Capped Oracle And Cached Price

AUDIT REPORT

Version 1.0.0

Serial No. 20250319000012025

Presented by Fairyproof

March 19, 2025

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus CappedOracle And CachedPrice project.

Audit Start Time:

March 10, 2025

Audit End Time:

March 15, 2025

Audited Code's Github Repository:

https://github.com/VenusProtocol/oracle/pull/239

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/ResilientOracle.sol
contracts/lib/Transient.sol
contracts/oracles/common/CorrelatedTokenOracle.sol
contracts/interfaces/ICappedOracle.sol

Specific correlated oracles:
contracts/oracles
├── AnkrBNBOracle.sol
├── BNBxOracle.sol
├── ERC4626Oracle.sol
├── EtherfiAccountantOracle.sol
├── OneJumpOracle.sol
├── PendleOracle.sol
├── SFraxOracle.sol
├── SlisBNBOracle.sol
├── StkBNBOracle.sol
├── WBETHOracle.sol
├── WeETHAccountantOracle.sol
├── WeETHOracle.sol
├── WstETHOracleV2.sol
├── ZkETHOracle.sol
```

Note: For the specific correlated oracles, the implementation of the `getUnderlyingAmount` function has been audited incrementally. Although only the function name and visibility were changed, each method of fetching the exchange rate has been re-verified (some oracle implementations had been audited previously).

The goal of this audit is to review Venus's solidity implementation for its CappedOracle And CachedPrice 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/

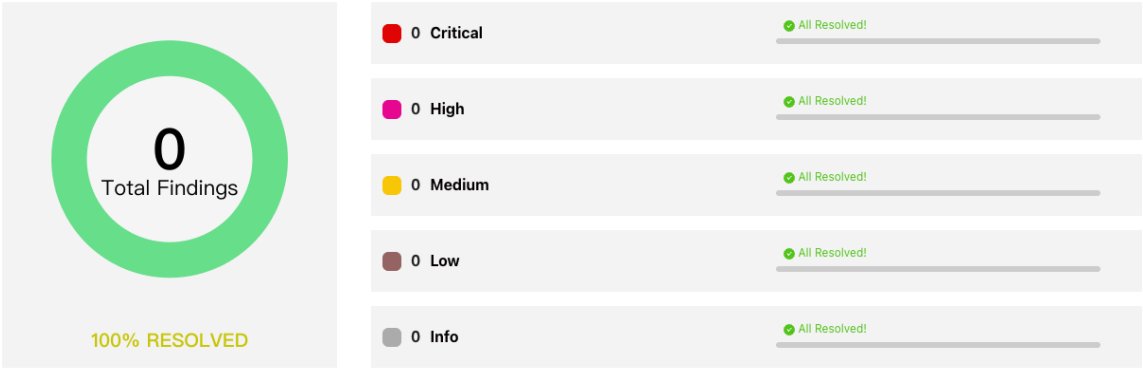
Whitepaper:https://github.com/VenusProtocol/venus-protocol-documentation/tree/main/whitepapers

Source Code: https://github.com/VenusProtocol/oracle/pull/239

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
2025031900012025	Fairyproof Security Team	Mar 10, 2025 - Mar 15, 2025	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 current audit includes two parts: CappedOracle and CachedPrice, which are interconnected.

Main Audit Points:

1. A caching mechanism was added to the `ResilientOracle` using opcodes like `tstore/tload`. This caches the current price so that multiple queries within a single transaction save on gas.
2. To address issues where the exchange rate for certain ERC4626-like tokens might fluctuate excessively, a maximum exchange rate cap was introduced in the `CorrelatedTokenOracle` contract. This cap is calculated based on the protocol-specific yield and compounding interval. When the queried exchange rate exceeds this cap, the capped exchange rate is used instead. In order to calculate compound interest, a snapshot mechanism was added to record the current maximum computed exchange rate.
3. The CorrelatedTokenOracle also uses the `tstore/tload` opcodes to cache the maximum exchange rate. However, since this cache only updates when a snapshot is updated, it is rarely used. In most cases, simply caching the price in the `ResilientOracle` is sufficient—only a single exchange rate query is needed without relying on caching.

Price Caching Workflow:
For example, when obtaining the price of `AnkrBNB` multiple times within a single transaction:

1. The user calls the `updateAssetPrice(ankrBNB)` function in the `ResilientOracle` contract to fetch the current price of ankrBNB.
2. This triggers the `updateSnapshot` function in the `AnkrBNBOracle` to update the snapshot.
3. The `_getPrice` function is then called to retrieve the ankrBNB price from the three sources: PivotOracle, MainOracle, and FallbackOracle.
4. The `MainOracle` is the `AnkrBNBOracle` itself, so its `getPrice` function is invoked.
5. Within `getPrice`, the final exchange rate is first calculated, and then the ResilientOracle's `getPrice` function is called again to fetch the BNB price.
6. Inside the `ResilientOracle`, the price fetching process for BNB is analogous to that for ankrBNB, resulting in the BNB price.
7. Based on the BNB price and the exchange rate, the final price for ankrBNB is computed and returned to the `_getPrice` function.
8. The ankrBNB price is then cached, and the `updateAssetPrice(ankrBNB)` call completes.
9. Subsequent calls to `ResilientOracle.getPrice(ankrBNB)` return the cached price, saving gas.

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.

- N/A

11. Appendices

11.1 Unit Test

1. MockAnkrBNB.sol

```
// SPDX-License-Identifier: BSD-3-Clause
pragma solidity 0.8.25;

import { IAnkrBNB } from "../interfaces/IAnkrBNB.sol";
import {ProxyAdmin} from "@openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol";
import {TransparentUpgradeableProxy} from "@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";


contract MockAnkrBNB is IAnkrBNB {
    uint8 public constant decimals = 18;

    uint256 public exchange_rate;

    constructor(uint initial_rate) {
        exchange_rate = initial_rate;
    }

    function setExchangeRate(uint256 rate) external {
        exchange_rate = rate;
    }

    function sharesToBonds(uint256 amount) public view override returns (uint256) {
        return amount * exchange_rate / (10**decimals);
    }
}
```

2. MockOracle.sol

```
// SPDX-License-Identifier: BSD-3-Clause
pragma solidity 0.8.25;

import {OracleInterface,ResilientOracleInterface} from "../interfaces/OracleInterface.sol";

interface ICounter {
    function addCounter() external;
}


contract MockMainBNBOracle is OracleInterface {
    uint public counter = 0;
    uint public price0 = 300 ether;
    uint public price1 = 302 ether;

    function addCounter() external {
        counter++;
    }

    function getPrice(address asset) external view returns (uint256) {
        asset;
        if(counter % 2 == 0) {
            return price0;
        } else {
            return price1;
        }
    }
}

contract MockPivotBNBOracle is OracleInterface{
    uint public counter = 0;
    uint public price0 = 301 ether;
    uint public price1 = 303 ether;

    function addCounter() external {
        counter++;
    }
    function getPrice(address asset) external view returns (uint256) {
```

```
        asset;
        if(counter % 2 == 0) {
            return price0;
        } else {
            return price1;
        }
    }
}


contract MockFallbackBNBOracle is OracleInterface {
     uint public counter = 0;
    uint public price0 = 209 ether;
    uint public price1 = 208 ether;

    function addCounter() external {
        counter++;
    }
    function getPrice(address asset) external view returns (uint256) {
        asset;
        if(counter % 2 == 0) {
            return price0;
        } else {
            return price1;
        }
    }
}

contract MockComptroller {
    ResilientOracleInterface public resilientOracle;
    ICounter public mainOracle;
    ICounter public pivotOracle;
    ICounter public fallbackOracle;

    constructor(
        ResilientOracleInterface _ResilientOracle,
        ICounter _mainOracle,
        ICounter _pivotOracle,
        ICounter _fallbackOracle
    ) {
        resilientOracle = _ResilientOracle;
        mainOracle = _mainOracle;
        pivotOracle = _pivotOracle;
        fallbackOracle = _fallbackOracle;
    }

    function getPrices(address asset,uint times) external returns(uint256[] memory result) {
        result = new uint256[](times);
        for(uint i=0;i<times;i++) {
            result[i] = resilientOracle.getPrice(asset);
            mainOracle.addCounter();
            pivotOracle.addCounter();
            fallbackOracle.addCounter();
        }
    }
    function getPricesWithCache(address asset,uint times) external returns(uint256[] memory result) {
        result = new uint256[](times);
        resilientOracle.updateAssetPrice(asset);
        for(uint i=0;i<times;i++) {
            result[i] = resilientOracle.getPrice(asset);
            mainOracle.addCounter();
            pivotOracle.addCounter();
            fallbackOracle.addCounter();
        }
    }
}
```


3. OracleTest.js

```
const {
    time,
    loadFixture,
  } = require("@nomicfoundation/hardhat-toolbox/network-helpers");
const { anyValue } = require("@nomicfoundation/hardhat-chai-matchers/withArgs");
const { expect } = require("chai");
const { ethers } = require("hardhat");
const { assert } = require("ethers");
```

```
describe("OracleUnitTest", function () {
    const BNB_USD_PRICE = ethers.parseEther("300");
    const BNB_AMOUNT_FOR_ONE_ANKRBNB = ethers.parseEther("1.075370795716558975");
    const ANNUAL_GROWTH_RATE = ethers.parseEther("0.05");
    const GROWTH_RATE_PER_SECOND = ANNUAL_GROWTH_RATE / BigInt(365 * 24 * 3600);
    const ANKRBNB_USD_PRICE_DENOMINATOR = ethers.parseEther("1");
    const SNAPSHOT_UPDATE_INTERVAL = 10;
    const ANKRBNB_USD_PRICE = BNB_USD_PRICE * BNB_AMOUNT_FOR_ONE_ANKRBNB / ANKRBNB_USD_PRICE_DENOMINATOR;
    const ZERO_ADDRESS = ethers.ZeroAddress;
    const BNB_TOKEN_ADDRESS = "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB";

    async function deployFixture() {
        const [owner,...users] = await ethers.getSigners();
        // 1 deploy acm
        const AccessControlManager = await ethers.getContractFactory("AccessControlManager");
        const acm = await AccessControlManager.deploy();
        // 2 deploy bound validator
        const BoundValidator = await ethers.getContractFactory("BoundValidator");
        const boundValidator = await upgrades.deployProxy(BoundValidator, [acm.target], {
          initializer: "initialize",
        });
        // 3 deploy ResilientOracle
        // can't use deployProxy here, because the constructor of ResilientOracle has 3 parameters
        const ResilientOracle = await ethers.getContractFactory("ResilientOracle");
        const resilient_impl = await ResilientOracle.deploy(ZERO_ADDRESS,ZERO_ADDRESS,boundValidator.target);
        let initData = resilient_impl.interface.encodeFunctionData("initialize", [acm.target]);
        const ProxyAdmin = await ethers.getContractFactory("ProxyAdmin");
        const proxyAdmin = await ProxyAdmin.deploy();
        const TransparentUpgradeableProxy = await ethers.getContractFactory("TransparentUpgradeableProxy");
        let resilientOracle = await TransparentUpgradeableProxy.deploy(resilient_impl.target, proxyAdmin.target,
initData);
        resilientOracle = ResilientOracle.attach(resilientOracle.target);

        // 4 deploy Mock BNBOracle
        const MockMainBNBOracle = await ethers.getContractFactory("MockMainBNBOracle");
        const mockMainBNBOracle = await MockMainBNBOracle.deploy();
        const MockPivotBNBOracle = await ethers.getContractFactory("MockPivotBNBOracle");
        const mockPivotBNBOracle = await MockPivotBNBOracle.deploy();
        const MockFallbackBNBOracle = await ethers.getContractFactory("MockFallbackBNBOracle");
        const mockFallbackBNBOracle = await MockFallbackBNBOracle.deploy();

        // set oracle and permission
        await acm.giveCallPermission(
            resilientOracle.target,
            "setTokenConfig(TokenConfig)",
            owner.address
        );
        await acm.giveCallPermission(
            boundValidator.target,
            "setValidateConfig(ValidateConfig)",
            owner.address
        );
        await resilientOracle.setTokenConfig({
            asset:BNB_TOKEN_ADDRESS,
            oracles:[
                mockMainBNBOracle.target,
                mockPivotBNBOracle.target,
                mockFallbackBNBOracle.target
            ],
            enableFlagsForOracles:[true,true,true]
        });
        await boundValidator.setValidateConfig({
            asset:BNB_TOKEN_ADDRESS,
            upperBoundRatio:ethers.parseEther("1.05"), // 5%
            lowerBoundRatio:ethers.parseEther("0.95"), // 5%
        })
        // 5 deploy AnkrBNB Oracle
        const AnkrBNB = await ethers.getContractFactory("MockAnkrBNB");
        const ankrBNB = await AnkrBNB.deploy(BNB_AMOUNT_FOR_ONE_ANKRBNB);
        const AnkrBNBOracle = await ethers.getContractFactory("AnkrBNBOracle");
        const block = await ethers.provider.getBlock("latest");
        const ankrBNBOracle = await AnkrBNBOracle.deploy(
            ankrBNB.target,
            resilientOracle.target,
            ANNUAL_GROWTH_RATE,
            SNAPSHOT_UPDATE_INTERVAL,
            BNB_AMOUNT_FOR_ONE_ANKRBNB,
            block.timestamp
        );
        // set ankrBNB oracle
```

```
        await resilientOracle.setTokenConfig({
            asset:ankrBNB.target,
            oracles:[
                ankrBNBOracle.target,
                ZERO_ADDRESS,
                ZERO_ADDRESS
            ],
            enableFlagsForOracles:[true,false,false]
        });

        // 6 deploy MockComptroller
        const MockComptroller = await ethers.getContractFactory("MockComptroller");
        const mockComptroller = await MockComptroller.deploy(
            resilientOracle.target,
            mockMainBNBOracle.target,
            mockPivotBNBOracle.target,
            mockFallbackBNBOracle.target
        );

        return {
            owner,
            users,
            acm,
            boundValidator,
            resilientOracle,
            ankrBNBOracle,
            mockComptroller,
            ankrBNB
        }
    }
    describe("ResilientOracle getPrice unit test", function () {
        it("query bnb price without cache", async function () {
            const { mockComptroller } = await deployFixture();
            let prices = await mockComptroller.getPrices.staticCall(BNB_TOKEN_ADDRESS,2);
            let price0 = prices[0];
            let price1 = prices[1];
            expect(price0).to.equal(BNB_USD_PRICE);
            expect(price0).to.not.equal(price1);
        });

        it("query bnb price with cache", async function () {
            const { mockComptroller } = await deployFixture();
            let prices = await mockComptroller.getPricesWithCache.staticCall(BNB_TOKEN_ADDRESS,5);
            let price0 = prices[0];
            expect(price0).to.equal(BNB_USD_PRICE);
            for(let i=1;i<5;i++){
                expect(prices[i]).to.equal(price0);
            }
        });

        it("query ankrbnb price without cache", async function () {
            const { mockComptroller,ankrBNB } = await deployFixture();
            let prices = await mockComptroller.getPrices.staticCall(ankrBNB.target,2);
            let price0 = prices[0];
            let price1 = prices[1];
            expect(price0).to.equal(ANKRBNB_USD_PRICE);
            expect(price0).to.not.equal(price1);
        });

        it("query ankrbnb price with cache", async function () {
            const { mockComptroller,ankrBNB } = await deployFixture();
            let prices = await mockComptroller.getPricesWithCache.staticCall(ankrBNB.target,5);
            let price0 = prices[0];
            expect(price0).to.equal(ANKRBNB_USD_PRICE);
            for(let i=1;i<5;i++){
                expect(prices[i]).to.equal(price0);
            }
        });

        it("query ankrbnb price with normal exchange rate", async function () {
            const { mockComptroller,ankrBNB,ankrBNBOracle } = await deployFixture();
            // check old price
            let prices = await mockComptroller.getPricesWithCache.staticCall(ankrBNB.target,1);
            let old_price = prices[0];
            expect(old_price).to.equal(ANKRBNB_USD_PRICE);
            // set new exchange rate beyond max allowed rate
            await ankrBNB.setExchangeRate(ethers.parseEther("1.1"));
            // Advance block time by SNAPSHOT_UPDATE_INTERVAL seconds
            const block = await ethers.provider.getBlock("latest");
            let snapshotTimestamp = await ankrBNBOracle.snapshotTimestamp();
            let differ = block.timestamp - Number(snapshotTimestamp);
```

```
        // Advance block time by SNAPSHOT_UPDATE_INTERVAL seconds
        await time.increase(SNAPSHOT_UPDATE_INTERVAL - differ);

        // Verify the new timestamp
        const newBlock = await ethers.provider.getBlock("latest");
        expect(newBlock.timestamp).to.equal(Number(snapshotTimestamp) + SNAPSHOT_UPDATE_INTERVAL);
        // Verify the max_allowed_rate
        let max_allowed_rate = await ankrBNBOracle.getMaxAllowedExchangeRate();
        expect(max_allowed_rate).to.equal(
            BNB_AMOUNT_FOR_ONE_ANKRBNB +
            BNB_AMOUNT_FOR_ONE_ANKRBNB * GROWTH_RATE_PER_SECOND
            * BigInt(SNAPSHOT_UPDATE_INTERVAL)
            / ANKRBNB_USD_PRICE_DENOMINATOR
        ,"max_allowed_rate error");
        // Verify the underlyingAmount
        let underlyingAmount = await ankrBNBOracle.getUnderlyingAmount();
        expect(underlyingAmount).to.equal(ethers.parseEther("1.1"),"underlyingAmount error");
        // Verify the max_allowed_rate < underlyingAmount
        expect(max_allowed_rate).to.lt(underlyingAmount,"max_allowed_rate < underlyingAmount error");

        // Get price after time advancement
        prices = await mockComptroller.getPricesWithCache.staticCall(ankrBNB.target, 1);
        let new_price = prices[0];
        // Verify the new price
        expect(new_price).to.gt(old_price);
        expect(new_price).to.equal(BNB_USD_PRICE * max_allowed_rate / ANKRBNB_USD_PRICE_DENOMINATOR);
    });

    it("query ankrbnb price with abnormal exchange rate", async function () {
        const { mockComptroller,ankrBNB,ankrBNBOracle } = await deployFixture();
        // check old price
        let prices = await mockComptroller.getPricesWithCache.staticCall(ankrBNB.target,1);
        let old_price = prices[0];
        expect(old_price).to.equal(ANKRBNB_USD_PRICE);
        // set new exchange rate beyond max allowed rate
        await ankrBNB.setExchangeRate(ethers.parseEther("1.075370802766451091"));
        // Advance block time by SNAPSHOT_UPDATE_INTERVAL seconds
        const block = await ethers.provider.getBlock("latest");
        let snapshotTimestamp = await ankrBNBOracle.snapshotTimestamp();
        let differ = block.timestamp - Number(snapshotTimestamp);
        // Advance block time by SNAPSHOT_UPDATE_INTERVAL seconds
        await time.increase(SNAPSHOT_UPDATE_INTERVAL - differ);

        // Verify the new timestamp
        const newBlock = await ethers.provider.getBlock("latest");
        expect(newBlock.timestamp).to.equal(Number(snapshotTimestamp) + SNAPSHOT_UPDATE_INTERVAL);
        // Verify the max_allowed_rate
        let max_allowed_rate = await ankrBNBOracle.getMaxAllowedExchangeRate();
        expect(max_allowed_rate).to.equal(
            ethers.parseEther("1.075370812766451091")  //  max_allowed_rate not qual snapshotExchangeRate
        ,"max_allowed_rate error");
        // Verify the underlyingAmount
        let underlyingAmount = await ankrBNBOracle.getUnderlyingAmount();
        expect(underlyingAmount).to.equal(ethers.parseEther("1.075370802766451091"),"underlyingAmount error");
        // Verify the max_allowed_rate >=  underlyingAmount
        expect(max_allowed_rate).to.gt(underlyingAmount,"max_allowed_rate < underlyingAmount error");
        // Get price after time advancement
        prices = await mockComptroller.getPricesWithCache.staticCall(ankrBNB.target, 1);
        let new_price = prices[0];
        // Verify the new price
        expect(new_price).to.gt(old_price);
        expect(new_price).to.equal(BNB_USD_PRICE * underlyingAmount / ANKRBNB_USD_PRICE_DENOMINATOR);
    });
});

describe("updateSnapshot and getMaxAllowedExchangeRate unit test", function () {
    it("updateSnapshot and getMaxAllowedExchangeRate", async function () {
        const { mockComptroller,ankrBNB,ankrBNBOracle } = await deployFixture();
        let snapshotTimestamp = await ankrBNBOracle.snapshotTimestamp();
        let snapshotExchangeRate = await ankrBNBOracle.snapshotExchangeRate();
        let block = await ethers.provider.getBlock("latest");
        let differ = block.timestamp - Number(snapshotTimestamp);
        assert(differ < SNAPSHOT_UPDATE_INTERVAL,"differ < SNAPSHOT_UPDATE_INTERVAL error");
        await ankrBNBOracle.updateSnapshot();
        // check not updated
        expect(await ankrBNBOracle.snapshotTimestamp()).to.equal(snapshotTimestamp);
        expect(await ankrBNBOracle.snapshotExchangeRate()).to.equal(snapshotExchangeRate);
        // set exchange rate
        await ankrBNB.setExchangeRate(ethers.parseEther("1.1"));

        // Advance block time by SNAPSHOT_UPDATE_INTERVAL seconds
```

```
            await time.increase(SNAPSHOT_UPDATE_INTERVAL - differ);
            // Verify the new timestamp
            const newBlock = await ethers.provider.getBlock("latest");
            expect(newBlock.timestamp).to.gte(Number(snapshotTimestamp) + SNAPSHOT_UPDATE_INTERVAL);
            // updateSnapshot
            await ankrBNBOracle.updateSnapshot();
            // check updated
            expect(await ankrBNBOracle.snapshotTimestamp()).to.equal(newBlock.timestamp + 1);
            diff = newBlock.timestamp + 1 - Number(snapshotTimestamp);
            // check new snapshotExchange rate
            snapshotExchangeRate = await ankrBNBOracle.snapshotExchangeRate()
            expect(snapshotExchangeRate).to.equal(
                BNB_AMOUNT_FOR_ONE_ANKRBNB +
                BNB_AMOUNT_FOR_ONE_ANKRBNB * GROWTH_RATE_PER_SECOND
                * BigInt(diff)
                / ANKRBNB_USD_PRICE_DENOMINATOR
            );
            await time.increase(SNAPSHOT_UPDATE_INTERVAL);
            // updateSnapshot
            await ankrBNBOracle.updateSnapshot();
            // check updated
            let new_snapshotExchangeRate = await ankrBNBOracle.snapshotExchangeRate();
            expect(new_snapshotExchangeRate).to.equal(
                snapshotExchangeRate +
                snapshotExchangeRate * GROWTH_RATE_PER_SECOND
                * BigInt(11)
                / ANKRBNB_USD_PRICE_DENOMINATOR
            );
            expect(await ankrBNBOracle.snapshotTimestamp()).to.equal((await
ethers.provider.getBlock("latest")).timestamp);
        });
    });
});
```

11.2 External Functions Check Points

1. CorrelatedTokenOracle.sol.md

File: contracts/oracles/common/CorrelatedTokenOracle.sol

contract: CorrelatedTokenOracle is OracleInterface, ICappedOracle

(Empty fields in the table represent things that are not required or relevant)

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	isCapped()	view					
2	updateSnapshot()				Yes	Passed	
3	getPrice(address)	view					
4	getMaxAllowedExchangeRate()	view					
5	getUnderlyingAmount()	view					

2. ResilientOracle.sol.md

File: contracts/ResilientOracle.sol

contract: ResilientOracle is PausableUpgradeable, AccessControlledV8, ResilientOracleInterface

(Empty fields in the table represent things that are not required or relevant)

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	initialize(address)		`initializer`				OnlyOnce
2	pause()						_checkAccessAllowed("pause()")
3	unpause()						_checkAccessAllowed("unpause()")
4	setTokenConfigs(TokenConfig[])					Passed	_checkAccessAllowed("setTokenConfig(T
5	setOracle(address,address,OracleRole)		`notNullAddress(asset), checkTokenConfigExistence(asset)`				_checkAccessAllowed("setOracle(addres
6	enableOracle(address,OracleRole,bool)		`notNullAddress(asset), checkTokenConfigExistence(asset)`				_checkAccessAllowed("enableOracle(adc
7	updatePrice(address)						
8	updateAssetPrice(address)					Passed	
9	getTokenConfig(address)	view					
10	getUnderlyingPrice(address)	view			Yes	Passed	NOtPaused
11	getPrice(address)	view			Yes	Passed	NotPaused
12	setTokenConfig(TokenConfig)		`notNullAddress(tokenConfig.asset), notNullAddress(tokenConfig.oracles[uint256(OracleRole.MAIN)])`			Passed	_checkAccessAllowed("setTokenConfig(T
13	getOracle(address,OracleRole)	view					

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