FAIRYPROOF

Venus Prime

AUDIT REPORT

Version 1.0.0

Serial No. 20230810000012019

Presented by Fairyproof

August 10, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Prime Token project.

Audit Start Time:

July 28, 2023

Audit End Time:

August 4, 2023

Audited Code's Github Repository:

https://github.com/VenusProtocol/venus-protocol/pull/196

Audited Code's Github Commit Number When Audit Started:

7db2fa6767a8e1e97ddfa99109e057848ea507b8

Audited Code's Github Commit Number When Audit Ended:

7cc82326e0f5e07d3997a3bec659b81d6254e29e

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/Comptroller/ComptrollerStorage.sol
contracts/Comptroller/Diamond/facets/PolicyFacet.sol
contracts/Comptroller/Diamond/facets/SetterFacet.sol
contracts/XVSVault/XVSVault.sol
contracts/XVSVault/XVSVaultStorage.sol
contracts/Tokens/Prime/IPrime.sol
contracts/Tokens/Prime/Prime.sol. The key contract of the feature
contracts/Tokens/Prime/PrimeStorage.so
contracts/Tokens/Prime/libs/Scores.sol
contracts/Tokens/Prime/libs/FixedMath.sol
contracts/Tokens/Prime/libs/FixedMath0x.sol
```

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

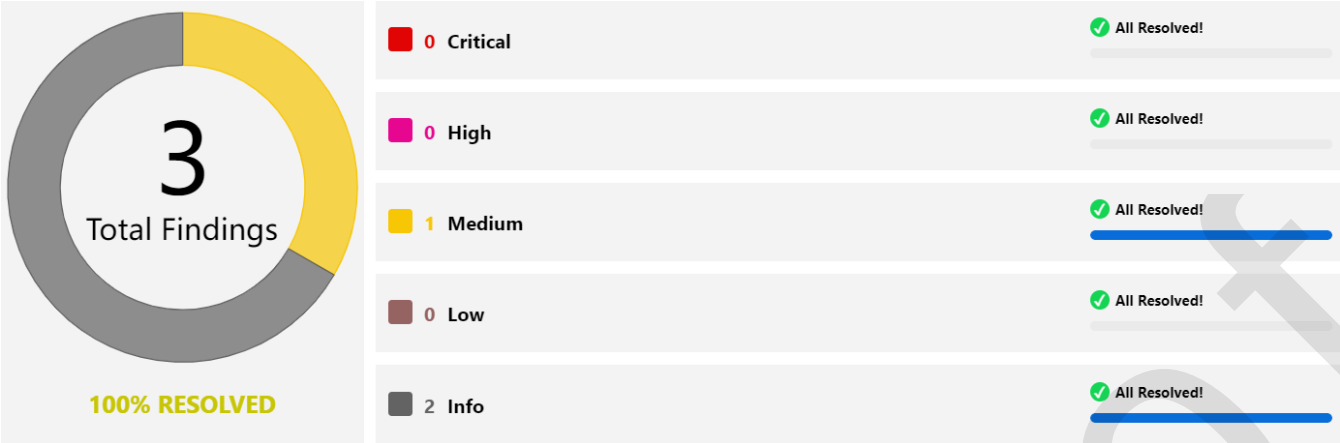
Whitepaper:https://github.com/VenusProtocol/venus-protocol-documentation/tree/main/whitepapers

Source Code: https://github.com/VenusProtocol/venus-protocol/pull/196

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Venus Prime team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023081000012019	Fairyproof Security Team	Jul 28, 2023 - Aug 4, 2023	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of medium-severity and two issues of info-severity were uncovered. The Venus Prime 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 Prime

Venus Prime is a revolutionary incentive program aimed to bolster user engagement and growth within the protocol. Venus Prime aims to enhance yields and promote $XVS staking, focusing on markets including USDT, BNB, BTC and ETH.

The above description is quoted from relevant documents of Venus Prime.

04. Major functions of audited code

The audited code mainly enhances yields for users with the Prime token and promotes $XVS staking

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 found one issue, for more details please refer to [FP-1] 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-2,FP-3] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Slot Conflict	Contract Upgrade/Migration	Medium	✓ Fixed
FP-2	Code Redundancy	Code Improvement	Info	✓ Fixed
FP-3	Code Improvement	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Slot Conflict

Contract Upgrade/Migration Medium ✓ Fixed

Issue/Risk: Contract Upgrade/Migration

Description:

In ComptrollerStorage.sol, a new variable `prime` is defined in ComptrollerV10Storage as follows:

```
contract ComptrollerV10Storage is ComptrollerV9Storage {
    /// @notice The rate at which venus is distributed to the corresponding borrow market
(per block)
    mapping(address => uint) public venusBorrowSpeeds;

    /// @notice The rate at which venus is distributed to the corresponding supply market
(per block)
    mapping(address => uint) public venusSupplySpeeds;

    /// @notice Prime token address
    IPrime public prime;
}
```

However in the contract deployed at https://bscscan.com/address/0x909dd16b24cef96c7be13065a9a0eaf8a126ffa5#code, the slot is allocated to `approvedDelegates`. Therefore a contract upgrade will lead to a conflict.

Recommendation:

Consider defining `prime` in either `ComptrollerV11Storage` or `ComptrollerV12Storage`.

```
contract ComptrollerV11Storage is ComptrollerV10Storage {
    /// @notice Whether the delegate is allowed to borrow on behalf of the borrower
    //mapping(address borrower => mapping (address delegate => bool approved)) public
approvedDelegates;
    mapping(address => mapping(address => bool)) public approvedDelegates;
    /// @notice Prime token address
    IPrime public prime;
}
```

Update/Status:

The Venus team had fixed the issue:

https://github.com/VenusProtocol/venus-protocol/pull/196/commits/df3cf7364f23466d35c7cae0d79631694293f8e6

[FP-2] Code Redundancy

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

- `interface IVToken`'s `balanceOf()` reads a state. Consider adding a `view` to its definition.
- None of the functions `borrowRatePerBlock()`, `reserveFactorMantissa()`, `totalBorrows()`, `accrueInterest()` in `interface IVToken`' is used.

```
interface IVToken {
    function borrowRatePerBlock() external view returns (uint);
```

```
    function reserveFactorMantissa() external returns (uint);

    function totalBorrows() external returns (uint);

    function accrueInterest() external returns (uint);

    function borrowBalanceStored(address account) external returns (uint);

    function exchangeRateStored() external returns (uint);

    function balanceOf(address account) external returns (uint);

    function underlying() external view returns (address);
}
```

Recommendation:

Consider removing all of them.

Update/Status:

The Venus team had fixed the issue:

https://github.com/VenusProtocol/venus-protocol/pull/196/commits/3c3e2eef0aa99d28485ef0b288f85dc328c2fefc

[FP-3] Code Improvement

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

- `function accrueInterest(address vToken)` repeatedly calls `_getUnderlying(vToken)` to read underlying.

Recommendation:

Consider adding a local variable address underlying = _getUnderlying(vToken) to read it. This will reduce gas consumption.

Update/Status:

The Venus team had fixed the issue:

https://github.com/VenusProtocol/venus-protocol/pull/196/commits/355b7a717d3c877213e5ef537cc7f957e38e4792

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. PrimeTest

```
const { ethers } = require("hardhat");
const {expect, use } = require("chai");
const { FakeContract, MockContract, smock } = require ("@defi-wonderland/smock");
const {
    loadFixture,
    time,
    mine,
  } = require("@nomicfoundation/hardhat-network-helpers");
const { convertToUnit } = require("./helpers");

use(smock.matchers);

async function deployProtocol(){
    const [owner, Alice, Bob, user3] = await ethers.getSigners();

    const xvsVaultProxy_address = "0x051100480289e704d20e9DB4804837068f3f9204";
    const XVSVault = await ethers.getContractFactory("XVSVault")
    const xvsVaultProxy =  XVSVault.attach(xvsVaultProxy_address)

    const XVS_address = "0xcF6BB5389c92Bdda8a3747Ddb454cB7a64626C63";
    let XVS = await ethers.getContractAt("XVS", XVS_address)

    const vBNB_address = "0xA07c5b74C9B40447a954e1466938b865b6BBea36";
    let vBNB = await ethers.getContractAt("contracts/Tokens/VTokens/VBep20.sol:VBep20",
vBNB_address)
```

```
    const wBNB_address = "0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c";
    let BNB = await ethers.getContractAt("BEP20Interface", wBNB_address)

    const oracleAddr = "0x6592b5DE802159F3E74B2486b091D11a8256ab8A"
    let Oracle = await
ethers.getContractAt("contracts/Oracle/ResilientOracleInterface.sol:ResilientOracleInterfac
e", wBNB_address)

    const IProtocolShareReserve = await
smock.fake("contracts/Tokens/Prime/Prime.sol:IProtocolShareReserve");
    const accessControl = await
smock.fake("contracts/Governance/IAccessControlManager.sol:IAccessControlManager");
    const PriceOracle = await smock.fake("PriceOracle");
    accessControl.isAllowedToCall.returns(true);

    const Comptroller_address = "0xf6C14D4DFE45C132822Ce28c646753C54994E59C";
    let Comptroller = await
ethers.getContractAt("contracts/Comptroller/ComptrollerInterface.sol:IComptroller",
Comptroller_address)

    const Prime = await ethers.getContractFactory("Prime")
    let prime = await Prime.deploy(wBNB_address,vBNB_address)

    const UpgradeableBeacon = await ethers.getContractFactory("UpgradeableBeacon")
    const upgradeableBeacon = await UpgradeableBeacon.deploy(prime.address)

    const BeaconProxy = await
ethers.getContractFactory("contracts/test/BeaconProxy.sol:BeaconProxy");
    const beaconProxy1 = await BeaconProxy.deploy(upgradeableBeacon.address, "0x");

    prime = await Prime.attach(beaconProxy1.address)

    await prime.initialize(
      xvsVaultProxy.address,
      XVS.address,
      0,
      1,
      2,
      accessControl.address,
      IProtocolShareReserve.address,
      Comptroller.address,
    )

    // addMarket
    const vBUSD_addr = "0x95c78222B3D6e262426483D42CfA53685A67Ab9D"
    const vBUSD = await ethers.getContractAt("contracts/Tokens/VTokens/VBep20.sol:VBep20",
vBUSD_addr);
    const BUSD_address = "0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56";
    const BUSD = await ethers.getContractAt("BEP20Interface", BUSD_address)

    // setLimit
    await prime.setLimit(10,10)
```

```
    await prime.addMarket(vBUSD_addr,convertToUnit(1,18),convertToUnit(2,18))

    // bob alice  get xvs
    const imPerson = await
ethers.getImpersonatedSigner("0xf977814e90da44bfa03b6295a0616a897441acec");
    await XVS.connect(imPerson).transfer(Alice.address, ethers.utils.parseEther("10000"))
    await XVS.connect(imPerson).transfer(Bob.address, ethers.utils.parseEther("10000"))

    // bob alice get vBUSD
    const imPersonvBusd = await
ethers.getImpersonatedSigner("0x29fec057b86ef46d240bd271837369f5715335ef");
    await vBUSD.connect(imPersonvBusd).transfer(Alice.address,
ethers.utils.parseEther("0.01"))
    await vBUSD.connect(imPersonvBusd).transfer(Bob.address,
ethers.utils.parseEther("0.01"))

    // Alice get BUSD
    const imPersonBusd = await
ethers.getImpersonatedSigner("0x28c6c06298d514db089934071355e5743bf21d60")
    await BUSD.connect(imPersonBusd).transfer(owner.address,
ethers.utils.parseEther("100000"))
    return {owner,Alice,Bob, xvsVaultProxy, XVS, vBNB, BNB, prime,
       beaconProxy1, vBUSD,BUSD, IProtocolShareReserve,Comptroller,PriceOracle,Oracle}
}

describe("Prime Test", function(){

  describe("Initial test", function(){
    it("Initial value should be equaled to contructor value", async function(){
      const {owner, xvsVaultProxy, XVS, vBNB, BNB, prime, beaconProxy1, vBUSD} = await
 loadFixture(deployProtocol);
      expect(await prime.WBNB()).to.be.equal(BNB.address);
      expect(await prime.vBNB()).to.be.equal(vBNB.address);
      const vBUSD_Market = await prime.markets(vBUSD.address)
      const vBUSD_underlying = await vBUSD.underlying()
      expect(await vBUSD_Market.supplyMultiplier).to.be.equal(convertToUnit(1,18))
      expect(await vBUSD_Market.borrowMultiplier).to.be.equal(convertToUnit(2,18))
      expect(await prime.vTokenForAsset(vBUSD_underlying)).to.be.equal(vBUSD.address)
    })
  })

  describe("mint and burn test", function(){
    it("mint test", async function(){
      const {owner, Alice,Bob, xvsVaultProxy, XVS,vBUSD, prime} = await
 loadFixture(deployProtocol);
      await prime.issue(true, [owner.address])
      const ownerToken = await prime.tokens(owner.address)
      expect(ownerToken.exists).equal(true)
      expect(ownerToken.isIrrevocable).equal(true)
      await prime.issue(false, [Alice.address, Bob.address])
      const AliceToken = await prime.tokens(Alice.address)
      expect(AliceToken.exists).equal(true)
      expect(AliceToken.isIrrevocable).equal(false)
```

```
    })

    it("burn test", async function(){
      const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD, BUSD,
IProtocolShareReserve} = await  loadFixture(deployProtocol);
      expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))


      // stake xvs
      XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
      await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
      await prime.connect(Alice).xvsUpdated(Alice.address)

      // vtoken supply/borrow
      expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
      expect(await vBUSD.balanceOf(Bob.address)).equal(ethers.utils.parseEther("0.01"))
      await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
      expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))

      // Alice state and claim xvs
      XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
      await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
      await mine(90 * 24 * 60 * 60);
      await prime.connect(Alice).claim()
      // burn
      await xvsVaultProxy.connect(Alice).requestWithdrawal(XVS.address,0,
ethers.utils.parseEther('100'))
      await prime.connect(Alice).xvsUpdated(Alice.address)
      const token = await prime.tokens(Alice.address)
      expect(token.exists).equal(false)

    })

  })

  describe("Stake  test", function(){
    it("stake test", async function(){
      const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD} = await
 loadFixture(deployProtocol);
     expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))
     expect(await XVS.balanceOf(Bob.address)).equal(ethers.utils.parseEther("10000"))
     XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
     await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
     await expect(prime.connect(Alice).claim()).rejectedWith("IneligibleToClaim");
     await prime.connect(Alice).xvsUpdated(Alice.address)
     await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
     await mine(90 * 24 * 60 * 60);
     await prime.connect(Alice).claim()
     const AlickToken = await prime.tokens(Alice.address)
     expect(AlickToken.exists).equal(true);
```

```
      expect(AlickToken.isIrrevocable).equal(false);

    })
  })

  describe("Score calculate  test", function(){
    it("Score calculate  test", async function(){
      const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD} = await
 loadFixture(deployProtocol);
     expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))
     expect(await XVS.balanceOf(Bob.address)).equal(ethers.utils.parseEther("10000"))
     // stake xvs
     XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
     await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
     // vtoken supply/borrow
     expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
     expect(await vBUSD.balanceOf(Bob.address)).equal(ethers.utils.parseEther("0.01"))
     await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
     expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))
     const exchangeRate = await vBUSD.exchangeRateStored()
     const balanceOfAccount = await vBUSD.balanceOf(Alice.address)
     const supply = ethers.BigNumber.from(exchangeRate)
                       .mul(ethers.BigNumber.from(balanceOfAccount))
                       .div(ethers.utils.parseEther("1"))

     await prime.updateScore(Alice.address, vBUSD.address)

    // Alice state xvs
     await expect(prime.connect(Alice).claim()).rejectedWith("IneligibleToClaim");
     XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
     await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
     await prime.connect(Alice).xvsUpdated(Alice.address)
     await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
     await mine(90 * 24 * 60 * 60);
     await prime.connect(Alice).claim()

    })

  })

  describe("Interest  test", function(){
    it("claim interest about one user test", async function(){
        const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD, BUSD,
IProtocolShareReserve} = await  loadFixture(deployProtocol);
        expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))
        expect(await XVS.balanceOf(Bob.address)).equal(ethers.utils.parseEther("10000"))

        // stake xvs
        XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
```

```
        await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
        await prime.connect(Alice).xvsUpdated(Alice.address)

        // vtoken supply/borrow
        expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
        expect(await vBUSD.balanceOf(Bob.address)).equal(ethers.utils.parseEther("0.01"))
        await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
        expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))

        // Alice state and claim xvs
        XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
        await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
        await prime.connect(Alice).xvsUpdated(Alice.address)
        await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
        await mine(90 * 24 * 60 * 60);
        await prime.connect(Alice).claim()
        // get interest

 IProtocolShareReserve.getUnreleasedFunds.returns(ethers.BigNumber.from("200000049999993750
0015"));

 BUSD.connect(owner).transfer(prime.address,ethers.BigNumber.from("2000000499999937500015")
);
        expect(await
BUSD.balanceOf(prime.address)).equal(ethers.BigNumber.from("2000000499999937500015"))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Alice.address)).equal(ethers.BigNumber.from("2000000499999937500015"))
        await prime.connect(Alice).claimInterest(vBUSD.address)
        expect(await
BUSD.balanceOf(Alice.address)).equal(ethers.BigNumber.from("2000000499999937500015").add(et
hers.utils.parseEther("0.001")))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Alice.address)).equal(0)

      })

      it("claim interest about more users test", async function(){
        const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD, BUSD,
IProtocolShareReserve} = await  loadFixture(deployProtocol);
        expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))
        expect(await XVS.balanceOf(Bob.address)).equal(ethers.utils.parseEther("10000"))

        // Alice stake xvs
        XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("2000"))
        await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('2000'))
        await prime.connect(Alice).xvsUpdated(Alice.address)

        // Bob stake xvs
        XVS.connect(Bob).approve(xvsVaultProxy.address, ethers.utils.parseEther("2000"))
```

```
        await xvsVaultProxy.connect(Bob).deposit(XVS.address,0,
ethers.utils.parseEther('2000'))
        await prime.connect(Bob).xvsUpdated(Bob.address)
        //Alice vtoken supply/borrow

        expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
        await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
        expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))
        //Bob vtoken supply/borrow
        expect(await vBUSD.balanceOf(Bob.address)).equal(ethers.utils.parseEther("0.01"))
        await vBUSD.connect(Bob).borrow(ethers.utils.parseEther("0.001"))
        expect(await
vBUSD.borrowBalanceStored(Bob.address)).equal(ethers.utils.parseEther("0.001"))

        // Alice state and claim xvs
        await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
        await mine(90 * 24 * 60 * 60);
        await prime.connect(Alice).claim()

        // Bob state and claim xvs
        await prime.connect(Bob).claim()
        // iussue interest

 IProtocolShareReserve.getUnreleasedFunds.returns(ethers.BigNumber.from("200000049999993750
0015").add(ethers.BigNumber.from("4000000999999876891804")));

 BUSD.connect(owner).transfer(prime.address,ethers.BigNumber.from("2000000499999939391789")
);
        expect(await
BUSD.balanceOf(prime.address)).equal(ethers.BigNumber.from("2000000499999939391789"))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Alice.address)).equal(ethers.BigNumber.from("2000000499999939391789"))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Bob.address)).equal(ethers.BigNumber.from("200000049999937500015"))

        await prime.connect(Alice).claimInterest(vBUSD.address)
        expect(await
BUSD.balanceOf(Alice.address)).equal(ethers.BigNumber.from("2000000499999939391789").add(et
hers.utils.parseEther("0.001")))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Alice.address)).equal(0)


 BUSD.connect(owner).transfer(prime.address,ethers.BigNumber.from("200000049999937500015")
);
        expect(await
BUSD.balanceOf(prime.address)).equal(ethers.BigNumber.from("200000049999937500015"))
        const BobAccrueIn = await prime.callStatic.getInterestAccrued(vBUSD.address,
Bob.address)
        const beforeBalance = await BUSD.balanceOf(Bob.address)
        await prime.connect(Bob).claimInterest(vBUSD.address)
```

```
        expect(await
BUSD.balanceOf(Bob.address)).equal(ethers.BigNumber.from(BobAccrueIn).add(ethers.BigNumber.
from(beforeBalance)))
        expect(await prime.callStatic.getInterestAccrued(vBUSD.address,
Bob.address)).equal(0)

        // Bob second stake xvs
        const BobBeforeIn = await prime.interests(vBUSD.address, Bob.address)
        XVS.connect(Bob).approve(xvsVaultProxy.address, ethers.utils.parseEther("2000"))
        await xvsVaultProxy.connect(Bob).deposit(XVS.address,0,
ethers.utils.parseEther('2000'))
        await prime.connect(Bob).xvsUpdated(Bob.address)
        const BobAfterIn = await prime.interests(vBUSD.address, Bob.address)
        expect(BobAfterIn.score).above(BobBeforeIn.score)

    })


  })

  describe("Update parameter", function(){
    it("update Alpha test", async function(){
        const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD, BUSD,
IProtocolShareReserve, PriceOracle,Oracle} = await  loadFixture(deployProtocol);
        expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))

        // stake xvs
        XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
        await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
        await prime.connect(Alice).xvsUpdated(Alice.address)

        // vtoken supply/borrow
        expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
        await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
        expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))

        // Alice state and claim xvs
        XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
        await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
        await mine(90 * 24 * 60 * 60);
        await prime.connect(Alice).claim()
        const before = await prime.interests(vBUSD.address, Alice.address)
        // update Alpha
        await prime.updateAlpha(4,5)
        await prime.connect(Alice).xvsUpdated(Alice.address)
        const after = await prime.interests(vBUSD.address, Alice.address)
        expect(before.score).above(after.score)

    })
```

```
    it("update Multipliers test", async function(){
      const {owner, Alice,Bob, xvsVaultProxy, XVS, prime,vBUSD, BUSD,
IProtocolShareReserve, PriceOracle,Oracle} = await  loadFixture(deployProtocol);
      expect(await XVS.balanceOf(Alice.address)).equal(ethers.utils.parseEther("10000"))

      // stake xvs
      XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
      await xvsVaultProxy.connect(Alice).deposit(XVS.address,0,
ethers.utils.parseEther('1000'))
      await prime.connect(Alice).xvsUpdated(Alice.address)

      // vtoken supply/borrow
      expect(await vBUSD.balanceOf(Alice.address)).equal(ethers.utils.parseEther("0.01"))
      await vBUSD.connect(Alice).borrow(ethers.utils.parseEther("0.001"))
      expect(await
vBUSD.borrowBalanceStored(Alice.address)).equal(ethers.utils.parseEther("0.001"))

      // Alice state and claim xvs
      XVS.connect(Alice).approve(xvsVaultProxy.address, ethers.utils.parseEther("1000"))
      await expect(prime.connect(Alice).claim()).rejectedWith("WaitMoreTime");
      await mine(90 * 24 * 60 * 60);
      await prime.connect(Alice).claim()
      const before = await prime.interests(vBUSD.address, Alice.address)

      // update Alpha
      await prime.updateMultipliers(vBUSD.address, convertToUnit(2,18),convertToUnit(2,18))
      await prime.connect(Alice).xvsUpdated(Alice.address)
      const after = await prime.interests(vBUSD.address, Alice.address)
      expect(after.score).above(before.score)

  })
  })
})
```

11.2 External Functions Check Points

1. Prime

File: contracts/Tokens/Prime/Prime.sol

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

contract: Prime is IIncomeDestination, AccessControlledV8, PrimeStorageV1

Index	Function	Visibility	StateMutability	Permission Check	Param Check	IsUserInterface	Unit Test
1	initialize(address,address,uint256,uint128,uint128,address,address,address)	external		initializer			Passed

Index	Function	Visibility	StateMutability	Permission Check	Param Check	IsUserInterface	Unit Test
2	updateAlpha(uint128,uint128)	external		_checkAccessAllowed("updateAlpha(uint128,uint128)")	Checked		Passed
3	updateMultipliers(address,uint256,uint256)	external		_checkAccessAllowed("updateMultipliers(address,uint256,uint256)")	Checked		Passed
4	addMarket(address,uint256,uint256)	external		_checkAccessAllowed("addMarket(address,uint256,uint256)");	Checked		Passed
5	setLimit(uint256,uint256)	external		_checkAccessAllowed("setLimit(uint256,uint256)");	Checked		Passed
6	issue(bool,address[])	external		_checkAccessAllowed("issue(bool,address[])");	Checked		Passed
7	xvsUpdated(address)	external			NoNeed	YES	Passed
8	claim()	external			NoNeed	YES	Passed
9	executeBoost(address,address)	public		markets[vToken].exists and tokens[user].exists	Checked	YES	Passed
10	updateScore(address,address)	public		markets[vToken].exists and tokens[user].exists	Checked	YES	Passed
11	accrueInterest(address)	public		markets[vToken].exists	Checked	YES	Passed
12	getInterestAccrued(address,address)	public		markets[vToken].exists	Checked	YES	Passed
13	claimInterest(address)	external			NoNeed	YES	Passed
14	updateAssetsState(address,address)	external		Only called by protocolShareReserve	Checked		Passed
15	updateScores(address[])	external		tokens[user].exists	Checked	YES	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