Venus Correlated Oracles and Unlist Markets

AUDIT REPORT

Version 1.0.0

Serial No. 20240328000012024

Presented by Fairyproof

March 28, 2024

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Correlated oracles And Unlist markets project.

Audit Start Time:

March 21, 2024

Audit End Time:

March 28, 2024

Audited Source File's Address:

https://github.com/VenusProtocol/oracle/pull/165

https://github.com/VenusProtocol/venus-protocol/pull/429

https://github.com/VenusProtocol/isolated-pools/pull/349

https://github.com/VenusProtocol/venus-protocol/pull/438

The goal of this audit is to review Venus's solidity implementation for its Correlated oracles And Unlist markets function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Venus team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

Website:https://venus.io/

Source Code:

https://github.com/VenusProtocol/oracle/pull/165

https://github.com/VenusProtocol/venus-protocol/pull/429

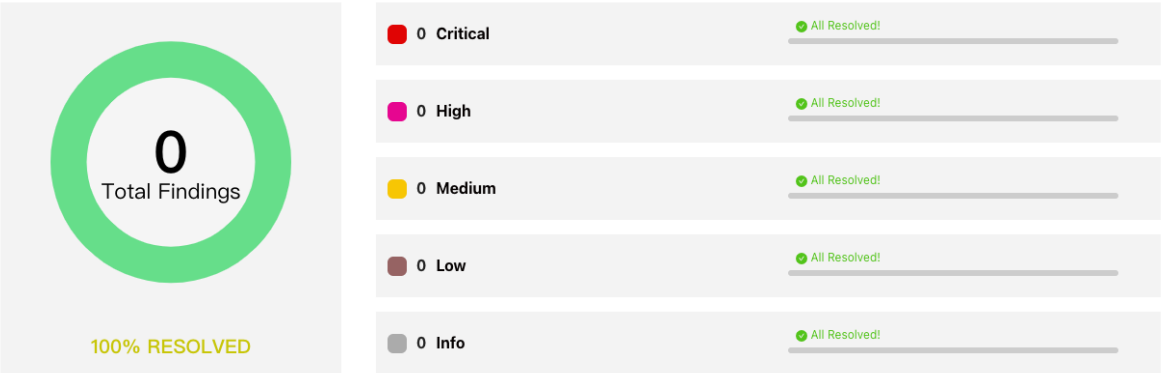
https://github.com/VenusProtocol/isolated-pools/pull/349

https://github.com/VenusProtocol/venus-protocol/pull/438

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Venus team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2024032800012024	Fairyproof Security Team	Mar 21, 2024 - Mar 28, 2024	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, no issues were uncovered.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Venus

Venus Protocol (“Venus”) is an algorithmic-based money market system designed to bring a complete decentralized finance-based lending and credit system onto Ethereum,Binance Smart Chain,opBNB and Arbitrum.

The above description is quoted from relevant documents of Venus.

04. Major functions of audited code

The audited code mainly implements the following functions:

1. Unlist markets
   Privilege functions to “tag” markets as unlisted. The isListed flag is checked in every function where the users interact
   with the markets. If isListed is false these operations are not allowed.
   For those cases where the markets are deprecated, and after following the usual process (announcements in the UI and
   the official channels, pause borrow and supplies, increase interest rates to incentivize borrowers to repay their debt, and
   finally if needed enable the forced liquidations), Venus team can have a mechanism to stop considering a market at all.
   The unlisted markets will be totally ignored by the UI (they won’t be shown).
2. Fix Borrow Cap 0 Logic
   In the Venus Core pool, a borrow cap equal 0 means unlimited.That is error-prone.Therefore, the Venus team modified the code to disable the `borrow` function when the borrow cap is set to 0.

3. Correlated oracles
   The changeset includes new custom oracles, with an interface compatible with the current oracles.These oracles are used to calculate the price of `CorrelatedToken`. Because it cannot be obtained directly from `Dex` or other oracles, it first obtains the price of the `UnderlyingToken` corresponding to the token, then obtains the exchange rate of `CorrelatedToken` and the `UnderlyingToken`, and finally multiplies the two to obtain price of `CorrelatedToken`.
   The price is scaled `10**(36 - CorrelatedToken decimal)`.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack

- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We didn't find issues or risks in these functions or areas at the time of writing.

08. issues by severity

- N/A

09. Issue descriptions

- N/A

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. OracleTestBase.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;
import {Test} from "forge-std/Test.sol";
import {OracleInterface} from "../src/interfaces/OracleInterface.sol";


contract MockOracle is OracleInterface {

    mapping(address => uint) public asset_prices;

    function setPrice(address asset, uint price) external {
        asset_prices[asset] = price;
    }

    function getPrice(address asset) external view returns (uint256) {
        return asset_prices[asset];
    }

}

enum Network {
    Mainnet,
    BSC
}

contract OracleTestBase is Test {
```

```
    uint bscFork;
    uint ethFork;
    MockOracle resilientOracle;

    function init(Network net) public {
        if(net == Network.BSC) {
            bscFork = vm.createSelectFork("https://rpc.ankr.com/bsc",37322271);
        } else if (net == Network.Mainnet) {
            ethFork = vm.createSelectFork("https://rpc.ankr.com/eth",19523720);
        } else {
            revert();
        }

        resilientOracle = new MockOracle();
    }
}
```

2. AnkrBNBOracle.t.sol

```
/**
AnkrBNBOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {AnkrBNBOracle} from "../src/oracles/AnkrBNBOracle.sol";
import {IAnkrBNB} from "../src/interfaces/IAnkrBNB.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract AnkrBNBOracleTest is OracleTestBase {
    AnkrBNBOracle public oracle;
    IAnkrBNB public ankr_bnb;

    address constant public NATIVE_TOKEN_ADDR =
0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB;
    address constant public ANKR_BNB =
0x52F24a5e03aee338Da5fd9Df68D2b6FAe1178827;

    function setUp() public  {
        super.init(Network.BSC);
        ankr_bnb = IAnkrBNB(ANKR_BNB);
        oracle = new AnkrBNBOracle(ANKR_BNB,address(resilientOracle));
    }

    function test_price() public {
        uint usd_decimal = 8;
        uint bnb_price = 572 * 10**usd_decimal + 123456;   // usd
        console.log("bnb_price_chainlink :%d",bnb_price);
        uint8 bnb_decimal = 18;
```

```
        uint scaled_price = bnb_price * 10**(18 - usd_decimal) * 10**(18 -
bnb_decimal);
        resilientOracle.setPrice(NATIVE_TOKEN_ADDR, scaled_price);

        uint bnb_amount = ankr_bnb.sharesToBonds(1 ether);
        // unused
        uint ankr_price = bnb_price * bnb_amount / 10**18;
        console.log("ankr_price_chainlink:%d",ankr_price);
        uint scaled_ankr_price =  bnb_price * bnb_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - ankr_bnb.decimals()) / 10**18;
        uint oracle_price = oracle.getPrice(ANKR_BNB);
        assertEq(scaled_ankr_price,oracle_price,"calculate error");
    }

}
```

3. BNBxOracle.t.sol

```
/**
BNBxOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {BNBxOracle} from "../src/oracles/BNBxOracle.sol";
import {IStaderStakeManager} from "../src/interfaces/IStaderStakeManager.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract BNBxOracleTest is OracleTestBase {
    BNBxOracle public oracle;
    IStaderStakeManager public stakeManager;

    address constant NATIVE_TOKEN_ADDR =
0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB;
    address constant bnbx_address = 0x1bdd3Cf7F79cfB8EdbB955f20ad99211551BA275;
    address constant stake_address = 0x7276241a669489E4BBB76f63d2A43Bfe63080F2F;

    function setUp() public  {
        super.init(Network.BSC);
        stakeManager = IStaderStakeManager(stake_address);
        oracle = new
BNBxOracle(stake_address,bnbx_address,address(resilientOracle));
    }

    function test_price() public {
        uint usd_decimal = 8;
```

```
        uint bnb_price = 572 * 10**usd_decimal + 123456;   // usd
        console.log("bnb_price_chainlink :%d",bnb_price);
        uint8 bnb_decimal = 18;
        uint scaled_price = bnb_price * 10**(18 - usd_decimal) * 10**(18 -
bnb_decimal);
        resilientOracle.setPrice(NATIVE_TOKEN_ADDR, scaled_price);

        uint8 bnbx_decimal = 18;
        uint bnb_amount = stakeManager.convertBnbXToBnb(1*10**bnbx_decimal);
        // unused
        uint bnbx_price = bnb_price * bnb_amount / 10**18;
        console.log("bnbx_price_chainlink:%d",bnbx_price);
        uint scaled_bnbx_price =  bnb_price * bnb_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - bnbx_decimal) / 10**18;
        uint oracle_price = oracle.getPrice(bnbx_address);
        assertEq(scaled_bnbx_price,oracle_price,"calculate error");
    }
}
```

4. SFraxOracle.t.sol

```
/**
SFraxOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {SFraxOracle} from "../src/oracles/SFraxOracle.sol";
import { ISFrax } from "../src/interfaces/ISFrax.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract SFraxOracleTest is OracleTestBase {
    SFraxOracle public oracle;
    ISFrax public sFrax;

    address constant public sFrax_address =
0xA663B02CF0a4b149d2aD41910CB81e23e1c41c32;
    address constant public Frax_address =
0x853d955aCEf822Db058eb8505911ED77F175b99e;

    function setUp() public  {
        super.init(Network.Mainnet);
        sFrax = ISFrax(sFrax_address);
        oracle = new
SFraxOracle(sFrax_address,Frax_address,address(resilientOracle));
    }
```

```
    function test_price() public {
        uint usd_decimal = 8;
        uint frax_price = 99784596;   // usd
        console.log("frax_price_chainlink :%d",frax_price);
        uint8 frax_decimal = 18;
        uint scaled_price = frax_price * 10**(18 - usd_decimal) * 10**(18 -
frax_decimal);
        resilientOracle.setPrice(Frax_address, scaled_price);

        uint8 sFrax_decimal = 18;
        uint frax_amount = sFrax.convertToAssets(10**sFrax_decimal);
        // unused
        uint sFrax_price = frax_price * frax_amount / 10**18;
        console.log("sFrax_price_chainlink:%d",sFrax_price);
        uint scaled_sFrax_price =  frax_price * frax_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - sFrax_decimal) / 10**18;
        uint oracle_price = oracle.getPrice(sFrax_address);
        assertEq(scaled_sFrax_price,oracle_price,"calculate error");
    }


}
```

5. SFrxETHOracle.t.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {SFraxOracle} from "../src/oracles/SFraxOracle.sol";
import { ISfrxETH } from "../src/interfaces/ISfrxETH.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";


contract SFrxETHOracleTest is OracleTestBase {
    SFraxOracle public oracle;
    ISfrxETH public sfrxETH;

    address constant public sfrxETH_address =
0xac3E018457B222d93114458476f3E3416Abbe38F;
    address constant public frxETH_address =
0x5E8422345238F34275888049021821E8E08CAa1f;

    function setUp() public  {
        super.init(Network.Mainnet);
        sfrxETH = ISfrxETH(sfrxETH_address);
        oracle = new
SFraxOracle(sfrxETH_address,frxETH_address,address(resilientOracle));
```

```
    }

    function test_price() public {
        uint usd_decimal = 8;
        uint8 frxETH_decimal = 18;
        uint frxETH_price = 3550 * 10**usd_decimal + 22000000;   // usd
        console.log("frxETH_price_chainlink :%d",frxETH_price);
        uint scaled_price = frxETH_price * 10**(18 - usd_decimal) * 10**(18 -
frxETH_decimal);
        resilientOracle.setPrice(frxETH_address, scaled_price);

        uint8 sFrxETH_decimal = 18;
        uint frxETH_amount = sfrxETH.convertToAssets(10**sFrxETH_decimal);
        // unused
        uint sFrxETH_price = frxETH_price * frxETH_amount / 10**18;
        console.log("sFrxETH_price_chainlink:%d",sFrxETH_price);
        uint scaled_sFrxETH_price =  frxETH_price * frxETH_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - sFrxETH_decimal) / 10**18;
        uint oracle_price = oracle.getPrice(sfrxETH_address);
        assertEq(scaled_sFrxETH_price,oracle_price,"calculate error");
    }
}
```

6. SlisBNBOracle.t.sol

```
/**
SlisBNBOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {SlisBNBOracle} from "../src/oracles/SlisBNBOracle.sol";
import { ISynclubStakeManager } from
"../src/interfaces/ISynclubStakeManager.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract SlisBNBOracleTest is OracleTestBase {
    SlisBNBOracle public oracle;
    ISynclubStakeManager public stakeManager;

    address constant NATIVE_TOKEN_ADDR =
0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB;
    address constant stake_mananger_address =
0x1adB950d8bB3dA4bE104211D5AB038628e477fE6;
    address constant slisBNB_address =
0xB0b84D294e0C75A6abe60171b70edEb2EFd14A1B;

    function setUp() public  {
        super.init(Network.BSC);
```

```
        stakeManager = ISynclubStakeManager(stake_mananger_address);
        oracle = new
SlisBNBOracle(stake_mananger_address,slisBNB_address,address(resilientOracle));
    }

    function test_price() public {
        // set the UNDERLYING_TOKEN scaled price
        uint usd_decimal = 8;
        uint bnb_price = 572 * 10**usd_decimal + 123456;   // usd
        console.log("bnb_price_chainlink :%d",bnb_price);
        uint8 bnb_decimal = 18;
        uint scaled_price = bnb_price * 10**(18 - usd_decimal) * 10**(18 -
bnb_decimal);
        resilientOracle.setPrice(NATIVE_TOKEN_ADDR, scaled_price);

        // get UNDERLYING_TOKEN token amount
        uint8 slisBNB_decimal = 18;
        uint bnb_amount = stakeManager.convertSnBnbToBnb(1*10**slisBNB_decimal);
        // unused
        uint slisBNB_price = bnb_price * bnb_amount / 10**bnb_decimal;
        console.log("slisBNB_price_chainlink:%d",slisBNB_price);
        // calculate the scaled  price of CORRELATED_TOKEN token
        uint scaled_slisBNB_price =  bnb_price * bnb_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - slisBNB_decimal) / 10**bnb_decimal;
        // get scaled price from oracle
        uint oracle_price = oracle.getPrice(slisBNB_address);
        // check eq
        assertEq(scaled_slisBNB_price,oracle_price,"calculate error");
    }

}
```

7. StkBNBOracle.t.sol

```
/**
StkBNBOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {StkBNBOracle} from "../src/oracles/StkBNBOracle.sol";
import { IPStakePool } from "../src/interfaces/IPStakePool.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract StkBNBOracleTest is OracleTestBase {
    StkBNBOracle public oracle;
```

```
    IPStakePool public stake_pool;

    address constant NATIVE_TOKEN_ADDR =
0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB;
    address constant stake_pool_address =
0xC228CefDF841dEfDbD5B3a18dFD414cC0dbfa0D8;
    address constant stkBNB_address = 0xc2E9d07F66A89c44062459A47a0D2Dc038E4fb16;

    function setUp() public  {
        super.init(Network.BSC);
        stake_pool = IPStakePool(stake_pool_address);
        oracle = new
StkBNBOracle(stake_pool_address,stkBNB_address,address(resilientOracle));
    }

    function _calcWeiWithdrawAmount(IPStakePool.Data memory self, uint256
poolTokens)
        internal
        pure
        returns (uint256)
    {
        uint256 numerator = poolTokens * self.totalWei;
        uint256 denominator = self.poolTokenSupply;

        if (numerator < denominator || denominator == 0) {
            return 0;
        }
        // TODO: later also take remainder into consideration
        return numerator / denominator;
    }

    function test_price() public {
        // set the UNDERLYING_TOKEN scaled price
        uint usd_decimal = 8;
        uint bnb_price = 572 * 10**usd_decimal + 1234567;   // usd
        console.log("bnb_price_chainlink :%d",bnb_price);
        uint8 bnb_decimal = 18;
        uint scaled_price = bnb_price * 10**(18 - usd_decimal) * 10**(18 -
bnb_decimal);
        resilientOracle.setPrice(NATIVE_TOKEN_ADDR, scaled_price);

        // get UNDERLYING_TOKEN token amount
        uint8 stkBNB_decimal = 18;
        uint bnb_amount =
_calcWeiWithdrawAmount(stake_pool.exchangeRate(),1*10**stkBNB_decimal);
        // unused
        uint stkBNB_price = bnb_price * bnb_amount / 10**bnb_decimal;
        console.log("stkBNB_price_chainlink:%d",stkBNB_price);
        // calculate the scaled  price of CORRELATED_TOKEN token
        uint scaled_stkBNB_price =  bnb_price * bnb_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - stkBNB_decimal) / 10**bnb_decimal;
        // get scaled price from oracle
        uint oracle_price = oracle.getPrice(stkBNB_address);
        // check eq
        assertEq(scaled_stkBNB_price,oracle_price,"calculate error");
```

```
    }

}
```

8. WBETHOracle.t.sol

```
/**
WBETHOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import {WBETHOracle} from "../src/oracles/WBETHOracle.sol";
import { IWBETH } from "../src/interfaces/IWBETH.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

contract WBETHOracleTest is OracleTestBase {
    WBETHOracle public oracle;
    IWBETH public wBETH;

    address constant eth_address = 0x2170Ed0880ac9A755fd29B2688956BD959F933F8;
    address constant wBETH_address = 0xa2E3356610840701BDf5611a53974510Ae27E2e1;
    uint256 public constant _EXCHANGE_RATE_UNIT = 1e18;

    function setUp() public  {
        super.init(Network.BSC);
        wBETH = IWBETH(wBETH_address);
        oracle = new
WBETHOracle(wBETH_address,eth_address,address(resilientOracle));
    }

    function test_price() public {
        uint usd_decimal = 8;
        uint8 eth_decimal = 18;
        uint eth_price = 3550 * 10**usd_decimal + 21020000;   // usd
        console.log("eth_price_chainlink  :%d",eth_price);
        uint scaled_eth_price = eth_price * 10**(18 - usd_decimal) * 10**(18 -
eth_decimal);
        resilientOracle.setPrice(eth_address, scaled_eth_price);

        uint8 wBETH_decimal = 18;
        uint eth_amount = wBETH.exchangeRate()* 10**wBETH_decimal /
_EXCHANGE_RATE_UNIT;
        // unused
        uint wBETH_price = eth_price * eth_amount / 10**eth_decimal;
        console.log("wBETH_price_chainlink:%d",wBETH_price);
        uint scaled_wBETH_price =  eth_price * eth_amount  * 10**(18 -
usd_decimal)
```

```
            * 10**(18 - wBETH_decimal) / 10**eth_decimal;
        uint oracle_price = oracle.getPrice(wBETH_address);
        assertEq(scaled_wBETH_price,oracle_price,"calculate error");
    }

}
```

9. WeETHOracle.t.sol

```
/**
WeETHOracle Forking test
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {console} from "forge-std/Test.sol";
import { WeETHOracle } from "../src/oracles/WeETHOracle.sol";
import { IEtherFiLiquidityPool } from
"../src/interfaces/IEtherFiLiquidityPool.sol";
import { Network,OracleTestBase } from "./OracleTestBase.sol";

interface IEETH {
    function liquidityPool() external view returns(address);
}

contract WeETHOracleTest is OracleTestBase {
    WeETHOracle public oracle;
    IEtherFiLiquidityPool public liquid_pool;

    address constant eETH_address = 0x35fA164735182de50811E8e2E824cFb9B6118ac2;
    address constant weETH_address = 0xCd5fE23C85820F7B72D0926FC9b05b43E359b7ee;


    function setUp() public  {
        super.init(Network.Mainnet);
        address pool = IEETH(weETH_address).liquidityPool();
        liquid_pool = IEtherFiLiquidityPool(pool);
        oracle = new
WeETHOracle(pool,weETH_address,eETH_address,address(resilientOracle));
    }

    function test_price() public {
        uint usd_decimal = 8;
        uint8 eth_decimal = 18;
        uint eth_price = 3550 * 10**usd_decimal + 31020000;   // usd
        console.log("eth_price_chainlink  :%d",eth_price);
        uint scaled_eth_price = eth_price * 10**(18 - usd_decimal) * 10**(18 -
eth_decimal);
        resilientOracle.setPrice(eETH_address, scaled_eth_price);
```

```
        uint8 WeETH_decimal = 18;
        uint eth_amount = liquid_pool.amountForShare(10**WeETH_decimal);
        // unused
        uint WeETH_price = eth_price * eth_amount / 10**eth_decimal;
        console.log("WeETH_price_chainlink:%d",WeETH_price);
        uint scaled_WeETH_price =  eth_price * eth_amount  * 10**(18 -
usd_decimal)
            * 10**(18 - WeETH_decimal) / 10**eth_decimal;
        uint oracle_price = oracle.getPrice(weETH_address);
        assertEq(scaled_WeETH_price,oracle_price,"calculate error");
    }
}
```

10. UnitTestOutput

```
Ran 1 test for test/SFrxETHOracle.t.sol:SFrxETHOracleTest
[PASS] test_price() (gas: 59355)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (2.02ms CPU
time)

Ran 1 test for test/SFraxOracle.t.sol:SFraxOracleTest
[PASS] test_price() (gas: 66136)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (2.30ms CPU
time)

Ran 1 test for test/WeETHOracle.t.sol:WeETHOracleTest
[PASS] test_price() (gas: 81179)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (2.11ms CPU
time)

Ran 1 test for test/StkBNBOracle.t.sol:StkBNBOracleTest
[PASS] test_price() (gas: 67758)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (1.06ms CPU
time)

Ran 1 test for test/WBETHOracle.t.sol:WBETHOracleTest
[PASS] test_price() (gas: 65031)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (1.01ms CPU
time)

Ran 1 test for test/BNBxOracle.t.sol:BNBxOracleTest
[PASS] test_price() (gas: 84587)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (1.41ms CPU
time)

Ran 1 test for test/AnkrBNBOracle.t.sol:AnkrBNBOracleTest
[PASS] test_price() (gas: 79571)
```

```
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (1.49ms CPU
time)

Ran 1 test for test/SlisBNBOracle.t.sol:SlisBNBOracleTest
[PASS] test_price() (gas: 82879)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.14s (1.48ms CPU
time)

Ran 8 test suites in 1.15s (9.14s CPU time): 8 tests passed, 0 failed, 0 skipped
(8 total tests)
```

11.2 External Functions Check Points

- N/A

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech