Libraries**:
  - requests: For making HTTP requests to the Etherscan API.
  - networkx: For building and analyzing the transaction graph.
  - matplotlib: For plotting the bar chart.
- **Etherscan API Key**: Required for accessing the Etherscan API. Sign up at etherscan.io to obtain a free key.

Installation

Install Python: Ensure Python 3.6+ is installed. Verify with: python --version

```
python --version
>>
Python 3.13.1
```

Image of python version command

Install Required Libraries: Run the following command in your terminal: pip install requests networkx matplotlib

```
pip install requests networkx matplotlib
```

Image of python library installation command

Set Up Etherscan API Key:

- Register at etherscan.io.
- Generate a free API key from the API section of your account.

Etherscan

Sign Up

Already have an account? Sign In here

Username

Username

Email Address

A confirmation code will be sent to this address

Confirm Email Address

Re-enter your email address

Password

****

Confirm Password

****

I agree to the Terms and Conditions.

I would like to receive the Etherscan newsletter and understand that I can unsubscribe at any time.

Verifying...

CLOUDFLARE

Privacy • Terms

Image of Etherscan sign up page

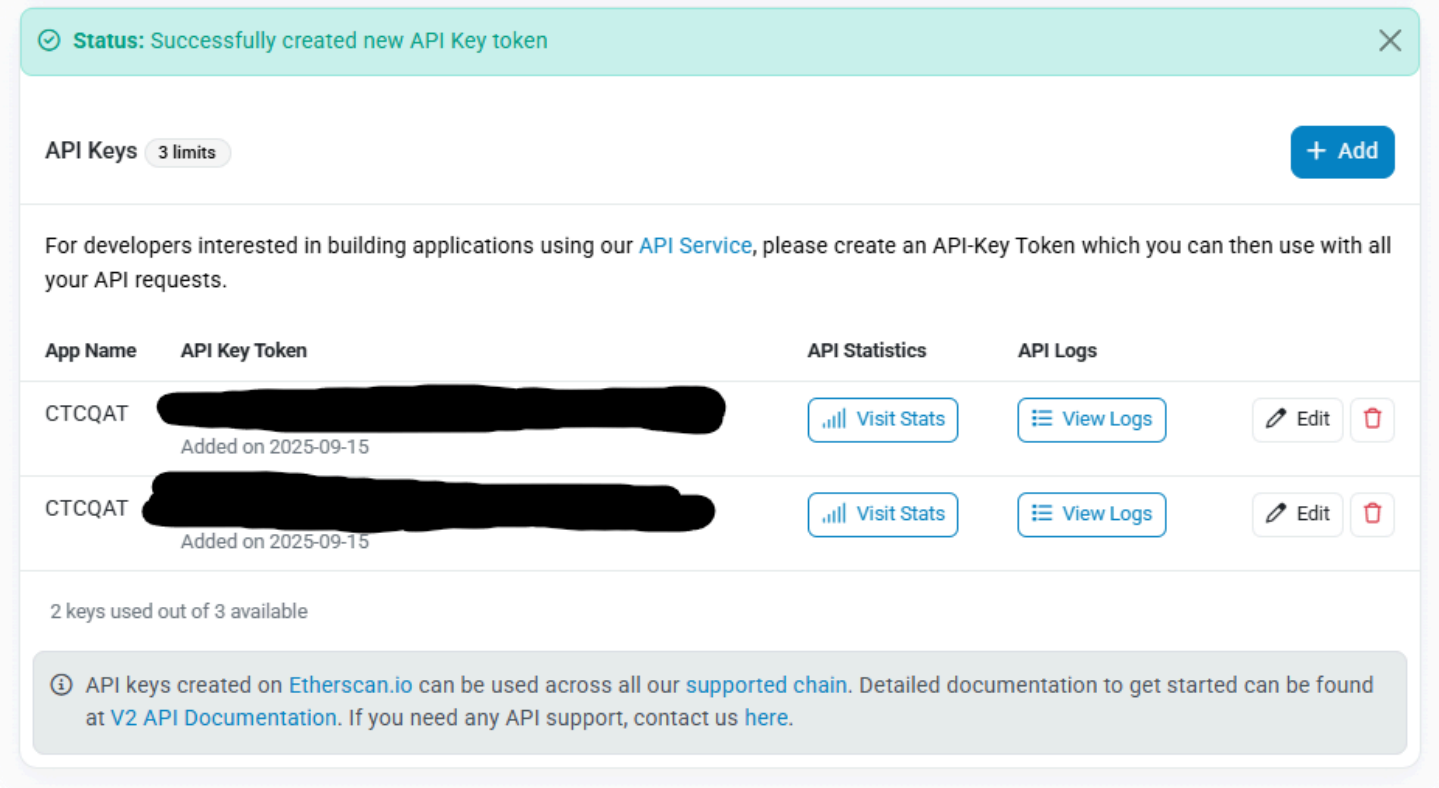

Image of Etherscan api token creation section

Usage

1. **Save the Script**: Save the script to any project folder directory.
2. **Run the Script**: In a terminal (e.g., VS Code integrated terminal), navigate to the script's directory and run: python CTCQAT.py
3. **Provide Inputs**:

- **Etherscan API Key**: Enter your API key when prompted.
- **Wallet Address**: Enter a valid Ethereum wallet address (e.g., 0x09750ad360fdb7a2ee23669c4503c974d86d8694).
- **Number of Transactions**: Enter a number between 1 and 100 (e.g., 20).

Example input:

```
PS D:\HTX C2\CTCQAT> & C:/Python313/python.exe [redacted]/CTCQAT.py"
Enter your Etherscan API key: 5N[redacted]62N4
Enter the wallet address to analyze: 0x9d[redacted]012b68
Enter the number of transactions to fetch (e.g., 10-20): 20
```

Image of Example input

View Output:

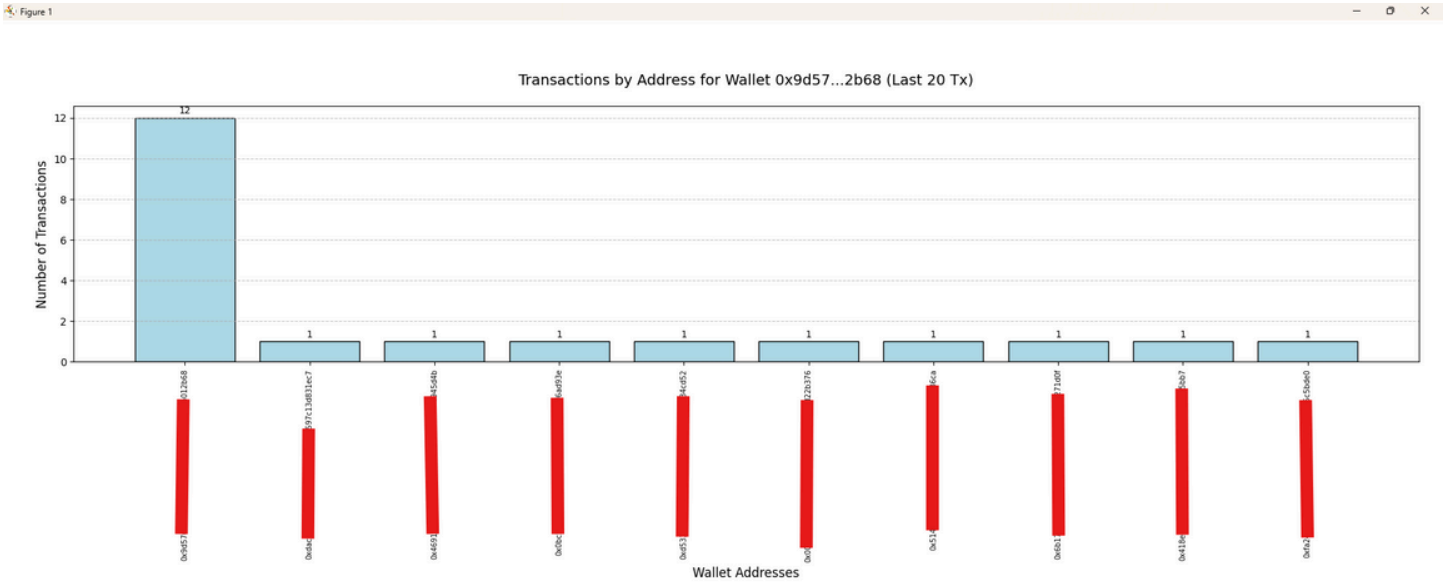

Image of Bar Graph generated by the code

```
[redacted] C:/Python313/python.exe "[redacted]CTCQAT.py"
Enter your Etherscan API key: 5N[redacted]2N4
Enter the wallet address to analyze: 0x9[redacted]2b68
=== Key Insights and Metrics ===
Wallet Address: 0x9[redacted]2b68
Number of Transactions Analyzed: 20
Unique Addresses Involved: 13
Total Value Transferred (ETH): 0.0431
Average Transaction Value (ETH): 0.0022
Transactions in Last 24 Hours: 0
Insight: Low recent activity.

=== Illicit Activity Detection (Basic) ===
Potential Flag: 1 addresses are involved in many transactions (more than 3). These could be popular wallets, exchanges, or services that act as hubs for sending or receiving ETH.
  - Address: 0x9d5[redacted]12b68

Data Source: Etherscan API
```

Image of code Output

Outputs

- **A bar chart:** displaying the number of transactions for up to 10 wallet addresses (full 42-character addresses).
  - X-axis: Full wallet addresses (e.g., 0x09750ad360fdb7a2ee23669c4503c974d86d8694).
  - Y-axis: Number of transactions (total degree).
  - Bars labeled with transaction counts, limited to the top 10 addresses by count.
- **Printed insights:** including transaction metrics and potential illicit activity flags.

Function Descriptions

get_user_inputs()

- **Purpose**: Collects and validates user inputs for the Etherscan API key, wallet address, and number of transactions.
- **Inputs**:
  - API key (string, required).
  - Ethereum wallet address (string, must match 0x followed by 40 hexadecimal characters).
  - Number of transactions (integer, 1–100).
- **Output**: Tuple of (api_key, wallet_address, num_transactions).
- **Error Handling**: Validates inputs using regex for addresses and checks for valid integer range. Exits on invalid input with clear error messages.

```python
def get_user_inputs():
    """
    Get and validate user inputs for API key, wallet address, and number of transactions.
    Handles input errors and validates formats.
    """
    try:
        api_key = input("Enter your Etherscan API key: ").strip()
        if not api_key:
            raise ValueError("API key cannot be empty. Please sign up at https://etherscan.io/apis for a free key.")

        wallet_address = input("Enter the wallet address to analyze: ").strip().lower()
        if not re.match(r'^0x[a-f0-9]{40}$', wallet_address):
            raise ValueError("Invalid Ethereum wallet address. It should start with '0x' followed by 40 hexadecimal characters.")

        num_transactions_str = input("Enter the number of transactions to fetch (e.g., 10-20): ").strip()
        num_transactions = int(num_transactions_str)
        if num_transactions < 1 or num_transactions > 100:  # Arbitrary upper limit to avoid API abuse
            raise ValueError("Number of transactions must be a positive integer between 1 and 100.")

        return api_key, wallet_address, num_transactions
    except ValueError as e:
        print(f"Input Error: {e}")
        exit(1)
    except Exception as e:
        print(f"Unexpected error during input: {e}")
        exit(1)
```

Image of get_user_inputs()

fetch_transactions(api_key, wallet_address, num_transactions)

- **Purpose**: Fetches recent transactions from the Etherscan API for the specified wallet.
- **Inputs**:
  - api_key: Etherscan API key.
  - wallet_address: Ethereum wallet address.
  - num_transactions: Number of transactions to fetch.
- **Output**: List of transaction dictionaries (or empty list if none found).
- **Error Handling**: Handles network errors, API errors, and invalid responses, exiting with descriptive messages.

```python
def fetch_transactions(api_key, wallet_address, num_transactions):
    """
    Fetch recent transactions from Etherscan API.
    Handles API errors and returns the list of transactions.
    """
    url = (
        f"https://api.etherscan.io/api?"
        f"module=account&action=txlist&"
        f"address={wallet_address}&"
        f"startblock=0&endblock=99999999&"
        f"sort=desc&"
        f"apikey={api_key}"
    )
    try:
        response = requests.get(url)
        response.raise_for_status()  # Raise error for bad HTTP status
        data = response.json()

        if data['status'] != '1':
            raise ValueError(f"API Error: {data.get('message', 'Unknown error')}. Result: {data.get('result')}")

        transactions = data['result'][:num_transactions]
        if not transactions:
            print("No transactions found for this address.")

        return transactions
    except requests.exceptions.RequestException as e:
        print(f"Network error fetching data: {e}")
        exit(1)
    except ValueError as e:
        print(e)
        exit(1)
    except Exception as e:
        print(f"Unexpected error during API fetch: {e}")
        exit(1)
```

Image of fetch_transactions(api_key, wallet_address, num_transactions)

build_graph(transactions)

- **Purpose**: Constructs a directed graph where nodes are wallet addresses and edges are transactions weighted by ETH value.
- **Input**: List of transaction dictionaries from Etherscan.
- **Output**: Tuple of (G, total_value_transferred, unique_addresses), where:
  - G: networkx.DiGraph with nodes (addresses) and edges (transactions).
  - total_value_transferred: Sum of ETH values in transactions.
  - unique_addresses: Set of unique wallet addresses.
- **Error Handling**: Catches missing transaction keys and unexpected errors, exiting with messages.

```python
def build_graph(transactions):
    """
    Build a directed graph from transactions.
    Nodes are addresses, edges are transactions with weights as ETH values.
    Returns the graph, total value transferred, and unique addresses.
    """
    try:
        G = nx.DiGraph()
        total_value_transferred = 0.0
        unique_addresses = set()

        for tx in transactions:
            from_addr = tx['from'].lower()
            to_addr = tx['to'].lower()
            value_wei = int(tx['value'])
            value_eth = value_wei / 1e18  # Convert Wei to ETH
            timestamp = datetime.fromtimestamp(int(tx['timeStamp']))

            G.add_edge(from_addr, to_addr, weight=value_eth, tx_hash=tx['hash'], timestamp=timestamp)
            unique_addresses.add(from_addr)
            unique_addresses.add(to_addr)
            total_value_transferred += value_eth

        return G, total_value_transferred, unique_addresses
    except KeyError as e:
        print(f"Error in transaction data: Missing key {e}")
        exit(1)
    except Exception as e:
        print(f"Unexpected error building graph: {e}")
        exit(1)
```

Image of build_graph(transactions)

plot_graph(G, wallet_address, num_transactions)

- **Purpose**: Plots a bar chart showing the number of transactions per wallet address.
- **Inputs**:
  - G: Directed graph from build_graph.
  - wallet_address: Queried wallet address (for title).
  - num_transactions: Number of transactions analyzed (for title).
- **Output**: Bar chart with:
  - X-axis: Full wallet addresses (up to top 10 by transaction count).
  - Y-axis: Transaction count (total degree = in-degree + out-degree).
  - Features: Rotated labels (90°), y-axis grid, transaction count labels on bars.
- **Error Handling**: Catches plotting errors and skips chart with a warning.

```python
def plot_graph(G, wallet_address, num_transactions):
    """
    Plot a bar chart showing the number of transactions per wallet address.
    X-axis: Full wallet addresses.
    Y-axis: Number of transactions (total degree = in-degree + out-degree).
    Limits to top 10 addresses by transaction count if too many for clarity.
    """
    try:
        plt.figure(figsize=(16, 6))  # Wider figure to fit full addresses

        # Calculate transaction count (total degree) for each address
        addresses = list(G.nodes())
        transaction_counts = [G.degree(node) for node in addresses]

        # Sort addresses by transaction count (descending) and limit to top 10 for clarity
        address_count_pairs = sorted(zip(addresses, transaction_counts), key=lambda x: x[1], reverse=True)
        if len(address_count_pairs) > 10:
            print(f"Warning: {len(address_count_pairs)} unique addresses found. Displaying top 10 by transaction count for clarity.")
            address_count_pairs = address_count_pairs[:10]
        addresses, transaction_counts = zip(*address_count_pairs)

        # Use full addresses for labels
        address_labels = list(addresses)  # Full addresses, no shortening

        # Plot bar chart
        plt.bar(address_labels, transaction_counts, color='lightblue', edgecolor='black')
        plt.xlabel('Wallet Addresses', fontsize=12)
        plt.ylabel('Number of Transactions', fontsize=12)
        plt.title(
            f'Transactions by Address for Wallet {wallet_address[:6]}...{wallet_address[-4:]} (Last {num_transactions} Tx)',
            fontsize=14, pad=20
        )

        # Rotate x-axis labels 90 degrees and adjust font size for full addresses
        plt.xticks(rotation=90, ha='center', fontsize=7)
        plt.grid(True, axis='y', linestyle='--', alpha=0.7)  # Add y-axis grid for clarity
        plt.tight_layout(pad=2.0)  # Adjust margins to fit labels

        # Add value labels on top of bars
        for i, count in enumerate(transaction_counts):
            plt.text(i, count + 0.1, str(count), ha='center', va='bottom', fontsize=9)

        plt.show()
    except Exception as e:
        print(f"Error plotting bar chart: {e}. Plotting skipped.")
```

Image of plot_graph(G, wallet_address, num_transactions)

print_insights(transactions, wallet_address, total_value_transferred, unique_addresses)

- **Purpose**: Prints transaction metrics and flags potential illicit activity.
- **Inputs**:
  - transactions: List of transaction dictionaries.
  - wallet_address: Queried wallet address.
  - total_value_transferred: Total ETH transferred.
  - unique_addresses: Set of unique addresses.
- **Output**: Printed insights, including:
  - Wallet address, transaction count, unique addresses, total and average ETH transferred.
  - Recent activity (last 24 hours).
  - Illicit activity flags for high-value transactions (>10 ETH) and high-degree nodes (>3 connections), with full addresses for high-degree nodes.
- **Error Handling**: Catches errors during insight calculation and prints a message.

```python
def print_insights(transactions, wallet_address, total_value_transferred, unique_addresses):
    """
    Print key insights, metrics, and basic illicit activity detection.
    Uses heuristics for flags, displaying full wallet addresses for high-degree nodes.
    """
    try:
        print("=== Key Insights and Metrics ===")
        print(f"Wallet Address: {wallet_address}")
        print(f"Number of Transactions Analyzed: {len(transactions)}")
        print(f"Unique Addresses Involved: {len(unique_addresses)}")
        print(f"Total Value Transferred (ETH): {total_value_transferred:.4f}")
        avg_value = total_value_transferred / len(transactions) if transactions else 0
        print(f"Average Transaction Value (ETH): {avg_value:.4f}")

        # Time-based insight (assuming recent = last 24 hours)
        recent_threshold = datetime.now() - timedelta(hours=24)
        recent_count = sum(1 for tx in transactions if datetime.fromtimestamp(int(tx['timeStamp'])) > recent_threshold)
        print(f"Transactions in Last 24 Hours: {recent_count}")

        # Simple activity insight
        if recent_count > 5:
            print("Insight: High activity detected (multiple transactions in the last day).")
        else:
            print("Insight: Low recent activity.")

        # Detection of possible illicit activity (basic heuristics)
        # Note: This is simplistic; real detection requires blacklists (e.g., from Chainalysis) and more analysis.
        # Flag high-value single tx (>10 ETH) or high-degree nodes (potential mixers/hubs).
        high_value_txs = [tx for tx in transactions if int(tx['value']) / 1e18 > 10]
        G = nx.DiGraph()  # Rebuild minimal graph for degree calculation
        for tx in transactions:
            G.add_edge(tx['from'].lower(), tx['to'].lower())
        high_degree_nodes = [node for node, degree in dict(G.degree()).items() if degree > 3]

        print("\n=== Illicit Activity Detection (Basic) ===")
        if high_value_txs:
            print(f"Potential Flag: {len(high_value_txs)} high-value transactions (>10 ETH) detected. Could indicate large transfers or wash trading.")
            for tx in high_value_txs[:3]:  # Show top 3
                print(f"  - Tx Hash: {tx['hash'][:10]}... Value: {int(tx['value'])/1e18:.2f} ETH")
        if high_degree_nodes:
            print(f"Potential Flag: {len(high_degree_nodes)} addresses are involved in many transactions (more than 3). These could be popular wallets, exchanges, or services that act as hubs for sending or receiving ETH.")
            for node in high_degree_nodes[:3]:
                print(f"  - Address: {node}")  # Display full address
        if not high_value_txs and not high_degree_nodes:
            print("No basic red flags detected. Activity appears routine.")

        print("\nData Source: Etherscan API")
    except Exception as e:
        print(f"Error printing insights: {e}")
```

Image of print_insights(transactions, wallet_address, total_value_transferred, unique_addresses)

Limitations

- **API Dependency**: Requires a valid Etherscan API key and internet connection. The free tier has rate limits (5 requests/second).
- **Illicit Activity Detection**: Uses simplistic heuristics (high-value transactions >10 ETH, high-degree nodes >3). Real detection requires blacklists (e.g., Chainalysis) or advanced analysis.
- **Chart Clarity**: Full 42-character addresses may cause overlap in the bar chart if many unique addresses exist. Limited to top 10 addresses to mitigate this.
- **Transaction Scope**: Only analyzes ETH transfers (not ERC-20 tokens or smart contract interactions) via Etherscan's txlist endpoint.
- **Recent Activity**: The 24-hour threshold for recent transactions assumes the system's local time (e.g., UTC+08 on September 15, 2025).
-

Potential Improvements

- **Enhanced Illicit Detection**: Integrate a blacklist API (e.g., Chainalysis, TRM Labs) for more accurate flagging.
- **Token Support**: Extend to analyze ERC-20 token transfers using Etherscan's tokentx endpoint.
- **Interactive Chart**: Use a library like plotly for interactive bar charts with hoverable full addresses.
- **Customizable Display**: Allow users to adjust the number of addresses shown in the bar chart or sort by criteria other than transaction count.
- **Export Options**: Save insights or the chart to a file (e.g., CSV, PNG).

Troubleshooting

- **ModuleNotFoundError**: If you see No module named 'requests', ensure libraries are installed:
  - pip install requests networkx matplotlib
- Verify the correct Python interpreter in VS Code (Ctrl + Shift + P, Python: Select Interpreter).
- **API Errors**: Check your Etherscan API key and internet connection. Ensure the wallet address is valid.
- **Chart Overlap**: If addresses overlap, reduce the number of transactions (e.g., 10) to limit unique addresses.
- **No Transactions**: If no transactions are found, verify the wallet address on etherscan.io or try a different address.

Example Use Case

To analyze a wallet suspected of phishing (e.g., 0x09750ad360fdb7a2ee23669c4503c974d86d8694):

Run the script and enter:

Enter your Etherscan API key: YOUR_API_KEY

Enter the wallet address to analyze: 0x09750ad360fdb7a2ee23669c4503c974d86d8694

Enter the number of transactions to fetch (e.g., 10-20): 20

Review the bar chart for transaction counts and check the illicit activity section for flagged addresses (shown in full).

License

This script is provided for educational purposes. Ensure compliance with Etherscan's API terms of service.