FAIRYPROOF

ERC4626 VToken Wrapper

AUDIT REPORT

Version 1.0.0

Serial No. 2025041400012021

Presented by Fairyproof

April 14, 2025

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus ERC4626 VToken Wrapper project.

Audit Start Time:

April 8, 2025

Audit End Time:

April 14, 2025

Audited Source File's Address:

https://github.com/VenusProtocol/isolated-pools/pull/497

https://github.com/VenusProtocol/protocol-reserve/pull/137

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/ERC4626/Interfaces/IComptroller.sol
contracts/ERC4626/Interfaces/IRewardsDistributor.sol
contracts/ERC4626/VenusERC4626.sol
contracts/ERC4626/VenusERC4626Factory.sol

contracts/Interfaces/IProtocolShareReserve.sol
```

The goal of this audit is to review Venus's solidity implementation for its ERC4626 VToken Wrapper 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:

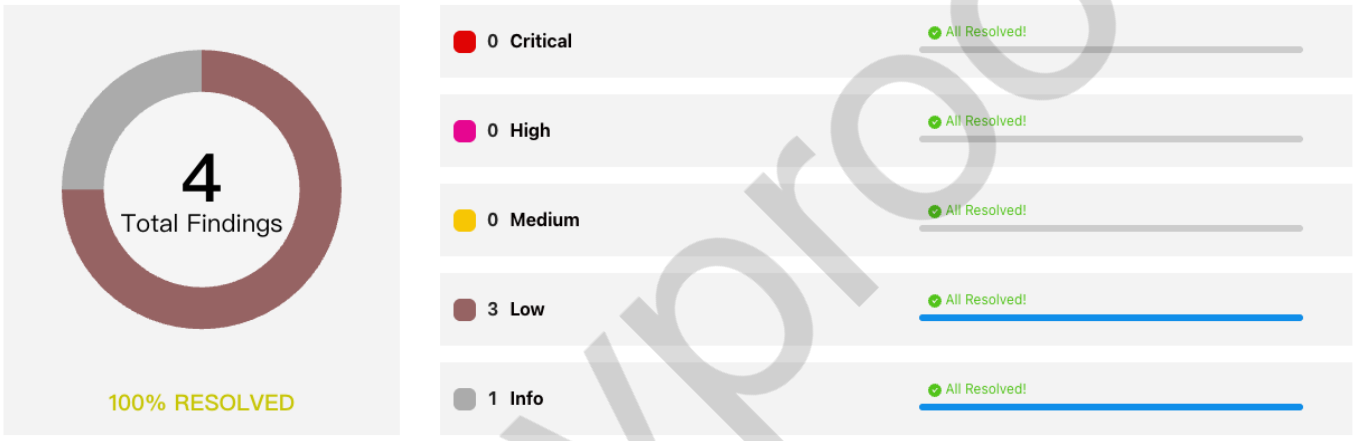
https://github.com/VenusProtocol/isolated-pools/pull/497

https://github.com/VenusProtocol/protocol-reserve/pull/137

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
2025041400012021	Fairyproof Security Team	Apr 8, 2025 - Apr 14, 2025	Passed



Summary:

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

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,Arbitrum and Unichain.

The above description is quoted from relevant documents of Venus.

04. Major functions of audited code

The audit code provide wrappers for the `VToken` contracts, following the ERC-4626 vault standard.

Basic flows:

- Deposits:
  - Users deposit funds to the 4626 wrapper contract. In the same transaction, those funds are supplied to the associated VToken contract (if it's allowed)
  - The 4626 wrapper contract holds the minted VTokens
  - The user holds the 4626 wrapper tokens
- Withdrawals:
  - Users redeem their 4626 wrapper tokens, interacting with the wrapper contract. In the same transaction, the wrapper redeems the required VTokens to withdraw underlying funds from the VToken contract and transfer them to the user
- Rewards:
  - The 4626 wrapper contract accrues rewards associated with the VToken contract. These rewards are sent to the Venus ProtocolShareReserve contract, with a new income type: `ERC4626_WRAPPER_REWARDS`

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 found some issues, for more details please refer to [FP-1,FP-3] in "09. Issue description".

- 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-4] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Implementation contracts can be initialized	Implementation Vulnerability	Low	✓ Fixed
FP-2	Missing Non-Zero Value Validation	Code Improvement	Low	✓ Fixed
FP-3	Missing Additional Reentrancy Protection Measures	Re-entrancy Attack	Low	✓ Fixed
FP-4	Miss address parameter	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Implementation contracts can be initialized

Implementation Vulnerability Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In the VenusERC4626 and VenusERC4626Factory contracts, the implementation contracts can be initialized, which may lead to unexpected results in specific scenarios.

Recommendation:

Add a constructor to both VenusERC4626 and VenusERC4626Factory to prevent the Implementation contracts from being initialized:

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
  // Note that the contract is upgradeable. Use initialize() or reinitializers
  // to set the state variables.
  _disableInitializers();
}
```

Update:

The venus team has fixed the issue at commit `c1202ca0de5d92f9812c3cd3303c25ce8a4fb7d2`.

Status:

Fixed

[FP-2] Missing Non-Zero Value Validation

Code Improvement Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `VenusERC4626.sol`, missing some non-zero value validations. Specifically:

- In the `deposit` function, add a check that `shares != 0`
- In the `mint` function, add a check that the minted amount is not zero
- In the `withdraw` function, add a check that `shares != 0`
- In the `redeem` function, add a check that `assets != 0`

Recommendation:

Add appropriate checks in the corresponding functions to ensure relevant variables are not zero.

Update:

The venus team has fixed the issue at commit `c1202ca0de5d92f9812c3cd3303c25ce8a4fb7d2`

Status:

Fixed

[FP-3] Missing Additional Reentrancy Protection Measures

Re-entrancy Attack Low ✓ Fixed

Issue/Risk: Re-entrancy Attack

Description:

In `VenusERC4626`, the functions `deposit`, `mint`, `withdraw`, and `redeem` do not have additional reentrancy protection measures. Although the inherited `ERC4626Upgradeable` already handles reentrancy attack scenarios, considering potential expansion to multiple token types in the future, it is still recommended to add extra reentrancy check measures.

Recommendation:

The `VenusERC4626` contract inherits `ReentrancyGuardUpgradeable` to use the `nonReentrant` modifier on specific functions to prevent reentrancy.

Update:

The venus team has fixed the issue at commit `ae56b6424bcd56c8e8d35ea39da0a1cb89f8b590` and `9cfef3df33df1c04b191f5f17f153bdd5efed3af`.

Status:

Fixed

[FP-4] Miss address parameter

Code Improvement Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `VenusERC4626.sol`, the `ClaimRewards` event has missed an address parameter that represents the address of `rewardToken`. When there are multiple `ClaimRewards` events, it's impossible to distinguish which amount belongs to which `rewardToken`.

Recommendation:

Modify the definition of ClaimRewards by adding a parameter of type address that represents the address of rewardToken.

Update:

The venus team has fixed the issue at commit c1202ca0de5d92f9812c3cd3303c25ce8a4fb7d2.

Status:

Fixed

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.

- All `VenusERC4626` contracts use the same template, pointing to the same `beacon` address, essentially following a clone pattern. The current `VenusERC4626Factory` contract lacks the mechanism to upgrade the `beacon` implementation address.

  Reply:

  VenusERC4626Factory is upgradable. If it’s required to update beacon implementation address, the required setter function could be added in the future.

- Rewards of `VenusERC4626` contracts are collected via the `claimRewards` function, and then the `ProtocolShareReserve` is responsible for recording and distributing these rewards. However, the current contract does not show any implementation of reward distribution. Since staking has no time limit, measures should be taken to prevent speculative behavior where staking occurs only just before rewards are distributed.

  Reply:

  Rewards generated by the underlying tokens deposited into the ERC4626 contract (and then deposited into the associated VToken) are sent to the protocol (specifically to the ProtocolShareReserve, where there will be some rules configured to distribute these funds, as the DAO decides).

  These rewards are not sent to the ERC4626 users.

11. Appendices

11.1 Unit Test

1. MockProtocolShareReserve.sol

```
pragma solidity 0.8.13;

interface IProtocolShareReserve {
    /// @notice it represents the type of vToken income
    enum IncomeType {
        SPREAD,
        LIQUIDATION
    }

    function updateAssetsState(
        address comptroller,
        address asset,
        IncomeType incomeType
    ) external;
}


contract ProtocolShareReserve is IProtocolShareReserve {

    event UpdateAssetsState(
        address indexed comptroller,
        address indexed asset,
        IncomeType incomeType
    );

    function updateAssetsState(
        address comptroller,
        address asset,
        IncomeType incomeType
    ) external {
        emit UpdateAssetsState(comptroller, asset, incomeType);
    }
}
```

2. MockPoolRegistery.sol

```
pragma solidity 0.8.25;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MockPoolRegistry {
    mapping(address => address) public assets;
    /// @notice Get the address of the VToken contract in the Pool where the underlying token is the provided
asset
    function getVTokenForAsset(address comptroller, address asset) external view returns (address) {
        comptroller;
        return assets[asset];
    }

    function setVTokenForAsset(address asset, address vToken) external {
        assets[asset] = vToken;
    }
}

contract MockComptroller {}

contract MockToken  is ERC20 {
    constructor() ERC20("MockToken", "MTK") {}

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) external {
        _burn(from, amount);
    }
}

contract MockVToken {
    address public underlying;
    address public comptroller;

    constructor(address _underlying, address _comptroller) {
        underlying = _underlying;
        comptroller = _comptroller;
    }
}
```

3. ProtocolShareReserve.js

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

describe("ProtocolShareReserve unit test", function () {
    const IncomeType = {
        SPREAD:0,
        LIQUIDATION:1,
        ERC4626_WRAPPER_REWARDS:2,
        FLASHLOAN:3,

    }
    async function deployFixture() {
        const [alice,bob] = await ethers.getSigners();
        const ProtocolShareReserve = await ethers.getContractFactory("ProtocolShareReserve");
        const protocolShareReserve = await ProtocolShareReserve.deploy();

        return {protocolShareReserve, alice,bob};
    }

    it("Should emit event when setProtocolShareReserve", async function () {
        const {protocolShareReserve, alice,bob} = await loadFixture(deployFixture);

        let incomeType = IncomeType.LIQUIDATION;

        await expect(protocolShareReserve.updateAssetsState(alice.address, bob.address, incomeType))
```

```
            .to.emit(protocolShareReserve, "UpdateAssetsState")
            .withArgs(alice.address, bob.address, incomeType);
    });

    it("Should be failed if IncomeType is not in range", async function () {
        const {protocolShareReserve, alice,bob} = await loadFixture(deployFixture);

        let incomeType = IncomeType.ERC4626_WRAPPER_REWARDS;

        await expect(protocolShareReserve.updateAssetsState(alice.address, bob.address, incomeType))
            .to.be.reverted;

    });
});
```

4. VenusERC4626Factory.js

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

describe("VenusERC4626Factory", function () {
    async function deployFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        const rewardRecipientAddress = alice.address;
        const loopsLimitNumber = 5;
        // 1 deploy acm
        const AccessControlManager = await ethers.getContractFactory("AccessControlManager");
        const acm = await AccessControlManager.deploy();
        // 2 deploy MockPoolRegistry
        const MockPoolRegistry = await ethers.getContractFactory("MockPoolRegistry");
        const poolRegistry = await MockPoolRegistry.deploy();
        // 3 deploy MockComptroller
        const MockComptroller = await ethers.getContractFactory("MockComptroller");
        const comptroller = await MockComptroller.deploy();
        // 4 deploy MockToken and MockVToken
        const ERC20 = await ethers.getContractFactory("ERC20");
        const WBNB = await ERC20.deploy("Wrapped BNB", "WBNB");
        const MockVToken = await ethers.getContractFactory("MockVToken");
        const vToken = await MockVToken.deploy(WBNB,comptroller.target);
        // 5 setup poolRegistry
        await poolRegistry.setVTokenForAsset(WBNB, vToken.target);

        // 3 deploy VenusERC4626 implementation
        const VenusERC4626 = await ethers.getContractFactory("VenusERC4626");
        const venusERC4626 = await VenusERC4626.deploy();
        // 4 deploy VenusERC4626Factory
        const VenusERC4626Factory = await ethers.getContractFactory("VenusERC4626Factory");
        const factory = await upgrades.deployProxy(VenusERC4626Factory,[
            acm.target,
            poolRegistry.target,
            rewardRecipientAddress,
            venusERC4626.target,
            loopsLimitNumber
        ]);
        await acm.giveCallPermission(
            factory.target,
            "setRewardRecipient(address)",
            owner.address
        )
        const BeaconProxy = await ethers.getContractFactory("BeaconProxy");
        return { factory, owner, alice, bob, users, acm, poolRegistry,vToken, venusERC4626,BeaconProxy};
    }

    describe("Deployment", function () {
        it("Should set the right owner", async function () {
            const { factory, owner } = await loadFixture(deployFixture);
            expect(await factory.owner()).to.equal(owner.address);
        });

        it("initialize twice should fail", async function () {
            const { factory, acm,alice,poolRegistry,venusERC4626 } = await loadFixture(deployFixture);
            await expect(factory.initialize(
                acm.target,
                poolRegistry.target,
                alice.address,
                venusERC4626.target,
                5
            )).to.be.revertedWith("Initializable: contract is already initialized");
        });
    });

    describe("setRewardRecipient", function () {
        it("Should set the right rewardRecipient", async function () {
            const { factory, alice } = await loadFixture(deployFixture);
            await factory.setRewardRecipient(alice.address);
            expect(await factory.rewardRecipient()).to.equal(alice.address);
        });

        it("Should revert if not owner", async function () {
            const { factory, bob } = await loadFixture(deployFixture);
            await expect(factory.connect(bob).setRewardRecipient(bob.address)).to.be.revertedWithCustomError(
                factory,"Unauthorized"
            ).withArgs(bob.address, factory.target,"setRewardRecipient(address)");
        });
    });

    describe("computeVaultAddress", function () {
        it("should compute the right vault address", async function () {
            const { factory,venusERC4626, alice,BeaconProxy } = await loadFixture(deployFixture);
            // 1 compute the vault address on chain
            let vault = await factory.computeVaultAddress(alice.address);
           // 2 compute the expected vault address
            let encoder = ethers.AbiCoder.defaultAbiCoder();
            let interface = venusERC4626.interface;
            let code = interface.encodeFunctionData("initialize", [
                alice.address,
                alice.address,
                5
            ]);
            let beacon = await factory.beacon();
            let constructorParams = encoder.encode(
                ["address", "bytes"],
                [beacon, code]
            );
            let creationCode = BeaconProxy.bytecode;
            let initcode_hash = ethers.solidityPackedKeccak256(
                ["bytes", "bytes"],
                [creationCode, constructorParams]
            )
            let expect_vault = ethers.getCreate2Address(
                factory.target,
                ethers.id("Venus-ERC4626 Vault"),
                initcode_hash
            );
            assert.equal(vault, expect_vault);
        });
    });

    describe("createERC4626Vault", function () {
        it("should create a new vault on expected address", async function () {
            const { factory,alice,vToken } = await loadFixture(deployFixture);
            let vault = await factory.computeVaultAddress(vToken.target);
            await expect(factory.createERC4626(vToken.target))
            .to.emit(factory, "CreateERC4626").withArgs(
                vToken.target,
                vault,
            )
        });
        it("should create a new vault on expected address with other creator", async function () {
            const { factory,alice,vToken } = await loadFixture(deployFixture);
            let vault = await factory.connect(alice).computeVaultAddress(vToken.target);
            await expect(factory.createERC4626(vToken.target))
            .to.emit(factory, "CreateERC4626").withArgs(
                vToken.target,
                vault,
            )
        });
    });
});
```

5. VenusERC4626Fork.js

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

/// Fork test on BNB chain at Block:48206947
describe("VenusERC4626 Fork Unit Test", function () {
    const BNBx = "0x1bdd3Cf7F79cfB8EdbB955f20ad99211551BA275";  // decimals 18
    const BNBx_HOLDER = "0xE8322f6234B6F1e6e3489600f8b1297aB3dE22ab";
    const vBNBx_TOKEN = "0x5E21bF67a6af41c74C1773E4b473ca5ce8fd3791"; // decimals 8
    const vBNBx_TOKEN_HOLDER = "0x2B518fC21ca652943Cd4014323B8d9eE3c1778f2";
    const COMPTROLLER = "0xd933909A4a2b7A4638903028f44D1d38ce27c352"
    const ProtocolShareReserve = "0xCa01D5A9A248a830E9D93231e791B1afFed7c446";
    const ACM_ADDRESS = "0x4788629ABc6cFCA10F9f969efdEAa1cF70c23555";
    const POOL_REGISTRY = "0x9F7b01A536aFA00EF10310A162877fd792cD0666";
    const loopsLimitNumber = 5;

    async function deployFixture() {
        const [owner, alice, bob, ...users] = await ethers.getSigners();
        // 1 get acm
        const AccessControlManager = await ethers.getContractFactory("AccessControlManager");
        const acm = AccessControlManager.attach(ACM_ADDRESS);
        // 2 get poolRegistry
        const MockPoolRegistry = await ethers.getContractFactory("MockPoolRegistry");
        const poolRegistry = MockPoolRegistry.attach(POOL_REGISTRY);
        // 3 get comptroller
        let Comptroller = await ethers.getContractFactory("Comptroller");
        const comptroller = Comptroller.attach(COMPTROLLER);
        // 4 get vToken
        const VToken = await ethers.getContractFactory("VToken");
        const vToken = VToken.attach(vBNBx_TOKEN);
        // 5 deploy VenusERC4626
        const VenusERC4626 = await ethers.getContractFactory("VenusERC4626");
        const venusERC4626_template = await VenusERC4626.deploy();
        // 6 deploy factory
        const VenusERC4626Factory = await ethers.getContractFactory("VenusERC4626Factory");
        const factory = await upgrades.deployProxy(VenusERC4626Factory,[
            acm.target,
            poolRegistry.target,
            ProtocolShareReserve,
            venusERC4626_template.target,
            loopsLimitNumber
        ]);

        // 7 create venusERC4626
        let vault = await factory.computeVaultAddress(vToken.target);
        await expect(factory.createERC4626(vToken.target))
        .to.emit(factory, "CreateERC4626").withArgs(
            vToken.target,
            vault,
        )
        let venusERC4626 = VenusERC4626.attach(vault);

        // 8 transfer bnbx to owner and alice
        const MockToken = await ethers.getContractFactory("MockToken");
        const bnbx = MockToken.attach(BNBx);
        await impersonateAccount(BNBx_HOLDER);
        const bnbxHolder = await ethers.getSigner(BNBx_HOLDER);
        await bnbx.connect(bnbxHolder).transfer(owner.address, ethers.parseUnits("1000", 18));
        await bnbx.connect(bnbxHolder).transfer(alice.address, ethers.parseUnits("1000", 18));

        // 9 approve bnbx to venusERC4626
        bnbx.connect(owner).approve(venusERC4626.target, ethers.parseUnits("100000000", 18));
        bnbx.connect(alice).approve(venusERC4626.target, ethers.parseUnits("100000000", 18));
        // 10 transfer vToken to bob
        await impersonateAccount(vBNBx_TOKEN_HOLDER);
        const vTokenHolder = await ethers.getSigner(vBNBx_TOKEN_HOLDER);
        await vToken.connect(vTokenHolder).transfer(bob.address, ethers.parseUnits("10", 8));

        return { owner, alice, bob, users, vToken, bnbx,venusERC4626,factory ,comptroller};
    }

    describe("Deployment", function () {
        it("createERC4626 again should fail", async function () {
            const { factory, vToken } = await loadFixture(deployFixture);
            await expect(factory.createERC4626(vToken.target)).to.be.reverted;
        });
        it("initialize twice should fail", async function () {
            const { venusERC4626,vToken,alice } = await loadFixture(deployFixture);
            await expect(venusERC4626.initialize(
                vToken.target,
                alice.address,
                5
            )).to.be.revertedWith("Initializable: contract is already initialized");
        });

        it("parameter check", async function () {
            const { venusERC4626,vToken } = await loadFixture(deployFixture);
            expect(await venusERC4626.vToken()).to.equal(vToken.target);
            expect(await venusERC4626.asset()).to.equal(BNBx);
            expect(await venusERC4626.comptroller()).to.equal(COMPTROLLER);
            expect(await venusERC4626.rewardRecipient()).to.equal(ProtocolShareReserve);
            let name = await venusERC4626.name();
            console.log("name", name);
            let symbol = await venusERC4626.symbol();
            console.log("symbol", symbol);
        });
    });

    describe("Deposit and Redeem", function () {
        it("Deploy and Redeem in normal case", async function () {
            const { owner, alice,bob, bnbx,vToken,venusERC4626 } = await loadFixture(deployFixture);

            // 1 alice deposit 10 bnbx
            let depositAmount = ethers.parseUnits("10", 18);
            const shares = await venusERC4626.previewDeposit(depositAmount);
            expect(shares).to.equal(depositAmount);
            await expect(venusERC4626.connect(alice).deposit(depositAmount, alice.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(alice.address, alice.address, depositAmount, shares);
            expect(await venusERC4626.balanceOf(alice.address)).to.equal(shares);

            // 2 owner deposit 5 bnbx
            depositAmount = depositAmount / 2n
            let shares2 = await venusERC4626.previewDeposit(depositAmount);
            await venusERC4626.deposit(depositAmount, owner.address);
            console.log("shares2",ethers.formatUnits(shares2, 18));
            expect(await venusERC4626.balanceOf(owner.address)).to.equal(shares2);

            let exchangeRateStored = await vToken.exchangeRateStored();
            console.log("exchangeRateStored", ethers.formatUnits(exchangeRateStored, 18));
            let vToken_decimals = await vToken.decimals();
            console.log("vToken_decimals", vToken_decimals);
            let vToken_balance = await vToken.balanceOf(venusERC4626.target);
            console.log("vToken_balance", ethers.formatUnits(vToken_balance, vToken_decimals));
            let totalAssets = await venusERC4626.totalAssets();
            console.log("totalAssets", ethers.formatUnits(totalAssets, 18));

            // 3 owner redeem all shares to bob
            let max_redeem = await venusERC4626.maxRedeem(owner.address);
            console.log("max_redeem", ethers.formatUnits(max_redeem, 18));
            expect(max_redeem).to.equal(shares2);
            let previewRedeem = await venusERC4626.previewRedeem(max_redeem);
            console.log("previewRedeem", ethers.formatUnits(previewRedeem, 18));
            await venusERC4626.redeem(max_redeem, bob.address, owner.address);
            // check bob's balance
            let balance_bob = await bnbx.balanceOf(bob.address);
            console.log("balance_bob", ethers.formatUnits(balance_bob, 18));
            expect(balance_bob).to.equal(previewRedeem);

            let alice_max_redeem = await venusERC4626.maxRedeem(alice.address);
            console.log("alice_max_redeem", ethers.formatUnits(alice_max_redeem, 18));
            expect(alice_max_redeem).to.equal(shares);
            await venusERC4626.connect(alice).redeem(alice_max_redeem, alice.address, alice.address);
            let balance_vtoken = await vToken.balanceOf(venusERC4626.target);
            console.log("balance_vtoken", ethers.formatUnits(balance_vtoken, vToken_decimals));
            expect(balance_vtoken).to.equal(0);
        });

        it("Redeem in case of redis vToken in venusERC4626", async function () {
            const { owner, alice, bob, bnbx, vToken, venusERC4626 } = await loadFixture(deployFixture);
            // 1 alice deposit 10 bnbx
            let depositAmount = ethers.parseUnits("10", 18);
            const shares = await venusERC4626.previewDeposit(depositAmount);
            expect(shares).to.equal(depositAmount);
            await expect(venusERC4626.connect(alice).deposit(depositAmount, alice.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(alice.address, alice.address, depositAmount, shares);
            expect(await venusERC4626.balanceOf(alice.address)).to.equal(shares);
            let totalAssets = await venusERC4626.totalAssets();
            console.log("totalAssets0:", ethers.formatUnits(totalAssets, 8));
            // 2 bob transfer 1 vToken to venusERC4626
            await vToken.connect(bob).transfer(venusERC4626.target, ethers.parseUnits("1", 8));
            totalAssets = await venusERC4626.totalAssets();
            console.log("totalAssets1:", ethers.formatUnits(totalAssets, 8));

            // 3 owner deposit 5 bnbx
            depositAmount = depositAmount / 2n
            let share2 = await venusERC4626.previewDeposit(depositAmount);
            console.log("share2:", ethers.formatUnits(share2, 18));
            await expect(venusERC4626.deposit(depositAmount, owner.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(owner.address, owner.address, depositAmount, share2);
            // bob transfer 1 vToken to venusERC4626
            await vToken.connect(bob).transfer(venusERC4626.target, ethers.parseUnits("1", 8));

            // 4 owner redeem all shares to bob
            let max_redeem = await venusERC4626.maxRedeem(owner.address);
            console.log("max_redeem:", ethers.formatUnits(max_redeem, 18));
            await venusERC4626.redeem(max_redeem, bob.address, owner.address);

            // 5 check bob's balance
            let balance_bob = await bnbx.balanceOf(bob.address);
            console.log("balance_bob:", ethers.formatUnits(balance_bob, 18));

            let alice_redeem = await venusERC4626.previewRedeem(shares);
            console.log("alice_redeem:", ethers.formatUnits(alice_redeem, 18));
            let max_redeem_alice = await venusERC4626.maxRedeem(alice.address);
            console.log("max_redeem_alice:", ethers.formatUnits(max_redeem_alice, 18));
            expect(max_redeem_alice).to.equal(shares);
            await venusERC4626.connect(alice).redeem(max_redeem_alice, alice.address, alice.address);
            let balance_vtoken = await vToken.balanceOf(venusERC4626.target);
            console.log("balance_vtoken:", ethers.formatUnits(balance_vtoken, 8));

        });
    });

    describe("mint and withdraw", function () {
        it("mint and withdraw in normal case", async function () {
            const { owner, alice, bob, bnbx, vToken, venusERC4626 } = await loadFixture(deployFixture);
            // alice mint 10 shares
            let shares = ethers.parseUnits("10", 18);
            let assets = await venusERC4626.previewMint(shares);
            console.log("assets", ethers.formatUnits(assets, 18));
            expect(assets).to.equal(shares);
            await expect(venusERC4626.connect(alice).mint(shares, alice.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(alice.address, alice.address, assets, shares);

            // owner mint 5 shares
            shares = shares / 2n;
            assets = await venusERC4626.previewMint(shares);
            console.log("assets", ethers.formatUnits(assets, 18));
            await expect(venusERC4626.mint(shares, owner.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(owner.address, owner.address, assets, shares);
            // alice withdraw all shares
            let maxWithdraw = await venusERC4626.maxWithdraw(alice.address);
            console.log("maxWithdraw", ethers.formatUnits(maxWithdraw, 18));
            // expect(maxWithdraw).to.equal(shares * 2n);
            let previewWithdraw_shares = await venusERC4626.previewWithdraw(maxWithdraw);
            console.log("previewWithdraw_shares", ethers.formatUnits(previewWithdraw_shares, 18));

            await expect(venusERC4626.connect(alice).withdraw(maxWithdraw, alice.address, alice.address))
                .to.emit(venusERC4626, "Withdraw")
                .withArgs(alice.address,alice.address, alice.address, maxWithdraw, previewWithdraw_shares);

            // owner withdraw all shares
            let maxWithdraw_owner = await venusERC4626.maxWithdraw(owner.address);
            console.log("maxWithdraw_owner", ethers.formatUnits(maxWithdraw_owner, 18));
            previewWithdraw_shares = await venusERC4626.previewWithdraw(maxWithdraw_owner);
            console.log("previewWithdraw_shares", ethers.formatUnits(previewWithdraw_shares, 18));
            await expect(venusERC4626.withdraw(maxWithdraw_owner, owner.address, owner.address))
                .to.emit(venusERC4626, "Withdraw")
                .withArgs(owner.address,owner.address, owner.address, maxWithdraw_owner,
previewWithdraw_shares);

            let vToken_balance = await vToken.balanceOf(venusERC4626.target);
            console.log("vToken_balance", ethers.formatUnits(vToken_balance, 8));

        });

        it("mint and withdraw in case of redis vToken in venusERC4626", async function () {
            const { owner, alice, bob, bnbx, vToken, venusERC4626 } = await loadFixture(deployFixture);
            // alice mint 10 shares
            let shares = ethers.parseUnits("10", 18);
            let assets = await venusERC4626.previewMint(shares);
            console.log("assets", ethers.formatUnits(assets, 18));
            expect(assets).to.equal(shares);
            await expect(venusERC4626.connect(alice).mint(shares, alice.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(alice.address, alice.address, assets, shares);

            // owner mint 5 shares
            shares = shares / 2n;
            assets = await venusERC4626.previewMint(shares);
            console.log("assets", ethers.formatUnits(assets, 18));
            await expect(venusERC4626.mint(shares, owner.address))
                .to.emit(venusERC4626, "Deposit")
                .withArgs(owner.address, owner.address, assets, shares);

            // bob transfer 1 vToken to venusERC4626
            await vToken.connect(bob).transfer(venusERC4626.target, ethers.parseUnits("1", 8));

            // alice withdraw all shares
            let maxWithdraw = await venusERC4626.maxWithdraw(alice.address);
            console.log("maxWithdraw", ethers.formatUnits(maxWithdraw, 18));
            // expect(maxWithdraw).to.equal(shares * 2n);
            let previewWithdraw_shares = await venusERC4626.previewWithdraw(maxWithdraw);
            console.log("previewWithdraw_shares", ethers.formatUnits(previewWithdraw_shares, 18));

            await expect(venusERC4626.connect(alice).withdraw(maxWithdraw, alice.address, alice.address))
                .to.emit(venusERC4626, "Withdraw")
                .withArgs(alice.address,alice.address, alice.address, maxWithdraw, previewWithdraw_shares);

            // owner withdraw all shares
            let maxWithdraw_owner = await venusERC4626.maxWithdraw(owner.address);
            console.log("maxWithdraw_owner", ethers.formatUnits(maxWithdraw_owner, 18));
            previewWithdraw_shares = await venusERC4626.previewWithdraw(maxWithdraw_owner);
            console.log("previewWithdraw_shares", ethers.formatUnits(previewWithdraw_shares, 18));
            await expect(venusERC4626.withdraw(maxWithdraw_owner, owner.address, owner.address))
                .to.emit(venusERC4626, "Withdraw")
                .withArgs(owner.address,owner.address, owner.address, maxWithdraw_owner,
previewWithdraw_shares);

            let vToken_balance = await vToken.balanceOf(venusERC4626.target);
            console.log("vToken_balance", ethers.formatUnits(vToken_balance, 8));

        });
    });

    describe("claimRewards", function () {
        it("claimRewards should work", async function () {
            const {comptroller,venusERC4626,alice,bnbx} = await loadFixture(deployFixture);
            let distributors = await comptroller.getRewardDistributors();
            expect(distributors.length).to.equal(loopsLimitNumber);
            const MockToken = await ethers.getContractFactory("MockToken");
            const RewardsDistributor = await ethers.getContractFactory("RewardsDistributor");
            for (let i = 0; i < distributors.length; i++) {
                let distributor = RewardsDistributor.attach(distributors[i]);
                let reward_token = await distributor.rewardToken();

                let token = MockToken.attach(reward_token);
                let balance = await token.balanceOf(ProtocolShareReserve);
                console.log("distributor", i, "reward_token:", reward_token,"balance:",
ethers.formatUnits(balance, 18));
            }

            await venusERC4626.claimRewards();
            console.log("claimRewards done");
            for (let i = 0; i < distributors.length; i++) {
                let distributor = RewardsDistributor.attach(distributors[i]);
                let reward_token = await distributor.rewardToken();
                let token = MockToken.attach(reward_token);
                let balance = await token.balanceOf(ProtocolShareReserve);
                console.log("distributor", i, "reward_token:", reward_token,"balance:",
ethers.formatUnits(balance, 18));
            }
        });
    });
});
```

6. UnitTestOutPut.md

```
  ProtocolShareReserve unit test
    ✔ Should emit event when setProtocolShareReserve (1880ms)
    ✔ Should be failed if IncomeType is not in range

  VenusERC4626Factory
    Deployment
      ✔ Should set the right owner (8055ms)
      ✔ initialize twice should fail
    setRewardRecipient
      ✔ Should set the right rewardRecipient
      ✔ Should revert if not owner (779ms)
    computeVaultAddress
      ✔ should compute the right vault address
    createERC4626Vault
      ✔ should create a new vault on expected address (324ms)
      ✔ should create a new vault on expected address with other creator

  VenusERC4626 Fork Unit Test
    Deployment
      ✔ createERC4626 again should fail (3211ms)
      ✔ initialize twice should fail
name ERC4626-Wrapped Venus Liquid Staking BNB
symbol v4626BNBx
      ✔ parameter check
    Deposit and Redeem
shares2 5.000000003073671335
exchangeRateStored 10041836134.411951134658496174
vToken_decimals 8n
vToken_balance 14.93750724
totalAssets 14.999999996067213321
max_redeem 5.000000003073671335
previewRedeem 5.000000000738185329
balance_bob 5.000000000738185329
alice_max_redeem 10.0
balance_vtoken 0.0
      ✔ Deploy and Redeem in normal case (3846ms)
totalAssets0: 99999999938.52657332
totalAssets1: 110041836069.39829031
share2: 4.543726439503182711
max_redeem: 4.543726439503182711
balance_bob: 5.313725347405215506
alice_redeem: 11.694641879103208809
max_redeem_alice: 10.0
balance_vtoken: 0.0
      ✔ Redeem in case of redis vToken in venusERC4626 (43ms)
    mint and withdraw
assets 10.0
assets 4.999999996926328667
maxWithdraw 9.999999990942632606
previewWithdraw_shares 10.0
maxWithdraw_owner 5.000000000507323184
previewWithdraw_shares 5.0
vToken_balance 0.0
      ✔ mint and withdraw in normal case
assets 10.0
assets 4.999999996926328667
maxWithdraw 10.669455733254104787
previewWithdraw_shares 10.0
maxWithdraw_owner 5.334727870230614484
previewWithdraw_shares 5.0
vToken_balance 0.0
      ✔ mint and withdraw in case of redis vToken in venusERC4626
    claimRewards
distributor 0 reward_token: 0x52F24a5e03aee338Da5fd9Df68D2b6FAe1178827 balance: 0.000014582125699264
distributor 1 reward_token: 0xc2E9d07F66A89c44062459A47a0D2Dc038E4fb16 balance: 0.000000000000000001
distributor 2 reward_token: 0x3BC5AC0dFdC871B365d159f728dd1B9A0B5481E8 balance: 0.0
distributor 3 reward_token: 0x3BC5AC0dFdC871B365d159f728dd1B9A0B5481E8 balance: 0.0
distributor 4 reward_token: 0x0782b6d8c4551B9760e74c0545a9bCD90bdc41E5 balance: 8.083300344366557448
claimRewards done
distributor 0 reward_token: 0x52F24a5e03aee338Da5fd9Df68D2b6FAe1178827 balance: 0.000014582125699264
distributor 1 reward_token: 0xc2E9d07F66A89c44062459A47a0D2Dc038E4fb16 balance: 0.000000000000000001
distributor 2 reward_token: 0x3BC5AC0dFdC871B365d159f728dd1B9A0B5481E8 balance: 0.0
distributor 3 reward_token: 0x3BC5AC0dFdC871B365d159f728dd1B9A0B5481E8 balance: 0.0
distributor 4 reward_token: 0x0782b6d8c4551B9760e74c0545a9bCD90bdc41E5 balance: 8.083300344366557448
      ✔ claimRewards should work (3113ms)
  17 passing (21s)
```

11.2 External Functions Check Points

1. VenusERC4626

File: contracts/ERC4626/VenusERC4626.sol

contract: VenusERC4626 is ERC4626Upgradeable, MaxLoopsLimitHelper

(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,address,uint256)		`initializer`	Yes		Passed	Only Once
2	claimRewards()			Yes	Yes	Passed	
3	deposit(uint256,address)		`nonReentrant`	Yes	Yes	Passed	
4	mint(uint256,address)		`nonReentrant`	Yes	Yes	Passed	
5	withdraw(uint256,address,address)		`nonReentrant`	Yes	Yes	Passed	
6	redeem(uint256,address,address)		`nonReentrant`	Yes	Yes	Passed	
7	totalAssets()	view				Passed	
8	maxDeposit(address)	view				Passed	
9	maxMint(address)	view				Passed	
10	maxWithdraw(address)	view				Passed	
11	maxRedeem(address)	view				Passed	

2. VenusERC4626Factory

File: contracts/ERC4626/VenusERC4626Factory.sol

contract: VenusERC4626Factory is Ownable2StepUpgradeable, 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,address,address,address,uint256)		`initializer`	Yes		Passed	Only Once
2	setRewardRecipient(address)			Yes		Passed	_checkAccessAllowed("setRewardRecipient(address)")
3	createERC4626(address)			Yes	Yes	Passed	
4	computeVaultAddress(address)	view				Passed	

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