Risk Oracle Integration

AUDIT REPORT

Version 1.0.0

Serial No. 20250226000012022

Presented by Fairyproof

February 26, 2025

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Risk Oracle Integration and CorePool Comptroller Interface Changing project.

Audit Start Time:

February 18, 2025

Audit End Time:

February 24, 2025

Audited Source File's Address:

https://github.com/VenusProtocol/governance-contracts/pull/115

https://github.com/VenusProtocol/venus-protocol/pull/548

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/RiskSteward
├── IRiskSteward.sol
├── IRiskStewardReceiver.sol
├── MarketCapsRiskSteward.sol
└── RiskStewardReceiver.sol

Diamond/facets
├── MarketFacet.sol
├── PolicyFacet.sol
├── SetterFacet.sol

1 directory, 3 files
```


The goal of this audit is to review Venus's solidity implementation for its RiskOracle Integration and CorePool Comptroller Interface Changing 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:

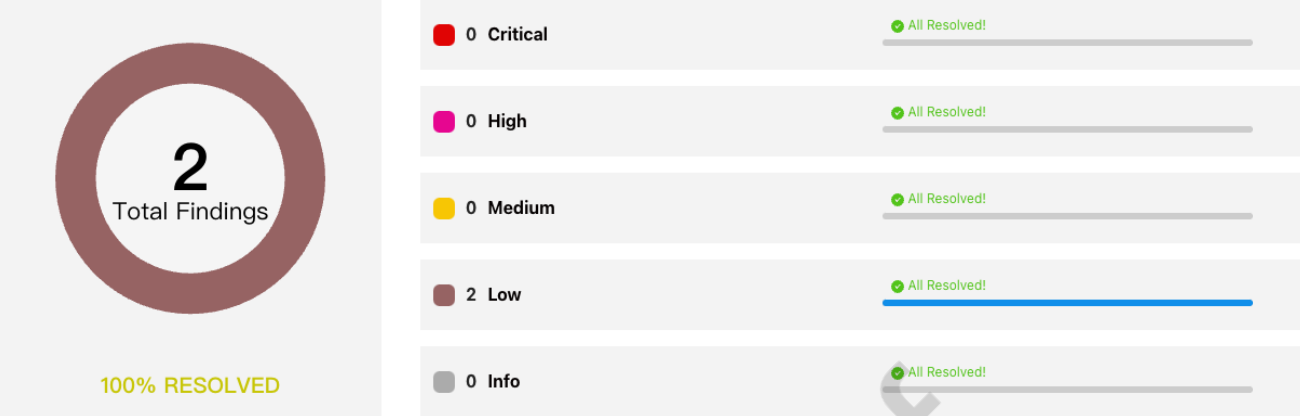
https://github.com/VenusProtocol/governance-contracts/pull/115

https://github.com/VenusProtocol/venus-protocol/pull/548

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
2025022600012022	Fairyproof Security Team	Feb 18, 2025 - Feb 24, 2025	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, two issues of low-severity were uncovered. The Venus team fixed two issues of low.

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: RiskOracle integration and CorePool Comptroller interface changing, which are interconnected.

RiskOracle Functionality Overview

RiskOracle, launched by Chaos Labs, provides recommended parameter settings based on on-chain transactions and actual conditions. Currently, Venus only uses its `MarketSupplyCap` and `MarketBorrowCap` recommendations. After recommendations are published, anyone can call the `processUpdateById` or `processUpdateByParameterAndMarket` functions in the `RiskStewardReceiver` contract to update these settings. The `RiskStewardReceiver` contract verifies whether updates are expired, checks update frequency, and confirms if settings have been previously updated. The `MarketCapsRiskSteward` contract performs simple validation on proposed updates, ensuring new values don't exceed a change magnitude of `maxDeltaBps` basis points compared to original values.

Note that the `UPDATE_EXPIRATION_TIME` constant in `RiskStewardReceiver.sol` serves two purposes:

1. Expiration time for recommended settings - updates older than `UPDATE_EXPIRATION_TIME` cannot be processed
2. Market setting update frequency - time between two updates must exceed `UPDATE_EXPIRATION_TIME`

Using the same constant for these different concepts may reduce flexibility, such as preventing custom update frequency settings. However, it has the additional benefit of preventing old recommendations from being updated. For example, after updating ID 5, ID 4 recommendations cannot be updated because the time between updates must exceed UPDATE_EXPIRATION_TIME, and by the time the next update is possible, ID 4's recommendations would have expired.

CorePool Comptroller Interface Changes Overview

This functionality primarily aims to standardize interface names between CorePool Comptroller and IsolatedPool Comptroller for functions with similar functionality. Historically, some CorePool Comptroller's interfaces had an additional `_` prefix compared to their IsolatedPool Comptroller counterparts, increasing complexity when handling both Comptroller types. The new functionality adds prefix-free identical interfaces in CorePool Comptroller, maintaining backward compatibility while achieving interface consistency.

Added interfaces without the `_` prefix include:

- supportMarket
- setCloseFactor
- setCollateralFactor
- setLiquidationIncentive
- setMarketBorrowCaps
- setMarketSupplyCaps
- setActionsPaused
- setPriceOracle
- setPrimeToken
- setForcedLiquidation

In implementation, functions like `setPriceOracle` (new) and `_setPriceOracle` (existing) differ only in name, sharing identical parameter lists, return values, and function bodies. The only exception is `setCollateralFactor`, which has an additional parameter compared to `_setCollateralFactor` but maintains the same core functionality.

Additionally, CorePool Comptroller added these interfaces to match IsolatedPool Comptroller:

- isMarketListed
- getBorrowingPower

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 found one issue, for more details please refer to [FP-2] in "09. Issue description".

- 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 found one issue, for more details please refer to [FP-1] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Length of `__gap` is incorrect	Code Improvement	Low	✓ Fixed
FP-2	The updating may not be the latest	Implementation Vulnerability	Low	✓ Fixed

09. Issue descriptions

[FP-1] Length of `__gap` is incorrect

Code Improvement Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `MarketCapsRiskSteward.sol`, we defined a gap as `uint256[50] private __gap;`. Typically, this gap is used to fix the number of slots used in an upgradeable contract for inheritance, where the length of the gap plus the used state variables conventionally equals 50. Since this contract defines a state variable `maxDeltaBps`, the gap length should be `49`. We recommend modifying its definition to: `uint256[49] private __gap;`. This maintains consistency with the `__gap` meaning in `AccessControlledV8` and `OwnableUpgradeable`, benefiting code risk coherence.
Reference code:https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.8/contracts/access/OwnableUpgradeable.sol

```
/**
 * @dev This empty reserved space is put in place to allow future versions to add new
 * variables without shifting down storage in the inheritance chain.
 * See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
 */
uint256[49] private __gap;
```

Similarly, the `__gap` in `RiskStewardReceiver.sol` should be modified to `uint256[47] private __gap;`, because it has three state variables (non-immutable) defined earlier: `riskParameterConfigs`, `lastProcessedTime`, and `processedUpdates`.

Recommendation:

Modifying its definition to: `uint256[49] private __gap;`.

Update/Status:

The Venus team has fixed this issue.

[FP-2] The updating may not be the latest

Implementation Vulnerability | Low | ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `RiskStewardReceiver.sol`, consider a scenario where RiskOracle publishes multiple `supplyCap` recommendations for a market, such as ID4 and ID5. Since the `processUpdateById` function has no call permission restrictions, third parties could call `processUpdateById(4)` to update settings. However, due to update frequency limitations, others cannot call `processUpdateById(5)` within UPDATE_EXPIRATION_TIME to update the `supplyCap` to the latest recommended value. Although Venus management can directly call the relevant comptroller functions to manually update this value, it creates some inconvenience. Therefore, we recommend either removing the `processUpdateById` function and keeping only `processUpdateByParameterAndMarket` (which always updates to the latest recommended settings), or adding call permission restrictions to both functions to prevent misuse.

Recommendation:

Increase limit to ensure updated configuration is up to date

Update/Status:

The venus team has fixed the issue.

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. MarketCapsRiskSteward.t.js

```
const { expect } = require("chai");
const { ethers } = require("hardhat");
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");
```

```
describe("MarketCapsRiskSteward Unit Test", function () {
    async function deployFixture()  {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        // 1. Deploy MockRiskOracle
        const MockRiskOracle = await ethers.getContractFactory("MockRiskOracle");
        const oracle = await MockRiskOracle.deploy("MockRiskOracle",[alice.address],["supplyCap","borrowCap"]);
        // 2. Deploy AccessControlManager
        const AccessControlManager = await ethers.getContractFactory("AccessControlManager");
        const acm = await AccessControlManager.deploy();
        // 3. Deploy MockCoreComptroller And MockComptroller
        const MockCoreComptroller = await ethers.getContractFactory("MockCoreComptroller");
        const coreComptroller = await MockCoreComptroller.deploy();
        const MockComptroller = await ethers.getContractFactory("MockComptroller");
        const comptroller = await MockComptroller.deploy();
        // 4. Deploy RiskStewardReceiver
        const RiskStewardReceiver = await ethers.getContractFactory("RiskStewardReceiver");
        let implementation = await RiskStewardReceiver.deploy(oracle.target);
        let initData = implementation.interface.encodeFunctionData("initialize", [acm.target]);
        // 5. Deploy ProxyAdmin
        const ProxyAdmin = await ethers.getContractFactory("ProxyAdmin");
        const proxyAdmin = await ProxyAdmin.deploy();
        // 6. Depoly TransparentUpgradeableProxy
        const TransparentUpgradeableProxy = await ethers.getContractFactory("TransparentUpgradeableProxy");
        let receiver = await TransparentUpgradeableProxy.deploy(implementation.target, proxyAdmin.target, initData);
        receiver = RiskStewardReceiver.attach(receiver.target);

        // 7. Deploy MarketCapsRiskSteward
        const MarketCapsRiskSteward = await ethers.getContractFactory("MarketCapsRiskSteward");
        implementation = await MarketCapsRiskSteward.deploy(coreComptroller.target,receiver.target);
        initData = implementation.interface.encodeFunctionData("initialize", [acm.target,5000]);
        let steward = await TransparentUpgradeableProxy.deploy(implementation.target, proxyAdmin.target, initData);
        steward = MarketCapsRiskSteward.attach(steward.target);

        // 8 deploy MockVToken
        const MockVToken = await ethers.getContractFactory("MockVToken");
        const vToken1 = await MockVToken.deploy(coreComptroller.target);
        const vToken2 = await MockVToken.deploy(comptroller.target);

        // initialize twice should be failed
        await expect(receiver.initialize(acm.target)).to.be.revertedWith(
            "Initializable: contract is already initialized"
        );

        // initialize twice should be failed
        await expect(steward.initialize(acm.target,5000)).to.be.revertedWith(
            "Initializable: contract is already initialized"
        );

        return {owner,alice,bob,users,oracle,acm,vToken1,vToken2,
            coreComptroller,comptroller,receiver,steward};
    }

    describe("Deployment", function () {
        it("Should deploy the MarketCapsRiskSteward", async function () {
            const {steward} = await loadFixture(deployFixture);
            console.log("MarketCapsRiskSteward deployed to:", steward.target);
        });
    });

    describe("Features unit test", function () {

        it("setMaxDeltaBps test", async function () {
            const {owner,steward,alice,bob,acm} = await loadFixture(deployFixture);
            await expect(steward.connect(alice).setMaxDeltaBps(100)).to.be.revertedWithCustomError(
                steward, "Unauthorized"
            ).withArgs(alice.address,steward.target,"setMaxDeltaBps(uint256)");

            await acm.giveCallPermission(steward.target,"setMaxDeltaBps(uint256)",alice.address);
            await
expect(steward.connect(alice).setMaxDeltaBps(100)).to.emit(steward,"MaxDeltaBpsUpdated").withArgs(5000,100);
```

```
            expect(await steward.maxDeltaBps()).to.be.equal(100);
        });
    });

    describe("processUpdate test", function () {
        let encoder = ethers.AbiCoder.defaultAbiCoder();
        const params = {
            timestamp:parseInt(new Date().getTime() / 1000),
            newValue:encoder.encode(["uint256"],[10000]),
            referenceId:"test",
            previousValue:encoder.encode(["uint256"],[9000]),
            updateType:"supplyCap",
            updateId:5,
            market:ethers.getAddress("0x8ba1f109551bd432803012645ac136ddd64dba72"),
            additionalData:"0x"
        }

        it("only RISK_STEWARD_RECEIVER can call processUpdate", async function () {
            const {owner,steward,alice,bob,acm} = await loadFixture(deployFixture);
            await expect(steward.connect(alice).processUpdate(params)).to.be.revertedWithCustomError(
                steward, "OnlyRiskStewardReceiver"
            );
        });

        it("only can update supplyCap or borrowCap", async function () {
            const {owner,steward,receiver,alice,bob,acm} = await loadFixture(deployFixture);
            await hre.network.provider.request({
                method: "hardhat_impersonateAccount",
                params: [receiver.target],
            });
            await hre.network.provider.request({
                method: "hardhat_setBalance",
                params: [
                  receiver.target,
                  // 10 ETH
                  "0x8AC7230489E80000"
                ],
            });
            const signer = await ethers.getSigner(receiver.target);
            await
expect(steward.connect(signer).processUpdate({...params,updateType:"totalCap"})).to.be.revertedWithCustomError(
                steward, "UnsupportedUpdateType"
            );
        });
    });

    describe("update SupplyCap test", function () {
        let encoder = ethers.AbiCoder.defaultAbiCoder();
        let params,vToken1,vToken2;
        let signer,receiver,steward,coreComptroller,comptroller;
        beforeEach(async function () {
            ({steward,receiver,coreComptroller,comptroller,vToken1,vToken2} = await loadFixture(deployFixture));
            await hre.network.provider.request({
                method: "hardhat_impersonateAccount",
                params: [receiver.target],
            });
            await hre.network.provider.request({
                method: "hardhat_setBalance",
                params: [
                  receiver.target,
                  // 10 ETH
                  "0x8AC7230489E80000"
                ],
            });
            signer = await ethers.getSigner(receiver.target);
            params = {
                timestamp:parseInt(new Date().getTime() / 1000),
                newValue:encoder.encode(["uint256"],[10000]),
                referenceId:"test",
                previousValue:encoder.encode(["uint256"],[9000]),
                updateType:"supplyCap",
```

```
            updateId:5,
            market:vToken1.target,
            additionalData:"0x"
        }
    });
    it("update SupplyCap should not exceed MAX_BPS", async function () {
        await coreComptroller._supportMarket(params.market);
        await coreComptroller._setMarketSupplyCaps([params.market],[6000]);
        await expect(steward.connect(signer).processUpdate(params)).to.be.revertedWithCustomError(
            steward, "UpdateNotInRange"
        );
    });

    it("update SupplyCap should change state and emit event", async function () {
        await coreComptroller._supportMarket(params.market);
        await coreComptroller._setMarketSupplyCaps([params.market],[7000]);
        let old_cap = await coreComptroller.supplyCaps(params.market);
        expect(old_cap).to.be.equal(7000);
        await expect(steward.connect(signer).processUpdate(params)).to.emit(
            steward,"SupplyCapUpdated"
        ).withArgs(params.market,10000);
        let new_cap = await coreComptroller.supplyCaps(params.market);
        expect(new_cap).to.be.equal(10000);
    });

    it("update SupplyCap on IsolatedPoolsComptroller", async function () {
        params.market = vToken2.target;
        await comptroller.supportMarket(params.market);
        await comptroller.setMarketSupplyCaps([params.market],[7000]);
        let old_cap = await comptroller.supplyCaps(params.market);
        expect(old_cap).to.be.equal(7000);
        await expect(steward.connect(signer).processUpdate(params)).to.emit(
            steward,"SupplyCapUpdated"
        ).withArgs(params.market,10000);
        let new_cap = await comptroller.supplyCaps(params.market);
        expect(new_cap).to.be.equal(10000);
    });

});

describe("update BorrowCap test", function () {
    let encoder = ethers.AbiCoder.defaultAbiCoder();
    let params,vToken1,vToken2;
    let signer,receiver,steward,coreComptroller,comptroller;
    beforeEach(async function () {
        ({steward,receiver,coreComptroller,comptroller,vToken1,vToken2} = await loadFixture(deployFixture));
        await hre.network.provider.request({
            method: "hardhat_impersonateAccount",
            params: [receiver.target],
        });
        await hre.network.provider.request({
            method: "hardhat_setBalance",
            params: [
              receiver.target,
              // 10 ETH
              "0x8AC7230489E80000"
            ],
        });
        signer = await ethers.getSigner(receiver.target);
        params = {
            timestamp:parseInt(new Date().getTime() / 1000),
            newValue:encoder.encode(["uint256"],[10000]),
            referenceId:"test",
            previousValue:encoder.encode(["uint256"],[9000]),
            updateType:"borrowCap",
            updateId:5,
            market:vToken1.target,
            additionalData:"0x"
        }
    });
    it("update BorrowCap should not exceed MAX_BPS", async function () {
```

```
            await coreComptroller._supportMarket(params.market);
            await coreComptroller._setMarketBorrowCaps([params.market],[6000]);
            await expect(steward.connect(signer).processUpdate(params)).to.be.revertedWithCustomError(
                steward, "UpdateNotInRange"
            );
        });

        it("update BorrowCap should change state and emit event", async function () {
            await coreComptroller._supportMarket(params.market);
            await coreComptroller._setMarketBorrowCaps([params.market],[7000]);
            let old_cap = await coreComptroller.borrowCaps(params.market);
            expect(old_cap).to.be.equal(7000);
            await expect(steward.connect(signer).processUpdate(params)).to.emit(
                steward,"BorrowCapUpdated"
            ).withArgs(params.market,10000);
            let new_cap = await coreComptroller.borrowCaps(params.market);
            expect(new_cap).to.be.equal(10000);
        });

        it("update BorrowCap on IsolatedPoolsComptroller", async function () {
            params.market = vToken2.target;
            await comptroller.supportMarket(params.market);
            await comptroller.setMarketBorrowCaps([params.market],[7000]);
            let old_cap = await comptroller.borrowCaps(params.market);
            expect(old_cap).to.be.equal(7000);
            await expect(steward.connect(signer).processUpdate(params)).to.emit(
                steward,"BorrowCapUpdated"
            ).withArgs(params.market,10000);
            let new_cap = await comptroller.borrowCaps(params.market);
            expect(new_cap).to.be.equal(10000);
        });

    });
});
```

2. RiskStewardReceiver.t.js

```
const { expect } = require("chai");
const { ethers } = require("hardhat");
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");


describe("RiskStewardReceiver Unit Test", function () {
    async function deployFixture()  {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        // 1. Deploy MockRiskOracle
        const MockRiskOracle = await ethers.getContractFactory("MockRiskOracle");
        const oracle = await MockRiskOracle.deploy("MockRiskOracle",[alice.address],["supplyCap","borrowCap"]);
        // 2. Deploy AccessControlManager
        const AccessControlManager = await ethers.getContractFactory("AccessControlManager");
        const acm = await AccessControlManager.deploy();
        // 3. Deploy MockCoreComptroller And MockComptroller
        const MockCoreComptroller = await ethers.getContractFactory("MockCoreComptroller");
        const coreComptroller = await MockCoreComptroller.deploy();

        const MockComptroller = await ethers.getContractFactory("MockComptroller");
        const comptroller = await MockComptroller.deploy();
        // 4. Deploy RiskStewardReceiver
        const RiskStewardReceiver = await ethers.getContractFactory("RiskStewardReceiver");
        let implementation = await RiskStewardReceiver.deploy(oracle.target);
        let initData = implementation.interface.encodeFunctionData("initialize", [acm.target]);
        // 5. Deploy ProxyAdmin
        const ProxyAdmin = await ethers.getContractFactory("ProxyAdmin");
        const proxyAdmin = await ProxyAdmin.deploy();
        // 6. Depoly TransparentUpgradeableProxy
        const TransparentUpgradeableProxy = await ethers.getContractFactory("TransparentUpgradeableProxy");
        let receiver = await TransparentUpgradeableProxy.deploy(implementation.target, proxyAdmin.target, initData);
```

```
        receiver = RiskStewardReceiver.attach(receiver.target);

        // 7. Deploy MarketCapsRiskSteward
        const MarketCapsRiskSteward = await ethers.getContractFactory("MarketCapsRiskSteward");
        implementation = await MarketCapsRiskSteward.deploy(coreComptroller.target,receiver.target);
        initData = implementation.interface.encodeFunctionData("initialize", [acm.target,5000]);
        let steward = await TransparentUpgradeableProxy.deploy(implementation.target, proxyAdmin.target, initData);
        steward = MarketCapsRiskSteward.attach(steward.target);

        // 8 deploy MockVToken
        const MockVToken = await ethers.getContractFactory("MockVToken");
        const vToken1 = await MockVToken.deploy(coreComptroller.target);
        const vToken2 = await MockVToken.deploy(comptroller.target);

        // initialize twice should be failed
        await expect(receiver.initialize(acm.target)).to.be.revertedWith(
            "Initializable: contract is already initialized"
        );

        // initialize twice should be failed
        await expect(steward.initialize(acm.target,5000)).to.be.revertedWith(
            "Initializable: contract is already initialized"
        );

        await coreComptroller._supportMarket(vToken1.target);
        await coreComptroller._setMarketSupplyCaps([vToken1.target],[10000]);
        await coreComptroller._setMarketBorrowCaps([vToken1.target],[10000]);

        await comptroller.supportMarket(vToken2.target);
        await comptroller.setMarketSupplyCaps([vToken2.target],[10000]);
        await comptroller.setMarketBorrowCaps([vToken2.target],[10000]);

        let encoder = ethers.AbiCoder.defaultAbiCoder();
        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [10000]
            ),
            "supplyCap",
            vToken1.target,
            "0x"
        );
        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [10000]
            ),
            "borrowCap",
            vToken1.target,
            "0x"
        );

        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [10000]
            ),
            "supplyCap",
            vToken2.target,
            "0x"
        );

        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [10000]
            ),
```

```
            "borrowCap",
            vToken2.target,
            "0x"
        );

        return {owner,alice,bob,users,oracle,acm,vToken1,vToken2,
            coreComptroller,comptroller,receiver,steward,encoder};
    }

    describe("Deployment", function () {
        it("Should deploy the MarketCapsRiskSteward", async function () {
            const {receiver} = await loadFixture(deployFixture);
            console.log("RiskStewardReceiver deployed to:", receiver.target);
        });
    });

    describe("pause and unpause test", function () {
        it("Only grant role can set", async function () {
            const {owner,receiver,alice,bob,acm} = await loadFixture(deployFixture);
            expect(await receiver.paused()).to.be.false;

            await expect(receiver.connect(alice).pause()).to.be.revertedWithCustomError(
                receiver, "Unauthorized"
            ).withArgs(alice.address,receiver.target,"pause()");


            await acm.giveCallPermission(receiver.target,"pause()",alice.address);
            await expect(receiver.connect(alice).pause()).to.emit(receiver,"Paused").withArgs(alice.address);
            expect(await receiver.paused()).to.be.true;
            // pause again
            await expect(receiver.connect(alice).pause()).to.be.revertedWith(
                "Pausable: paused"
            );

            await expect(receiver.connect(alice).unpause()).to.be.revertedWithCustomError(
                receiver, "Unauthorized"
            ).withArgs(alice.address,receiver.target,"unpause()");
            // upause
            await acm.giveCallPermission(receiver.target,"unpause()",alice.address);
            await expect(receiver.connect(alice).unpause()).to.emit(receiver,"Unpaused").withArgs(alice.address);
            expect(await receiver.paused()).to.be.false;

            // unpause again
            await expect(receiver.connect(alice).unpause()).to.be.revertedWith(
                "Pausable: not paused"
            );
        });
    });

    describe("setRiskParameterConfig test", function () {
        it("Only grant role can set", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 24*3600+1,
            )).to.be.revertedWithCustomError(
                receiver, "Unauthorized"
            ).withArgs(alice.address,receiver.target,"setRiskParameterConfig(string,address,uint256)");
        });

        it("empty string should be failed", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "",steward.target, 24*3600+1,
            )).to.be.revertedWithCustomError(
                receiver, "UnsupportedUpdateType"
            );
        });

        it("Wrong debounce should be failed", async function () {
```

```
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 24*3600,
            )).to.be.revertedWithCustomError(
                receiver, "InvalidDebounce"
            );
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 24*3600 - 1,
            )).to.be.revertedWithCustomError(
                receiver, "InvalidDebounce"
            );
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 0,
            )).to.be.revertedWithCustomError(
                receiver, "InvalidDebounce"
            );
        });

        it("Zero address should be failed", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",ethers.ZeroAddress, 24*3600+1,
            )).to.be.revertedWithCustomError(
                receiver, "ZeroAddressNotAllowed"
            );
        });

        it("setRiskParameterConfig should change state and emit event", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 24*3600+1,
            )).to.emit(receiver,"RiskParameterConfigSet").withArgs(
                "supplyCap",
                ethers.ZeroAddress,
                steward.target,
                0,
                24*3600+1,
                false,
                true
            );
            // set again
            await expect(receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",acm.target, 24*3600+10,
            )).to.emit(receiver,"RiskParameterConfigSet").withArgs(
                "supplyCap",
                steward.target,
                acm.target,
                24*3600+1,
                24*3600+10,
                true,
                true
            );
        });
    });

    describe("toggleConfigActive test", function () {
        it("Only grant role can set", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await expect(receiver.connect(alice).toggleConfigActive(
                "supplyCap"
            )).to.be.revertedWithCustomError(
                receiver, "Unauthorized"
            ).withArgs(alice.address,receiver.target,"toggleConfigActive(string)");
        });
```

```
        it("riskSteward must not be set zero address", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await acm.giveCallPermission(receiver.target,"toggleConfigActive(string)",alice.address);
            await expect(receiver.connect(alice).toggleConfigActive(
                "supplyCap"
            )).to.be.revertedWithCustomError(
                receiver, "UnsupportedUpdateType"
            );
        });

        it("toggleConfigActive should change state and emit event", async function () {
            const {owner,receiver,alice,bob,acm,steward} = await loadFixture(deployFixture);
            await acm.giveCallPermission(receiver.target,"toggleConfigActive(string)",alice.address);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);

            await receiver.connect(alice).setRiskParameterConfig(
                "supplyCap",steward.target, 24*3600+1,
            );
            await expect(receiver.connect(alice).toggleConfigActive(
                "supplyCap"
            )).to.emit(receiver,"ToggleConfigActive").withArgs(
                "supplyCap",
                false
            );
            // toggle again
            await expect(receiver.connect(alice).toggleConfigActive(
                "supplyCap"
            )).to.emit(receiver,"ToggleConfigActive").withArgs(
                "supplyCap",
                true
            );
        });
    });

    describe("processUpdateById test", function () {

        it("config must be active", async function () {
            const {
                receiver,steward,alice,acm,token1,oracle
            } = await loadFixture(deployFixture);
            await expect(receiver.processUpdateById(1)).to.be.revertedWithCustomError(
                receiver, "ConfigNotActive"
            );

        });

        it("must in execution window", async function () {
            const {
                receiver,steward,alice,acm,token1,oracle
            } = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await receiver.connect(alice).setRiskParameterConfig("supplyCap",steward.target, 24*3600+1);

            await ethers.provider.send("evm_increaseTime", [24*3600+3]);
            await expect(receiver.processUpdateById(1)).to.be.revertedWithCustomError(
                receiver, "UpdateIsExpired"
            );
        });

        it("processUpdateById should change state and emit event", async function () {
            const {
                receiver,steward,alice,acm,vToken1,oracle,encoder
            } = await loadFixture(deployFixture);
            await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
            await receiver.connect(alice).setRiskParameterConfig("supplyCap",steward.target, 24*3600+1);

            await ethers.provider.send("evm_increaseTime", [23*3600]);
            await expect(receiver.processUpdateById(1)).to.be.emit(
```

```
            receiver, "RiskParameterUpdated"
        ).withArgs(1);
        expect(await receiver.processedUpdates(1)).to.be.true;


        // should be failed it is already processed
        await expect(receiver.processUpdateById(1)).to.be.revertedWithCustomError(
            receiver,"ConfigAlreadyProcessed"
        );
        // add 1 hour
        await ethers.provider.send("evm_increaseTime", [1*3600]);
        // should be failed if update is too frequent
        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [12000]
            ),
            "supplyCap",
            vToken1.target,
            "0x"
        );
        await expect(receiver.processUpdateById(5)).to.be.revertedWithCustomError(
            receiver,"UpdateTooFrequent"
        );
        // should be successful if update is not too frequent and in execution window
        await ethers.provider.send("evm_increaseTime", [24*3600 -10]);
        expect(await receiver.processedUpdates(5)).to.be.false;
        await expect(receiver.processUpdateById(5)).to.be.emit(
            receiver, "RiskParameterUpdated"
        ).withArgs(5);
        expect(await receiver.processedUpdates(5)).to.be.true;
    });

    it("Should be failed if update is not latest", async function () {
        const {
            receiver,steward,alice,acm,vToken1,oracle,encoder
        } = await loadFixture(deployFixture);
        await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
        await oracle.connect(alice).publishRiskParameterUpdate(
            "test referenceId",
            encoder.encode(
                ["uint256"],
                [10000]
            ),
            "supplyCap",
            vToken1.target,
            "0x"
        );
        await receiver.connect(alice).setRiskParameterConfig("supplyCap",steward.target, 24*3600+1);
        await expect(receiver.processUpdateById(1)).to.be.revertedWithCustomError(
            receiver,"UpdateIsExpired"
        );
        await expect(receiver.processUpdateById(5)).to.be.emit(
            receiver, "RiskParameterUpdated"
        ).withArgs(5);
        expect(await receiver.processedUpdates(5)).to.be.true;
    });
});

describe("processUpdateByParameterAndMarket test", function () {
    it("update should change state and emit event", async function () {
        const {
            receiver,steward,alice,acm,vToken2,oracle
        } = await loadFixture(deployFixture);
        await
acm.giveCallPermission(receiver.target,"setRiskParameterConfig(string,address,uint256)",alice.address);
        await receiver.connect(alice).setRiskParameterConfig("supplyCap",steward.target, 24*3600+1);
        await receiver.connect(alice).setRiskParameterConfig("borrowCap",steward.target, 24*3600+1);
```

```
            await ethers.provider.send("evm_increaseTime", [23*3600-5]);
            await expect(receiver.processUpdateByParameterAndMarket("supplyCap",vToken2.target)).to.be.emit(
                receiver, "RiskParameterUpdated"
            ).withArgs(3);

            expect(await receiver.processedUpdates(3)).to.be.true;
            await expect(receiver.processUpdateByParameterAndMarket("borrowCap",vToken2.target)).to.be.emit(
                receiver, "RiskParameterUpdated"
            ).withArgs(4);

            expect(await receiver.processedUpdates(4)).to.be.true;
        });
    });

    describe("renounceOwnership test", function () {
        it("should be failed anyhow", async function () {
            const {owner,receiver,alice} = await loadFixture(deployFixture);
            expect(await receiver.owner()).to.be.equal(owner.address);
            await expect(receiver.connect(alice).renounceOwnership()).to.be.revertedWith(
                " renounceOwnership() is not allowed"
            );

            await expect(receiver.renounceOwnership()).to.be.revertedWith(
                " renounceOwnership() is not allowed"
            );
        });
    });
});
```

3. UnitTestOutput

```
  MarketCapsRiskSteward Unit Test
    Deployment
MarketCapsRiskSteward deployed to: 0x2279B7A0a67DB372996a5FaB50D91eAA73d2eBe6
      ✔ Should deploy the MarketCapsRiskSteward (441ms)
    Features unit test
      ✔ setMaxDeltaBps test
    processUpdate test
      ✔ only RISK_STEWARD_RECEIVER can call processUpdate
      ✔ only can update supplyCap or borrowCap
    update SupplyCap test
      ✔ update SupplyCap should not exceed MAX_BPS
      ✔ update SupplyCap should change state and emit event
      ✔ update SupplyCap on IsolatedPoolsComptroller
    update BorrowCap test
      ✔ update BorrowCap should not exceed MAX_BPS
      ✔ update BorrowCap should change state and emit event
      ✔ update BorrowCap on IsolatedPoolsComptroller

  RiskStewardReceiver Unit Test
    Deployment
RiskStewardReceiver deployed to: 0x59b670e9fA9D0A427751Af201D676719a970857b
      ✔ Should deploy the MarketCapsRiskSteward
    pause and unpause test
      ✔ Only grant role can set
    setRiskParameterConfig test
      ✔ Only grant role can set
      ✔ empty string should be failed
      ✔ Wrong debounce should be failed
      ✔ Zero address should be failed
      ✔ setRiskParameterConfig should change state and emit event
    toggleConfigActive test
      ✔ Only grant role can set
      ✔ riskSteward must not be set zero address
      ✔ toggleConfigActive should change state and emit event
    processUpdateById test
```

```
      ✔ config must be active
      ✔ must in execution window
      ✔ processUpdateById should change state and emit event
      ✔ Should be failed if update is not latest
    processUpdateByParameterAndMarket test
      ✔ update should change state and emit event
    renounceOwnership test
      ✔ should be failed anyhow


  26 passing (563ms)
```

11.2 External Functions Check Points

1. MarketCapsRiskSteward.sol.md

File: contracts/RiskSteward/MarketCapsRiskSteward.sol

contract: MarketCapsRiskSteward is IRiskSteward, AccessControlledV8

(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,uint256)		`initializer`	Yes	False	Passed	Only Once
2	setMaxDeltaBps(uint256)			Yes	False	Passed	_checkAccessAllowed("setMaxDeltaBps(uint256)")
3	processUpdate(RiskParameterUpdate)				False	Passed	Only msg.sender == RISK_STEWARD_RECEIVER
4	renounceOwnership()				False	Passed	Always Revert
5	setAccessControlManager(address)		`onlyOwner`		False		
6	accessControlManager()	view			False		

2. RiskStewardReceiver.sol.md

File: contracts/RiskSteward/RiskStewardReceiver.sol

contract: RiskStewardReceiver is IRiskStewardReceiver, PausableUpgradeable, AccessControlledV8

(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`		False	Passed	Only Once
2	pause()				False	Passed	_checkAccessAllowed("pause()")
3	unpause()				False	Passed	_checkAccessAllowed("unpause()")
4	setRiskParameterConfig(string,address,uint256)			Yes	False	Passed	_checkAccessAllowed("setRiskParameterConfig(string,address,uint256
5	toggleConfigActive(string)				False	Passed	_checkAccessAllowed("toggleConfigActive(string)")
6	processUpdateById(uint256)		`whenNotPaused`	Yes	Yes	Passed	
7	processUpdateByParameterAndMarket(string,address)		`whenNotPaused`		Yes	Passed	
8	renounceOwnership()				False	Passed	Always Revert
9	setAccessControlManager(address)		`onlyOwner`		False		
10	accessControlManager()	view			False		

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