Venus Diamond Proxy

AUDIT REPORT

Version 1.0.0

Serial No. 20230625000120 23

Presented by Fairyproof

June 25, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Venus Diamond Upgrade project.

Audit Start Time:

June 9, 2023

Audit End Time:

June 25, 2023

Audited Code's Github Repository:

https://github.com/VenusProtocol/venus-protocol/tree/feat/diamond-proxy

Audited Code's Github Commit Number When Audit Started:

295609e440286ffc04ec9ca7c0ff9f60885bed39

Audited Code's Github Commit Number When Audit Ended:

a46b20594e86d09e73f9049205ed8cec57f39896

Audited Source Files:

The source files audited include all the files as follows:

```

- contracts/Comptroller/ComptrollerStorage.sol
- contracts/Comptroller/Diamond/Diamond.sol
- contracts/Comptroller/Diamond/facets/FacetBase.sol
- contracts/Comptroller/Diamond/facets/MarketFacet.sol
- contracts/Comptroller/Diamond/facets/PolicyFacet.sol
- contracts/Comptroller/Diamond/facets/RewardFacet.sol
- contracts/Comptroller/Diamond/facets/SetterFacet.sol
- contracts/Comptroller/Diamond/facets/XVSRewardsHelper.sol
- contracts/Comptroller/Diamond/interfaces/IDiamondCut.sol
- contracts/Comptroller/Diamond/interfaces/IMarketFacet.sol
- contracts/Comptroller/Diamond/interfaces/IPolicyFacet.sol
- contracts/Comptroller/Diamond/interfaces/IRewardFacet.sol
- contracts/Comptroller/Diamond/interfaces/ISetterFacet.sol
- contracts/Comptroller/Diamond/interfaces/IXVS.sol

```

The goal of this audit is to review Venus' solidity implementation for its Diamond Upgrade 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/

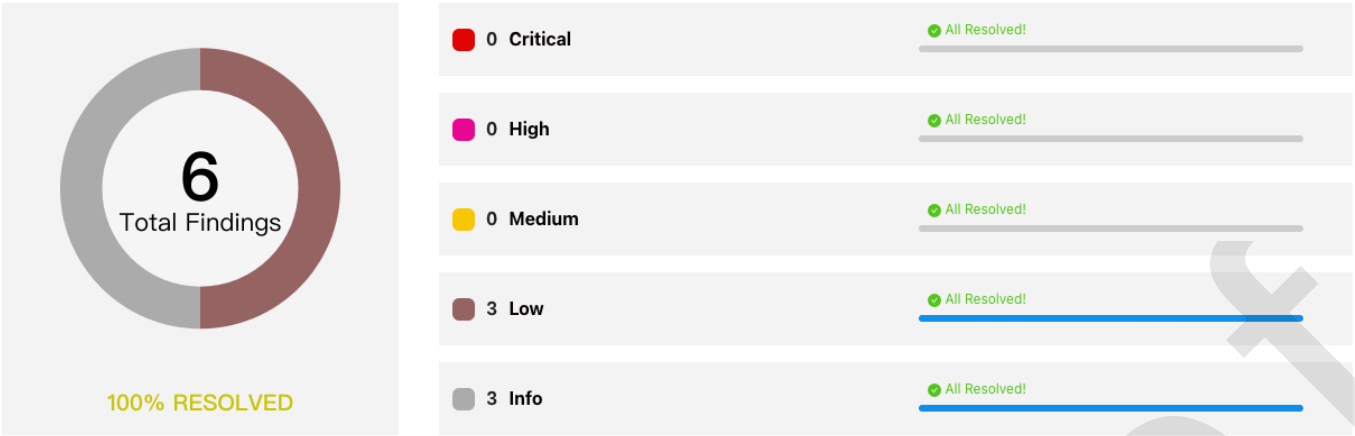
Whitepaper:https://venus.io/Whitepaper.pdf

Source Code: https://github.com/VenusProtocol/venus-protocol/tree/feat/diamond-proxy

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
2023062500012023	Fairyproof Security Team	Jun 9, 2023 - Jun 25, 2023	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, three issues of low-severity and 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 applies the Diamond protocol to re-implement the Comptroller contract and makes it scalable to future upgrades.

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-4] 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-5,FP-6] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Unnecessary `immutable` Function	Implementation Vulnerability	Low	✓ Fixed
FP-2	Missing Functions	Implementation Vulnerability	Low	✓ Fixed
FP-3	Inappropriate Event Triggering	Implementation Vulnerability	Low	✓ Fixed
FP-4	Missing Consideration for Re-entrancy Risks	Code Improvement	Info	✓ Fixed
FP-5	Inappropriate Parameter Names	Shadow Variable	Info	✓ Fixed
FP-6	Missing LoupeFacet Function	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Unnecessary `immutable` Function

Implementation Vulnerability Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Diamond.sol` , the removeFunction function has the following code `require(_facetAddress !=address(this),"LibDiamondCut: Can't remove immutable function");`.
According to EIP-2535, An immutable function is an external function that cannot be replaced or removed . However in practice, very few contracts would delegatecall their external interfaces which are publicly accessible. UniswapV3's Router contracts have
calls like address(this).delegatecall , however they are not in fallback functions.

This might cause the following two issues:

- When calling `addFunctions` , if `facetAddress` is set to this contract's address, the added function cannot be removed.
- Assume `addFunctions` is called and `facetAddress` is set to this contract's address. If the added selector points to an existing function, the function cannot be called by the default callback `function()` . If the added selector doesn't point to an existing function, when the function is called it will recursively nest itself in a loop until gas runs out.

Recommendation:

Consider removing `require(_facetAddress != address(this), "LibDiamondCut: Can't remove immutable function");` , since the contract doesn't need an immutable function.

Update/Status:

The Venus team has removed this line.

[FP-2] Missing Functions

Implementation Vulnerability Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Comptroller` , there are functions `updateDelegate` , `liquidateVAICalculateSeizeTokens` , `getAllMarkets` . None of them is defined in any `facet` contract.

Recommendation:

Consider adding a function definition for each of them.

Update/Status:

The Venus team has added these functions.

[FP-3] Inappropriate Event Triggering

Implementation Vulnerability Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `MarketFacet` , the `enterMarkets` function checks whether or not the return value is `0` and then decides whether or not to trigger the `MarketEntered` event. In `FacetBase` , the `addToMarketInternal` function always returns `0` no matter whether or not a market is newly entered.
In addition we notice that `emit MarketEntered(vToken, borrower);` has been commented out.
Therefore we assume that only when a market is newly entered this event should be triggered. If our assumption is correct we suggest to trigger events in `addToMarketInternal` . Since the event triggering is removed from `addToMarketInternal` , this event will not be triggered when calling `borrowAllowed function` in PolicyFacet.sol.

Recommendation:

Consider uncommenting the event from the `addToMarketInternal` and removing the event from `enterMarkets`

Update/Status:

The Venus team has fixed the issue.

[FP-4] Missing Consideration for Re-entrancy Risks

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `RewardFacet` , the claimVenus function has the following code snippets:

```
venusAccrued[holder] = grantXVSInternal(holder, venusAccrued[holder], shortfall,collateral);
```

The `grantXVSInternal` function interacted with the external contract `xvs` and therefore a state update would happens after the contract interaction. Although the `xvs` contract is immune to re-entrancy risks, in order to strictly confirm to the checks-effects-interactions rule, consider updating the state before the contract interaction.

Recommendation:

Consider adding a temporary variable to update the state before calling `grantXVSInternal` .

Update/Status:

The Venus team updated the state variable `venusAccrued[holder]` before calling the `grantXVSInternal()` function to prevent the reentrancy attack.

[FP-5] Inappropriate Parameter Names

Shadow Variable | Info | ✓ Fixed

Issue/Risk: Shadow Variable

Description:

In `SetterFacet` , the `_setActionsPaused` function has a `markets` parameter and it has the same name as state variable `markets` . It is confusing and the state variable will be shadowed.

Recommendation:

Consider renaming the parameter.

Update/Status:

The Venus team renamed parameter `market` to `market_` .

[FP-6] Missing LoupeFacet Function

Code Improvement | Info | ✓ Fixed

Issue/Risk: Code Improvement

Description:

The Diamond protocol, in general, has a `LoupeFacet` function to retrieve information, such as retrieving all `facet` addresses. The existing implementation didn't have such a `Facet` function.

Recommendation:

Consider adding a `Facet` function. Here is a reference:

```
contract MockLoupeFacet is MockStorageBase {

    function getFacetFunctionSelectors(address _facet) external view returns (bytes4[] memory _facetFunctionSelectors) {
        _facetFunctionSelectors =  facetFunctionSelectors[_facet].functionSelectors;
    }

    function getFacetPosition(address _facet) external view returns(uint256) {
        return facetFunctionSelectors[_facet].facetAddressPosition;
    }

    function getAllFacetAddresses() external view returns (address[] memory facetAddresses_) {
        facetAddresses_ = facetAddresses;
    }

    function getFacetAddressAndPosition(bytes4 _functionSelector) external view returns (FacetAddressAndPosition memory) {
        return  selectorToFacetAndPosition[_functionSelector];
    }
}
```

Update/Status:

The Venus team has added these functions in `Diamond.sol` instead of `Facet`.

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.

- Consider managing the admin's access control with great care and transferring it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. MockDiamondTest.js

```
const {
    time,
    getStorageAt,
    loadFixture,
} = require("@nomicfoundation/hardhat-network-helpers");

const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Mock Diamond Unit Test", function () {
    const FacetCutAction = {
        Add:0,
        Replace:1,
        Remove:2
    };
    async function deployFixture() {
        const [Owner,...users] = await ethers.getSigners();
        const MockDiamond = await ethers.getContractFactory("MockDiamond");
        const impl = await MockDiamond.deploy();
        const MockProxy = await ethers.getContractFactory("MockProxy");
        const proxy = await MockProxy.deploy(impl.address);
        const instance = MockDiamond.attach(proxy.address);
        return {
            Owner,instance,users
        };
    }

    describe("Add Facet test", function() {
        it("Facet must be contract", async () => {
            const {instance,users} = await loadFixture(deployFixture);
            let facetCut = {
                facetAddress:users[0].address,
                action:FacetCutAction.Add,
                functionSelectors:["0x12345678","0x22345678"]
            }
            await expect(instance.diamondCut([facetCut])).to.revertedWith(
                "Diamond: New facet has no code"
            );
        });

        it("Facet must not be zero address", async () => {
            const {instance} = await loadFixture(deployFixture);
            let facetCut = {
```

```
                facetAddress:ethers.constants.AddressZero,
                action:FacetCutAction.Add,
                functionSelectors:["0x12345678","0x22345678"]
            }
            await expect(instance.diamondCut([facetCut])).to.revertedWith(
                "LibDiamondCut: Add facet can't be address(0)"
            );
        });

        it("Facet must have selectors", async () => {
            const {instance} = await loadFixture(deployFixture);
            let facetCut = {
                facetAddress:ethers.constants.AddressZero,
                action:FacetCutAction.Add,
                functionSelectors:[]
            }
            await expect(instance.diamondCut([facetCut])).to.revertedWith(
                "LibDiamondCut: No selectors in facet to cut"
            );
        });

        it("Call nonexistent facet should be failed", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let diamond = MockOtherFacet.attach(instance.address);
            await expect(diamond.setX(20)).to.revertedWith(
                "Diamond: Function does not exist"
            );
        });

        it("Method of facet must be called successfully", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let facet2 = await MockOtherFacet.deploy();
            let facet3 = await MockOtherFacet.deploy();
            let selectors = [];
            for(let func in facet3.interface.functions) {
                let selector = facet3.interface.getSighash(func);
                // skip comptrollerImplementation
                if(selector === "0xbb82aa5e") {
                    continue;
                }
                selectors.push(selector);
            }

            let args = [
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x11111111","0x22222222"]
                },
                {
                    facetAddress:facet2.address,
```

```
                    action:FacetCutAction.Add,
                    functionSelectors:["0x33333333","0x44444444"]
                },
                {
                    facetAddress:facet3.address,
                    action:FacetCutAction.Add,
                    functionSelectors:selectors
                }
            ];
            await instance.diamondCut(args);
            let data = await getStorageAt(instance.address,4);
            let bn = ethers.BigNumber.from(data);
            expect(bn).to.equal(3);

            // call method of other facet
            let diamond = MockOtherFacet.attach(instance.address);
            await diamond.setX(10);
            expect(await diamond.getX()).to.equal(10);

            // call method of loupe
            let addresses = await instance.getAllFacetAddresses();
            let sources = [
                facet1.address,facet2.address,facet3.address
            ];
            for(let i=0;i<addresses.length;i++) {
                expect(addresses[i]).to.equal(sources[i]);
            }
            let _facetFunctionSelectors = await
instance.getFacetFunctionSelectors(facet1.address);
            expect(_facetFunctionSelectors[0]).to.equal("0x11111111");
            expect(_facetFunctionSelectors[1]).to.equal("0x22222222");
            let position = await instance.getFacetPosition(facet3.address);
            expect(position).to.equal(2);
            (position = await instance.getFacetPosition(facet2.address));
            expect(position).to.equal(1);

            let {facetAddress,functionSelectorPosition} = await
instance.getFacetAddressAndPosition("0x11111111");
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(0);
            ({facetAddress,functionSelectorPosition} = await
instance.getFacetAddressAndPosition("0x22222222"));
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(1);
        });

        it("Add same selector should be failed", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let facet2 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
```

```
                    action:FacetCutAction.Add,
                    functionSelectors:["0x11111111","0x22222222"]
                },
                {
                    facetAddress:facet2.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x22222222","0x44444444"]
                }
            ];
            await expect(instance.diamondCut(args)).to.revertedWith(
                "LibDiamondCut: Can't add function that already exists"
            );
        });
    });


    describe("Remove Facet test", function() {
        it("Remove nonexistent facet should be failed", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Remove,
                    functionSelectors:["0x11111111","0x22222222"]
                }
            ];
            await expect(instance.diamondCut(args)).to.revertedWith(
                "LibDiamondCut: Remove facet address must be address(0)"
            );
            args[0].facetAddress = ethers.constants.AddressZero;
            await expect(instance.diamondCut(args)).to.revertedWith("LibDiamondCut: Can't
remove function that doesn't exist");
        });

        it("Remove function should change state", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x11111111","0x22222222","0x33333333"]
                }
            ];
            await instance.diamondCut(args);
            // let diamond = await addLoupeFacet(instance,loupeFacet)
            let {facetAddress,functionSelectorPosition} = await
instance.getFacetAddressAndPosition("0x22222222");
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(1);
```

```
            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x33333333"));
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(2);

            // remove "0x22222222"
            await instance.diamondCut([
                {
                    facetAddress:ethers.constants.AddressZero,
                    action:FacetCutAction.Remove,
                    functionSelectors:["0x22222222"]
                }
            ]);
            let selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(2);
            let position = await instance.getFacetPosition(facet1.address);
            expect(position).to.equal(0);

            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x11111111"));
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(0);

            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x22222222"));
            expect(facetAddress).to.equal(ethers.constants.AddressZero);
            expect(functionSelectorPosition).to.equal(0);

            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x33333333"));
            expect(facetAddress).to.equal(facet1.address);
            expect(functionSelectorPosition).to.equal(1);
            // remove again
            await instance.diamondCut([
                {
                    facetAddress:ethers.constants.AddressZero,
                    action:FacetCutAction.Remove,
                    functionSelectors:["0x33333333"]
                }
            ]);

            let addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(1);

            selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(1);

            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x33333333"));
            expect(facetAddress).to.equal(ethers.constants.AddressZero);
            expect(functionSelectorPosition).to.equal(0);

            await instance.diamondCut([
                {
```

```
                    facetAddress:ethers.constants.AddressZero,
                    action:FacetCutAction.Remove,
                    functionSelectors:["0x11111111"]
                }
            ]);

            ({facetAddress,functionSelectorPosition} = await instance.getFacetAddressAndPosition("0x11111111"));
            expect(facetAddress).to.equal(ethers.constants.AddressZero);
            expect(functionSelectorPosition).to.equal(0);
            selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(0);
            addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(0);
        });
    });



    describe("Replace test", function() {
        it("Replaced zero facet should be failed", async ()=> {
            const {instance} = await loadFixture(deployFixture);
            let args = [
                {
                    facetAddress:ethers.constants.AddressZero,
                    action:FacetCutAction.Replace,
                    functionSelectors:["0x11111111","0x22222222"]
                }
            ];
            await expect(instance.diamondCut(args)).to.revertedWith("LibDiamondCut: Add facet can't be address(0)");
        });

        it("Replace no selectors should be failed", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Replace,
                    functionSelectors:[]
                }
            ];
            await expect(instance.diamondCut(args)).to.revertedWith("LibDiamondCut: No selectors in facet to cut");
        });
        it("Replaced function must exist", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
```

```
                    action:FacetCutAction.Replace,
                    functionSelectors:["0x11111111","0x22222222"]
                },
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Replace,
                    functionSelectors:["0x33333333","0x44444444"]
                }
            ];
            await expect(instance.diamondCut(args)).to.revertedWith(
                "LibDiamondCut: Can't remove function that doesn't exist"
            );
        });

        it("Replaced function must change state", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let facet2 = await MockOtherFacet.deploy();
            let args = [
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x11111111","0x22222222"]
                },
                {
                    facetAddress:facet1.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x33333333","0x44444444"]
                }
            ];
            await instance.diamondCut(args);
            let addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(1);
            let selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(4);
            args = [
                {
                    facetAddress:facet2.address,
                    action:FacetCutAction.Replace,
                    functionSelectors:["0x33333333","0x22222222"]
                }
            ]
            await instance.diamondCut(args);
            addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(2);
            selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(2);
            expect(selectors[0]).to.equal("0x11111111");
            expect(selectors[1]).to.equal("0x44444444");

            selectors = await instance.getFacetFunctionSelectors(facet2.address);
            expect(selectors.length).to.equal(2);
            expect(selectors[0]).to.equal("0x33333333");
```

```
            expect(selectors[1]).to.equal("0x22222222");
            // replace all
            args = [
                {
                    facetAddress:facet2.address,
                    action:FacetCutAction.Replace,
                    functionSelectors:["0x11111111","0x44444444"]
                }
            ]
            await instance.diamondCut(args);

            addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(1);

            selectors = await instance.getFacetFunctionSelectors(facet2.address);
            expect(selectors.length).to.equal(4);

            selectors = await instance.getFacetFunctionSelectors(facet1.address);
            expect(selectors.length).to.equal(0);

        });
    });

    describe("Bug demo", function() {
        it("Add facet of self can't remove", async () => {
            const {instance} = await loadFixture(deployFixture);
            let addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(0);
            // add facet of self
            let args = [
                {
                    facetAddress:instance.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x11111111","0x22222222"]
                },
                {
                    facetAddress:instance.address,
                    action:FacetCutAction.Add,
                    functionSelectors:["0x33333333","0x44444444"]
                }
            ];
            await instance.diamondCut(args);
            addresses = await instance.getAllFacetAddresses();
            expect(addresses.length).to.equal(1);
            let selectors = await instance.getFacetFunctionSelectors(instance.address);
            expect(selectors.length).to.equal(4);

            // remove
            args = [
                {
                    facetAddress:ethers.constants.AddressZero,
                    action:FacetCutAction.Remove,
                    functionSelectors:["0x11111111","0x22222222"]
                },
```

```
            ];
            await instance.diamondCut(args);
        });

        it("Infinite fallback", async () => {
            const {instance} = await loadFixture(deployFixture);
            const MockOtherFacet = await ethers.getContractFactory("MockOtherFacet");
            let facet1 = await MockOtherFacet.deploy();
            let selectors = [];
            for(let func in facet1.interface.functions) {
                let selector = facet1.interface.getSighash(func);
                // skip comptrollerImplementation
                if(selector === "0xbb82aa5e") {
                    continue;
                }
                selectors.push(selector);
            }
            let args = [
                {
                    facetAddress:instance.address,
                    action:FacetCutAction.Add,
                    functionSelectors:selectors
                }
            ];
            await instance.diamondCut(args);
            selectors = await instance.getFacetFunctionSelectors(instance.address);
            expect(selectors.length).to.equal(2);
            let facet = MockOtherFacet.attach(instance.address);
            await expect(facet.setX(30)).to.reverted;
        });
    });
});


```

2. MockDiamond.sol

```
pragma solidity 0.5.16;
pragma experimental ABIEncoderV2;

interface IDiamondCut {
    enum FacetCutAction {
        Add,
        Replace,
        Remove
    }
    // Add=0, Replace=1, Remove=2

    struct FacetCut {
        address facetAddress;
```

```
        FacetCutAction action;
        bytes4[] functionSelectors;
    }

    /// @notice Add/replace/remove any number of functions and optionally execute
    ///         a function with delegatecall
    /// @param _diamondCut Contains the facet addresses and function selectors
    function diamondCut(FacetCut[] calldata _diamondCut) external;

    event DiamondCut(FacetCut[] _diamondCut);
}

contract MockStorageBase {
    struct FacetAddressAndPosition {
        address facetAddress;
        uint96 functionSelectorPosition; // position in
facetFunctionSelectors.functionSelectors array
    }

    struct FacetFunctionSelectors {
        bytes4[] functionSelectors;
        uint256 facetAddressPosition; // position of facetAddress in facetAddresses array
    }
    // must be the first var
    address public comptrollerImplementation;

    // other state vars
    uint internal x;

    mapping(bytes4 => FacetAddressAndPosition) internal selectorToFacetAndPosition;
    // maps facet addresses to function selectors
    mapping(address => FacetFunctionSelectors) internal facetFunctionSelectors;
    // facet addresses
    address[] internal facetAddresses;
}

contract MockOtherFacet is MockStorageBase {
    function getX() external view returns(uint256) {
        return x;
    }

    function setX(uint _x) external {
        x = _x;
    }
}

contract MockLoupeFacet is MockStorageBase {

    function getFacetFunctionSelectors(address _facet) external view returns (bytes4[]
memory _facetFunctionSelectors) {
        _facetFunctionSelectors =  facetFunctionSelectors[_facet].functionSelectors;
    }

    function getFacetPosition(address _facet) external view returns(uint256) {
```

```
66          return facetFunctionSelectors[_facet].facetAddressPosition;
67      }
68
69      function getAllFacetAddresses() external view returns (address[] memory
    facetAddresses_) {
70          facetAddresses_ = facetAddresses;
71      }
72
73      function getFacetAddressAndPosition(bytes4 _functionSelector) external view returns
    (FacetAddressAndPosition memory) {
74          return  selectorToFacetAndPosition[_functionSelector];
75      }
76  }
77
78  contract MockDiamond is IDiamondCut, MockLoupeFacet {
79      /**
80       * @notice To add function selectors to the facets' mapping.
81       * @param _diamondCut IDiamondCut contains facets address, action and function
    selectors.
82       */
83      function diamondCut(FacetCut[] memory _diamondCut) public {
84          libDiamondCut(_diamondCut);
85      }
86
87      /**
88       * @notice To add function selectors to the facets' mapping.
89       * @param _diamondCut IDiamondCut contains facets address, action and function
    selectors.
90       */
91      function libDiamondCut(FacetCut[] memory _diamondCut) internal {
92          for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex++) {
93              FacetCutAction action = _diamondCut[facetIndex].action;
94              if (action == FacetCutAction.Add) {
95                  addFunctions(_diamondCut[facetIndex].facetAddress,
    _diamondCut[facetIndex].functionSelectors);
96              } else if (action == FacetCutAction.Replace) {
97                  replaceFunctions(_diamondCut[facetIndex].facetAddress,
    _diamondCut[facetIndex].functionSelectors);
98              } else if (action == FacetCutAction.Remove) {
99                  removeFunctions(_diamondCut[facetIndex].facetAddress,
    _diamondCut[facetIndex].functionSelectors);
100             } else {
101                 revert("LibDiamondCut: Incorrect FacetCutAction");
102             }
103         }
104         emit DiamondCut(_diamondCut);
105     }
106
107     /**
108      * @notice Add function selectors to the facet's address mapping.
109      * @param _facetAddress Address of the facet.
110      * @param _functionSelectors Array of function selectors need to add in the mapping.
111      */
```

```
112     function addFunctions(address _facetAddress, bytes4[] memory _functionSelectors)
internal {
113         require(_functionSelectors.length > 0, "LibDiamondCut: No selectors in facet to
cut");
114         require(_facetAddress != address(0), "LibDiamondCut: Add facet can't be
address(0)");
115         uint96 selectorPosition =
uint96(facetFunctionSelectors[_facetAddress].functionSelectors.length);
116         // add new facet address if it does not exist
117         if (selectorPosition == 0) {
118             addFacet(_facetAddress);
119         }
120         for (uint256 selectorIndex; selectorIndex < _functionSelectors.length;
selectorIndex++) {
121             bytes4 selector = _functionSelectors[selectorIndex];
122             address oldFacetAddress = selectorToFacetAndPosition[selector].facetAddress;
123             require(oldFacetAddress == address(0), "LibDiamondCut: Can't add function that
already exists");
124             addFunction(selector, selectorPosition, _facetAddress);
125             selectorPosition++;
126         }
127     }
128
129     /**
130      * @notice Replace facet's address mapping for function selectors i.e selectors
already associate to any other existing facet.
131      * @param _facetAddress Address of the facet.
132      * @param _functionSelectors Array of function selectors need to replace in the
mapping.
133      */
134     function replaceFunctions(address _facetAddress, bytes4[] memory _functionSelectors)
internal {
135         require(_functionSelectors.length > 0, "LibDiamondCut: No selectors in facet to
cut");
136         require(_facetAddress != address(0), "LibDiamondCut: Add facet can't be
address(0)");
137         uint96 selectorPosition =
uint96(facetFunctionSelectors[_facetAddress].functionSelectors.length);
138         // add new facet address if it does not exist
139         if (selectorPosition == 0) {
140             addFacet(_facetAddress);
141         }
142         for (uint256 selectorIndex; selectorIndex < _functionSelectors.length;
selectorIndex++) {
143             bytes4 selector = _functionSelectors[selectorIndex];
144             address oldFacetAddress = selectorToFacetAndPosition[selector].facetAddress;
145             require(oldFacetAddress != _facetAddress, "LibDiamondCut: Can't replace
function with same function");
146             removeFunction(oldFacetAddress, selector);
147             addFunction(selector, selectorPosition, _facetAddress);
148             selectorPosition++;
149         }
150     }
151
```

```
152     /**
153      * @notice Remove function selectors to the facet's address mapping.
154      * @param _facetAddress Address of the facet.
155      * @param _functionSelectors Array of function selectors need to remove in the
mapping.
156      */
157     function removeFunctions(address _facetAddress, bytes4[] memory _functionSelectors)
internal {
158         require(_functionSelectors.length > 0, "LibDiamondCut: No selectors in facet to
cut");
159         // if function does not exist then do nothing and return
160         require(_facetAddress == address(0), "LibDiamondCut: Remove facet address must be
address(0)");
161         for (uint256 selectorIndex; selectorIndex < _functionSelectors.length;
selectorIndex++) {
162             bytes4 selector = _functionSelectors[selectorIndex];
163             address oldFacetAddress = selectorToFacetAndPosition[selector].facetAddress;
164             removeFunction(oldFacetAddress, selector);
165         }
166     }
167
168     /**
169      * @notice Add new facet to the proxy.
170      * @param _facetAddress Address of the facet.
171      */
172     function addFacet(address _facetAddress) internal {
173         enforceHasContractCode(_facetAddress, "Diamond: New facet has no code");
174         facetFunctionSelectors[_facetAddress].facetAddressPosition =
facetAddresses.length;
175         facetAddresses.push(_facetAddress);
176     }
177
178     /**
179      * @notice Add function selector to the facet's address mapping.
180      * @param _selector funciton selector need to be added.
181      * @param _selectorPosition funciton selector position.
182      * @param _facetAddress Address of the facet.
183      */
184     function addFunction(bytes4 _selector, uint96 _selectorPosition, address
_facetAddress) internal {
185         selectorToFacetAndPosition[_selector].functionSelectorPosition =
_selectorPosition;
186         facetFunctionSelectors[_facetAddress].functionSelectors.push(_selector);
187         selectorToFacetAndPosition[_selector].facetAddress = _facetAddress;
188     }
189
190     /**
191      * @notice Remove function selector to the facet's address mapping.
192      * @param _facetAddress Address of the facet.
193      * @param _selector function selectors need to remove in the mapping.
194      */
195     function removeFunction(address _facetAddress, bytes4 _selector) internal {
196         require(_facetAddress != address(0), "LibDiamondCut: Can't remove function that
doesn't exist");
```

```
197         // an immutable function is a function defined directly in a diamond
198         // require(_facetAddress != address(this), "LibDiamondCut: Can't remove immutable
    function");
199         // replace selector with last selector, then delete last selector
200         uint256 selectorPosition =
    selectorToFacetAndPosition[_selector].functionSelectorPosition;
201         uint256 lastSelectorPosition =
    facetFunctionSelectors[_facetAddress].functionSelectors.length - 1;
202         // if not the same then replace _selector with lastSelector
203         if (selectorPosition != lastSelectorPosition) {
204             bytes4 lastSelector =
    facetFunctionSelectors[_facetAddress].functionSelectors[lastSelectorPosition];
205             facetFunctionSelectors[_facetAddress].functionSelectors[selectorPosition] =
    lastSelector;
206             selectorToFacetAndPosition[lastSelector].functionSelectorPosition =
    uint96(selectorPosition);
207         }
208         // delete the last selector
209         facetFunctionSelectors[_facetAddress].functionSelectors.pop();
210         delete selectorToFacetAndPosition[_selector];
211
212         // if no more selectors for facet address then delete the facet address
213         if (lastSelectorPosition == 0) {
214             // replace facet address with last facet address and delete last facet address
215             uint256 lastFacetAddressPosition = facetAddresses.length - 1;
216             uint256 facetAddressPosition =
    facetFunctionSelectors[_facetAddress].facetAddressPosition;
217             if (facetAddressPosition != lastFacetAddressPosition) {
218                 address lastFacetAddress = facetAddresses[lastFacetAddressPosition];
219                 facetAddresses[facetAddressPosition] = lastFacetAddress;
220                 facetFunctionSelectors[lastFacetAddress].facetAddressPosition =
    facetAddressPosition;
221             }
222             facetAddresses.pop();
223             delete facetFunctionSelectors[_facetAddress];
224         }
225     }
226
227     function enforceHasContractCode(address _contract, string memory _errorMessage)
    internal view {
228         uint256 contractSize;
229         assembly {
230             contractSize := extcodesize(_contract)
231         }
232         require(contractSize > 0, _errorMessage);
233     }
234
235     // Find facet for function that is called and execute the
236     // function if a facet is found and return any value.
237     function() external payable {
238         address facet = selectorToFacetAndPosition[msg.sig].facetAddress;
239         require(facet != address(0), "Diamond: Function does not exist");
240         // Execute public function from facet using delegatecall and return any value.
241         assembly {
```

```
            // copy function selector and any arguments
            calldatacopy(0, 0, calldatasize())
            // execute function call using the facet
            let result := delegatecall(gas(), facet, 0, calldatasize(), 0, 0)
            // get any return value
            returndatacopy(0, 0, returndatasize())
            // return any return value or error back to the caller
            switch result
            case 0 {
                revert(0, returndatasize())
            }
            default {
                return(0, returndatasize())
            }
        }
    }
}

contract MockProxy {
    address public  comptrollerImplementation;
    constructor(address impl) public {
        comptrollerImplementation = impl;
    }
    function() external payable {
        // delegate all other functions to current implementation
        (bool success, ) = comptrollerImplementation.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)
            }
        }
    }
}

```

3. UnitTestOutput

```
Mock Diamond Unit Test
   Add Facet test
     ✔ Facet must be contract (1617ms)
     ✔ Facet must not be zero address
     ✔ Facet must have selectors
     ✔ Call nonexistent facet should be failed
```

```
      ✔ Method of facet must be called successfully (296ms)
      ✔ Add same selector should be failed (113ms)
    Remove Facet test
      ✔ Remove nonexistent facet should be failed (82ms)
      ✔ Remove function should change state (287ms)
    Replace test
      ✔ Replaced zero facet should be failed
      ✔ Replace no selectors should be failed (43ms)
      ✔ Replaced function must exist (53ms)
      ✔ Replaced function must change state (279ms)
    Bug demo
      ✔ Add facet of self can't remove (105ms)
      ✔ Infinite fallback (599ms)


  14 passing (4s)

```

11.2 External Functions Check Points

1. MarketFacet.sol.output

File: Comptroller/Diamond/facets/MarketFacet.sol

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

contract: MarketFacet

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	isComptroller()	public	pure				
2	getAssetsIn(address)	external	view				
3	getAllMarkets()	external	view				
4	liquidateCalculateSeizeTokens(address,address,uint)	external	view				
5	liquidateVAICalculateSeizeTokens(address,uint)	external	view				
6	checkMembership(address,VToken)	external	view				
7	enterMarkets(address[])	external			Yes		
8	exitMarket(address)	external			Yes		
9	_supportMarket(VToken)	external		ensureAllowed			
10	updateDelegate(address,bool)	external			Yes		

2. FacetBase.sol.output

File: Comptroller/Diamond/facets/FacetBase.sol

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

contract: FacetBase

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	actionPaused(address,Action)	public	view				
2	getBlockNumber()	public	view				

3. PolicyFacet.sol_output

File: Comptroller/Diamond/facets/PolicyFacet.sol

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

contract: PolicyFacet is XVSRewardsHelper

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	mintAllowed(address,address,uint)	external					CalledByVToken
2	mintVerify(address,address,uint,uint)	external					CalledByVToken
3	redeemAllowed(address,address,uint)	external					CalledByVToken
4	redeemVerify(address,address,uint,uint)	external	pure				CalledByVToken
5	borrowAllowed(address,address,uint)	external					CalledByVToken
6	borrowVerify(address,address,uint)	external					CalledByVToken
7	repayBorrowAllowed(address,address,address,uint)	external					CalledByVToken
8	repayBorrowVerify(address,address,address,uint,uint)	external					CalledByVToken
9	liquidateBorrowAllowed(address,address,address,address,uint)	external	view				CalledByVToken
10	liquidateBorrowVerify(address,address,address,address,uint,uint)	external					CalledByVToken
11	seizeAllowed(address,address,address,address,uint)	external					CalledByVToken
12	seizeVerify(address,address,address,address,uint)	external					CalledByVToken
13	transferAllowed(address,address,address,uint)	external					CalledByVToken
14	transferVerify(address,address,address,uint)	external					CalledByVToken
15	getAccountLiquidity(address)	external	view				
16	getHypotheticalAccountLiquidity(address,address,uint,uint)	external	view				
17	_setVenusSpeeds(VToken[],uint[],uint[])	external		ensureAdminOr			
18	releaseToVault()	public					Public
19	getXVSAddress()	public	pure				

4. SetterFacet.sol.output

File: Comptroller/Diamond/facets/SetterFacet.sol

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

contract: SetterFacet

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	_setPriceOracle(PriceOracle)	external		ensureAdmin			
2	_setCloseFactor(uint)	external		ensureAdmin			
3	_setAccessControl(address)	external		ensureAdmin			
4	_setCollateralFactor(VToken,uint)	external		ensureAllowed			
5	_setLiquidationIncentive(uint)	external		ensureAllowed			
6	_setLiquidatorContract(address)	external		ensureAdmin			
7	_setPauseGuardian(address)	external		ensureAdmin			
8	_setMarketBorrowCaps(VToken[],uint[])	external		ensureAllowed			
9	_setMarketSupplyCaps(VToken[],uint256[])	external		ensureAllowed			
10	_setProtocolPaused(bool)	external		ensureAllowed			
11	_setActionsPaused(address[],Action[],bool)	external		ensureAllowed			
12	_setVAIController(VAIControllerInterface)	external		ensureAdmin			
13	_setVAIMintRate(uint)	external		ensureAdmin			
14	setMintedVAIOf(address,uint)	external		vaiController			
15	_setTreasuryData(address,address,uint)	external		ensureAdminOr			
16	_setComptrollerLens(ComptrollerLensInterface)	external		ensureAdmin			
17	_setVenusVAIVaultRate(uint)	external		ensureAdmin			
18	_setVAIVaultInfo(address,uint256,uint256)	external		ensureAdmin			

5. RewardFacet.sol.output

File: Comptroller/Diamond/facets/RewardFacet.sol

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

contract: RewardFacet is XVSRewardsHelper

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	claimVenus(address)	public			Yes		
2	claimVenus(address,VToken[])	public			Yes		
3	claimVenus(address[],VToken[],bool,bool)	public			Yes		
4	claimVenusAsCollateral(address)	external			Yes		
5	_grantXVS(address,uint)	external		ensureAdminOr			
6	getXVSVTokenAddress()	public	pure				
7	claimVenus(address[],VToken[],bool,bool,bool)	public			Yes		

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