Block Rate Update

AUDIT REPORT

Version 1.0.0

Serial No. 20250414000022022

Presented by Fairyproof

April 14, 2025

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Block Rate Update project.

Audit Start Time:

April 9, 2025

Audit End Time:

April 14, 2025

Audited Source File's Address:

https://github.com/VenusProtocol/venus-protocol/pull/576

https://github.com/VenusProtocol/governance-contracts/pull/139

https://github.com/VenusProtocol/venus-protocol/pull/574

https://github.com/VenusProtocol/solidity-utilities/pull/32

Audited Source Files:

The source files audited include all the files as follows:

```
- contracts/Utils/CheckpointView.sol
- contracts/Governance/GovernorBravoDelegate.sol
- contracts/Governance/GovernorBravoInterfaces.sol
- contracts/Tokens/VAI/VAIController.sol
- contracts/XVSVault/XVSVault.sol
- contracts/TimeManagerV5.sol
```

The goal of this audit is to review Venus's solidity implementation for its Block Rate Update 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/venus-protocol/pull/576

https://github.com/VenusProtocol/governance-contracts/pull/139

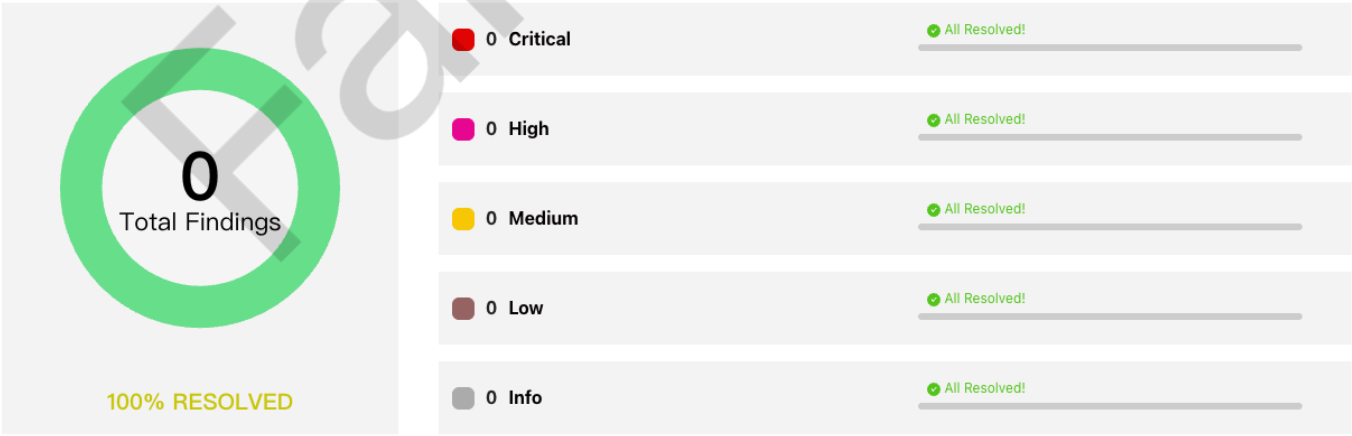
https://github.com/VenusProtocol/venus-protocol/pull/574

https://github.com/VenusProtocol/solidity-utilities/pull/32

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
2025041400022022	Fairyproof Security Team	Apr 9, 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, no issues were uncovered.

02. About Fairyproof

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

03. Introduction to Venus

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

The above description is quoted from relevant documents of Venus.

04. Major functions of audited code

This audit mainly focuses on the countermeasures for the potential reduction of block times that BNB Chain may adopt. The primary changes involve adjusting the relevant functions and adding configuration options for parameters that are tied to the block count.

- Specifically, the `GovernorBravoDelegate` contract is planned for an upgrade (its corresponding proxy being the `GovernorBravoDelegator` contract). The upgrade consolidates certain constants related to block times into a state variable (stored in `GovernorBravoDelegateStorageV3`) without affecting the original storage slot order. A new `setValidationParams` function is added to configure the new state variables, and a separate `setProposalConfigs` function is extracted to facilitate resetting proposal configurations. Note that when calling `setValidationParams`, you must ensure that the new values do not conflict with those already present in the current `proposalConfigs` (a pre-check is required).

- The `TimeManagerV5` contract primarily has two new events added and is already utilized in the Venus Protocol SDK. Due to Solidity version limitations, `TimeManagerV5` cannot be marked as an abstract contract, yet it is not meant to be used independently. This should be clearly indicated in the comments.

- The `CheckpointView` contract is used to automatically switch the data source based on a specific timestamp—e.g., reading the number of blocks per year. However, the actual use case for it has not been clearly observed so far.

- The `VAIController` contract requires an upgrade. The only modification in the contract is the change to the annual block count, and the affected function is `getVAIRepayRatePerBlock()`. Currently, the `getBlocksPerYear` function is currently hard-coded.
- The `XVSVault` contract needs an upgrade as well. It introduces a `setBlocksPerYear` function (inheriting from `TimeManagerV5`) to set the annual block count, thereby achieving self-sufficiency without relying on an external data source.

05. Coverage of issues

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

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

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

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

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

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

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

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

07. Major areas that need attention

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

- Function Implementation

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

- Access Control

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

- Token Issuance & Transfer

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

- State Update

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

- Asset Security

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

- Miscellaneous

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

08. issues by severity

- N/A

09. Issue descriptions

- N/A

10. Recommendations to enhance the overall security

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

- In the definition of the TimeManagerV5 contract, add a comment to indicate that it should not be deployed directly but must be inherited:

```
/**
 * @title TimeManagerV5
 * @dev THIS CONTRACT IS NOT MEANT TO BE DEPLOYED DIRECTLY.
 * It should only be used through inheritance by other contracts.
 */
contract TimeManagerV5 {
```

- Before the `uint256[48] private __gap` declaration in `TimeManagerV5`, add a similar comment:

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

```
 *
 * Storage layout explanation:
 * - blocksOrSecondsPerYear (uint256): Occupies slot 0 (full 32 bytes)
 * - Slot 1 has the following variables packed from right to left (low to high):
 *   - isTimeBased (bool): 1 byte at lowest position
 *   - isInitialized (bool): 1 byte
 *   - __deprecatedSlot1 (bytes8): 8 bytes
 *
 * Total used slots: 2
 * Therefore __gap size is 48 (50 - 2 = 48)
 */
```

11. Appendices

11.1 Unit Test

1. MockDataSource.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.25;

contract MockDataSource {
    uint256 public rate;

    constructor(uint256 initialRate) {
        rate = initialRate;
    }

    function setRate(uint256 newRate) external {
        rate = newRate;
    }

    function getRate() external view returns (uint256) {
        return rate;
    }
}
```

2. ScenarioTimeManagerV5.SOL

```
pragma solidity 0.5.16;

import { TimeManagerV5 } from "../TimeManagerV5.sol";

contract ScenarioTimeManagerV5 is TimeManagerV5 {
    address public owner;
```

```
    function initializeTimeManager(bool timeBased_, uint256 blocksPerYear_) external {
        _initializeTimeManager(timeBased_, blocksPerYear_);
        owner = msg.sender;
    }

    function setBlocksPerYear(uint256 blocksPerYear_) external {
        require(msg.sender == owner, "Only owner can set blocks per year");
        _setBlocksPerYear(blocksPerYear_);
    }
}
```

3. GovernorBravoDelegator.sol

```
pragma solidity ^0.5.16;
pragma experimental ABIEncoderV2;

import "./GovernorBravoInterfaces.sol";

/**
 * @title GovernorBravoDelegator
 * @author Venus
 * @notice The `GovernorBravoDelegator` contract.
 */
contract GovernorBravoDelegator is GovernorBravoDelegatorStorage, GovernorBravoEvents {
    constructor(
        address timelock_,
        address xvsVault_,
        address admin_,
        address implementation_,
        uint votingPeriod_,
        uint votingDelay_,
        uint proposalThreshold_,
        address guardian_
    ) public {
        // Admin set to msg.sender for initialization
        admin = msg.sender;

        delegateTo(
            implementation_,
            abi.encodeWithSignature(
                "initialize(address,address,uint256,uint256,uint256,address)",
                timelock_,
                xvsVault_,
                votingPeriod_,
                votingDelay_,
                proposalThreshold_,
                guardian_
            )
        );

        _setImplementation(implementation_);

        admin = admin_;
    }
```

```
    /**
     * @notice Called by the admin to update the implementation of the delegator
     * @param implementation_ The address of the new implementation for delegation
     */
    function _setImplementation(address implementation_) public {
        require(msg.sender == admin, "GovernorBravoDelegator::_setImplementation: admin only");
        require(
            implementation_ != address(0),
            "GovernorBravoDelegator::_setImplementation: invalid implementation address"
        );

        address oldImplementation = implementation;
        implementation = implementation_;

        emit NewImplementation(oldImplementation, implementation);
    }

    /**
     * @notice Internal method to delegate execution to another contract
     * @dev It returns to the external caller whatever the implementation returns or forwards reverts
     * @param callee The contract to delegatecall
     * @param data The raw data to delegatecall
     */
    function delegateTo(address callee, bytes memory data) internal {
        (bool success, bytes memory returnData) = callee.delegatecall(data);
        assembly {
            if eq(success, 0) {
                revert(add(returnData, 0x20), returndatasize)
            }
        }
    }

    /**
     * @dev Delegates execution to an implementation contract.
     * It returns to the external caller whatever the implementation returns
     * or forwards reverts.
     */
    function() external payable {
        // delegate all other functions to current implementation
        (bool success, ) = implementation.delegatecall(msg.data);

        assembly {
            let free_mem_ptr := mload(0x40)
            returndatacopy(free_mem_ptr, 0, returndatasize)

            switch success
            case 0 {
                revert(free_mem_ptr, returndatasize)
            }
            default {
                return(free_mem_ptr, returndatasize)
            }
        }
    }
}
```

4. CheckpointView.js

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

describe("CheckpointView unit test", function () {
    async function deployFixture() {
        const [owner, alice] = await ethers.getSigners();
        const MockDataSource = await ethers.getContractFactory("MockDataSource");
        const mockDataSource1 = await MockDataSource.deploy(55);
        const mockDataSource2 = await MockDataSource.deploy(100);
        let block = await ethers.provider.getBlock();
        let blockTimestamp = block.timestamp;

        const CheckpointView = await ethers.getContractFactory("CheckpointView");
        const checkpointView = await CheckpointView.deploy(
            mockDataSource1.target,
            mockDataSource2.target,
            blockTimestamp + 20
        );
        return { checkpointView, mockDataSource1, mockDataSource2,MockDataSource, owner, alice };
    }

    describe("Deploy metadata test", function () {
        it("State Variable check", async function () {
            const { checkpointView,mockDataSource1,mockDataSource2 } = await
loadFixture(deployFixture);
            const dataSource1 = await checkpointView.DATA_SOURCE_1();
            const dataSource2 = await checkpointView.DATA_SOURCE_2();
            const checkpoint_timestamp = await checkpointView.CHECKPOINT_TIMESTAMP();

           expect(dataSource1).to.equal(mockDataSource1.target);
           expect(dataSource2).to.equal(mockDataSource2.target);
           let block = await ethers.provider.getBlock();
           let blockTimestamp = block.timestamp;
           // deploy should consume 1 second
           expect(checkpoint_timestamp).to.equal(blockTimestamp + 19);
           expect(await checkpointView.currentDataSource()).to.equal(mockDataSource1.target);

        });
    });

    describe("DataSource Switch test",function () {
        it("Switch DataSource", async function () {
            const { checkpointView, mockDataSource1, mockDataSource2 } = await
loadFixture(deployFixture);
            await ethers.provider.send("evm_increaseTime", [19]);
            await ethers.provider.send("evm_mine");
            expect(await checkpointView.currentDataSource()).to.equal(mockDataSource2.target);
        });

        it("Switch DataSource with no time pass", async function () {
            const { checkpointView, mockDataSource1, mockDataSource2 } = await
loadFixture(deployFixture);
            expect(await checkpointView.currentDataSource()).to.equal(mockDataSource1.target);

            await ethers.provider.send("evm_increaseTime", [18]);
```

```
            await ethers.provider.send("evm_mine");
            expect(await checkpointView.currentDataSource()).to.equal(mockDataSource1.target);

        });

        it("Should get different data from different data sources", async function () {
            const { checkpointView, mockDataSource1,MockDataSource } = await
loadFixture(deployFixture);
            let new_checkpoint = MockDataSource.attach(checkpointView.target);

            let input = mockDataSource1.interface.encodeFunctionData("getRate", []);
            let instraction = {
                to: checkpointView.target,
                data: input,
            }
            let result = await ethers.provider.call(instraction);
            let [decoded] = mockDataSource1.interface.decodeFunctionResult("getRate", result);
            expect(decoded).to.equal(55);
            expect(await new_checkpoint.getRate()).to.equal(55);

            await ethers.provider.send("evm_increaseTime", [19]);
            await ethers.provider.send("evm_mine");
            result = await ethers.provider.call(instraction);
            ([decoded] = mockDataSource1.interface.decodeFunctionResult("getRate", result));
            expect(decoded).to.equal(100);
            expect(await new_checkpoint.getRate()).to.equal(100);
        });
    });

});
```

5. TimeManagerV5.js

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

describe("TimeManagerV5 unit test", function () {
    const SECONDS_PER_YEAR = 31_536_000;
    const BLOCKS_PER_YEAR = 21_024_000; // 1.5s block time

    async function deployFixture() {
        const [owner, alice] = await ethers.getSigners();

        const ScenarioTimeManagerV5 = await ethers.getContractFactory("ScenarioTimeManagerV5");
        const timeManagerV5 = await ScenarioTimeManagerV5.deploy();
        return { timeManagerV5,owner, alice};
    }

    describe("Deploy and Initialize",function() {
        it("Initialize with timeBased", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await
expect(timeManagerV5.initializeTimeManager(true,SECONDS_PER_YEAR)).to.revertedWith("Invalid time based
configuration");
            await expect(timeManagerV5.initializeTimeManager(true,0)).to.emit(
```

```
                timeManagerV5,
                "InitializeTimeManager"
            ).withArgs(true,SECONDS_PER_YEAR);
            expect(await timeManagerV5.isTimeBased()).to.equal(true);
            expect(await timeManagerV5.blocksOrSecondsPerYear()).to.equal(SECONDS_PER_YEAR);
            // init twice should be failed
            await expect(timeManagerV5.initializeTimeManager(true,0)).to.revertedWith("Already
initialized TimeManager");
            await
expect(timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR)).to.revertedWith("Already
initialized TimeManager");
        });

        it("Initialize with blockBased", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await expect(timeManagerV5.initializeTimeManager(false,0)).to.revertedWith("Invalid blocks
per year");
            await expect(timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR)).to.emit(
                timeManagerV5,
                "InitializeTimeManager"
            ).withArgs(false,BLOCKS_PER_YEAR);
            expect(await timeManagerV5.isTimeBased()).to.equal(false);
            expect(await timeManagerV5.blocksOrSecondsPerYear()).to.equal(BLOCKS_PER_YEAR);
            // init twice should be failed
            await expect(timeManagerV5.initializeTimeManager(true,0)).to.revertedWith("Already
initialized TimeManager");
            await
expect(timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR)).to.revertedWith("Already
initialized TimeManager");
        });

        it("length of __gap  check", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR);
            let slot0 = await ethers.provider.getStorage(timeManagerV5.target, 0);
            let value0= ethers.getBigInt(slot0);
            expect(value0).to.equal(BLOCKS_PER_YEAR);
            let slot1 = await ethers.provider.getStorage(timeManagerV5.target, 1);
            let isTimeBased = slot1.substring(64,66);
            expect(isTimeBased).to.equal("00");
            let isInitialized = slot1.substring(62,64);
            expect(isInitialized).to.equal("01");
            let deprecatedSlot1 = slot1.substring(46,62);
            expect(deprecatedSlot1).to.equal("123456789abcdef0");
            let slot2 = await ethers.provider.getStorage(timeManagerV5.target, 2);
            expect(slot2).to.equal("0x" + "00".repeat(32));
        });
    });

    describe("setBlocksPerYear",function() {
        it("Contract must be initialized", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await expect(timeManagerV5.setBlocksPerYear(0)).to.revertedWith("Only owner can set blocks
per year");
        });

        it("only owner can set blocks per year", async function () {
            const { timeManagerV5, alice } = await loadFixture(deployFixture);
            await timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR);
```

```
            await expect(timeManagerV5.connect(alice).setBlocksPerYear(0)).to.revertedWith("Only owner
can set blocks per year");
        });

        it("set can change state and emit event with blockBased", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await timeManagerV5.initializeTimeManager(false,BLOCKS_PER_YEAR);
            await expect(timeManagerV5.setBlocksPerYear(BLOCKS_PER_YEAR + 1)).to.be.emit(
                timeManagerV5,"SetBlocksPerYear"
            ).withArgs(BLOCKS_PER_YEAR,BLOCKS_PER_YEAR + 1);
            expect(await timeManagerV5.blocksOrSecondsPerYear()).to.equal(BLOCKS_PER_YEAR + 1);
            await expect(timeManagerV5.setBlocksPerYear(0)).to.be.revertedWith("Blocks per year cannot
be zero");
        });

        it("set will be failed with timeBased", async function () {
            const { timeManagerV5 } = await loadFixture(deployFixture);
            await timeManagerV5.initializeTimeManager(true,0);
            await expect(timeManagerV5.setBlocksPerYear(SECONDS_PER_YEAR)).to.revertedWith("Cannot
update for time based network");
        });
    });
});
```

6. GovernorBravoForking.js

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

/// Fork test on BNB chain at Block:48281468
describe("GovernorBravoDelegate unit test", function () {
    const DELEGATOR_ADDRESS = "0x2d56dC077072B53571b8252008C60e945108c75a";
    const ADMIN_ADDRESS = "0x1C2CAc6ec528c20800B2fe734820D87b581eAA6B";   // SAFE WALLET
    const IMPLEMENTATION_ADDRESS = "0x360ac19648efC29d2b7b70baC227C35e909272Fd";

    const old_validationParams = {
        minVotingPeriod:20 * 60 * 3,  // 3 hours , 3 secs per block ,should be changed while 1.5 secs
per block
        maxVotingPeriod:20 * 60 * 24 * 14, // About 2 weeks, 3 secs per block, should be changed while
1.5 secs per block
        minVotingDelay:1,
        maxVotingDelay:20 * 60 * 24 * 7 // About 1 week, 3 secs per block, should be changed while 1.5
secs per block
    }
    const new_validationParams = {
        minVotingPeriod:old_validationParams.minVotingPeriod * 2,
        maxVotingPeriod:old_validationParams.maxVotingPeriod * 2,
        minVotingDelay:old_validationParams.minVotingDelay * 1, // 1 sec ,not set 2 secs to avoid
exceeding existing settings
        maxVotingDelay:old_validationParams.maxVotingDelay * 2
    }

    //  note: proposal of index 1 is the same as proposal index 0
    const proposal_configs = [
```

```
        {
            votingDelay:1,
            votingPeriod:28800,
            proposalThreshold : ethers.parseEther("300000.0")
        },
        {
            votingDelay:1,
            votingPeriod:28800,
            proposalThreshold : ethers.parseEther("300000.0")
        },
        {
            votingDelay:1,
            votingPeriod:7200,
            proposalThreshold : ethers.parseEther("300000.0")
        }
    ]

    const proposal_timelocks = [
        "0x939bD8d64c0A9583A7Dcea9933f7b21697ab6396",
        "0x555ba73dB1b006F3f2C7dB7126d6e4343aDBce02",
        "0x213c446ec11e45b15a6E29C1C1b402B8897f606d"
    ]

    async function deployFixture() {
        const [owner, alice] = await ethers.getSigners();
        const GovernorBravoDelegator = await ethers.getContractFactory("GovernorBravoDelegator");
        const delegator = GovernorBravoDelegator.attach(DELEGATOR_ADDRESS);

        const GovernorBravoDelegate = await ethers.getContractFactory("GovernorBravoDelegate");

        await impersonateAccount(ADMIN_ADDRESS);
        const admin = await ethers.getSigner(ADMIN_ADDRESS);

        return { delegator, owner, alice,admin,GovernorBravoDelegate };
    }

    describe("Deploy metadata test", function () {
        it("State Variable check", async function () {
            const { delegator, admin } = await loadFixture(deployFixture);
            const implementation = await delegator.implementation();
            expect(implementation).to.equal(IMPLEMENTATION_ADDRESS);

            const adminAddress = await delegator.admin();
            expect(adminAddress).to.equal(admin.address);

            const pendingAdmin = await delegator.pendingAdmin();
            expect(pendingAdmin).to.equal(ethers.ZeroAddress);

        });

        it("Mock Set with admin", async function () {
            const { delegator, admin,GovernorBravoDelegate,alice } = await loadFixture(deployFixture);
            let delegate = GovernorBravoDelegate.attach(delegator.target);
            await expect(delegate.connect(admin)._setPendingAdmin(alice.address))
                .to.emit(delegate, "NewPendingAdmin")
                .withArgs(ethers.ZeroAddress, alice.address);
            expect(await delegate.pendingAdmin()).to.equal(alice.address);
        });
    });
```

```
describe("Upgrade and set setValidationParams/setProposalConfigs test", function () {
    it("Upgrade with reinitialize should be failed", async function () {
        const { delegator, admin,GovernorBravoDelegate } = await loadFixture(deployFixture);
        let delegate = GovernorBravoDelegate.attach(delegator.target);

        let xvsVault = await delegate.xvsVault();
        let guardian = await delegate.guardian();
        let new_implement = await GovernorBravoDelegate.deploy();
        // upgrade  with admin
        await expect(delegator.connect(admin)._setImplementation(new_implement.target)).to.emit(
            delegator,
            "NewImplementation"
        ).withArgs(IMPLEMENTATION_ADDRESS, new_implement.target);
        // reinitialize with admin should be failed
        let initial_params = [
            xvsVault,
            old_validationParams,
            proposal_configs,
            proposal_timelocks,
            guardian
        ]
        await expect(delegate.connect(admin).initialize(...initial_params)).to.be.revertedWith(
            "GovernorBravo::initialize: cannot initialize twice"
        )
    });

    it("Upgrade with setValidationParams should be success", async function () {
        const { delegator, admin,GovernorBravoDelegate } = await loadFixture(deployFixture);
        let delegate = GovernorBravoDelegate.attach(delegator.target);

        let xvsVault = await delegate.xvsVault();
        let guardian = await delegate.guardian();
        let new_implement = await GovernorBravoDelegate.deploy();
        // upgrade  with admin
        await expect(delegator.connect(admin)._setImplementation(new_implement.target)).to.emit(
            delegator,
            "NewImplementation"
        ).withArgs(IMPLEMENTATION_ADDRESS, new_implement.target);

        // first setValidationParams with admin should be success
        await expect(delegate.connect(admin).setValidationParams(old_validationParams))
            .to.emit(delegate, "SetValidationParams")
            .withArgs(
                0,
                old_validationParams.minVotingPeriod,
                0,
                old_validationParams.maxVotingPeriod,
                0,
                old_validationParams.minVotingDelay,
                0,
                old_validationParams.maxVotingDelay
            );

        // check the latest slot's value of old implementation
        for(let i=0;i<3;i++){
            let proposal_config = await delegate.proposalConfigs(i);
            let proposal_timelock = await delegate.proposalTimelocks(i);
            expect(proposal_config.votingDelay).to.equal(proposal_configs[i].votingDelay);
```

```
                expect(proposal_config.votingPeriod).to.equal(proposal_configs[i].votingPeriod);

 expect(proposal_config.proposalThreshold).to.equal(proposal_configs[i].proposalThreshold);
                expect(proposal_timelock).to.equal(proposal_timelocks[i]);
            }

            // set again;
            await expect(delegate.connect(admin).setValidationParams(new_validationParams))
                .to.emit(delegate, "SetValidationParams")
                .withArgs(
                    old_validationParams.minVotingPeriod,
                    new_validationParams.minVotingPeriod,
                    old_validationParams.maxVotingPeriod,
                    new_validationParams.maxVotingPeriod,
                    old_validationParams.minVotingDelay,
                    new_validationParams.minVotingDelay,
                    old_validationParams.maxVotingDelay,
                    new_validationParams.maxVotingDelay
                );

            // set proposal configs again
            await expect(delegate.connect(admin).setProposalConfigs(proposal_configs))
                .to.emit(delegate, "SetProposalConfigs")
                .withArgs(
                    proposal_configs[0].votingPeriod,
                    proposal_configs[0].votingDelay,
                    proposal_configs[0].proposalThreshold
                )
                .withArgs(
                    proposal_configs[1].votingPeriod,
                    proposal_configs[1].votingDelay,
                    proposal_configs[1].proposalThreshold
                )
                .withArgs(
                    proposal_configs[2].votingPeriod,
                    proposal_configs[2].votingDelay,
                    proposal_configs[2].proposalThreshold
                );

            expect(await delegate.xvsVault()).to.equal(xvsVault);
            expect(await delegate.guardian()).to.equal(guardian);
        });
    });

});
```

7. UnitTestOutput.md

```
hh test ./test/CheckpointView.js


  CheckpointView unit test
    Deploy metadata test
      ✔ State Variable check (219ms)
    DataSource Switch test
```

```
      ✔ Switch DataSource
      ✔ Switch DataSource with no time pass
      ✔ Should get different data from different data sources


  4 passing (225ms)

hh test ./test/TimeManagerV5.js


  TimeManagerV5 unit test
    Deploy and Initialize
      ✔ Initialize with timeBased (219ms)
      ✔ Initialize with blockBased
      ✔ length of __gap  check
    setBlocksPerYear
      ✔ Contract must be initialized
      ✔ only owner can set blocks per year
      ✔ set can change state and emit event with blockBased
      ✔ set will be failed with timeBased


  7 passing (234ms)


# forking block: 48281468
hh test ./test/GovernorBravoForking.js


  GovernorBravoDelegate unit test
    Deploy metadata test
      ✔ State Variable check (1653ms)
      ✔ Mock Set with admin
    Upgrade and set setValidationParams/setProposalConfigs test
      ✔ Upgrade with reinitialize should be failed
      ✔ Upgrade with setValidationParams should be success


  4 passing (2s)
```

11.2 External Functions Check Points

1. CheckpointView

File: contracts/CheckpointView.sol

contract: CheckpointView

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	fallback(bytes)				Yes	Passed	StaticCall
2	currentDataSource()	view				Passed	

2. ScenarioTimeManagerV5

File: contracts/Mock/ScenarioTimeManagerV5.sol

contract: ScenarioTimeManagerV5 is TimeManagerV5

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	initializeTimeManager(bool,uint256)			Yes		Passed	Only Once
2	setBlocksPerYear(uint256)			Yes	No	Passed	
3	getBlockNumberOrTimestamp()	view				Passed	

3. GovernorBravoDelegate

File: contracts/Governance/GovernorBravoDelegate.sol

contract: GovernorBravoDelegate is GovernorBravoDelegateStorageV3, GovernorBravoEvents

(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,ValidationParams,ProposalConfig[],TimelockInterface[],address)				False	Passed	Only Once
2	setValidationParams(ValidationParams)				False	Passed	Only Admin
3	setProposalConfigs(ProposalConfig[])				False	Passed	Only Admin
4	propose(address[],uint[],string[],bytes[],string,ProposalType)						
5	queue(uint)						
6	execute(uint)						
7	cancel(uint)						
8	getActions(uint)	view					
9	getReceipt(uint,address)	view					
10	state(uint)	view					
11	castVote(uint,uint8)						
12	castVoteWithReason(uint,uint8,string)						
13	castVoteBySig(uint,uint8,uint8,bytes32,bytes32)						
14	_setGuardian(address)						
15	_initiate(address)						
16	_setProposalMaxOperations(uint)						
17	_setPendingAdmin(address)				False	Passed	Only Admin
18	_acceptAdmin()						

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