Цель

Добавить в наш конвейер новый, третий этап анализа для 5-10 самых перспективных кандидатов, прошедших первичный отбор. Этот этап будет использовать прямое подключение к блокчейну (через RPC-ноду, например, бесплатный тариф Alchemy) для проверки ключевых ончейн-метрик, недоступных через агрегаторы.

Новые Инструменты

1. **RPC-провайдер:** Alchemy или Infura (бесплатный тариф).
2. **Библиотека:** `web3.py` для взаимодействия с Ethereum-совместимыми сетями.
3. `pip install web3`

Новый Компонент: `OnChainAgent`

Мы создадим нового, узкоспециализированного агента — `OnChainAgent`. Его единственная задача — принимать на вход отчет `PumpAnalysisReport` и обогащать его ончейн-данными.

Что именно мы будем анализировать? (Два самых важных сигнала)

Мы сосредоточимся на двух самых важных проверках, которые дают максимальную ценность и легко реализуются.

1. Проверка Блокировки Ликвидности (LP Lock)

- **Проблема:** Команда может в любой момент вынуть всю ликвидность из пула (rug pull).
- **Решение:** Мы проверим, кто является крупнейшими держателями **LP-токенов**. Если >90% этих токенов находятся на адресе известного "локкера" (контракта-замка, например, Unicrypt, Pinksale) или на "мертвом" адресе (`0x00...dead`), это **сигнал высочайшего доверия**.
- **Ценность:** Мгновенно повышает "очки безопасности" токена.

2. Анализ Концентрации Держателей (Holder Concentration)

- **Проблема:** Если один кошелек (не считая биржи или контракта) держит 40% всех токенов, он может обрушить цену в любой момент.
- **Решение:** Мы получим список топ-10 держателей токена и рассчитаем, какой процент эмиссии они контролируют.
- **Ценность:** Дает нам понимание распределения токенов.
    - **Красный флаг:** Очень высокая концентрация в руках нескольких EOA-кошельков (не контрактов).
    - **Зеленый флаг:** Равномерное распределение между сотнями/тысячами кошельков.

План Интеграции в Оркестратор

Наш `SimpleOrchestrator` будет обновлен. После Этапа 2 (обогащение данных из CoinGecko/GoPlus) появится новый **Этап 2.5**.

1. **`run_analysis_pipeline:`**
2. ... (Этап 1: Discovery)
3. ... (Этап 2: Enrichment)
4. **`IF score > 60 THEN:`** (Если токен достаточно интересен)
    - Вызвать `OnChainAgent.analyze(token_report)`.
    - Добавить полученные ончейн-данные в финальный алерт.
5. ... (Этап 3: Scoring & Alerting)

Это позволит нам использовать дорогие RPC-запросы только для самых достойных кандидатов, оставаясь в рамках бесплатных лимитов.

```
import os
from web3 import Web3
from dotenv import load_dotenv

load_dotenv()

# Словарь с RPC-эндпоинтами из .env
# Пример: ETH_RPC_URL=https://eth-mainnet.g.alchemy.com/v2/your_key
RPC_URLS = {
    "ethereum": os.getenv("ETH_RPC_URL"),
    "base": os.getenv("BASE_RPC_URL"),
    "arbitrum": os.getenv("ARBITRUM_RPC_URL"),
    # Solana требует отдельного клиента, для MVP фокусируемся на EVM
}

class RPCClient:
    """Простой клиент для взаимодействия с EVM-совместимыми сетями."""
    def __init__(self):
        self.providers = {
            chain: Web3(Web3.HTTPProvider(url))
            for chain, url in RPC_URLS.items() if url
        }
        if not self.providers:
            print("⚠️ ПРЕДУПРЕЖДЕНИЕ: RPC_URL не настроены в .env. OnChainAgent
будет работать в mock-режиме.")

    def get_provider(self, chain_id: str) -> Web3:
        """Получить web3 провайдер для указанной сети."""
        provider = self.providers.get(chain_id)
        if not provider:
            raise ValueError(f"RPC для сети '{chain_id}' не настроен.")
        if not provider.is_connected():
```

```
            raise ConnectionError(f"Не удалось подключиться к RPC для сети
'{chain_id}'.")
        return provider

    def get_token_balance(self, chain_id: str, contract_address: str,
holder_address: str) -> float:
        """Получить баланс токена для конкретного держателя."""
        # Упрощенная реализация - для получения баланса нужен ABI контракта
        # В реальной системе здесь будет более сложная логика
        print(f"DEBUG: Запрос баланса {contract_address} для {holder_address}")
        return 0.0 # Возвращаем mock-данные

    def get_top_holders(self, chain_id: str, contract_address: str):
        """
        Получить топ-держателей токена.
        ПРИМЕЧАНИЕ: Это очень сложная задача для прямого RPC.
        Обычно для этого используют сторонние API-индексаторы (Etherscan,
Covalent).
        В данном примере мы имитируем результат.
        """
        print(f"DEBUG: Запрос топ-держателей для {contract_address}")
        return [
            {"address": "0x123...abc", "percentage": 15.5},
            {"address": "0x456...def", "percentage": 8.2},
        ]


import asyncio
from typing import Dict, Any
from pydantic import BaseModel

from tools.blockchain.rpc_client import RPCClient
# Предполагаем, что отчет передается в виде словаря или Pydantic модели
from agents.pump_analysis.pump_models import PumpAnalysisReport

class OnChainSignalReport(BaseModel):
    """Отчет с ончейн-сигналами."""
    is_liquidity_locked: bool = False
    lp_lock_percentage: float = 0.0
    holder_concentration_top_10: float = 0.0
    red_flags: list[str] = []
    positive_signals: list[str] = []

class OnChainAgent:
    """Агент для глубокого ончейн-анализа."""
    def __init__(self):
        self.rpc_client = RPCClient()
```

```
    def analyze_liquidity_lock(self, chain_id: str, lp_token_address: str) ->
tuple[bool, float]:
        """
        Анализирует, заблокирована ли ликвидность.
        Упрощенная логика для MVP.
        """
        # В реальной системе мы бы проверили топ-держателей LP токена
        # и сравнили их адреса с известными адресами локеров.
        print(f"DEBUG: Проверка блокировки ликвидности для {lp_token_address}")
        # Имитируем результат: 85% ликвидности заблокировано
        return True, 85.0

    def analyze_holder_concentration(self, chain_id: str, token_address: str) ->
float:
        """
        Анализирует концентрацию токенов у топ-держателей.
        """
        top_holders = self.rpc_client.get_top_holders(chain_id, token_address)
        concentration = sum(holder['percentage'] for holder in top_holders)
        return concentration

    async def analyze(self, report: PumpAnalysisReport) -> OnChainSignalReport:
        """
        Основной метод анализа.
        Принимает отчет и обогащает его ончейн-данными.
        """
        loop = asyncio.get_running_loop()
        return await loop.run_in_executor(
            None, self.analyze_sync, report
        )

    def analyze_sync(self, report: PumpAnalysisReport) -> OnChainSignalReport:
        """Синхронная версия для выполнения в executor."""
        try:
            # Для анализа ликвидности нам нужен адрес LP-пары
            # В `PumpAnalysisReport` у нас пока нет этого поля, но его легко
добавить
            lp_address = report.pair_address # Используем адрес пары как LP-адрес

            is_locked, lock_percent = self.analyze_liquidity_lock(report.chain_id,
lp_address)
            concentration = self.analyze_holder_concentration(report.chain_id,
report.contract_address)

            onchain_report = OnChainSignalReport(
                is_liquidity_locked=is_locked,
```

```
                lp_lock_percentage=lock_percent,
                holder_concentration_top_10=concentration,
            )

            if is_locked and lock_percent > 80:
                onchain_report.positive_signals.append(f"✅ Ликвидность
заблокирована ({lock_percent}%)")
            else:
                onchain_report.red_flags.append(f"⚠️ Ликвидность не заблокирована
или заблокирована частично ({lock_percent}%)")

            if concentration > 50:
                onchain_report.red_flags.append(f"⚠️ Высокая концентрация: топ-10
держат {concentration}%")
            elif concentration > 30:
                onchain_report.positive_signals.append(f"ℹ️ Умеренная концентрация:
топ-10 держат {concentration}%")
            else:
                onchain_report.positive_signals.append(f"✅ Низкая концентрация:
топ-10 держат {concentration}%")

            return onchain_report

        except Exception as e:
            print(f"ОШИБКА в OnChainAgent: {e}")
            return OnChainSignalReport(red_flags=["Ошибка ончейн-анализа"])
```