Venus Automatic Allocation

AUDIT REPORT

Version 1.0.0

Serial No. 20230803000012021

Presented by Fairyproof

August 3, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Automatic Allocation project.

Audit Start Time:

July 9, 2023

Audit End Time:

August 3, 2023

Audited Code's Github Repository:

Core Market: https://github.com/VenusProtocol/venus-protocol/pull/262

Audited Code's Github Commit Number When Audit Started:

42ae2afb00b88a4de41ad0f16496665965714827

Audited Code's Github Commit Number When Audit Ended:

476dafce560b10642ea8a802523dc9d6f251369a

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/Tokens/VTokens/VToken.sol
contracts/Tokens/VTokens/VTokenInterfaces.sol
contracts/Utils/ErrorReporter.sol
```

Audited Code's Github Repository:

VBNBAdmin: https://github.com/VenusProtocol/venus-protocol/pull/289

Audited Code's Github Commit Number When Audit Started:

43fd6684c038a3b84fe029de349da16c822b2b4c

Audited Code's Github Commit Number When Audit Ended:

8e63e9a126e4a29071c4bbb4a0d5274227d0a892

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/Admin/VBNBAdmin.sol
contracts/Admin/VBNBAdminStorage.sol
```

Audited Code's Github Repository:

Isolated Market: https://github.com/VenusProtocol/isolated-pools/pull/207

Audited Code's Github Commit Number When Audit Started:

5e597b2f9b0939d091cb2c3fe40decba66f5f19a

Audited Code's Github Commit Number When Audit Ended:

bbced9f3ccef3e149aab9ba1ab21c63c5cbcb2c2

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/VToken.sol
contracts/VTokenInterfaces.sol
```

Audited Code's Github Repository:

ProtocolShareReserve: https://github.com/VenusProtocol/protocol-reserve/pull/2/commits

Audited Code's Github Commit Number When Audit Started:

535933b122e88aee133b9df8f9cbf30a432d3a39

Audited Code's Github Commit Number When Audit Ended:

5d2b7a7c4e3d76a5365ff9738902964ce2d43dad

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/ProtocolShareReserve.sol
contracts/Interfaces/*.sol
```

The goal of this audit is to review Venus's solidity implementation for its Automatic Allocation 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/

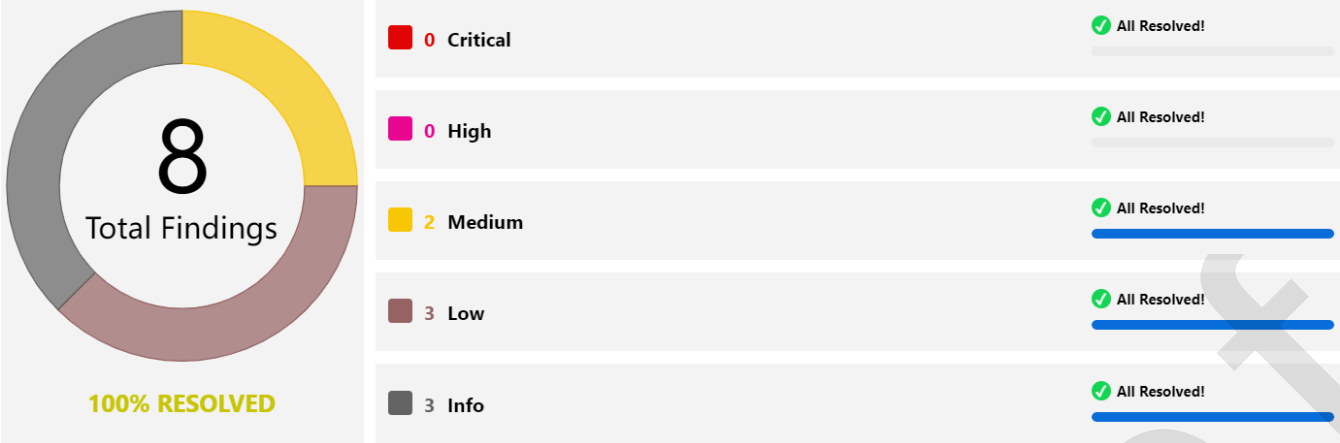
Whitepaper:https://venus.io/Whitepaper.pdf

Source Code: https://github.com/VenusProtocol/venus-protocol/pull/262

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
2023080300012021	Fairyproof Security Team	Jul 9, 2023 - Aug 3, 2023	Passed



Summary:

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

The above description is quoted from relevant documents of Venus.

04. Major functions of audited code

The audited code mainly implements a function of automatic income allocation, upgrades relevant contracts and transfers all income to `ProtocolShareReserve` for future re-allocation.

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 some issues, for more details please refer to [FP-1,FP-2,FP-3] 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 one issue, for more details please refer to [FP-5] 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 some issues, for more details please refer to [FP-4,FP-6,FP-7,FP-8] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Unnecessary `kind` Parameter	Implementation Vulnerability	Medium	✓ Fixed
FP-2	Incorrect Algorithm in `_seize`	Implementation Vulnerability	Medium	✓ Fixed
FP-3	Unable to Allocate Income Correctly	Implementation Vulnerability	Low	✓ Fixed
FP-4	Inappropriate `__gap` Value	Code Improvement	Low	✓ Fixed
FP-5	Unnecessary `reinitializer`	Contract Upgrade/Migration	Low	✓ Fixed
FP-6	Typo of `IncomeType`	Code Improvement	Info	✓ Fixed
FP-7	Confusing Variable Names	Code Improvement	Info	✓ Fixed
FP-8	Incorrect Comments	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Unnecessary `kind` Parameter

Implementation Vulnerability Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In the isolated market's `VToken.sol` contract , the `reduceReserves` function has an `IncomeType`'s parameter `kind`. This function has the following issue:

1. The income comes from `accrueInterest` but the type of the income generated by `accrueInterest` is `SPREAD` .
   Therefore, the data type of the income sent by `reduceReserves` is `SPREAD` rather than a data type defined by a parameter.

Recommendation:

Consider removing the `kind` parameter from the `reduceReserves` function and changing `_reduceReservesFresh(reduceAmount, kind);` to `_reduceReservesFresh(reduceAmount, IProtocolShareReserve.IncomeType.SPREAD);`.

Update/Status:

The Venus team has removed the parameter named `kind`.

[FP-2] Incorrect Algorithm in `_seize`

Implementation Vulnerability | Medium | ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In the isolated market's `VToken.sol` contract , with regard to `_seize`, compared with its original code, the new code simplifies the calculation of `protocolSeizeTokens` and doesn't use `liquidationIncentiveMantissa` (or uses a default value of 1e18). However, the `setProtocolSeizeShare` function uses `liquidationIncentiveMantissa` and does comparisons.

Recommendation:

Consider redesigning the algorithm.

Update/Status:

The Venus team has fixed the issue by merging develop in feature.

[FP-3] Unable to Allocate Income Correctly

Implementation Vulnerability | Low | ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `ProtocolShareReserve.sol` , the `addOrUpdateDistributionConfig` function can only add/set one `target` each time and the function doesn't check if the sum of all the `target`s percentages is 100%. In order to make sure the sum is 100%, each target's percentage should be adjusted twice. However after the first adjustment, the income will be allocated on the second adjustment. However on the second adjustment the sum of all the percentages may not be 100% and this may lead to incorrect income allocation.

Recommendation:

Consider redesigning the function by allowing multiple `target`s to be added/set and verifying that the sum of all the percentages is 100%.

Update/Status:

The Venus team has fixed the issue.

[FP-4] Inappropriate `__gap` Value

Code Improvement | Low | ✓ Fixed

Issue/Risk: Code Improvement

Description:

When using `gap`, the total number of the state variables (excluding the inherited ones) defined in a contract + gap is 50 (whenever a new state variable is defined, gap = gap -1).
In the core market's `VTokenStorage.sol` contract, it already has 24 state variables, it is better to change `uint256[50] private __gap;` to `uint256[26] private __gap;`.

In `VBNBAdminStorage.sol`, consider changing `VBNBAdminStorage`'s `uint256[49] private __gap;` to `uint256[47] private __gap;` since the contract uses 3 slots. In general this design defines 50 slots.

Recommendation:

In `VTokenStorage.sol`, considering the need for future scalability, reserving 26 slots may not be sufficient. Consider either of the two options:

1. Changing the total number of slots to 74 in the comment for `uint256[50] private __gap;` . This is for the convenience of future maintenance.
2. Inheriting `VTokenStorageBase` as follows:

```
contract VTokenStorageBase {
    bool internal _notEntered;
    ......
}
contract VTokenStorage is VTokenStorageBase {
    uint256[50] private __gap;
}
```

This way, new state variables can be defined in `VTokenStorage` and whenever a new state variable is defined `__gap` = `__gap - 1`. This will make the code maintenance easy.

Update/Status:

The Venus team has fixed the issue.

[FP-5] Unnecessary `reinitializer`

Contract Upgrade/Migration Low ✓ Fixed

Issue/Risk: Contract Upgrade/Migration

Description:

In the isolated market's `VToken.sol` contract, the modifier of `initialize` has been changed to `reinitializer(2)`.
So after an upgrade, `initialize` can be called once.

Recommendation:

Consider disallowing a second initialization to avoid core parameters to be mistakenly reset.

Update/Status:

The Venus team has changed `reinitializer(2)` to `initializer`.

[FP-6] Typo of `IncomeType`

Code Improvement Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

In the isolated market's `VTokenInterface.sol` contract, `IncomeType` is defined as follows:

```
enum IncomeType {
    LIQUIDATION,
    SPREAD
}
```

This definition has different order of elements compared with the `IncomeType` defined in `interface IProtocolShareReserve`, which is defined as follows:

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

However `vToken` uses `IProtocolShareReserve.IncomeType`. Therefore, consider removing `VTokenInterface`'s `IncomeType`

Recommendation:

Consider removing `VTokenInterface`'s `IncomeType`

Update/Status:

The Venus team has fixed the issue.

[FP-7] Confusing Variable Names

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In the core market's `VToken.sol` contract, consider changing `accessControl` to `accessControlManager` and `setAccessControl` to `setAccessControlManager`. This is to avoid confusing the names with names in `AccessControlledv8`.

Recommendation:

Consider changing confusing names.

Update/Status:

The Venus team had changed the name of `accessControl`.

[FP-8] Incorrent Comments

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In the isolated market's `VToken.sol` contract, some comments are incorrect, which don't describe their code's behavior correctly. Here are some examples:

- `setProtocolSeizeShare`'s comment, `must be less than liquidation incentive - 1` should be `must equal or  be less than liquidation incentive - 1`
- succeeded should be `suceeded`
- `increaseAllowance` and `decreaseAllowance` have a redundant `tokens`.
- In `addReserves` 's comment `fo` should be `of`.

Recommendation:

Consider updating these comments.

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. MockProtocolShareReserve.sol

```
pragma solidity 0.8.13;
import "@openzeppelin/contracts/proxy/beacon/BeaconProxy.sol";
import "@openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol";
import "./MockErc20.sol";
import "../Interfaces/IIncomeDestination.sol";
import "../Interfaces/IPrime.sol";
import "../Interfaces/IVToken.sol";
import "../Interfaces/PoolRegistryInterface.sol";
import "../Interfaces/ComptrollerInterface.sol";
import "../Interfaces/IProtocolShareReserve.sol";


contract MockProtocolShareReserve is IProtocolShareReserve {

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

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

contract MockInCome is IIncomeDestination {
    // comptroller => asset => amount;
    mapping(address => mapping(address => uint)) public assetState;

    function updateAssetsState(address comptroller, address asset) virtual external {
        assetState[comptroller][asset] = IERC20(asset).balanceOf(address(this));
    }
}
```

```
contract MockVToken is IVToken,MockERC20 {
    address public underlying;
    constructor(
        address asset,
        string memory name,
        string memory symbol,
        uint256 initialSupply
    ) MockERC20(name, symbol,initialSupply) {
        underlying = asset;
    }
}

contract MockPrime is IPrime,MockInCome {
    address[] public all_markets;
    // asset => vToken
    mapping(address => address ) private _vTokenForAsset;

    bool private flag;

    function setAllMarkets(address[] calldata markets) external {
        all_markets = markets;
    }

    function setVTokenForAsset(address[] calldata assets,address[] calldata vTokens)
external {
        for(uint i=0;i<vTokens.length;i++) {
            address vToken = vTokens[i];
            address asset = assets[i];
            _vTokenForAsset[asset] = vToken;
        }
    }

    function vTokenForAsset(address asset) external view returns (address) {
        return _vTokenForAsset[asset];
    }

    function accrueInterest(address vToken) external {
        vToken;
        if(false) {
            flag = flag;
        }
    }

    function allMarkets() external view returns (address[] memory) {
        return all_markets;
    }
}

contract MockPoolRegistry is PoolRegistryInterface {
    // comptroller => asset => vToken
    mapping(address => mapping(address => address)) private _vTokenForAsset;
```

```
    function setVTokenForAsset(
        address comptroller,
        address[] calldata assets,
        address[] calldata vTokens
    ) external {
        for(uint i=0;i<vTokens.length;i++) {
            address vToken = vTokens[i];
            address asset = assets[i];
            _vTokenForAsset[comptroller][asset] = vToken;
        }
    }

    function getVTokenForAsset(address comptroller, address asset) external view returns
(address) {
        return _vTokenForAsset[comptroller][asset];
    }
}

contract MockComptroller is ComptrollerInterface {
    uint private x;
    function isComptroller() external view returns (bool) {
        x;
        return true;
    }
}
```

2. common.js

```
const beacon_abis = [
    "function implementation() public view returns (address)",
    "function upgradeTo(address newImplementation) public"
]

const acm_abis = [
    "function isAllowedToCall(address user, string signature) external view returns (bool)"
];

const token_abis = [
    "function balanceOf(address user) external view returns(uint256)"
];

const delegator_abis = [
    "function _setImplementation(address implementation_, bool allowResign, bytes memory
becomeImplementationData) public"
]

const IncomeType = {
    SPREAD:0,
    LIQUIDATION:1
```

```
};

module.exports = {
    delegator_abis,
    beacon_abis,
    acm_abis,
    token_abis,
    IncomeType
}
```

3. core_upgrade.t.js

```
const {
    loadFixture,
    time,
    mine,
    impersonateAccount,
    stopImpersonatingAccount,
} = require("@nomicfoundation/hardhat-network-helpers");
const {deployMockContract} = require('@ethereum-waffle/mock-contract');
const { expect } = require("chai");
const { ethers } = require("hardhat");
const {old_abis,new_abis} = require("../../scripts/VBep20Delegator_state_abi");
const all_core_markets = require("../../scripts/all_core_markets");
const core_impl_to_proxy = require("../../scripts/core_markets_impl_to_proxy");
const axios = require('axios');
const {
    acm_abis,delegator_abis,token_abis,
} = require("./common");


describe("Core Pool(vAAVE) upgrade Forking Test", function () {
    const aave_address = "0xfb6115445Bff7b52FeB98650C87f44907E58f802";
    const vAAVE_address = "0x26DA28954763B92139ED49283625ceCAf52C6f94";
    const old_implement = "0x13f816511384D3534783241ddb5751c4b7a7e148";
    const time_lock = "0x939bD8d64c0A9583A7Dcea9933f7b21697ab6396";

    function sleep(ms) {
        return new Promise(resolve => setTimeout(resolve, ms));
    }

    async function deployAndLoad() {
        // get signer
        const [Owner,Alice,Bob,...users] = await ethers.getSigners();
        const MockProtocolShareReserve = await
ethers.getContractFactory("MockProtocolShareReserve");
        const protocolShareReserve = await MockProtocolShareReserve.deploy();
        const mock_acm = await deployMockContract(Owner,acm_abis);
```

```
        const Old_VToken = await
ethers.getContractFactory("contracts/Markets/core/vAAVE/VBep20Delegate.sol:VBep20Delegate")
;
        const vToken = Old_VToken.attach(vAAVE_address);
        const New_VToken = await
ethers.getContractFactory("contracts/Markets/core/latest/VBep20Delegate.sol:VBep20Delegate"
);
        const nToken = await New_VToken.deploy();
        return {
            nToken,Owner,mock_acm,vToken,New_VToken,protocolShareReserve,Alice,Bob
        };
    }

    it("Upgrade and accrueInterest", async () => {
        const {nToken,Owner,mock_acm,vToken,New_VToken,protocolShareReserve,Alice,Bob} =
await loadFixture(deployAndLoad);
        let token_contract = new ethers.Contract(aave_address,token_abis,ethers.provider);
        expect(await vToken.implementation()).equal(old_implement);
        expect(await vToken.admin()).equal(time_lock);
        await impersonateAccount(time_lock);
        let comptroller = await vToken.comptroller();
        let underlying = await vToken.underlying();
        expect(underlying).equal(aave_address);
        let signer = await ethers.getSigner(time_lock);
        const interface = new ethers.utils.Interface(delegator_abis);
        await signer.sendTransaction({
            to:vToken.address,
            data:interface.encodeFunctionData("_setImplementation",
[nToken.address,false,"0x"])
        });
        expect(await vToken.implementation()).equal(nToken.address);
        console.log();
        console.log("test set up")
        const instance = New_VToken.attach(vToken.address);
        expect(await
instance.callStatic.setProtcolShareReserve(protocolShareReserve.address)).equal(1);
        await expect(signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setProtcolShareReserve",
[protocolShareReserve.address])
        })).emit(
            instance,"NewProtocolShareReserve"
        ).withArgs(ethers.constants.AddressZero,protocolShareReserve.address);
        expect(await instance.protocolShareReserve()).equal(protocolShareReserve.address);
        await expect(signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setReduceReservesBlockDelta",[10])
        })).emit(
            instance,"NewReduceReservesBlockDelta"
        ).withArgs(0,10);
        expect(await instance.reduceReservesBlockDelta()).equal(10);
        await stopImpersonatingAccount(time_lock);
        // accrueInterest should emit UpdateAssetsState
```

```
        console.log("accrueInterest should emit UpdateAssetsState");
        console.log("The aave balance of protocolShareReserve should be zero");
        expect(await token_contract.balanceOf(protocolShareReserve.address)).equal(0);
        // after first accrueInterest()
        await expect(instance.accrueInterest()).emit(
            protocolShareReserve,"UpdateAssetsState"
        ).withArgs(comptroller,aave_address,IncomeType.SPREAD);
        console.log(await token_contract.balanceOf(protocolShareReserve.address));

        let last_number = await time.latestBlock();
        expect(await instance.reduceReservesBlockNumber()).equal(last_number);
        console.log("after five blocks");
        await mine(5,{interval:3});
        await instance.accrueInterest();
        console.log(await token_contract.balanceOf(protocolShareReserve.address));

        expect(await instance.reduceReservesBlockNumber()).equal(last_number);
        console.log("after ten blocks");
        await mine(10,{interval:3});
        await instance.accrueInterest();
        last_number = await time.latestBlock();
        expect(await instance.reduceReservesBlockNumber()).equal(last_number);
        console.log(await token_contract.balanceOf(protocolShareReserve.address));
    });

    it("_reduceReserves should distribute interest", async () => {
        const {nToken,Owner,mock_acm,vToken,New_VToken,protocolShareReserve,Alice,Bob} =
await loadFixture(deployAndLoad);
        let token_contract = new ethers.Contract(aave_address,token_abis,ethers.provider);
        await impersonateAccount(time_lock);
        let signer = await ethers.getSigner(time_lock);
        const interface = new ethers.utils.Interface(delegator_abis);
        // upgrade
        await signer.sendTransaction({
            to:vToken.address,
            data:interface.encodeFunctionData("_setImplementation",
[nToken.address,false,"0x"])
        });
        // set up
        const instance = New_VToken.attach(vToken.address);
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setProtcolShareReserve",
[protocolShareReserve.address])
        });
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setReduceReservesBlockDelta",[10])
        });
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setAccessControlManager",
[mock_acm.address])
```

```
        });
        await stopImpersonatingAccount(time_lock);
        // after first accrueInterest()
        await instance.accrueInterest();
        let last_number = await time.latestBlock();
        expect(await instance.reduceReservesBlockNumber()).equal(last_number);
        let balance_before = await token_contract.balanceOf(protocolShareReserve.address);
        console.log("balance_before:",balance_before);
        console.log("after five blocks");
        await mine(5,{interval:3});
        await instance.accrueInterest();
        expect(await instance.reduceReservesBlockNumber()).equal(last_number);
        let totalReserves = await instance.totalReserves();
        expect(totalReserves).gt(0);
        await mock_acm.mock.isAllowedToCall.returns(true);
        await instance._reduceReserves(totalReserves);
        let balance_after = await token_contract.balanceOf(protocolShareReserve.address);
        expect(balance_after.sub(balance_before)).equal(totalReserves);
        console.log("balance_after :",balance_after);
    });

    it("Checking state vars of all_markets",async () => {
        const {nToken,Owner,mock_acm,vToken,New_VToken,protocolShareReserve,Alice,Bob} =
await loadFixture(deployAndLoad);
        console.log("all_core_markets:",all_core_markets.length);
        for(let j=0;j<all_core_markets.length;j++) {
            let market = all_core_markets[j];
            let old_token = vToken.attach(market);
            let admin = await old_token.admin();
            if(admin === ethers.constants.AddressZero) {
                throw("zero admin");
            }
            console.log("index:",j,"market:",market,"admin:",admin);
            // get old vars;
            let old_vars = [];
            let old_interface = new ethers.utils.Interface(old_abis);
            for(let name of Object.keys(old_interface.functions)){
                let call = name.substring(0,name.length-2);
                let data = await old_token[call]();
                old_vars.push(data)
            }
            expect(old_vars.length).equal(old_abis.length);
            await impersonateAccount(admin);
            let signer = await ethers.getSigner(admin);
            const interface = new ethers.utils.Interface(delegator_abis);
            // upgrade
            await signer.sendTransaction({
                to:market,
                data:interface.encodeFunctionData("_setImplementation",
[nToken.address,false,"0x"])
            });
            // set up
            const instance = New_VToken.attach(market);
```

```
            await signer.sendTransaction({
                to:instance.address,
                data:instance.interface.encodeFunctionData("setProtcolShareReserve",
[protocolShareReserve.address])
            });
            await signer.sendTransaction({
                to:instance.address,
                data:instance.interface.encodeFunctionData("setReduceReservesBlockDelta",
[10])
            });
            await signer.sendTransaction({
                to:instance.address,
                data:instance.interface.encodeFunctionData("setAccessControlManager",
[mock_acm.address])
            });
            await stopImpersonatingAccount(admin);
            // differ vars;
            let new_vars = []
            for(let name of Object.keys(old_interface.functions)){
                let call = name.substring(0,name.length-2);
                let data = await old_token[call]();
                new_vars.push(data)
            }
            expect(new_vars.length).equal(old_abis.length);
            for(let i=0;i<new_vars.length;i++) {
                if(i !== old_abis.length -1) {
                    expect(old_vars[i]).equal(new_vars[i])
                } else {
                    expect(new_vars[i]).equal(nToken.address);
                }
            }
            console.log("prepare accrueInterest");
            // check extra vars
            await instance.accrueInterest();
            let new_market = new ethers.Contract(market,new_abis,ethers.provider);
            let last_number = await time.latestBlock();
            expect(await new_market.reduceReservesBlockDelta()).equal(10);
            expect(await new_market.reduceReservesBlockNumber()).equal(last_number);
            expect(await
new_market.protocolShareReserve()).equal(protocolShareReserve.address);
            expect(await new_market.accessControlManager()).equal(mock_acm.address);
            //  sleep 4 seconds to prevent timeout
            await sleep(4000);
        }
    });

    it("check interface of all implements", async () => {
        const {New_VToken} = await loadFixture(deployAndLoad);
        let new_names = Object.keys(New_VToken.interface.functions).sort();
        let all_impl = Object.keys(core_impl_to_proxy);
        const url = "https://api.bscscan.com/api?
module=contract&action=getabi&address=contract_address&apikey=apikey";
        let all_reduces = {}
```

```
        for(let i=0;i<all_impl.length;i++) {
            let extras = []
            let impl = all_impl[i];
            let query = url.replace("contract_address",impl);
            let {result} = (await axios.get(query)).data
            let interface = new ethers.utils.Interface(result);
            let sort_names = Object.keys(interface.functions).sort();
            let k = 0;
            for(let j=0;j<new_names.length;j++) {
                let new_name = new_names[j];
                let old_name = ""
                if(k < sort_names.length) {
                    old_name = sort_names[k]
                }
                if(new_name === old_name) {
                    k++
                } else {
                    extras.push(new_name)
                }
            }
            all_reduces[impl] = extras;
            await sleep(2000);
        }
        console.log(all_reduces);
    });
});
```

4. isolated_upgrade.t.js

```
const {
    loadFixture,
    time,
    mine,
    impersonateAccount,
    stopImpersonatingAccount,
} = require("@nomicfoundation/hardhat-network-helpers");
const {deployMockContract} = require('@ethereum-waffle/mock-contract');
const { expect } = require("chai");
const { ethers } = require("hardhat");
const {
    acm_abis,token_abis,beacon_abis,IncomeType
} = require("./common");


describe("Isolated Pool upgrade Forking Test", function () {
    const vUSDT = "0x5e3072305F9caE1c7A82F6Fe9E38811c74922c3B";
    const beacon_address = "0x2b8A1C539ABaC89CbF7E2Bc6987A0A38A5e660D4";
    const old_implement = "0x3E4F3F90fD01766472E654748509C6eD81e7C62c";
```

```
    const beacon_owner = "0x939bD8d64c0A9583A7Dcea9933f7b21697ab6396";  // time lock
    const old_protocolShareReserve = "0xF322942f644A996A617BD29c16bd7d231d9F35E9";

    async function deployAndLoad() {
        // get signer
        const [Owner,...users] = await ethers.getSigners();
        const MockProtocolShareReserve = await
ethers.getContractFactory("MockProtocolShareReserve");
        const protocolShareReserve = await MockProtocolShareReserve.deploy();
        // const BeaconProxy = await ethers.getContractFactory("BeaconProxy");
        const beacon = new ethers.Contract(beacon_address,beacon_abis,ethers.provider);
        const Old_VToken = await
ethers.getContractFactory("contracts/Markets/Isolated/vUSDT/VToken.sol:VToken");
        const vToken = Old_VToken.attach(vUSDT);
        const New_VToken = await
ethers.getContractFactory("contracts/Markets/Isolated/latest/VToken.sol:VToken");
        const new_implement = await New_VToken.deploy();

        return {
            new_implement,Owner,beacon,vToken,New_VToken,protocolShareReserve
        };
    }

    it("upgrade and set test", async () => {
        const {new_implement,Owner,beacon,vToken,New_VToken,protocolShareReserve} = await
loadFixture(deployAndLoad);
        expect(await beacon.implementation()).equal(old_implement);
        console.log("test upgrade")
        await impersonateAccount(beacon_owner);
        let signer = await ethers.getSigner(beacon_owner);
        await signer.sendTransaction({
            to:beacon_address,
            data:beacon.interface.encodeFunctionData("upgradeTo",[new_implement.address])
        });
        await stopImpersonatingAccount(beacon_owner);
        expect(await beacon.implementation()).equal(new_implement.address);
        console.log();

        console.log("test setProtocolShareReserve");
        const instance = New_VToken.attach(vToken.address);
        expect(await instance.protocolShareReserve()).equal(old_protocolShareReserve);
        await expect(instance.setProtocolShareReserve(protocolShareReserve.address))
            .to.revertedWith("Ownable: caller is not the owner")

        let owner = await instance.owner();
        expect(owner).equal(beacon_owner); // same as beacon_owner
        await impersonateAccount(owner);
        signer = await ethers.getSigner(owner);
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setProtocolShareReserve",
[protocolShareReserve.address])
        });
```

```
        await stopImpersonatingAccount(beacon_owner);
        expect(await instance.protocolShareReserve()).equal(protocolShareReserve.address);
        console.log()
        console.log("test setReduceReservesBlockDelta")
        await
expect(instance.setReduceReservesBlockDelta(10)).rejectedWith("Unauthorized");
        // let acm_address = await instance.accessControlManager();
        const mock_acm = await deployMockContract(Owner,acm_abis);
        // reset acm
        console.log("reset acm");
        await impersonateAccount(owner);
        signer = await ethers.getSigner(owner);
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setAccessControlManager",
[mock_acm.address])
        });
        await stopImpersonatingAccount(beacon_owner);
        expect(await instance.accessControlManager()).equal(mock_acm.address);
        // skip ts link;
        await mock_acm.mock.isAllowedToCall.returns(true);
        await expect(instance.setReduceReservesBlockDelta(10)).emit(
            instance,"NewReduceReservesBlockDelta"
        ).withArgs(0,10);

        expect(await instance.reduceReservesBlockDelta()).equal(10);
        console.log("upgrade test over");
    });

    it("reduceReserves unit test", async () => {
        // first upgrade
        const {new_implement,Owner,beacon,vToken,New_VToken,protocolShareReserve} = await
loadFixture(deployAndLoad);
        const instance = New_VToken.attach(vToken.address);
        await impersonateAccount(beacon_owner);
        let signer = await ethers.getSigner(beacon_owner);
        await signer.sendTransaction({
            to:beacon_address,
            data:beacon.interface.encodeFunctionData("upgradeTo",[new_implement.address])
        });

        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setProtocolShareReserve",
[protocolShareReserve.address])
        });

        const mock_acm = await deployMockContract(Owner,acm_abis);
        await signer.sendTransaction({
            to:instance.address,
            data:instance.interface.encodeFunctionData("setAccessControlManager",
[mock_acm.address])
        });
```

```
        await mock_acm.mock.isAllowedToCall.returns(true);
        await instance.setReduceReservesBlockDelta(10);
        await stopImpersonatingAccount(beacon_owner);

        // first call reduceReservesBlockNumber == _getBlockNumber;
        let token = await instance.underlying();

        let token_contract = new ethers.Contract(token,token_abis,ethers.provider);
        let balance = await token_contract.balanceOf(protocolShareReserve.address);
        console.log("balance before:",balance);
        let amount = await instance.totalReserves();
        console.log("totalReserves:",amount);
        expect(await instance.reduceReservesBlockNumber()).equal(0);
        await instance.reduceReserves(amount);
        let last = await time.latestBlock();
        console.log("last:",last);
        expect(await instance.reduceReservesBlockNumber()).equal(last);
        balance = await token_contract.balanceOf(protocolShareReserve.address);
        console.log("balance after reduceReserves:",balance);
        // after five block
        console.log("after five block");
        await mine(5,{interval:3});
        await instance.accrueInterest();
        amount = await instance.totalReserves();
        console.log("totalReserves:",amount);
        let comptroller = await instance.comptroller();
        await expect(instance.reduceReserves(amount)).emit(
            protocolShareReserve,"UpdateAssetsState"
        ).withArgs(comptroller,token,IncomeType.SPREAD);

        expect(await instance.reduceReservesBlockNumber()).equal(last);
        let balance_final = await token_contract.balanceOf(protocolShareReserve.address);
        console.log("balance final:",balance_final);
        expect(balance_final.sub(balance)).equal(amount);

        amount = await instance.totalReserves();
        console.log("totalReserves after final:",amount);
        console.log("after five block");
        await mine(5,{interval:3});
        await instance.accrueInterest();
        last = await time.latestBlock();
        console.log("last:",last);
        expect(await instance.reduceReservesBlockNumber()).equal(last);
        balance_final = await token_contract.balanceOf(protocolShareReserve.address);
        console.log("balance final:",balance_final);
    });
});
```

5. vBNB_admin.t.js

```
const {
    time,
    loadFixture,
    impersonateAccount,
    stopImpersonatingAccount,
} = require("@nomicfoundation/hardhat-network-helpers");
const { expect } = require("chai");
const { ethers } = require("hardhat");
const {deployMockContract} = require('@ethereum-waffle/mock-contract');
const {
    acm_abis,token_abis,
} = require("./common");


describe("VBNBAdmin Fork Test", function () {
    const vBNB_address = "0xA07c5b74C9B40447a954e1466938b865b6BBea36";
    const wbnb = "0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c";
    const UNAUTHORIZED = 1;

    async function deployAndLoadFixture() {
        // get signer
        const [Owner,Alice,Bob,...users] = await ethers.getSigners();
        const MockProtocolShareReserve = await
ethers.getContractFactory("MockProtocolShareReserve");
        const protocolShareReserve = await MockProtocolShareReserve.deploy();
        const VBNB = await
ethers.getContractFactory("contracts/Markets/core/vBNB/VBNB.sol:VBNB");
        const vBNB = VBNB.attach(vBNB_address);
        const mock_acm = await deployMockContract(Owner,acm_abis);
        // deploy Proxy Admin
        const ProxyAdmin = await ethers.getContractFactory("ProxyAdmin");
        const proxy_admin = await ProxyAdmin.deploy();
        // deploy VBNBAdmin and init
        const VBNBAdmin = await ethers.getContractFactory("VBNBAdmin");
        const vbnb_admin_impl = await VBNBAdmin.deploy();
        const TransparentUpgradeableProxy = await
ethers.getContractFactory("TransparentUpgradeableProxy");
        let vbnb_admin = await
TransparentUpgradeableProxy.deploy(vbnb_admin_impl.address,proxy_admin.address,"0x");
        let instance = VBNBAdmin.attach(vbnb_admin.address);
        await
instance.initialize(vBNB.address,protocolShareReserve.address,wbnb,mock_acm.address);
        // return;
        return {
            Owner,Alice,Bob,users,vBNB,mock_acm,wbnb,instance,protocolShareReserve,
        };
    }

    describe("Initial and state check", function() {
        it("Initial state should be correct", async () => {
```

```
            const {
                vBNB,protocolShareReserve,instance
            } = await loadFixture(deployAndLoadFixture);
            // check state
            expect(await instance.vBNB()).equal(vBNB.address);
            expect(await instance.WBNB()).equal(wbnb);
            expect(await
instance.protocolShareReserve()).equal(protocolShareReserve.address);
        });

        it("Init twice should be failed", async () => {
            const {
                mock_acm,vBNB,protocolShareReserve,instance
            } = await loadFixture(deployAndLoadFixture);

            await expect(instance.initialize(
                vBNB.address,
                protocolShareReserve.address,
                wbnb,
                mock_acm.address
            )).to.revertedWith("Initializable: contract is already initialized");
        });
    });


    describe("setProtocolShareReserve test", function(){
        it("Must call by setProtocolShareReserve(address)", async () => {
            const {instance,Bob,mock_acm} = await loadFixture(deployAndLoadFixture);
            await mock_acm.mock.isAllowedToCall.returns(false);
            await
expect(instance.setProtocolShareReserve(Bob.address)).rejectedWith("Unauthorized");
        });

        it("Must not be zero address", async () => {
            const {instance,mock_acm} = await loadFixture(deployAndLoadFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await
expect(instance.setProtocolShareReserve(ethers.constants.AddressZero)).revertedWith("PSR
address invalid");
        });

        it("Should change state and emit event", async () => {
            const {instance,protocolShareReserve,Bob,Alice,mock_acm} = await
loadFixture(deployAndLoadFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await expect(instance.setProtocolShareReserve(Bob.address)).emit(
                instance,"ProtocolShareReserveUpdated"
            ).withArgs(protocolShareReserve.address,Bob.address);
            expect(await instance.protocolShareReserve()).equal(Bob.address);

            // set again
            await expect(instance.setProtocolShareReserve(Alice.address)).emit(
                instance,"ProtocolShareReserveUpdated"
```

```
            ).withArgs(Bob.address,Alice.address);
            expect(await instance.protocolShareReserve()).equal(Alice.address);
        });
    });


    describe("acceptVBNBAdmin test", function() {
        it("only admin of vBNB can change amin ship", async () => {
            const {instance,vBNB} = await loadFixture(deployAndLoadFixture);
            expect(await
vBNB.callStatic._setPendingAdmin(instance.address)).equal(UNAUTHORIZED);
        });

        it("acceptVBNBAdmin should change the admin of vBNB", async () => {
            const {instance,Alice,vBNB} = await loadFixture(deployAndLoadFixture);
            let old_admin = await vBNB.admin();
            expect(await vBNB.pendingAdmin()).equal(ethers.constants.AddressZero);
            // change pendingAdmin
            await impersonateAccount(old_admin);
            let signer = await ethers.getSigner(old_admin);
            // send gas fee
            await Alice.sendTransaction({
                to:signer.address,
                value:ethers.constants.WeiPerEther
            });
            await vBNB.connect(signer)._setPendingAdmin(instance.address);
            await stopImpersonatingAccount(old_admin);
            // check state
            expect(await vBNB.admin()).equal(old_admin);
            expect(await vBNB.pendingAdmin()).equal(instance.address);
            // acceptVBNBAdmin
            await expect(instance.connect(Alice).acceptVBNBAdmin()).revertedWith("only
owner can accept admin");
            await instance.acceptVBNBAdmin();
            expect(await vBNB.admin()).equal(instance.address);
            expect(await vBNB.pendingAdmin()).equal(ethers.constants.AddressZero);
        });
    });


    describe("fallback test", function() {
        it("Must owner can call fallback", async () => {
            const {instance,Alice} = await loadFixture(deployAndLoadFixture);
            let tx = {
                to:instance.address,
                data:0x00000011
            };
            await expect(Alice.sendTransaction(tx)).revertedWith("only owner can call vBNB
admin functions");
        });

        it("VBNBAdmin can call vBNB after accept admin", async () => {
            const {instance,vBNB,Owner,Alice} = await loadFixture(deployAndLoadFixture);
```

```
            let data = vBNB.interface.encodeFunctionData("_setPendingAdmin",
[Alice.address]);
            let tx = {
                to:instance.address,
                data:data,
            };
            // before change admin;
            await Owner.sendTransaction(tx);
            expect(await vBNB.pendingAdmin()).equal(ethers.constants.AddressZero);
            // change admin
            let old_admin = await vBNB.admin();
            await impersonateAccount(old_admin);
            let signer = await ethers.getSigner(old_admin);
            await Alice.sendTransaction({
                to:signer.address,
                value:ethers.constants.WeiPerEther
            });
            await vBNB.connect(signer)._setPendingAdmin(instance.address);
            await stopImpersonatingAccount(old_admin);
            await instance.acceptVBNBAdmin();
            // call vBNB admin functions
            await Owner.sendTransaction(tx);
            expect(await vBNB.pendingAdmin()).equal(Alice.address);
        });
    });


    describe("receive test", function() {
        const value = ethers.utils.parseEther("1.0");
        it("Only vBNB can send bnb to this contract", async () => {
            const {instance,Owner,vBNB} = await loadFixture(deployAndLoadFixture);
            let tx = {
                to:instance.address,
                value:value
            };
            await expect(Owner.sendTransaction(tx)).revertedWith("only vBNB can send BNB to
this contract");
            let balance = await ethers.provider.getBalance(vBNB.address);
            expect(balance).gt(value);
            // mock call by vBNB
            await impersonateAccount(vBNB.address);
            const signer = await ethers.getSigner(vBNB.address);
            // send transaction
            await signer.sendTransaction(tx);
            await stopImpersonatingAccount(vBNB.address);
            // check balance
            expect(await ethers.provider.getBalance(instance.address)).equal(value);
        });
    });


    describe("reduceReserves test", async () => {
```

```
        it("only the admin of vBNB can call _reduceReserves", async () => {
            const {instance,vBNB} = await loadFixture(deployAndLoadFixture);
            let totalReserves = await vBNB.totalReserves();
            await
expect(instance.reduceReserves(totalReserves)).revertedWith("reduceReserves failed");
        });

        it("call should transfer wbnb to share reserve", async () => {
            const {instance,Owner,vBNB,wbnb,protocolShareReserve} = await
loadFixture(deployAndLoadFixture);
            // change admin
            let old_admin = await vBNB.admin();
            await impersonateAccount(old_admin);
            let signer = await ethers.getSigner(old_admin);
            await Owner.sendTransaction({
                to:signer.address,
                value:ethers.constants.WeiPerEther
            });
            await vBNB.connect(signer)._setPendingAdmin(instance.address);
            await stopImpersonatingAccount(old_admin);
            await instance.acceptVBNBAdmin();
            // reduceReserves
            let totalReserves = await vBNB.totalReserves();
            expect(totalReserves).gt(0);
            await expect(instance.reduceReserves(totalReserves)).emit(
                instance,"ReservesReduced"
            ).withArgs(totalReserves);
            let wbnb_contract = new ethers.Contract(wbnb,token_abis,ethers.provider);
            expect(await
wbnb_contract.balanceOf(protocolShareReserve.address)).equal(totalReserves);
        });
    });
});
```

6. ProtocolShareReserve.t.js

```
const {
    loadFixture,
} = require("@nomicfoundation/hardhat-network-helpers");
const {deployMockContract} = require('@ethereum-waffle/mock-contract');
const { expect } = require("chai");
const { ethers } = require("hardhat");
const {acm_abis} = require("../fork/common");

describe("ProtocolShareReserve Unit Test", function () {
    const Schema  = {
        DEFAULT:0,
        SPREAD_PRIME_CORE:1
    };
```

```
    const IncomeType = {
        SPREAD:0,
        LIQUIDATION:1
    };

    const MAX_PERCENT = 100;

    async function deployFixture() {
        // get signer
        const [Owner,Alice,Bob,...users] = await ethers.getSigners();
        // deploy wbnb
        const WETH9 = await ethers.getContractFactory("WETH9");
        const wbnb = await WETH9.deploy();
        // deploy MockERC20
        const MockERC20 = await ethers.getContractFactory("MockERC20");
        const venus = await MockERC20.deploy("Venus","Venus",100000000);
        const cake = await MockERC20.deploy("Cake","Cake",100000000);
        // deploy mock income
        const MockInCome = await ethers.getContractFactory("MockInCome");
        const inComeA = await MockInCome.deploy();
        const inComeB = await MockInCome.deploy();
        // deploy MockPrime
        const MockPrime = await ethers.getContractFactory("MockPrime");
        const prime = await MockPrime.deploy();
        // deploy mock market
        const MockVToken = await ethers.getContractFactory("MockVToken");
        const vBNB = await MockVToken.deploy(wbnb.address,"vBNB","vBNB",100000000);
        const vVenus = await MockVToken.deploy(venus.address,"vVenus","vVenus",100000000);
        // set prime
        await prime.setAllMarkets([vBNB.address,vVenus.address]);
        await prime.setVTokenForAsset([wbnb.address,venus.address],
[vBNB.address,vVenus.address]);
        expect(await prime.vTokenForAsset(wbnb.address)).equal(vBNB.address);
        expect(await prime.vTokenForAsset(venus.address)).equal(vVenus.address);
        // deploy MockComptroller
        const MockComptroller = await ethers.getContractFactory("MockComptroller");
        const comptroller = await MockComptroller.deploy();
        const core_comptroller = await MockComptroller.deploy();
        // deploy pool Register
        const MockPoolRegistry = await ethers.getContractFactory("MockPoolRegistry");
        const poolRegistry = await MockPoolRegistry.deploy();
        await poolRegistry.setVTokenForAsset(
            comptroller.address,
            [wbnb.address,venus.address],
            [vBNB.address,vVenus.address]
        );
        // deploy Proxy Admin
        const ProxyAdmin = await ethers.getContractFactory("ProxyAdmin");
        const proxy_admin = await ProxyAdmin.deploy();
        const TransparentUpgradeableProxy = await
ethers.getContractFactory("TransparentUpgradeableProxy");
        // deploy mock access control manager
        const mock_acm = await deployMockContract(Owner,acm_abis);
```

```
        // deploy pool share reserve
        const ProtocolShareReserve = await
ethers.getContractFactory("ProtocolShareReserve");
        const share_reserve_impl = await
ProtocolShareReserve.deploy(core_comptroller.address,wbnb.address,vBNB.address);
        const share_reserve = await
TransparentUpgradeableProxy.deploy(share_reserve_impl.address,proxy_admin.address,"0x");
        const instance = ProtocolShareReserve.attach(share_reserve.address);
        // init
        await instance.initialize(mock_acm.address);

        return {
            Owner,Alice,Bob,users,instance,poolRegistry,prime,cake,
            wbnb,vBNB,vVenus,venus,comptroller,core_comptroller,
            mock_acm,inComeA,inComeB
        }
    }

    describe("Check initial state", async () => {
        it("State vars should are the initial params or zero value", async () => {
            const {instance,core_comptroller} = await loadFixture(deployFixture);
            expect(await instance.prime()).equal(ethers.constants.Zero);
            expect(await instance.poolRegistry()).equal(ethers.constants.Zero);
            expect(await instance.CORE_POOL_COMPTROLLER()).equal(core_comptroller.address);
        });

        it("Init twice should be failed", async () => {
            const {mock_acm,instance} = await loadFixture(deployFixture);
            await expect(instance.initialize( mock_acm.address))
                .revertedWith("Initializable: contract is already initialized");
        });
    });


    describe("setPoolRegistry test", function() {
        it("Must call by setPoolRegistry(address)", async () => {
            const {instance,mock_acm,poolRegistry} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(false);
            await
expect(instance.setPoolRegistry(poolRegistry.address)).rejectedWith("Unauthorized");;
        });

        it("Must not be zero address", async () => {
            const {instance,mock_acm} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await
expect(instance.setPoolRegistry(ethers.constants.AddressZero)).revertedWith("ProtocolShareR
eserve: Pool registry address invalid");
        });

        it("Should change state and emit event", async () => {
            const {instance,poolRegistry,Alice,mock_acm} = await
loadFixture(deployFixture);
```

```
            await mock_acm.mock.isAllowedToCall.returns(true);
            await expect(instance.setPoolRegistry(poolRegistry.address)).emit(
                instance,"PoolRegistryUpdated"
            ).withArgs(ethers.constants.AddressZero,poolRegistry.address);
            expect(await instance.poolRegistry()).equal(poolRegistry.address);

            // set again
            await expect(instance.setPoolRegistry(Alice.address)).emit(
                instance,"PoolRegistryUpdated"
            ).withArgs(poolRegistry.address,Alice.address);
            expect(await instance.poolRegistry()).equal(Alice.address);
        });
    });


    describe("setPrime test", function() {
        it("Must call by setPrime(address)", async () => {
            const {instance,prime,mock_acm,Alice} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(false);
            await
expect(instance.connect(Alice).setPrime(prime.address)).rejectedWith("Unauthorized");;
        });

        it("Must not be zero address", async () => {
            const {instance,mock_acm} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await
expect(instance.setPrime(ethers.constants.AddressZero)).revertedWith("ProtocolShareReserve:
Prime address invalid");
        });

        it("Should change state and emit event", async () => {
            const {instance,prime,Alice,mock_acm} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await expect(instance.setPrime(prime.address)).emit(
                instance,"PrimeUpdated"
            ).withArgs(ethers.constants.AddressZero,prime.address);
            expect(await instance.prime()).equal(prime.address);

            // set again
            await expect(instance.setPrime(Alice.address)).emit(
                instance,"PrimeUpdated"
            ).withArgs(prime.address,Alice.address);
            expect(await instance.prime()).equal(Alice.address);
        });
    });

    describe("addOrUpdateDistributionConfigs test", function() {
        it("Must call by operator", async () => {
            const {instance,inComeA,mock_acm} = await loadFixture(deployFixture);
            let config = {
                schema:Schema.DEFAULT,
                percentage:7,
```

```
                destination:inComeA.address
            }
            await mock_acm.mock.isAllowedToCall.returns(false);
            await expect(instance.addOrUpdateDistributionConfigs([config]))
                .rejectedWith("Unauthorized");
        });

        it("Destination must not be zero address", async () => {
            const {instance,prime,mock_acm} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPrime(prime.address);
            let config = {
                schema:Schema.DEFAULT,
                percentage:7,
                destination:ethers.constants.AddressZero
            }
            await expect(instance.addOrUpdateDistributionConfigs([config]))
                .revertedWith("ProtocolShareReserve: Destination address invalid");
        });

        it("Total percents must be zero or MAX_PERCENT", async () => {
            const {instance,mock_acm,prime,inComeA,inComeB} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPrime(prime.address);
            let config0 = {
                schema:Schema.DEFAULT,
                percentage:30,
                destination:inComeA.address
            };
            let config1 = {
                schema:Schema.DEFAULT,
                percentage:71,
                destination:inComeB.address
            };
            let config2 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:71,
                destination:inComeB.address
            };
            await
expect(instance.addOrUpdateDistributionConfigs([config0,config1,config2])).revertedWith(
                "ProtocolShareReserve: Total Percentage must be 0 or 100"
            );

            await expect(instance.addOrUpdateDistributionConfigs([config2])).revertedWith(
                "ProtocolShareReserve: Total Percentage must be 0 or 100"
            );
        });

        it("Add and update test", async () => {
            const {instance,prime,mock_acm,inComeA,inComeB} = await
loadFixture(deployFixture);
```

```
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPrime(prime.address);
            let config0 = {
                schema:Schema.DEFAULT,
                percentage:30,
                destination:inComeA.address
            };
            let config1 = {
                schema:Schema.DEFAULT,
                percentage:70,
                destination:inComeB.address
            };
            // add
            await expect(instance.addOrUpdateDistributionConfigs([config0,config1])).emit(
                instance,"DistributionConfigAdded"
            ).withArgs(inComeA.address,30,Schema.DEFAULT);
            // check state
            expect(await instance.totalDistributions()).equal(2);
            let { schema,percentage,destination } =  await instance.distributionTargets(0);
            expect(schema).equal(config0.schema);
            expect(percentage).equal(config0.percentage);
            expect(destination).equal(config0.destination);
            ({ schema,percentage,destination } =  await instance.distributionTargets(1));
            expect(schema).equal(config1.schema);
            expect(percentage).equal(config1.percentage);
            expect(destination).equal(config1.destination);
            // update
            config0.percentage = 40;
            config1.percentage = 70;
            await
expect(instance.addOrUpdateDistributionConfigs([config0,config1])).revertedWith(
                "ProtocolShareReserve: Total Percentage must be 0 or 100"
            );
            config1.percentage = 60;
            await expect(instance.addOrUpdateDistributionConfigs([config0,config1])).emit(
                instance,"DistributionConfigUpdated"
            ).withArgs(inComeB.address,70,60,Schema.DEFAULT);
            expect(await instance.totalDistributions()).equal(2);
            ({ schema,percentage,destination } =  await instance.distributionTargets(0));
            expect(schema).equal(config0.schema);
            expect(destination).equal(config0.destination);
            expect(percentage).equal(40);

            ({ schema,percentage,destination } =  await instance.distributionTargets(1));
            expect(schema).equal(config1.schema);
            expect(destination).equal(config1.destination);
            expect(percentage).equal(60);
        });

        it("Add all schema test", async () => {
            const {instance,prime,mock_acm,inComeA,inComeB} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
```

```
            await instance.setPrime(prime.address);
            let config0 = {
                schema:Schema.DEFAULT,
                percentage:30,
                destination:inComeA.address
            };
            let config1 = {
                schema:Schema.DEFAULT,
                percentage:70,
                destination:inComeB.address
            };
            let config2 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:20,
                destination:inComeA.address
            };
            let config3 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:80,
                destination:inComeB.address
            };
            await
instance.addOrUpdateDistributionConfigs([config0,config1,config2,config3]);
            expect(await instance.totalDistributions()).equal(4);
        });
    });

    describe("updateAssetsState test", function () {
        it("Incorrect comptroller should be failed", async () => {
            const {instance,mock_acm,poolRegistry,wbnb} = await loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);

 expect(instance.updateAssetsState(poolRegistry.address,wbnb.address,IncomeType.SPREAD)).re
vertedWithPanic;
        });


        it("Asset must not be zero",async () => {
            const {instance,mock_acm,poolRegistry,comptroller} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await expect(instance.updateAssetsState(
                comptroller.address,
                ethers.constants.AddressZero,
                IncomeType.SPREAD
            )).revertedWith(
                "ProtocolShareReserve: Asset address invalid"
            );
        });

        it("Pool must support the asset", async () => {
```

```
            const {instance,mock_acm,comptroller,vBNB,poolRegistry} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await expect(instance.updateAssetsState(
                comptroller.address,
                vBNB.address,
                IncomeType.SPREAD
            )).revertedWith(
                "ProtocolShareReserve: The pool doesn't support the asset"
            );
        });;

        it("Core comptroller can add asset without vToken", async () => {
            const {instance,mock_acm,core_comptroller,cake,prime,poolRegistry} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await instance.setPrime(prime.address);
            await cake.transfer(instance.address,1000);
            await
expect(instance.updateAssetsState(core_comptroller.address,cake.address,IncomeType.SPREAD))
                .emit(instance,"AssetsReservesUpdated")

.withArgs(core_comptroller.address,cake.address,1000,IncomeType.SPREAD,Schema.DEFAULT);
            expect(await instance.assetsReserves(
                core_comptroller.address,
                cake.address,
                Schema.DEFAULT
            )).equal(1000);
            expect(await instance.totalAssetReserve(cake.address)).equal(1000);
        });

        it("add asset of Schema.SPREAD_PRIME_CORE", async () => {
            const {instance,mock_acm,core_comptroller,venus,prime,poolRegistry} = await
loadFixture(deployFixture);
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await instance.setPrime(prime.address);
            await venus.transfer(instance.address,10000);
            await
expect(instance.updateAssetsState(core_comptroller.address,venus.address,IncomeType.SPREAD)
)
                .emit(instance,"AssetsReservesUpdated")

.withArgs(core_comptroller.address,venus.address,10000,IncomeType.SPREAD,Schema.SPREAD_PRIM
E_CORE);
            expect(await instance.assetsReserves(
                core_comptroller.address,
                venus.address,
                Schema.SPREAD_PRIME_CORE
            )).equal(10000);
            expect(await instance.totalAssetReserve(venus.address)).equal(10000);
```

```
        });
    });


    describe("getUnreleasedFunds test", function() {
        it("can get after add config", async () => {
            const
{instance,mock_acm,core_comptroller,venus,inComeA,inComeB,prime,poolRegistry} = await
loadFixture(deployFixture);
            // first setup
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await instance.setPrime(prime.address);

            // add config
            let config0 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:30,
                destination:inComeA.address
            };

            let config1 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:70,
                destination:inComeB.address
            };
            await instance.addOrUpdateDistributionConfigs([config0,config1]);
            // update asset;
            await venus.transfer(instance.address,10000);
            await
expect(instance.updateAssetsState(core_comptroller.address,venus.address,IncomeType.SPREAD)
)
                .emit(instance,"AssetsReservesUpdated")

.withArgs(core_comptroller.address,venus.address,10000,IncomeType.SPREAD,Schema.SPREAD_PRIM
E_CORE);
            expect(await instance.assetsReserves(
                core_comptroller.address,
                venus.address,
                Schema.SPREAD_PRIME_CORE
            )).equal(10000);

            let value = await instance.getUnreleasedFunds(
                core_comptroller.address,
                Schema.SPREAD_PRIME_CORE,
                inComeA.address,
                venus.address
            );
            expect(value).equal(10000 * 30 / MAX_PERCENT);

            value = await instance.getUnreleasedFunds(
                core_comptroller.address,
                Schema.SPREAD_PRIME_CORE,
```

```
                inComeB.address,
                venus.address
            );
            expect(value).equal(10000 * 70 / MAX_PERCENT);
            expect(await venus.balanceOf(instance.address)).equal(10000);
            // add config again should emit release;
            config0.percentage = 60;
            config1.percentage = 40;
            await instance.addOrUpdateDistributionConfigs([config0,config1]);
            // check balance
            expect(await venus.balanceOf(instance.address)).equal(0);
            expect(await venus.balanceOf(inComeA.address)).equal(3000);
            expect(await venus.balanceOf(inComeB.address)).equal(7000);

            // check query result
            value = await instance.getUnreleasedFunds(
                core_comptroller.address,
                Schema.SPREAD_PRIME_CORE,
                inComeA.address,
                venus.address
            );
            expect(value).equal(0);
            value = await instance.getUnreleasedFunds(
                core_comptroller.address,
                Schema.SPREAD_PRIME_CORE,
                inComeB.address,
                venus.address
            );
            expect(value).equal(0);
        });
    });

    describe("releaseFunds test", function() {
        it("releaseFunds should transfer token and emit event", async () => {
            const
{instance,mock_acm,core_comptroller,venus,inComeA,inComeB,prime,poolRegistry} = await
loadFixture(deployFixture);
            // first setup
            await mock_acm.mock.isAllowedToCall.returns(true);
            await instance.setPoolRegistry(poolRegistry.address);
            await instance.setPrime(prime.address);
            // add config
            let config0 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:40,
                destination:inComeA.address
            };
            let config1 = {
                schema:Schema.SPREAD_PRIME_CORE,
                percentage:60,
                destination:inComeB.address
            };
```

```
            let config2 = {
                schema:Schema.DEFAULT,
                percentage:30,
                destination:inComeA.address
            };
            let config3 = {
                schema:Schema.DEFAULT,
                percentage:70,
                destination:inComeB.address
            };
            await
instance.addOrUpdateDistributionConfigs([config0,config1,config2,config3]);
            // update asset with Schema.DEFAULT;
            await venus.transfer(instance.address,10003);
            await
expect(instance.updateAssetsState(core_comptroller.address,venus.address,IncomeType.SPREAD)
)
                .emit(instance,"AssetsReservesUpdated")

.withArgs(core_comptroller.address,venus.address,10003,IncomeType.SPREAD,Schema.SPREAD_PRIM
E_CORE);
            expect(await instance.assetsReserves(
                core_comptroller.address,
                venus.address,
                Schema.SPREAD_PRIME_CORE
            )).equal(10003);

            // update asset with Schema.DEFAULT;
            await venus.transfer(instance.address,20000);
            await
expect(instance.updateAssetsState(core_comptroller.address,venus.address,IncomeType.LIQUIDA
TION))
            .emit(instance,"AssetsReservesUpdated")

.withArgs(core_comptroller.address,venus.address,20000,IncomeType.LIQUIDATION,Schema.DEFAUL
T);
            expect(await instance.assetsReserves(
                core_comptroller.address,
                venus.address,
                Schema.DEFAULT
            )).equal(20000);
            expect(await venus.balanceOf(instance.address)).equal(30003);
            expect(await venus.balanceOf(inComeA.address)).equal(0);
            expect(await venus.balanceOf(inComeB.address)).equal(0);
            let i1 = parseInt(10003 * 40 /100);
            let i2 = parseInt(10003 * 60 /100)
            let v1 = i1 + 20000 * 30 / 100;
            let v2 = i2 + 20000 * 70 / 100;

            await expect(instance.releaseFunds(core_comptroller.address,
[venus.address])).emit(
                instance,"AssetReleased"
```

```
).withArgs(inComeA.address,venus.address,Schema.SPREAD_PRIME_CORE,40,parseInt(10003 * 40
/100));
            // check balance
            expect(await venus.balanceOf(instance.address)).equal(10003 - i1 - i2);
            expect(await venus.balanceOf(inComeA.address)).equal(v1);
            expect(await venus.balanceOf(inComeB.address)).equal(v2);

            // release again
            await instance.releaseFunds(core_comptroller.address,[venus.address]);
            expect(await venus.balanceOf(instance.address)).equal(10003 - i1 - i2);
            expect(await venus.balanceOf(inComeA.address)).equal(v1);
            expect(await venus.balanceOf(inComeB.address)).equal(v2);
        });
    });
});
```

11.2 External Functions Check Points

1. ProtocolShareReserve.sol_output.md

File: contracts/ProtocolReserve/ProtocolShareReserve.sol

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

contract: ProtocolShareReserve is AccessControlledV8, IProtocolShareReserve

Index	Function	Visibility	StateMutability	Permission Check	Param Check	IsUserInterface	Unit Test
1	initialize(address)	external		OnlyOnce			Passed
2	setPoolRegistry(address)	external		_checkAccessAllowed("setPoolRegistry(address)")	Checked		Passed
3	setPrime(address)	external		_checkAccessAllowed("setPrime(address)");	Checked		Passed
4	addOrUpdateDistributionConfigs(DistributionConfig[])	external		_checkAccessAllowed("addOrUpdateDistributionConfigs(DistributionConfig)");	Checked		Passed
5	releaseFunds(address,address[])	external			NoNeed	Yes	Passed
6	getUnreleasedFunds(address,Schema,address,address)	external	view		NoNeed	Yes	Passed
7	totalDistributions()	external	view				Passed
8	updateAssetsState(address,address,IncomeType)	public			Checked	Yes	Passed

2. VBNBAdmin.sol_output.md

File: contracts/Markets/Admin/VBNBAdmin.sol

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

contract: VBNBAdmin is ReentrancyGuardUpgradeable, AccessControlledV8, VBNBAdminStorage

Index	Function	Visibility	StateMutability	Permission Check	Param Check	IsUserInterface	Unit Test
1	initialize(VTokenInterface,IProtocolShareReserve,IWBNB,address)	external		Only Once			Passed
2	setProtocolShareReserve(IProtocolShareReserve)	external		_checkAccessAllowed("setProtocolShareReserve(address)");	Checked		Passed
3	reduceReserves(uint)	external			Checked	Yes	Passed
4	acceptVBNBAdmin()	external		Only Owner			Passed
5	receive()	external	payable	Only vBNB			Passed
6	fallback(bytes)	external	payable	Only Owner			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