// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

import "@aave/core-v3/contracts/interfaces/IPool.sol";

import "@uniswap/v3-core/contracts/interfaces/callback/IUniswapV3FlashCallback.sol";

import "@balancer-labs/v2-vault/contracts/interfaces/IVault.sol";

contract FlashLoanExecutor is ReentrancyGuard, IUniswapV3FlashCallback {

address public immutable owner;

address public immutable profitWallet;

address public constant AAVE\_POOL = 0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2; // Aave V3 Pool (Ethereum)

address public constant UNISWAP\_V3\_FACTORY = 0x1F98431c8aD98523631AE4a59f267346ea31F984;

address public constant BALANCER\_VAULT = 0xBA12222222228d8Ba445958a75a0704d566BF2C8;

event FlashLoanExecuted(address indexed provider, address token, uint256 amount, uint256 profit);

event ProfitTransferred(address indexed to, uint256 amount);

modifier onlyOwner() {

require(msg.sender == owner, "Not owner");

\_;

}

constructor(address \_profitWallet) {

owner = msg.sender;

profitWallet = \_profitWallet;

}

// Aave Flash Loan

function executeAaveFlashLoan(address token, uint256 amount, bytes calldata params) external onlyOwner {

IPool(AAVE\_POOL).flashLoanSimple(address(this), token, amount, params, 0);

}

// Uniswap V3 Flash Swap

function executeUniswapFlashSwap(address pool, uint256 amount0, uint256 amount1, bytes calldata data) external onlyOwner {

IUniswapV3PoolActions(pool).flash(address(this), amount0, amount1, data);

}

// Balancer Flash Loan

function executeBalancerFlashLoan(address[] memory tokens, uint256[] memory amounts) external onlyOwner {

IVault(BALANCER\_VAULT).flashLoan(address(this), tokens, amounts, "");

}

// Aave callback

function executeOperation(

address token,

uint256 amount,

uint256 fee,

address,

bytes calldata params

) external nonReentrant returns (bool) {

require(msg.sender == AAVE\_POOL, "Unauthorized");

// Example: Arbitrage logic (swap on Uniswap, etc.)

// For simplicity, assume profit is made

uint256 totalDebt = amount + fee;

IERC20(token).approve(AAVE\_POOL, totalDebt);

uint256 profit = IERC20(token).balanceOf(address(this)) - totalDebt;

if (profit > 0) {

IERC20(token).transfer(profitWallet, profit);

emit ProfitTransferred(profitWallet, profit);

}

emit FlashLoanExecuted(AAVE\_POOL, token, amount, profit);

return true;

}

// Uniswap V3 callback

function uniswapV3FlashCallback(uint256 fee0, uint256 fee1, bytes calldata data) external override {

// Verify caller is Uniswap pool

// Arbitrage logic here

uint256 profit = IERC20(data).balanceOf(address(this)) - (fee0 + fee1);

if (profit > 0) {

IERC20(data).transfer(profitWallet, profit);

emit ProfitTransferred(profitWallet, profit);

}

emit FlashLoanExecuted(msg.sender, data, fee0 + fee1, profit);

}

// Balancer callback

function receiveFlashLoan(

IERC20[] memory tokens,

uint256[] memory amounts,

uint256[] memory fees,

bytes memory

) external {

require(msg.sender == BALANCER\_VAULT, "Unauthorized");

uint256 totalProfit = 0;

for (uint i = 0; i < tokens.length; i++) {

uint256 totalDebt = amounts[i] + fees[i];

tokens[i].approve(BALANCER\_VAULT, totalDebt);

uint256 profit = tokens[i].balanceOf(address(this)) - totalDebt;

if (profit > 0) {

tokens[i].transfer(profitWallet, profit);

totalProfit += profit;

}

}

emit FlashLoanExecuted(BALANCER\_VAULT, address(tokens[0]), amounts[0], totalProfit);

}

// Withdraw profits to external wallet

function withdrawProfit(address token, address to, uint256 amount) external onlyOwner {

IERC20(token).transfer(to, amount);

emit ProfitTransferred(to, amount);

}

}