**The 1000 Project Middleware: README**

Welcome to the middleware for The 1000 Project! This document outlines the purpose, roles, and execution flow for each script. This middleware facilitates a hybrid on-chain/off-chain solution for reward and burn wallet management.

**Overview**

The middleware is designed to:

* Perform daily wallet filtering for eligibility checks.
* Detect potentially malicious P2P activity.
* Randomly select eligible wallets using Chainlink VRF for rewards.
* Minimize on-chain computation to save gas costs by leveraging off-chain processing.

**Project File Structure**

The middleware comprises seven Python scripts, each with a distinct role in the pipeline:

1. **setupanddependencies.py**: Prepares the environment, including APIs, libraries, and configurations.
2. **fetch\_wallet\_data.py**: Pulls wallet and transaction metadata from the blockchain.
3. **detect\_p2p\_transfers.py**: Flags wallets for suspicious P2P activities.
4. **filter\_eligible\_wallets.py**: Filters wallets based on eligibility criteria.
5. **call\_chainlink\_vrf.py**: Randomly selects eligible wallets using Chainlink VRF.
6. **export\_selected\_wallets.py**: Logs the selected wallets for transparency and auditing.
7. **mainfunction.py**: Orchestrates the entire workflow and determines whether it’s a reward or burn day.

**Script Descriptions**

**1. setupanddependencies.py**

* **Purpose**: Prepares the environment by importing required libraries and connecting to APIs.
* **Inputs**: Blockchain endpoint, API keys, and configuration files.
* **Outputs**: Sets global variables for use by subsequent scripts.

**2. fetch\_wallet\_data.py**

* **Purpose**: Fetches on-chain wallet and transaction data.
* **Inputs**: Blockchain metadata (e.g., wallet balances, transactions).
* **Outputs**: Structured database or JSON file with wallet data.

**3. detect\_p2p\_transfers.py**

* **Purpose**: Analyzes transaction patterns to detect suspicious P2P transfers.
* **Inputs**: Transaction history.
* **Outputs**: List of flagged wallets for exclusion from rewards.

**4. filter\_eligible\_wallets.py**

* **Purpose**: Filters wallets to produce a list of eligible holders.
* **Inputs**: Wallet data, flagged wallets, and eligibility criteria.
* **Outputs**: Finalized list of eligible wallets for rewards.

**5. call\_chainlink\_vrf.py**

* **Purpose**: Interfaces with Chainlink VRF to ensure random wallet selection.
* **Inputs**: Eligible wallets list.
* **Outputs**: Randomized list of wallets selected for rewards.

**6. export\_selected\_wallets.py**

* **Purpose**: Saves the list of selected wallets for record-keeping and transparency.
* **Inputs**: Selected wallets and reward amounts.
* **Outputs**: Timestamped file or database entry.

**7. mainfunction.py**

* **Purpose**: Serves as the central workflow manager.
* **Logic**:
  + Determines whether it’s a reward or burn day.
  + Skips wallet filtering and Chainlink VRF calls on burn days.
  + Executes all scripts sequentially on reward days.
* **Inputs**: Configuration flags, reward/burn logic.
* **Outputs**: Executes the middleware pipeline.

**Workflow**

**Burn Days**

1. Skip wallet filtering and selection processes.
2. Execute burn logic directly, burning 1% of the reward wallet balance.

**Reward Days**

1. Fetch wallet and transaction data (fetch\_wallet\_data.py).
2. Detect suspicious P2P transfers (detect\_p2p\_transfers.py).
3. Filter wallets based on eligibility (filter\_eligible\_wallets.py).
4. Randomly select wallets using Chainlink VRF (call\_chainlink\_vrf.py).
5. Export the finalized reward distribution list (export\_selected\_wallets.py).

**Testing Instructions**

1. **Setup**:
   * Install dependencies: pip install -r requirements.txt.
   * Configure API keys and blockchain endpoints in setupanddependencies.py.
2. **Run the Middleware**:
   * Execute mainfunction.py as the entry point.
   * Use flags or configuration files to simulate reward and burn days.
3. **Simulate Reward Days**:
   * Ensure that wallet data is fetched, filtered, and processed through the Chainlink VRF logic.
4. **Simulate Burn Days**:
   * Verify that wallet filtering and Chainlink VRF calls are skipped.
5. **Validate Outputs**:
   * Check the exported files for accuracy.
   * Confirm that flagged wallets are excluded from rewards.

**Notes for Testing**

* Ensure blockchain endpoints are operational and API keys are valid.
* Test with sample data to validate the P2P detection logic.
* Use logging to debug each script and ensure proper sequencing.

**Next Steps**

1. Refactor code for optimization.
2. Test the system under live conditions with real blockchain data.
3. Implement further modularity and scalability features as needed.