FAIRYPROOF

wUSDM Liquidator

AUDIT REPORT

Version 1.0.0

Serial No. 20250508000012025

Presented by Fairyproof

May 08, 2025

01. Introduction

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

Audit Start Time:

April 30, 2025

Audit End Time:

May 3, 2025

Audited Code's Github Repository:

https://github.com/VenusProtocol/isolated-pools/pull/517

Audited Code's Github Commit Number When Audit Started:

c57bbf0ec66f606ede845a3d7820cffcbecbd410

Audited Code's Github Commit Number When Audit Ended:

c57bbf0ec66f606ede845a3d7820cffcbecbd410

Audited Source Files:

The source files audited include all the files as follows:

```
- contracts/ComptrollerStorage.sol
- contracts/WUSDMLiquidator.sol
```

The goal of this audit is to review Venus's solidity implementation for its WUSDMLiquidator function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Venus team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project’s source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project’s code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

Website:https://venus.io/

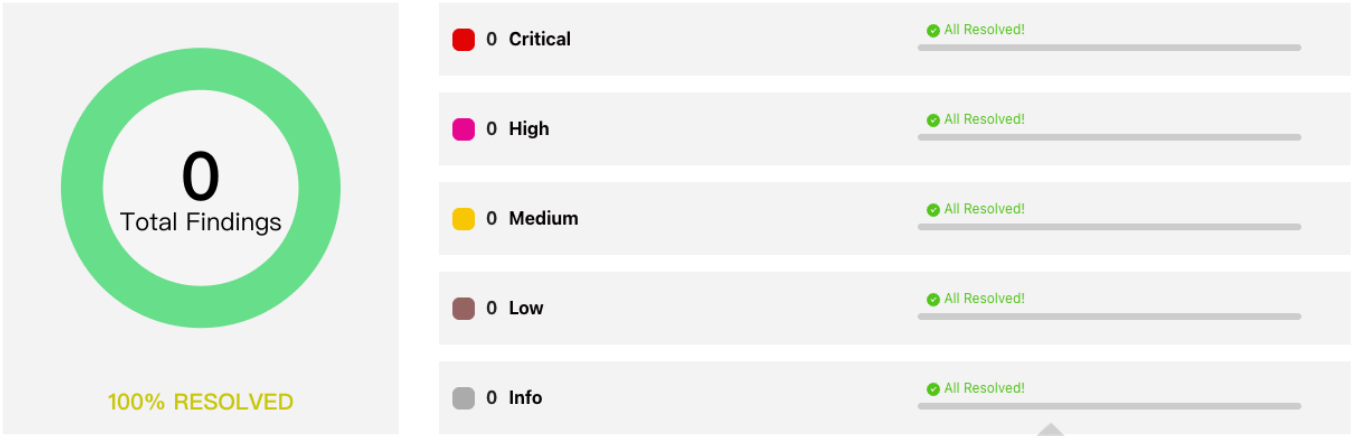
Whitepaper:https://github.com/VenusProtocol/venus-protocol-documentation/tree/main/whitepapers

Source Code: https://github.com/VenusProtocol/isolated-pools/pull/517

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
2025050800012025	Fairyproof Security Team	Apr 30, 2025 - May 3, 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

The main function of the audit code is to perform targeted liquidation of the `wUSDM` market on the `zkSync` blockchain for Venus protocol to uniformly handle bad debts. Currently, this market is in a paused state. During liquidation, the pause will be temporarily lifted to perform the liquidation, and after the liquidation is completed, the market will be set back to a paused state.

Note: To perform the liquidation, the `Comptroller` contract needs to be temporarily upgraded to modify the maximum `CloseFactor` to 100%, and after the liquidation is completed, it needs to be upgraded back again.

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.

- N/A

11. Appendices

11.1 Unit Test

1. LiquidatorTest.sol

```
// Forking test on block of 59859399
// SPDX-License-Identifier: MIT
pragma solidity 0.8.25;

import {Test, console} from "forge-std/Test.sol";

import {WUSDMLiquidator,Comptroller,VToken,IERC20Upgradeable} from
"../src/WUSDMLiquidator.sol";
import {BeaconProxy} from "@openzeppelin/contracts/proxy/beacon/BeaconProxy.sol";
import {UpgradeableBeacon} from
"@openzeppelin/contracts/proxy/beacon/UpgradeableBeacon.sol";
import {AccessControlManager} from "@venusprotocol/governance-
contracts/contracts/Governance/AccessControlManager.sol";

contract LiquidatorTest is Test {
    address comptroller_address = 0xddE4D098D9995B659724ae6d5E3FB9681Ac941B1;
    // beacon of comptroller
    address beacon_address = 0x0221415aF47FD261dD39B72018423dADe5d937c5;
    // beacon owner
    address beacon_owner = 0x093565Bc20AA326F4209eBaF3a26089272627613;

    address wusdm_address = 0xA900cbE7739c96D2B153a273953620A701d5442b;
    // market address
    address vwusdm_address = 0x183dE3C349fCf546aAe925E1c7F364EA6FB4033c;
```

```
    address vweth_address = 0x1Fa916C27c7C2c4602124A14C77Dbb40a5FF1BE8;
    address vusdce_address = 0x1aF23bD57c62A99C59aD48236553D0Dd11e49D2D;
    address vusdt_address = 0x69cDA960E3b20DFD480866fFfd377Ebe40bd0A46;
    // Venus Treasury ,
    address vTreasury_address = 0xB2e9174e23382f7744CebF7e0Be54cA001D95599;
    // only for upgrade comptroller
    address pool_register = 0xFD96B926298034aed9bBe0Cca4b651E41eB87Bc4;
    // latest acm address
    address acm_address = 0x526159A92A82afE5327d37Ef446b68FD9a5cA914;
    // admin of acm, safe wallet
    address acm_admin = beacon_owner;

    // instance of WUSDMLiquidator
    WUSDMLiquidator liquidator;

    // bad accounts
    address public constant A2 = 0x4C0e4B3e6c5756fb31886a0A01079701ffEC0561;
    address public constant A3 = 0x924EDEd3D010b3F20009b872183eec48D0111265;
    address public constant A4 = 0x2B379d8c90e02016658aD00ba2566F55E814C369;
    address public constant A5 = 0xfffAB9120d9Df39EEa07063F6465a0aA45a80C52;

    // only for test
    address[] public accounts = [A2,A3,A4,A5];
    address[2][] public markets = [
        [vweth_address,address(0)],
        [vusdt_address,vusdce_address],
        [vusdt_address,vusdce_address],
        [vusdt_address,vusdce_address]
    ];

    string[2][] public market_names = [
        ["vweth","empty"],
        ["vusdt","vusdce"],
        ["vusdt","vusdce"],
        ["vusdt","vusdce"]
    ];

    function setUp() public {
        // 1 deploy WUSDMLiquidator
        WUSDMLiquidator implement = new WUSDMLiquidator();
        UpgradeableBeacon beacon = new UpgradeableBeacon(address(implement));
        BeaconProxy proxy = new BeaconProxy(
            address(beacon),
            abi.encodeWithSignature(
                "initialize()"
            )
        );
        liquidator = WUSDMLiquidator(address(proxy));

        // 2 temp upgrade comptroller
        Comptroller tempComptroller = new Comptroller(pool_register);
        vm.startPrank(beacon_owner);
```

```
        UpgradeableBeacon(beacon_address).upgradeTo(address(tempComptroller));
        vm.stopPrank();

        // 3 get acm instance
        AccessControlManager acm = AccessControlManager(acm_address);
        vm.startPrank(acm_admin);

        // 4 grant WUSDMLiquidator to access acm
        acm.giveCallPermission(
            comptroller_address,
            "setMinLiquidatableCollateral(uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            comptroller_address,
            "setCloseFactor(uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            comptroller_address,
            "setCollateralFactor(address,uint256,uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            comptroller_address,
            "setActionsPaused(address[],uint256[],bool)",
            address(liquidator)
        );

        // 5. grant WUSDMLiquidator to access markets
        acm.giveCallPermission(
            vwusdm_address,
            "setProtocolSeizeShare(uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            vweth_address,
            "setProtocolSeizeShare(uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            vusdce_address,
            "setProtocolSeizeShare(uint256)",
            address(liquidator)
        );
        acm.giveCallPermission(
            vusdt_address,
            "setProtocolSeizeShare(uint256)",
            address(liquidator)
        );

        // 6 transfer wUSDM to liquidator
```

```
        vm.startPrank(vTreasury_address);
        uint balance = IERC20Upgradeable(wusdm_address).balanceOf(vTreasury_address);
        IERC20Upgradeable(wusdm_address).transfer(address(liquidator), balance);
        vm.stopPrank();
    }

    // check state of accounts
    function check_account_state(bool is_before) internal {
        if(is_before){
            console.log("=== check state before ===");
        }else{
            console.log("=== check state after ===");
        }
        for(uint i=0;i<accounts.length;i++){
            address account = accounts[i];
            (uint liquidity,uint shortfall) = _isUnderwater(account);
            console.log("liquidity of A%d is: %d",i+2,liquidity);
            console.log("shortfall of A%d is: %d",i+2,shortfall);
            uint balnace = VToken(vwusdm_address).balanceOf(account);
            console.log("vwUSDM balance of A%d is: %d", i+2,balnace);
            address[2] memory market = markets[i];
            string[2] memory names = market_names[i];
            for(uint j=0;j<market.length;j++){
                address market_address = market[j];
                if(market_address == address(0)){
                    continue;
                }
                uint debt = getDebt(account,market_address);
                string memory name = names[j];
                console.log("Market %s debt of A%d is:%d",name,i+2,debt);
            }
            console.log("");
        }
    }

    // check state of liquidator
    function check_state_of_liquidator() internal view {
        (uint liquidity,uint shortfall) = _isUnderwater(address(liquidator));
        console.log("Liquidity of liquidator is: %d",liquidity);
        console.log("Shortfall of liquidator is: %d",shortfall);
        uint balnace = VToken(vwusdm_address).balanceOf(address(liquidator));
        console.log("vwUSDM balance of liquidator is: %d",balnace);

        uint wethDebt = VToken(vweth_address).borrowBalanceStored(address(liquidator));
        uint usdceDebt = VToken(vusdce_address).borrowBalanceStored(address(liquidator));
        uint usdtDebt = VToken(vusdt_address).borrowBalanceStored(address(liquidator));
        console.log("WETH debt of liquidator: %d", wethDebt);
        console.log("USDC.e debt of liquidator: %d", usdceDebt);
        console.log("USDT debt of liquidator: %d", usdtDebt);
    }

    // check state of comptroller
```

```
    function get_comptroller_state() internal view returns (uint[] memory) {
        uint[] memory results = new uint[](4);
        uint minLiquidatableCollateral  =
Comptroller(comptroller_address).minLiquidatableCollateral();
        uint closeFactorMantissa = Comptroller(comptroller_address).closeFactorMantissa();
        (, uint256 wUSDMCollateralFactor, uint256 wUSDMLiquidationThreshold) =
Comptroller(comptroller_address).markets(vwusdm_address);
        results[0] = minLiquidatableCollateral;
        results[1] = closeFactorMantissa;
        results[2] = wUSDMCollateralFactor;
        results[3] = wUSDMLiquidationThreshold;
        return results;
    }

    // test run liquidator
    function testRun() external {
        check_account_state(true);
        uint[] memory state1 = get_comptroller_state();
        liquidator.run();
        check_account_state(false);
        check_state_of_liquidator();
        uint[] memory state2 = get_comptroller_state();
        assert(state1.length == state2.length);
        for(uint i=0;i<state1.length;i++){
            assert(state1[i] == state2[i]);
        }
    }


    function _isUnderwater(address account) internal view returns (uint256 liquidity,uint
shortfall) {
        (, liquidity, shortfall) =
Comptroller(comptroller_address).getAccountLiquidity(account);

    }

    function getDebt(address account,address market) internal returns (uint256) {
       uint debt = VToken(market).borrowBalanceCurrent(account);
       return debt;
    }


    // get the beacon address of comptroller
    function testGetBeaconAddress() public view {
        bytes32 beaconSlot =
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50;
        bytes32 value = vm.load(comptroller_address, beaconSlot);
        address beaconAddress = address(uint160(uint256(value)));
        console.log("Beacon address:", beaconAddress);
    }

}
```

2. ForkingTestOutput

```
Forking test on block of 59859399

Ran 2 tests for test/Liquidator.t.sol:LiquidatorTest
[PASS] testGetBeaconAddress() (gas: 9179)
Logs:
  Beacon address: 0x0221415aF47FD261dD39B72018423dADe5d937c5

[PASS] testRun() (gas: 16257247)
Logs:
  === check state before ===
  liquidity of A2 is: 0
  shortfall of A2 is: 60813704974064221501532
  vwUSDM balance of A2 is: 27600971020155
  Market vweth debt of A2 is:159344103812547681617

  liquidity of A3 is: 0
  shortfall of A3 is: 33045687714775514046380
  vwUSDM balance of A3 is: 4929412019494
  Market vusdt debt of A3 is:55253303744
  Market vusdce debt of A3 is:19388753561

  liquidity of A4 is: 0
  shortfall of A4 is: 26936230662235911869497
  vwUSDM balance of A4 is: 3528279214003
  Market vusdt debt of A4 is:21631131316
  Market vusdce debt of A4 is:35072750180

  liquidity of A5 is: 0
  shortfall of A5 is: 10888333336513344512552
  vwUSDM balance of A5 is: 15966710
  Market vusdt debt of A5 is:8366180527
  Market vusdce debt of A5 is:2521082990

  WETH debt of a2 before repay:  12081574383012029517
  USDC debt of a3 before repay:  1
  USDC debt of a4 before repay:  374953358
  USDC debt of a5 before repay:  2520925971
  USDT debt of a3 before repay:  26167701080
  USDT debt of a4 before repay:  21631131316
  USDT debt of a5 before repay:  8366180527

  === check state after ===
  liquidity of A2 is: 16876253176
  shortfall of A2 is: 0
```

```
  vwUSDM balance of A2 is: 2
  Market vweth debt of A2 is:1

  liquidity of A3 is: 0
  shortfall of A3 is: 1266058806663
  vwUSDM balance of A3 is: 87
  Market vusdt debt of A3 is:1
  Market vusdce debt of A3 is:1

  liquidity of A4 is: 0
  shortfall of A4 is: 1671088927125
  vwUSDM balance of A4 is: 39
  Market vusdt debt of A4 is:1
  Market vusdce debt of A4 is:1

  liquidity of A5 is: 0
  shortfall of A5 is: 1316687571721
  vwUSDM balance of A5 is: 81
  Market vusdt debt of A5 is:1
  Market vusdce debt of A5 is:1

  Liquidity of liquidator is: 179871760446603873565890
  Shortfall of liquidator is: 0
  vwUSDM balance of liquidator is: 72982327972910
  WETH debt of liquidator: 159344103812547681617
  USDC.e debt of liquidator: 56982586731
  USDT debt of liquidator: 85250615587

Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 309.44s (286.59s CPU time)

Ran 1 test suite in 309.99s (309.44s CPU time): 2 tests passed, 0 failed, 0 skipped (2
total tests)
```


11.2 External Functions Check Points

1. WUSDMLiquidator.sol

File: src/WUSDMLiquidator.sol

contract: WUSDMLiquidator

(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()		`initializer`		No	Passed	Only Once
2	run()		`onlyOwner`		No	Passed	
3	sweepToken(IERC20Upgradeable)		`onlyOwner`		No		

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