Smart Contract Security Audit Report

2018-10-19

On the 2018-10-11 day, the SlowMist Security Team received the Fragments team's application for smart contract security audit of the μFragments project. The following are the details and results of this smart contract security audit :

Project Name :

μFragments

Smart Contract Files:

- uFragments-master/contracts

├── UFragments.sol
├── UFragmentsPolicy.sol
├── lib
│ ├── SafeMathInt.sol
│ └── UInt256Lib.sol

- market-oracle-master/contracts

├── MarketOracle.sol
├── MarketSource.sol
├── MarketSourceFactory.sol

The Smart Contract Code Link:

https://github.com/frgprotocol/uFragments/tree/07437020b54c535ced2f4b5f1a0cc1a2ee6618e3

https://github.com/frgprotocol/market-oracle/tree/888fccaf05786f3f7f49e18ff040f911d44906f4

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed

		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False top-up” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed
10	Uninitialized Storage Pointers Audit	-	Passed
11	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X001810190001

Audit Date : October 19, 2018

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as “the provided information”). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects.)

Summary: The contract does not have the Overflow, the Race Conditions issue, Optimize gas usage, It is recommended to change "assert" to "require"

uFragments/lib/SafeMathInt.sol

```
/*
MIT License
```

```
/*
Copyright (c) 2018 requestnetwork
Copyright (c) 2018 Fragments, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

pragma solidity 0.4.24;


/**
 * @title SafeMathInt
 * @dev Math operations for int256 with overflow safety checks.
 */
library SafeMathInt {
    int256 private constant MIN_INT256 = int256(1) << 255;
    int256 private constant MAX_INT256 = ~(int256(1) << 255);

    /**
     * @dev Multiplies two int256 variables and fails on overflow.
     */
    function mul(int256 a, int256 b)
        internal
        pure
        returns (int256)
    {
```

```
    int256 c = a * b;

    // Detect overflow when multiplying MIN_INT256 with -1
    require(c != MIN_INT256 || (a & MIN_INT256) != (b & MIN_INT256));
    require((b == 0) || (c / b == a));
    return c;
}

/**
 * @dev Division of two int256 variables and fails on overflow.
 */
function div(int256 a, int256 b)
    internal
    pure
    returns (int256)
{
    // Prevent overflow when dividing MIN_INT256 by -1
    require(b != -1 || a != MIN_INT256);

    // Solidity already throws when dividing by 0.
    return a / b;
}

/**
 * @dev Subtracts two int256 variables and fails on overflow.
 */
function sub(int256 a, int256 b)
    internal
    pure
    returns (int256)
{
    int256 c = a - b;
    require((b >= 0 && c <= a) || (b < 0 && c > a));
    return c;
}

/**
 * @dev Adds two int256 variables and fails on overflow.
 */
function add(int256 a, int256 b)
    internal
    pure
```

```
        returns (int256)
    {
        int256 c = a + b;
        require((b >= 0 && c >= a) || (b < 0 && c < a));
        return c;
    }

    /**
     * @dev Converts to absolute value, and fails on overflow.
     */
    function abs(int256 a)
        internal
        pure
        returns (int256)
    {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }
}
```

uFragments/lib/UInt256Lib.sol

```
pragma solidity 0.4.24;



/**
 * @title Various utilities useful for uint256.
 */
library UInt256Lib {

    uint256 private constant MAX_INT256 = ~(uint256(1) << 255);

    /**
     * @dev Safely converts a uint256 to an int256.
     */
    function toInt256Safe(uint256 a)
        internal
        pure
        returns (int256)
    {
        require(a <= MAX_INT256);
        return int256(a);
    }
```

```
}
```

uFragments/contracts/UFragments.sol

```
pragma solidity 0.4.24;
```

//SlowMist// OpenZeppelin-zos's SafeMath, Ownable and DetailedERC20 modules are used, which is a recommended approach

```
import "openzeppelin-zos/contracts/math/SafeMath.sol";
import "openzeppelin-zos/contracts/ownership/Ownable.sol";
import "openzeppelin-zos/contracts/token/ERC20/DetailedERC20.sol";

import "./lib/SafeMathInt.sol";



/**
 * @title uFragments ERC20 token
 * @dev This is part of an implementation of the uFragments Ideal Money protocol.
 *      uFragments is a normal ERC20 token, but its supply can be adjusted by splitting and
 *      combining tokens proportionally across all wallets.
 *
 *      uFragment balances are internally represented with a hidden denomination, 'gons'.
 *      We support splitting the currency in expansion and combining the currency on contraction by
 *      changing the exchange rate between the hidden 'gons' and the public 'fragments'.
 */
contract UFragments is DetailedERC20, Ownable {
    // PLEASE READ BEFORE CHANGING ANY ACCOUNTING OR MATH
    // Anytime there is division, there is a risk of numerical instability from rounding errors. In
    // order to minimize this risk, we adhere to the following guidelines:
    // 1) The conversion rate adopted is the number of gons that equals 1 fragment.
    //    The inverse rate must not be used--TOTAL_GONS is always the numerator and _totalSupply is
    //    always the denominator. (i.e. If you want to convert gons to fragments instead of
    //    multiplying by the inverse rate, you should divide by the normal rate)
    // 2) Gon balances converted into Fragments are always rounded down (truncated).
    //
    // We make the following guarantees:
    // - If address 'A' transfers x Fragments to address 'B'. A's resulting external balance will
    //   be decreased by precisely x Fragments, and B's external balance will be precisely
    //   increased by x Fragments.
    //
```

```
    // We do not guarantee that the sum of all balances equals the result of calling totalSupply().
    // This is because, for any conversion function 'f()' that has non-zero rounding error,
    // f(x0) + f(x1) + ... + f(xn) is not always equal to f(x0 + x1 + ... xn).
    using SafeMath for uint256;
    using SafeMathInt for int256;

    event LogRebase(uint256 indexed epoch, uint256 totalSupply);
    event LogRebasePaused(bool paused);
    event LogTokenPaused(bool paused);

    // Used for authentication
    address public _monetaryPolicy;

    modifier onlyMonetaryPolicy() {
        require(msg.sender == _monetaryPolicy);
        _;
    }
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach

```
    // Precautionary emergency controls.
    bool public _rebasePaused;
    bool public _tokenPaused;

    modifier whenRebaseNotPaused() {
        require(!_rebasePaused);
        _;
    }

    modifier whenTokenNotPaused() {
        require(!_tokenPaused);
        _;
    }

    modifier validRecipient(address to) {
        require(to != address(0x0));
        require(to != address(this));
        _;
    }

    uint256 private constant DECIMALS = 9;
```

```
uint256 private constant MAX_UINT256 = ~uint256(0);
uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 50 * 10**6 * 10**DECIMALS;

// TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.
// Use the highest value that fits in a uint256 for max granularity.
uint256 private constant TOTAL_GONS = MAX_UINT256.sub(MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);

// MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2
uint256 private constant MAX_SUPPLY = ~uint128(0);  // (2^128) - 1

uint256 private _totalSupply;
uint256 private _gonsPerFragment;
mapping(address => uint256) private _gonBalances;

// This is denominated in Fragments, because the gons-fragments conversion might change before
// it's fully paid.
mapping (address => mapping (address => uint256)) private _allowedFragments;

/**
 * @param monetaryPolicy The address of the monetary policy contract to use for authentication.
 */
function setMonetaryPolicy(address monetaryPolicy)
    external
    onlyOwner
{
    _monetaryPolicy = monetaryPolicy;
}

/**
 * @dev Pauses or unpauses the execution of rebase operations.
 * @param paused Pauses rebase operations if this is true.
 */
function setRebasePaused(bool paused)
    external
    onlyOwner
{
    _rebasePaused = paused;
    emit LogRebasePaused(paused);
}

/**
 * @dev Pauses or unpauses execution of ERC-20 transactions.
```

```
     * @param paused Pauses ERC-20 transactions if this is true.
     */
    function setTokenPaused(bool paused)
        external
        onlyOwner
    {
        _tokenPaused = paused;
        emit LogTokenPaused(paused);
    }

    /**
     * @dev Notifies Fragments contract about a new rebase cycle.
     * @param supplyDelta The number of new fragment tokens to add into circulation via expansion.
     * @return The total number of fragments after the supply adjustment.
     */
    function rebase(uint256 epoch, int256 supplyDelta)
        external
        onlyMonetaryPolicy
        whenRebaseNotPaused
        returns (uint256)
    {
        if (supplyDelta == 0) {
            emit LogRebase(epoch, _totalSupply);
            return _totalSupply;
        }

        if (supplyDelta < 0) {
            _totalSupply = _totalSupply.sub(uint256(supplyDelta.abs()));
        } else {
            _totalSupply = _totalSupply.add(uint256(supplyDelta));
        }

        if (_totalSupply > MAX_SUPPLY) {
            _totalSupply = MAX_SUPPLY;
        }

        _gonsPerFragment = TOTAL_GONS.div(_totalSupply);

        // From this point forward, _gonsPerFragment is taken as the source of truth.
        // We recalculate a new _totalSupply to be in agreement with the _gonsPerFragment
        // conversion rate.
        // This means our applied supplyDelta can deviate from the requested supplyDelta,
```

```
    // but this deviation is guaranteed to be < (_totalSupply^2)/(TOTAL_GONS - _totalSupply).
    //
    // In the case of _totalSupply <= MAX_UINT128 (our current supply cap), this
    // deviation is guaranteed to be < 1, so we can omit this step. If the supply cap is
    // ever increased, it must be re-included.
    // _totalSupply = TOTAL_GONS.div(_gonsPerFragment)

    emit LogRebase(epoch, _totalSupply);
    return _totalSupply;
}

function initialize(address owner)
    public
    isInitializer("UFragments", "1.0.0" /* Version ID */)
{
    DetailedERC20.initialize("UFragments", "UFRG", uint8(DECIMALS));
    Ownable.initialize(owner);

    _rebasePaused = false;
    _tokenPaused = false;

    _totalSupply = INITIAL_FRAGMENTS_SUPPLY;
    _gonBalances[owner] = TOTAL_GONS;
    _gonsPerFragment = TOTAL_GONS.div(_totalSupply);

    emit Transfer(address(0x0), owner, _totalSupply);
}

/**
 * @return The total number of fragments.
 */
function totalSupply()
    public
    view
    returns (uint256)
{
    return _totalSupply;
}

/**
 * @param who The address to query.
 * @return The balance of the specified address.
```

```
 */
function balanceOf(address who)
    public
    view
    returns (uint256)
{
    return _gonBalances[who].div(_gonsPerFragment);
}

/**
 * @dev Transfer tokens to a specified address.
 * @param to The address to transfer to.
 * @param value The amount to be transferred.
 * @return True on success, false otherwise.
 */
function transfer(address to, uint256 value)
    public
    validRecipient(to)
    whenTokenNotPaused
    returns (bool)
{
    uint256 gonValue = value.mul(_gonsPerFragment);
    _gonBalances[msg.sender] = _gonBalances[msg.sender].sub(gonValue);
    _gonBalances[to] = _gonBalances[to].add(gonValue);
    emit Transfer(msg.sender, to, value);

    return true; //SlowMist// The returned value complies with the EIP20 specification

}

/**
 * @dev Function to check the amount of tokens that an owner has allowed to a spender.
 * @param owner The address which owns the funds.
 * @param spender The address which will spend the funds.
 * @return The number of tokens still available for the spender.
 */
function allowance(address owner, address spender)
    public
    view
    returns (uint256)
{
    return _allowedFragments[owner][spender];
}
```

```
/**
 * @dev Transfer tokens from one address to another.
 * @param from The address you want to send tokens from.
 * @param to The address you want to transfer to.
 * @param value The amount of tokens to be transferred.
 */
function transferFrom(address from, address to, uint256 value)
    public
    validRecipient(to)
    whenTokenNotPaused
    returns (bool)
{
    _allowedFragments[from][msg.sender] = _allowedFragments[from][msg.sender].sub(value);

    uint256 gonValue = value.mul(_gonsPerFragment);
    _gonBalances[from] = _gonBalances[from].sub(gonValue);
    _gonBalances[to] = _gonBalances[to].add(gonValue);
    emit Transfer(from, to, value);

    return true; //SlowMist// The returned value complies with the EIP20 specification
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf of
 * msg.sender. This method is included for ERC20 compatibility.
 * increaseApproval and decreaseApproval should be used instead.
 * Changing an allowance with this method brings the risk that someone may transfer both
 * the old and the new allowance - if they are both greater than zero - if a transfer
 * transaction is mined before the later approve() call is mined.
 *
 * @param spender The address which will spend the funds.
 * @param value The amount of tokens to be spent.
 */
function approve(address spender, uint256 value)
    public
    whenTokenNotPaused
    returns (bool)
{
    _allowedFragments[msg.sender][spender] = value;
    emit Approval(msg.sender, spender, value);
```

```
    return true; //SlowMist// The returned value complies with the EIP20 specification
}

/**
 * @dev Increase the amount of tokens that an owner has allowed to a spender.
 * This method should be used instead of approve() to avoid the double approval vulnerability
 * described above.
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(address spender, uint256 addedValue)
    public
    whenTokenNotPaused
    returns (bool)
{
    _allowedFragments[msg.sender][spender] =
        _allowedFragments[msg.sender][spender].add(addedValue);
    emit Approval(msg.sender, spender, _allowedFragments[msg.sender][spender]);
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner has allowed to a spender.
 *
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseAllowance(address spender, uint256 subtractedValue)
    public
    whenTokenNotPaused
    returns (bool)
{
    uint256 oldValue = _allowedFragments[msg.sender][spender];
    if (subtractedValue >= oldValue) {
        _allowedFragments[msg.sender][spender] = 0;
    } else {
        _allowedFragments[msg.sender][spender] = oldValue.sub(subtractedValue);
    }
    emit Approval(msg.sender, spender, _allowedFragments[msg.sender][spender]);
    return true;
}
```

```
}
```

uFragments/contracts/UFragmentsPolicy.sol

//SlowMist// OpenZeppelin-zos's SafeMath and Ownable modules are used, which is a recommended approach

```
pragma solidity 0.4.24;

import "openzeppelin-zos/contracts/math/SafeMath.sol";
import "openzeppelin-zos/contracts/ownership/Ownable.sol";

import "./lib/SafeMathInt.sol";
import "./lib/UInt256Lib.sol";
import "./UFragments.sol";


interface IMarketOracle {
    function getPriceAnd24HourVolume() external returns (uint256, uint256);
}


/**
 * @title uFragments Monetary Supply Policy
 * @dev This is an implementation of the uFragments Ideal Money protocol.
 *      uFragments operates symmetrically on expansion and contraction. It will both split and
 *      combine coins to maintain a stable unit price.
 *
 *      This component regulates the token supply of the uFragments ERC20 token in response to
 *      market oracles.
 */
contract UFragmentsPolicy is Ownable {
    using SafeMath for uint256;
    using SafeMathInt for int256;
    using UInt256Lib for uint256;

    event LogRebase(
        uint256 indexed epoch,
        uint256 exchangeRate,
        uint256 volume24hrs,
```

```
    int256 requestedSupplyAdjustment
);

UFragments public _uFrags;
IMarketOracle public _marketOracle;

// If the current exchange rate is within this absolute distance from the target, no supply
// update is performed. Fixed point number--same format as the rate.
uint256 public _deviationThreshold;

// The rebase lag parameter, used to dampen the applied supply adjustment by 1 / _rebaseLag
// Check setRebaseLag comments for more details.
uint256 public _rebaseLag;

// At least this much time must pass between rebase operations.
uint256 public _minRebaseTimeIntervalSec;

// Block timestamp of last rebase operation
uint256 public _lastRebaseTimestampSec;

// The number of rebase cycles since inception
uint256 public _epoch;

uint256 private constant RATE_DECIMALS = 18;

uint256 private constant TARGET_RATE = 1 * 10**RATE_DECIMALS;

int256 private constant TARGET_RATE_SIGNED = int256(TARGET_RATE);

// Due to the expression in computeSupplyDelta(), MAX_RATE * MAX_SUPPLY must fit into an int256.
// Both are 18 decimals fixed point numbers.
uint256 private constant MAX_RATE = 10**6 * 10**RATE_DECIMALS;
// MAX_SUPPLY = MAX_INT256 / MAX_RATE
uint256 private constant MAX_SUPPLY = ~(uint256(1) << 255) / MAX_RATE;

/**
 * @notice Anyone can call this function to initiate a new rebase operation, provided the
 *         minimum time period has elapsed.
 * @dev The supply adjustment equals (_totalSupply * DeviationFromTargetRate) / _rebaseLag
 *      Where DeviationFromTargetRate is (MarketOracleRate - TARGET_RATE) / TARGET_RATE
 */
function rebase() external {
```

```
        require(_lastRebaseTimestampSec.add(_minRebaseTimeIntervalSec) <= now);
        _epoch = _epoch.add(1);
        _lastRebaseTimestampSec = now;

        uint256 exchangeRate;
        uint256 volume;
        (exchangeRate, volume) = _marketOracle.getPriceAnd24HourVolume();
        if (exchangeRate > MAX_RATE) {
            exchangeRate = MAX_RATE;
        }

        int256 supplyDelta = computeSupplyDelta(exchangeRate);
        // Apply the Dampening factor.
        supplyDelta = supplyDelta.div(_rebaseLag.toInt256Safe());

        if (supplyDelta > 0 && _uFrags.totalSupply().add(uint256(supplyDelta)) > MAX_SUPPLY) {
            supplyDelta = (MAX_SUPPLY.sub(_uFrags.totalSupply())).toInt256Safe();
        }

        uint256 supplyAfterRebase = _uFrags.rebase(_epoch, supplyDelta);

        assert(supplyAfterRebase <= MAX_SUPPLY); //SlowMist// Optimize gas usage, It is
```

recommended to change assert to require

```
        emit LogRebase(_epoch, exchangeRate, volume, supplyDelta);
    }

    /**
     * @notice Sets the reference to the market oracle.
     * @param marketOracle The address of the market oracle contract.
     */
    function setMarketOracle(IMarketOracle marketOracle)
        external
        onlyOwner
    {
        _marketOracle = marketOracle;
    }

    /**
     * @notice Sets the deviation threshold. If the exchange rate given by the market
     *         oracle is within this absolute distance from the target, then no supply
     *         modifications are made. RATE_DECIMALS fixed point number.
```

```
     * @param deviationThreshold The new exchange rate threshold.
     */
    function setDeviationThreshold(uint256 deviationThreshold)
        external
        onlyOwner
    {
        _deviationThreshold = deviationThreshold;
    }

    /**
     * @notice Sets the minimum time period that must elapse between rebase cycles.
     * @param minRebaseTimeIntervalSec The new minimum time interval, in seconds.
     */
    function setMinRebaseTimeIntervalSec(uint256 minRebaseTimeIntervalSec)
        external
        onlyOwner
    {
        _minRebaseTimeIntervalSec = minRebaseTimeIntervalSec;
    }

    /**
     * @notice Sets the rebase lag parameter.
               It is used to dampen the applied supply adjustment by 1 / _rebaseLag
               If the rebase lag R, equals 1, the smallest value for R, then the full supply
               correction is applied on each rebase cycle.
               If it is greater than 1, then a correction of 1/R of is applied on each rebase.
     * @param rebaseLag The new rebase lag parameter.
     */
    function setRebaseLag(uint256 rebaseLag)
        external
        onlyOwner
    {
        require(rebaseLag > 0);
        _rebaseLag = rebaseLag;
    }

    /**
     * @dev ZOS upgradable contract initialization method.
     *      It is called at the time of contract creation to invoke parent class initializers and
     *      initialize the contract's state variables.
     */
    function initialize(address owner, UFragments uFrags)
```

```
    public
    isInitializer("UFragmentsPolicy", "1.0.0" /* Version ID */)
{
    Ownable.initialize(owner);

    _deviationThreshold = (5 * TARGET_RATE) / 100;  // 5% of target
    _rebaseLag = 30;
    _minRebaseTimeIntervalSec = 1 days;
    _lastRebaseTimestampSec = 0;
    _epoch = 0;

    _uFrags = uFrags;
}

/**
 * @return Computes the total supply adjustment in response to the exchange rate.
 */
function computeSupplyDelta(uint256 rate)
    private
    view
    returns (int256)
{
    if (withinDeviationThreshold(rate)) {
        return 0;
    }

    // (totalSupply * (rate - target)) / target
    return _uFrags.totalSupply().toInt256Safe().mul(
        rate.toInt256Safe().sub(TARGET_RATE_SIGNED)
    ).div(TARGET_RATE_SIGNED);
}

/**
 * @param rate The current exchange rate, an 18 decimal fixed point number.
 * @return If the rate is within the deviation threshold from the target rate, returns true.
 *         Otherwise, returns false.
 */
function withinDeviationThreshold(uint256 rate)
    private
    view
    returns (bool)
{
```

```
        return (rate >= TARGET_RATE && rate.sub(TARGET_RATE) < _deviationThreshold)
            || (rate < TARGET_RATE && TARGET_RATE.sub(rate) < _deviationThreshold);
    }
}
```

market-oracle/contracts/MarketOracle.sol

//SlowMist// OpenZeppelin-solidity SafeMath and Ownable modules are used, which is a recommended approach

```
pragma solidity 0.4.24;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "openzeppelin-solidity/contracts/ownership/Ownable.sol";

import "./MarketSource.sol";



/**
 * @title Market Oracle
 *
 * @dev Provides the exchange rate and volume data onchain using data from a whitelisted
 *      set of market source contracts.
 *      Exchange rate is the TOKEN:TARGET rate.
 *      Volume is a 24 hour trading volume in Token volume.
 */
contract MarketOracle is Ownable {
    using SafeMath for uint256;

    // Whitelist of sources
    MarketSource[] public _whitelist;

    event LogSourceAdded(MarketSource source);
    event LogSourceRemoved(MarketSource source);
    event LogSourceExpired(MarketSource source);

    /**
     * @dev Calculates the volume weighted average of exchange rates and total trade volume.
     *      Expired market sources are ignored.
     * @return exchangeRate: Volume weighted average of exchange rates.
```

```
     *         volume: Total trade volume of the last reported 24 hours in Token volume.
     */
    function getPriceAnd24HourVolume()
        external
        returns (uint256, uint256)
    {
        uint256 volumeWeightedSum = 0;
        uint256 volumeSum = 0;
        uint256 partialRate = 0;
        uint256 partialVolume = 0;
        bool isSourceFresh = false;

        for (uint256 i = 0; i < _whitelist.length; i++) {
            (isSourceFresh, partialRate, partialVolume) = _whitelist[i].getReport();

            if (!isSourceFresh) {
                emit LogSourceExpired(_whitelist[i]);
                continue;
            }

            volumeWeightedSum = volumeWeightedSum.add(partialRate.mul(partialVolume));
            volumeSum = volumeSum.add(partialVolume);
        }

        // No explicit fixed point normalization is done as dividing by volumeSum normalizes
        // to exchangeRate's format.
        uint256 exchangeRate = volumeWeightedSum.div(volumeSum);
        return (exchangeRate, volumeSum);
    }

    /**
     * @dev Adds a market source to the whitelist.
     * @param source Address of the MarketSource.
     */
    function addSource(MarketSource source)
        external
        onlyOwner
    {
        _whitelist.push(source);
        emit LogSourceAdded(source);
    }
```

```
/**
 * @dev Removes the provided market source from the whitelist.
 * @param source Address of the MarketSource.
 */
function removeSource(MarketSource source)
    external
    onlyOwner
{
    for (uint256 i = 0; i < _whitelist.length; i++) {
        if (_whitelist[i] == source) {
            removeSourceAtIndex(i);
            break;
        }
    }
}

/**
 * @dev Expunges from the whitelist any MarketSource whose associated contracts have been
 *      destructed.
 */
function removeDestructedSources()
    external
{
    uint256 i = 0;
    while (i < _whitelist.length) {
        if (isContractDestructed(_whitelist[i])) {
            removeSourceAtIndex(i);
        } else {
            i++;
        }
    }
}

/**
 * @return The number of market sources in the whitelist.
 */
function whitelistSize()
    public
    view
    returns (uint256)
{
    return _whitelist.length;
```

```
    }

    /**
     * @dev Checks if a contract has been destructed.
     * @param contractAddress Address of the contract.
     */
    function isContractDestructed(address contractAddress)
        private
        view
        returns (bool)
    {
        uint256 size;
        assembly { size := extcodesize(contractAddress) }
        return size == 0;
    }

   /**
    * @param index Index of the MarketSource to be removed from the whitelist.
    */
    function removeSourceAtIndex(uint256 index)
        private
    {
        emit LogSourceRemoved(_whitelist[index]);
        if (index != _whitelist.length-1) {
            _whitelist[index] = _whitelist[_whitelist.length-1];
        }
        _whitelist.length--;
    }
}
```

market-oracle/contracts/MarketSource.sol

//SlowMist// OpenZeppelin-solidity SafeMath and Destructible modules are used, which is a recommended approach

```
pragma solidity 0.4.24;

import "openzeppelin-solidity/contracts/lifecycle/Destructible.sol";
import "openzeppelin-solidity/contracts/math/SafeMath.sol";
```

```
/**
 * @title Market Source
 *
 * @dev Provides the exchange rate and the 24 hour trading volume of a trading pair on a market.
 *      This can only receive data from a single trusted source, the owner address.
 *
 */
contract MarketSource is Destructible {
    using SafeMath for uint256;

    event LogExchangeRateReported(
        uint128 exchangeRate,
        uint128 volume24hrs,
        uint64 indexed timestampSec
    );

    // Name of the source reporting exchange rates
    string public _name;

    // These are the three oracle values that are continuously reported.
    // Smaller types are used here locally to save on storage gas.
    uint128 private _exchangeRate;
    uint128 private _volume24hrs;
    uint64 private _timestampSec;

    // The number of seconds after which the report must be deemed expired.
    uint64 public _reportExpirationTimeSec;

    constructor(string name, uint64 reportExpirationTimeSec) public {
        _name = name;
        _reportExpirationTimeSec = reportExpirationTimeSec;
    }

    /**
     * @param exchangeRate The average exchange rate over the past 24 hours of TOKEN:TARGET.
     *                     18 decimal fixed point number.
     * @param volume24hrs The trade volume of the past 24 hours in Token volume.
     *                    18 decimal fixed point number.
     * @param timestampSec The off chain timestamp of the observation.
     */
    function reportRate(uint128 exchangeRate, uint128 volume24hrs, uint64 timestampSec)
```

```
        external
        onlyOwner
    {
        require(exchangeRate > 0);
        require(volume24hrs > 0);

        _exchangeRate = exchangeRate;
        _volume24hrs = volume24hrs;
        _timestampSec = timestampSec;

        emit LogExchangeRateReported(exchangeRate, volume24hrs, timestampSec);
    }

    /**
     * @return Most recently reported market information.
     *         isFresh: Is true if the last report is within the expiration window and
     *                  false if the report has expired.
     *         exchangeRate: The average exchange rate over the last reported 24 hours
     *                       of TOKEN:TARGET.
     *                       18 decimal fixed point number.
     *         volume24hrs:  The trade volume of last 24 hours reported in Token volume.
     *                       18 decimal fixed point number.
     */
    function getReport()
        public
        view
        returns (bool, uint256, uint256)
    {
        bool isFresh = (uint256(_timestampSec).add(_reportExpirationTimeSec) > now);
        return (
            isFresh,
            uint256(_exchangeRate),
            uint256(_volume24hrs)
        );
    }
}
```

market-oracle/contracts/MarketSourceFactory.sol

```
pragma solidity 0.4.24;
```

```
import "./MarketSource.sol";



/**
 * @title Market Source Factory
 */
contract MarketSourceFactory {
    event LogSourceCreated(address owner, MarketSource source);

    /**
     * @param name A human readable identifier for the source.
     * @param reportExpirationTimeSec The number of seconds after which the market data is deemed expired.
     * @return The address of the created MarketSource contract.
     */
    function createSource(string name, uint64 reportExpirationTimeSec)
        public
        returns (MarketSource)
    {
        MarketSource source = new MarketSource(name, reportExpirationTimeSec);
        source.transferOwnership(msg.sender);
        emit LogSourceCreated(msg.sender, source);
        return source;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account