HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Venus Protocol
Date: April 3, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Venus Protocol
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Lending Platform
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://venus.io/
Changelog	30.01.2023 - Initial Review 07.03.2023 - Second Review 03.04.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Venus Protocol (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/VenusProtocol/isolated-pools
Commit	d644aba716a4bbaaa645ead54c5777ef8bfd8a97
Whitepaper	Shared as a file
Functional Requirements	Shared as a file
Technical Requirements	https://github.com/VenusProtocol/isolated-pools
Contracts	File: ./contracts/BaseJumpRateModelV2.sol SHA3: b3444327a988798bfce75df4cc548328d350c46bfaf3b3a1cf79da713f4c3112 File: ./contracts/Comptroller.sol SHA3: 2ec03a2ed912bfb2d0d07ab3a38f14fe96c0b28162fb6b0f4ba9c5780a639ed8 File: ./contracts/ComptrollerInterface.sol SHA3: 419e3e70916d5b01bcb055bf8a858bc1af0ab9f533a79c6bcbede14b53fcb7db File: ./contracts/ComptrollerStorage.sol SHA3: e0c63d29c421f356b9c1792495bef109762fd079cd3543b6f17691887db6e7a3 File: ./contracts/ErrorReporter.sol SHA3: b4ce2d31b59cb01e317ac33541b6f182e99360567c2ef670b630999ea1631a1b File: ./contracts/ExponentialNoError.sol SHA3: d7aee280bd5f87020891db75f274085e32a539993dba73a8b4d9adaa1a6a13bc File: ./contracts/Factories/JumpRateModelFactory.sol SHA3: 0c4a01252c1705f177006d4acbd2083b97eb7989650c14eba702fc5035caaa60 File: ./contracts/Factories/VTokenProxyFactory.sol SHA3: 3909d9ae28cef63bdb907692de500811d011373c1200049111958ab1d0d73529 File: ./contracts/Factories/WhitePaperInterestRateModelFactory.sol SHA3: 9c9c2297837d5404df86261bd0648bc63dd0e6830866216a9cb703690a3f36b2

	File: ./contracts/Governance/AccessControlManager.sol SHA3: 1bfd604bd17a912291a23952f4d8c2e6e556b6da8de297dad6e2957ea331f2cb File: ./contracts/InterestRateModel.sol SHA3: f10b9f3d4d3cb3035bf6c8c7a34b0357fa9fd10b40e3a418d9a8c9aff2413464 File: ./contracts/IPancakeswapV2Router.sol SHA3: 355517858307adfb2d4108c35753af8bc46b89709179f914770529ba371767b0 File: ./contracts/JumpRateModelV2.sol SHA3: 2fb3dab5b18c32ab36aea5dfb3ebdd33087c485427fc56e449e0a0a894317fe2 File: ./contracts/Lens/PoolLens.sol SHA3: 8610ad9627f48ee24cb8beeeaa353ceb99018f23930d79ee2211f6e434303f9f File: ./contracts/Pool/PoolRegistry.sol SHA3: 190660e064ed40307e44a6130088e433b607ebacf482e1a57aebd88f0f8ad28c File: ./contracts/Pool/PoolRegistryInterface.sol SHA3: 028cebfabf3e0eed80062b19dec85126f00c9bf8d00762a199275fc90839f946 File: ./contracts/Proxy/UpgradeableBeacon.sol SHA3: 8e2e7e7530c6d69c2d2c23a57fcc21538f40325ace50cc39b6c008407dc8e24b File: ./contracts/Rewards/RewardsDistributor.sol SHA3: 25cafe482f056c50a878b08b16557e265a36544ab1dd9879310d58096f9d02c7 File: ./contracts/RiskFund/IProtocolShareReserve.sol SHA3: cb02c0976d343603cd26f492889d4da1fc8b69b102cb7ceb8e41cc14703bc238 File: ./contracts/RiskFund/IRiskFund.sol SHA3: 853cab97bdc67e2cd45da901c0d282c2b45d9e16ed50f7e87e0958d28e228d35 File: ./contracts/RiskFund/ProtocolShareReserve.sol SHA3: 4743bb12fc8bdd201cf036b4d7ca34acd756a7bd1150fb4214037c2532c299af File: ./contracts/RiskFund/ReserveHelpers.sol SHA3: 02793aa8040b7ba31495bc191c79f277c9727e2e26241e602928d1c8adbed908 File: ./contracts/RiskFund/RiskFund.sol SHA3: 2bf0c4ad49ac18348678260cd026bbef5345f0c84e26bd11035237c619904529 File: ./contracts/Shortfall/Shortfall.sol SHA3: 33d9f9dba0d7a91b01c9f1c748556a3e8d0397beda5a216f40b40164545f30c4 File: ./contracts/VToken.sol SHA3: f7deb0baaeae62f0b61578d3829268ad9b4ab20d36d61b6897cc67f94fb86dcc File: ./contracts/VTokenInterfaces.sol

	SHA3: ad15c22cc7fe30638e41a64743a2d128131987759310713 8f0e3d16753cdd161 File: ./contracts/WhitePaperInterestRateModel.sol SHA3: 70f0c0cda76ca3c30287604e4bc3e81b8517876d7c125234e9f577ab7897fbab

Second review scope

Repository	https://github.com/VenusProtocol/isolated-pools
Commit	3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7
Whitepaper	Shared as a file
Functional Requirements	Shared as a file
Technical Requirements	https://github.com/VenusProtocol/isolated-pools
Contracts	File: ./contracts/BaseJumpRateModelV2.sol SHA3: 398d8eae8583c0916ff14edd5d0ed8fe46919572422f8863ebb6dbc2e3ce018e File: ./contracts/Comptroller.sol SHA3: edabf529d5b31e8301cd419a17e09e25bb8d74b09e9ee5c7609bd3f0d129fd49 File: ./contracts/ComptrollerInterface.sol SHA3: 71a061a85e77d187a7d6d0b15ea8ad3f0c6abba5dec858acc1522566a042ff6b File: ./contracts/ComptrollerStorage.sol SHA3: 0de329c685455e96e42e1dd0fbfe37706b5a6c31d9d5a126bec314caac978ae6 File: ./contracts/ErrorReporter.sol SHA3: b4ce2d31b59cb01e317ac33541b6f182e99360567c2ef670b630999ea1631a1b File: ./contracts/ExponentialNoError.sol SHA3: a3501b72e64e3328f81e625f297ea40c4038934d3a10ce4dae34eab5ba93997c File: ./contracts/Factories/JumpRateModelFactory.sol SHA3: 0c4a01252c1705f177006d4acbd2083b97eb7989650c14eba702fc5035caaa60 File: ./contracts/Factories/VTokenProxyFactory.sol SHA3: 13c71cb616758151f36f94488f636c8a2d2c795b3335caff2e0c2bcfee2d5888 File: ./contracts/Factories/WhitePaperInterestRateModelFactory.sol SHA3: 9c9c2297837d5404df86261bd0648bc63dd0e6830866216a9cb703690a3f36b2 File: ./contracts/Governance/AccessControlManager.sol SHA3: 79b1ea35e1a071fbe41dfbf789e6563803eec64f0a0e2566f173c4988510d280 File: ./contracts/InterestRateModel.sol SHA3: 99b31b19e1cc6068ef923ab081b62df8542a64eff54947f0327e75d79007ae9b

File: ./contracts/IPancakeswapV2Router.sol
SHA3:
4d681de4770be514a4f9016f5dfde8cb7c77bde1b9c85af500ac07e34930649d

File: ./contracts/JumpRateModelV2.sol
SHA3:
359e66b07c98f5128ed00cbde29a8d9234b2ff3a1e6f2bc32cf0b2e32140e2e9

File: ./contracts/Lens/PoolLens.sol
SHA3:
1fab08eacc38939fa0e6152672d430716724915a5e9e9c427aaca422ab43fcf0

File: ./contracts/MaxLoopsLimitHelper.sol
SHA3:
5fbbc40326d32d09207c8afaceced3d9a04bcc0a8f8ef158e7ebf37b9f5a92f2

File: ./contracts/Pool/PoolRegistry.sol
SHA3:
872192dd69a50d8dae59d2df59efefc7930e7cb85d77df2e9d630530bccbb139

File: ./contracts/Pool/PoolRegistryInterface.sol
SHA3:
61df21adfa2f51a0cc332b1e7d8101199b1fb5e96687fdfe7c578166970cec32

File: ./contracts/Proxy/UpgradeableBeacon.sol
SHA3:
03f62d614586e73156fce410a1dbd4b1e643fb1cb251b6f1bb092e35238979b9

File: ./contracts/Rewards/RewardsDistributor.sol
SHA3:
2ba29ee2171d845186f3baab783bd232dfd7451d1386c3c842d34e6d3f31b469

File: ./contracts/RiskFund/IProtocolShareReserve.sol
SHA3:
cb02c0976d343603cd26f492889d4da1fc8b69b102cb7ceb8e41cc14703bc238

File: ./contracts/RiskFund/IRiskFund.sol
SHA3:
dc8538d5e092d2821d9fcf038ec0285b96be21f83ac75bb353a943e10e42f52c

File: ./contracts/RiskFund/ProtocolShareReserve.sol
SHA3:
1dd6fb20fd920cf9620c5bb277f74345f950f3ef012b001f1dc90fc2e4b739cd

File: ./contracts/RiskFund/ReserveHelpers.sol
SHA3:
904af19c78be05cb12517e3be209841864417d096dc9d607faa8f80b3bf2cad5

File: ./contracts/RiskFund/RiskFund.sol
SHA3:
3968e943b0859ad8e41dea4237456ef71d44eecc9a9e11957e1e38cdeb5cb997

File: ./contracts/Shortfall/IShortfall.sol
SHA3:
e7583cfc82a4aeec84eec6e08ead2fa82fd04fbb90ba8cc61229b799013425d2

File: ./contracts/Shortfall/Shortfall.sol
SHA3:
a7dccd4aea2f9e99a9917f5ed58090979dfe23961b189502c013f26a71ba229a

File: ./contracts/VToken.sol
SHA3:
f231d99fb5995cac65e976c37acedb6b0705f3808f2024b2b7046aecdc43756a

	File: ./contracts/VTokenInterfaces.sol SHA3: c5c7d723e3f70ee41e5704d4c763b28b0c8a889e546bf89be6e19fed4ce7b66d File: ./contracts/WhitePaperInterestRateModel.sol SHA3: e13e3906a13fcae559993ba2326168b4adfa10a655bb455aa5672f35028d266e

Third review scope

Repository	https://github.com/VenusProtocol/isolated-pools
Commit	ddd4656d9221c29a7892c1c95a2e692ceb45d807
Whitepaper	Shared as a file
Functional Requirements	Shared as a file
Technical Requirements	https://github.com/VenusProtocol/isolated-pools
Contracts	File: ./contracts/BaseJumpRateModelV2.sol SHA3: 7a77425ac2b502ea3d44d7be2bad7415d80e696fed2aef432686fbd1e7f48c3b File: ./contracts/Comptroller.sol SHA3: 312868a9f026cba632b584b5c32c58429fb35f66ce86039a6ac2644b715cad1f File: ./contracts/ComptrollerInterface.sol SHA3: b62176aa3923cfd8d58f04a0a4dd388e24a15be4cac23297218fe5303e82e5c4 File: ./contracts/ComptrollerStorage.sol SHA3: 43624df79fb50afdb2cc1a02937f2aa6340091e89297d9488d182f93059a5af2 File: ./contracts/ErrorReporter.sol SHA3: 67715cd8b3b23e2e6e572f10b9bc9ca09a7517d835e8dacde7c4a5c48dcd18f0 File: ./contracts/ExponentialNoError.sol SHA3: a3501b72e64e3328f81e625f297ea40c4038934d3a10ce4dae34eab5ba93997c File: ./contracts/Factories/JumpRateModelFactory.sol SHA3: 7a284364d72fde0abc0a454797dc1cecb7cef3e66212a1ed014a1e2620e72966 File: ./contracts/Factories/VTokenProxyFactory.sol SHA3: 1abd5d0d57b5470b3ccfc25fe528239457f45569cb5fc3932c1f4f49b4766d15 File: ./contracts/Factories/WhitePaperInterestRateModelFactory.sol SHA3: 9c9c2297837d5404df86261bd0648bc63dd0e6830866216a9cb703690a3f36b2 File: ./contracts/Governance/AccessControlled.sol SHA3: b6cc176065ec9a587b0155d6336096e628ecf30862cfa90879f41e8aae94f6fd File: ./contracts/Governance/AccessControlManager.sol

	SHA3: 5c1b31c46638cc989fd7d1f1f159bbd083590d11ad19a9ec93ad1b6162e64ae4 File: ./contracts/Governance/IAccessControlManager.sol SHA3: 7f94f040d8d3d0291b159e6df35d4d0015212b8899d42d751fcb0cbca9e6fc1c File: ./contracts/InterestRateModel.sol SHA3: 99b31b19e1cc6068ef923ab081b62df8542a64eff54947f0327e75d79007ae9b File: ./contracts/IPancakeswapV2Router.sol SHA3: 4d681de4770be514a4f9016f5dfde8cb7c77bde1b9c85af500ac07e34930649d File: ./contracts/JumpRateModelV2.sol SHA3: 9827013dc6909d6ca488315c9faaf342e16a2f03f710f284014c5b6f96330e56 File: ./contracts/Lens/PoolLens.sol SHA3: 71d53457e47a08f8f89613715b534bd99736795f1220b058eb119aae66b8e128 File: ./contracts/MaxLoopsLimitHelper.sol SHA3: 5fbbc40326d32d09207c8afaceced3d9a04bcc0a8f8ef158e7ebf37b9f5a92f2 File: ./contracts/Pool/PoolRegistry.sol SHA3: 399a6566d1f1f8510b1c29b0c40f68dd97a324654bd4c72b44bcb8bb28ab124e File: ./contracts/Pool/PoolRegistryInterface.sol SHA3: 61df21adfa2f51a0cc332b1e7d8101199b1fb5e96687fdfe7c578166970cec32 File: ./contracts/Proxy/UpgradeableBeacon.sol SHA3: 03f62d614586e73156fce410a1dbd4b1e643fb1cb251b6f1bb092e35238979b9 File: ./contracts/Rewards/RewardsDistributor.sol SHA3: 63ab805c37957eae3e1c1c1054881cc0dc6718c5d37f42453be9614c759d6096 File: ./contracts/RiskFund/IProtocolShareReserve.sol SHA3: cb02c0976d343603cd26f492889d4da1fc8b69b102cb7ceb8e41cc14703bc238 File: ./contracts/RiskFund/IRiskFund.sol SHA3: dc8538d5e092d2821d9fcf038ec0285b96be21f83ac75bb353a943e10e42f52c File: ./contracts/RiskFund/ProtocolShareReserve.sol SHA3: f0f4ce63c58b5c369bd63e6fea7c03096ba8f84200d93cc176b0f25c76ce48b3 File: ./contracts/RiskFund/ReserveHelpers.sol SHA3: 904af19c78be05cb12517e3be209841864417d096dc9d607faa8f80b3bf2cad5 File: ./contracts/RiskFund/RiskFund.sol SHA3: 6a332d9b853bb3474ab1bdea928afd4aea48b008d6374e203aadc10e52514c9f File: ./contracts/Shortfall/IShortfall.sol

	SHA3: e7583cfc82a4aeec84eec6e08ead2fa82fd04fbb90ba8cc61229b799013425d2 File: ./contracts/Shortfall/Shortfall.sol SHA3: daa84d17912bddcc5ee6fb1f414f82bf06559dc07bc94c86809f89756c7c7803

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- NatSpecs are generally satisfactory.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Instead of using onlyOwner modifier, direct check is used.

Test coverage

Code coverage of the project is **73.87%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage are partially missing.
- Interactions with several users are not tested thoroughly.
- Some contracts are not fully tested.

Security score

As a result of the audit, the code contains **1** medium security issue. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.4**.

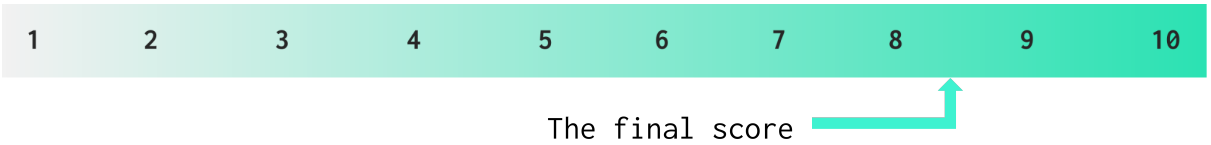

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
30 January 2023	14	12	10	0
1 March 2023	4	4	4	0
03 April 2023	0	1	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant

Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed

Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Venus Protocol (“Venus”) is an algorithmic-based money market system designed to bring a complete decentralized finance-based lending and credit system.
Venus enables users to utilize their cryptocurrencies by supplying collateral to the network that may be borrowed by pledging over-collateralized cryptocurrencies. This creates a secure lending environment where the lender receives a compounded interest rate annually (APY) paid per block, while the borrower pays interest on the borrowed cryptocurrency.
The Venus Protocol has been designed to give platform users a decentralized and secure marketplace to take out loans, earn an interest, and mint synthetic stablecoins.
The files tested in the audit’s scope:

- **BaseJumpRateModelV2.sol** - base contract for JumpRateModelV2.sol;
- **Comptroller.sol** - the Comptroller contract is the central contract for each lending pool. It contains functionality central to the borrowing activity in the pool, such as supplying and borrowing assets and liquidations. Configuration values for the pool, such as the liquidation incentive, close factor, and collateral factor, can be set and retrieved from the Comptroller. Account liquidity and positions can be retrieved from the Comptroller;
- **ComptrollerInterface.sol** - interface for Comptroller.sol;
- **ComptrollerStorage.sol** - storage for Comptroller.sol;
- **ErrorReporter.sol** - stores error codes for VToken.sol;
- **ExponentialNoError.sol** - exponential module for storing fixed-precision decimals;
- **JumpRateModelFactory.sol** - is a factory to deploy the jump rate interest rate model. When using this model, interest follows a linear curve until supply or demand reaches the kink, after which there is a steep increase in interest rates.
- **VTokenProxyFactory.sol** - when adding a market to pools, *VTokenProxyFactory* is deployed for each market. It is the token that represents the underlying supplied asset.
- **WhitePaperInterestRateModelFactory.sol** - is another interest rate model that can be deployed with a market. It is similar to the jump rate model but uses a base rate and does not include a kink.
- **AccessControlManager.sol** - grants account access to call specific functions on contracts. This contract is responsible for granting and revoking those permissions. It provides a getter to check if an address is allowed to call a specific function.
- **InterestRateModel.sol** - base interest rate model contract;
- **IPancakeswapV2Router.sol** - interface for interacting with PancakeswapV2Router;

- **JumpRateModelV2.sol** - compound's JumpRateModel Contract V2 for V2 vTokens;
- **PoolLens.sol** - to make querying pool data easier, Isolated Pools contain a lens that queries and formats pool data. These calls can be gas intensive, so the general rule of thumb is that this contract should not be used in transactions;
- **PoolRegistry.sol** - Creating and managing pools is done by PoolRegistry. It can add markets to pools, update pool metadata, and return pool information.
- **PoolRegistryInterface.sol** - interface for PoolRegistry.sol;
- **UpgradeableBeacon.sol** - used in conjunction with one or more instances of BeaconProxy to determine their implementation contract, which is where they will delegate all function calls;
- **RewardsDistributor.sol** - users are rewarded for borrowing and lending activities with a rewards token. RewardsDistributor manages these distributions using a configurable rate.
- **IProtocolShareReserve.sol** - interface for ProtocolShareReserve.sol;
- **IRiskFund.sol** -
- **ProtocolShareReserve.sol** - acts as a treasury where each isolated pool can transfer their revenue;
- **ReserveHelpers.sol** - stores additional functionality for ProtocolShareReserve.sol and RiskFund.sol;
- **RiskFund.sol** - lending comes with the inherent risk that borrowers will not be able to repay their loan, which is a threat to the protocol's insolvency. Venus V4 looks to mitigate this risk with a RiskFund. A percentage of the protocol’s revenues is transferred to the RiskFund. When bad debt is detected, this fund can be auctioned off and used to cover the bad debt;
- **Shortfall.sol** - when bad debt is auctioned off the Shortfall contract is responsible for running the action and paying the winner;
- **VToken.sol** - when a user supplies a token to the protocol, they are minted vTokens to represent their supply. The VToken contract contains methods that support lending activities for the asset including lending, borrowing and liquidating;
- **VTokenInterfaces.sol** - stores interfaces and storage variables for VToken.sol;
- **WhitePaperInterestRateModel.sol** - WhitePaperInterestRateModel is an interest rate model that can be deployed with markets . It is similar to JumpRateModel except it does not include a kink. Instead, it contains a fixed base rate.
- **AccessControlled.sol** - access control manager contract.
- **MaxLoopsLimitHelper.sol** - Limit for the loops to avoid the DOS.

Privileged roles

- VToken.sol:
  - owner - can set a new AccessControlManager and sweep accidental ERC-20 transfers to this contract;
  - shortfall - updates bad debt (Called only when bad debt is recovered from auction);
  - comptroller - can call the method healBorrow() which will repay a certain amount of debt, treat the rest of the borrow as bad debt, essentially "forgiving" the borrower and forceLiquidateBorrow() to liquidate the borrower's collateral;
  - AccessControlManager privilege roles:
    - setProtocolSeizeShare() method caller - can set protocol share accumulated from liquidations;
    - setReserveFactor() method caller - can set a new reserve factor for the protocol after accruing interest;
    - setInterestRateModel() method caller - can update the interest rate model after accruing interest;
- Comptroller.sol:
  - owner - can set the closeFactor to use when liquidating borrows, can add a new RewardsDistributor and initialize it with all markets, can set a new PriceOracle for the Comptroller;
  - vToken - allowed to call the method preBorrowHook() on borrows;
  - poolRegistry - can add the market to the markets mapping and set it as listed;
  - AccessControlManager privilege roles:
    - setCollateralFactor() method caller - can set collateralFactor for a market (collateralFactorMantissa - multiplier representing the most one can borrow against their collateral in this market and liquidationThresholdMantissa - multiplier representing the collateralization after which the borrow is eligible for liquidation);
    - setLiquidationIncentive() method caller - can set liquidationIncentive (representing the discount on collateral that a liquidator receives);
    - setMarketBorrowCaps() method caller - can set borrow caps for the given vToken markets.
    - setMarketSupplyCaps() method caller - can set the given supply caps for the given vToken markets
    - setActionsPaused() method caller - can pause/unpause specified actions;
    - setMinLiquidatableCollateral() method caller - can set the minimal collateral required for regular (non-batch) liquidations.

- ProtocolShareReserve:
  - owner - can release funds.
- RiskFund:
  - owner - can update pool registry address, can update convertible base asset, can update PancakeSwap router address and set min amount to convert;
  - shortfall - can transfer tokens for auction;
  - AccessControlManager privilege roles:
    - swapPoolsAssets() method caller - can swap an array of pool assets into the base asset's tokens of at least a minimum amount.
- PoolRegistry:
  - owner - The owner of the PoolRegistry contract has the capability to create a new pool and add a market to an existing pool. The owner can set the pool name and update metadata information.
- AccessControlManager:
  - owner - The owner of the AccessControlManager can grant and revoke the role.
- Shortfall:
  - owner - The owner of the Shortfall contract has the authority to specify the convertible base assets and the minimum pool bad debt variables. They can also set the address for the Pool Registry. Furthermore, the owner has the ability to initiate a new auction.

Risks

- This protocol is divided into 3 repos. This audit only covers one of them (isolated-pools). The other repos (concerning Oracles and Governance) and their interactions are not covered in this audit. **As the pools are interacting with the oracle part, any problem in the oracle repo will cause issues with the pools**.
- Reviewed contracts are upgradable but are **supposed to be used as a first implementation**.
- In VToken.sol, on every mint function preMintHook of the Comptroller contract is called. As preMintHook is called before the actual amount of tokens transferred is calculated it will not allow it to reach the actual max supply limit for tokens with a fee on transfers.

Recommendation

- The project uses OpenZeppelin's AccessControl and its own access control implementation. It is recommended to use only one of them for the entire project.

Findings

Critical

No critical severity issues were found.

High

H01. Highly Permissive Role Access

A malicious owner may alter the token address after bids have been placed, potentially causing the user with the highest bid to receive an undesired token.

Path: ./contracts/Shortfall/Shortfall.sol : setConvertableBaseAsset()

Recommendation: Prevent changing the address after bids are placed.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

H02. Access Control Violation

The closeAuction() function can be called by anyone for any auction. These functions allow users to close or restart any auction without checking for the caller's role.

This means that after placing a bid a malicious user can close the related auction or an auction can be restarted by the malicious bidder when the malicious bidder's amount is surpassed.

This can cause arbitrary parties to end or restart arbitrary auctions at any time.

Path: ./contracts/Shortfall/Shortfall.sol : closeAuction()

Recommendation: Implement access control to the issued functions.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

H03. Requirements Violation

The mathematical formulas for the following functions are different from the ones provided in the supplied documentation.

Paths: ./contracts/Shortfall/WhitePaperInterestRateModel.sol : getBorrowRate(), getSuppyRate()

./contracts/BaseJumpRateModelV2.sol : getBorrowRateInternal(), getSuppyRate()

Recommendation: Either update the documentation according to the implementation or vice versa.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

H04. Requirements Violation

It is possible to set close factor mantissa less than *closeFactorMinMantissa* and greater than *closeFactorMaxMantissa* as the method *setCloseFactor* does not limit the input parameters.

The code should not violate requirements provided by the Customer.

Path: ./contracts/Comptroller.sol : setCloseFactor()

Recommendation: Add a check for the *newCloseFactorMantissa* parameter.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

H05. Race Condition

Contract VToken.sol implements a standard *approve* method for the managing of the allowance. Changing an allowance with this method brings the risk that someone may use both the old and the new allowance via unfortunate transaction ordering.

It can lead to user fund loss.

Path: ./contracts/VToken.sol

Recommendation: Implement *decreaseAllowance* and *increaseAllowance* functions to mitigate the risk.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

H06. Undocumented Behavior

Method *setComptroller* allows the owner of the VToken.sol contract to change the actual Comptroller address.

According to provided documentation: “The Comptroller contract is the central contract for each lending pool. It contains functionality central to borrowing activity in the pool, such as supplying and borrowing assets and liquidations. Configuration values for the pool such as the liquidation incentive, close factor, and collateral factor can also be set and retrieved from the Comptroller.”

This behavior is not documented.

The code should not contain undocumented functionality.

Path: ./contracts/VToken.sol : function setComptroller()

Recommendation: Add documentation for the mentioned functionality or remove the possibility to change Comptroller and AccessControlManager from the VToken.sol contract.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

H07. Denial of Service - Loop Gas Limit

In Comptroller.sol, the functions *enterMarkets*, *setMarketBorrowCaps*, *setMarketSupplyCaps*, *setActionsPaused* loop over an array of VToken. These parameters are given to the function without any checks.

The functions *exitMarket*, *healAccount*, *liquidateAccount*, *_getHypotheticalLiquiditySnapshot* loop over an array of users' accountAssets without any checks.

The function *liquidateAccount* loops over an array of orders. These parameters are given to the function without any checks.

The functions *addRewardsDistributor*, *_addMarket* loop over the array allMarkets without any checks.

The functions *preMintHook*, *preRedeemHook*, *preBorrowHook*, *preRepayHook*, *preSeizeHook*, *preTransferHook*, *supportMarket* and *addRewardsDistributor* loop over an array of rewardsDistributor without any checks.

In RiskFund.sol, the function *swapPoolsAssets* loops over an array of underlyingAssets. These parameters are given to the function without any checks.

In RewardsDistributor.sol, the function *_claimRewardToken* loops over an array of VToken and over an array of holders. These parameters are given to the function without any checks.

In case of a very big array, this can lead to an out-of-gas exception, provoking a denial of service.

Paths: ./contracts/Comptroller.sol : enterMarkets(), exitMarket(), healAccount(), liquidateAccount(), addRewardsDistributor(), _addMarket(), _getHypotheticalLiquiditySnapshot()

./contracts/RiskFund/RiskFund.sol : swapPoolsAssets()

./contracts/Rewards/RewardsDistributor.sol : _claimRewardToken()

Recommendation: Set a limit on the array's length.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

H08. Access Control Violation

In VToken.sol, the function reduceReserves() reduces the contract's reserves by transferring them to the protocol reserve contract.

There is no access limitation to this function. This can allow a bad actor to manipulate the contract's reserves.

Path: ./contracts/VToken.sol

Recommendation: Access to functions with critical functionality should be limited.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

H09. Highly Permissive Role Access

In RewardsDistributor.sol, the owner can transfer available rewards tokens to any address with the function *grantRewardToken*. The owner can drain all reward tokens from the contract, making it impossible for the users to earn their rewards.

Path: ./contracts/RewardsDistributor.sol : grantRewardToken()

Recommendation: Owners should not have access to funds that belong to users. Either provide documentation about this functionality or remove it.

Status: Mitigated (The owner of the contract will be the Governance, so the function grantRewardToken will be executable only via VIP, with the votes of the community)

Medium

M01. Inefficient Gas Model - Uncontrolled Iterations

The numbers of iterations of the loop in the functions are uncontrolled as it depends on stored data and it makes external calls.

Path: ./contracts/Lens/PoolLens.sol : vTokenBalancesAll(), getAllPools(), vTokenUnderlyingPriceAll(), vTokenMetadataAll()

Recommendation: Implement loop length limitations.

Status: Reported.

M02. Contradiction - Requirement Contradiction

The requirement "require(bytes(name).length <= 100, "No pool name supplied.")" presents a contradiction with the stated requirement. An error explanation of "No pool name supplied." contradicts the requirement that the length of "bytes(name)" must not exceed 100.

Path: ./contracts/Pool/PoolRegistry.sol : _registerPool()

Recommendation: Either change the requirement statement or explanation.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M03. Contradiction - Missing Validation

The authorized address can set the supply cap to an amount that is lower than the token total supply.

Path: ./contracts/Comptroller.sol : setMarketSupplyCaps()

Recommendation: Implement checks which prevent the total cap to be lower than the total supply or document this possibility if it is a feature.

Status: Mitigated (The client states, Supply caps smaller than the current total supplies are accepted. This way, new supplies will not be allowed until the total supplies amount goes below the new supplycap)

M04. Inconsistent Data - Variable Is Not Limited

In Comptroller.sol, the function *setLiquidationIncentive* sets the new liquidation incentive without any restrictions. *(Fixed)*

Consider limiting the minLiquidatableCollateral values in order to prevent unexpected reverts. *(Mitigated)*

Path: ./contracts/Comptroller.sol : setMinLiquidatableCollateral()

Recommendation: Provide conscious limits for stored configuration values.

Status: Mitigated (The client states the minLiquidatableCollateral value will be set via the governance with the user votes.)

M05. Inconsistent Data - Unused Return Value

The function *supportMarket* performs a call to isToken() but ignores the return value.

The function *closeAuction* performs a call to *transferReserveForAuction*() but ignores the return value.

Paths: ./contracts/Comptroller.sol : supportMarket()

./contracts/Shortfall/Shortfall.sol : closeAuction()

Recommendation: Implement return value checks.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M06. Best Practice Violation - Check Effects Interaction Pattern Violation

The createRegistryPool() function updates state variables after external calls.

Path: ./contracts/Pool/PoolRegistry.sol : createRegistryPool()

Recommendation: Implement the function according to the Checks-Effects-Interaction pattern or use Reentrancy locks.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M07. Contradiction - NatSpec Contradiction

In the NatSpec, it is stated that the _poolsByID variable is an array that contains Venus pool comptroller addresses. However, it is a mapping.

Path: ./contracts/Pool/PoolRegistry.sol

Recommendation: Update the NatSpec or change the implementation.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M08. Contradiction - Name Contradiction

The contracts are defined as abstract, but filename includes interface keywords. Contract names should represent contract logic and should not be misleading.

Paths: ./contracts/Pool/PoolRegistryInterface.sol,

./contracts/ComptrollerInterface.sol

Recommendation: Define contracts with an interface keyword.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M10. Inconsistent Data - Missing Event for Critical Value Updates

Critical state changes should emit events for tracking things off-chain.

Otherwise, iIt can lead to inability for users to subscribeto events and check what is going on with the project.

Paths: ./contracts/VToken.sol : functions healBorrow(), sweepToken()

./contracts/RiskFund/RiskFund.sol : functions swapPoolsAssets(), transferReserveForAuction()

./contracts/RiskFund/ProtocolShareReserve.sol : function releaseFunds()

./contracts/Rewards/RewardsDistributor.sol : functions initializeMarket(), updateContributorRewards(), _updateRewardTokenSupplyIndex()

./contracts/Comptroller.sol : functions supportMarket(), addRewardsDistributor()

Recommendation: Emit events on critical state changes.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M11. Contradiction - Non-Finalized Code

The production code contains variables and functionality which is unused.

RewardsDistributor.sol contains a TODO comment.

The code should not contain TODO comments. Otherwise, it means that the code is not finalized, and additional changes will be introduced in the future.

In ComptrollerStorage.sol l 71, there is the following comment :

- “// NOTE: please remove this as it is not used anymore”.

The variables transferGuardianPaused, seizeGuardianPaused, mintGuardianPaused, borrowGuardianPaused and borrowCapGuardian are never used. The associated event NewBorrowCapGuardian is never used either.

The variable maxAssets is never assigned and can only be viewed.

Paths: ./contracts/Comptroller.sol : NewBorrowCapGuardian

./contracts/Rewards/RewardsDistributor.sol

./contracts/ComptrollerStorage.sol : transferGuardianPaused, seizeGuardianPaused, mintGuardianPaused, borrowGuardianPaused, borrowCapGuardian, maxAssets

Recommendation: Finalize code.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

M12. Contradiction - NatSpec Comments Contradiction

According to the NatSpec documentation: “The vToken must handle variations between ERC-20 and ETH underlying”. Version of VToken.sol under audit handle only ERC-20.

It can lead to incorrect assumptions about the code's purpose.

In Comptroller.sol, the natSpec says :

- @title Compound's Comptroller Contract
- @author Compound.

But the contract has been heavily modified. It should be specified.

Paths: ./contracts/VToken.sol : functions _mintFresh(), _redeemFresh(), _borrowFresh(), _repayBorrowFresh()

./contracts/Comptroller.sol

Recommendation: Fix the mismatch.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

Low

L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: ./contracts/ComptrollerInterface.sol

./contracts/InterestRateModel.sol

./contracts/IPancakeswapV2Router.sol

./contracts/JumpRateModelV2.sol

./contracts/VTokenInterfaces.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L02. Redundant Import

The usage of ClonesUpgradeable.sol, VTokenInterfaces.sol, ComptrollerInterface.sol, JumpRateModelV2.sol is unnecessary in the contract.

Unused imports should be removed from the contracts. Unused imports are allowed in Solidity and do not pose a direct security issue. However, it is best practice to avoid them as they can decrease readability.

The RiskFund contract imports "../Pool/PoolRegistry.sol", "../Pool/PoolRegistryInterface.sol".

Paths: ./contracts/Pool/PoolRegistry.sol

./contracts/RiskFund/RiskFund.sol

Recommendation: Remove the redundant import.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

L03. Missing Zero Address Validation

Address parameters are used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths: ./contracts/Pool/PoolRegistry.sol: addMarket(), setPriceOracle()

./contracts/Proxy/UpgradeableBeacon.sol: constructor()

./contracts/JumpRateModelV2.sol: constructor()

./contracts/Shortfall/Shortfall.sol : initialize()

./contracts/BaseJumpRateModelV2.sol : constructor()

./contracts/Comptroller.sol : constructor()

./contracts/RiskFund.sol : initialize(_accessControl)

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L05. Contradiction

The contract name is defined as *ComptrollerV1Storage*, but the filename is *ComptrollerStorage.sol*.

Path: ./contracts/ComptrollerStorage.sol

Recommendation: Rename the contract.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L06. Division by Zero

It is possible to perform zero divisions while calculating the utilization rate and multiplierPerBlock while updating the Jump Rate model,

Path: ./contracts/BaseJumpRateModelV2.sol : updateJumpRateModel(), utilizationRate()

Recommendation: Implement a check to prevent zero division.

Status: Mitigated (The client stated that there is no need for a specific error message.)

L07. Style Guide Violation

The provided projects should follow the official guidelines.

Inside each contract, library, or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)

4. external
5. public
6. internal
7. private

Within a grouping, place the view and pure functions last.

It is best practice to cover all functions with NatSpec annotation and to follow the Solidity naming convention. This will increase overall code quality and readability.

Path: ./contracts/

Recommendation: Follow the official Solidity guidelines.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L08. Functions that Can Be Declared External

“public” functions that are never called by the contract should be declared “external” to save Gas.

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths: ./contracts/VToken.sol : functions initialize(), borrowBalanceStored(), exchangeRateStored(), setComptroller(), setInterestRateModel()

./contracts/Comptroller.sol :setPriceOracle(), getAccountLiquidity(), getHypotheticalAccountLiquidity(), getAllMarkets(), isMarketListed()

./contracts/Rewards/RewardsDistributor.sol : functions initialize(), setRewardTokenSpeeds(), setContributorRewardTokenSpeed(), distributeSupplierRewardToken(), claimRewardToken(address holder)

./contracts/RiskFund/ProtocolShareReserve.sol : functions initialize()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

L09. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/Comptroller.sol : addRewardsDistributor()

Recommendation: Remove boolean equality.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L10. Variable Shadowing

Comptroller.liquidateAccount().markets shadows :

- ComptrollerV1Storage.markets

Path: ./contracts/Comptroller.sol

Recommendation: Rename related variables/arguments.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L11. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Path: ./contracts/VTokenInterface.sol : Mint, Redeem, Borrow, RepayBorrow, BadDebtIncreased, BadDebtRecovered, LiquidateBorrow,

Recommendation: Use the “indexed” keyword for the event parameters.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L12. Contract Should Be a Library

ExponentialNoError.sol should be a library. Some functions are not used in the code.

Path: ./contracts/ExponentialNoError.sol

Recommendation: Change it as a library or remove the unused functions.

Status: Fixed (Revised commit: ddd4656d9221c29a7892c1c95a2e692ceb45d807)

L13. Redundant Use

Unnecessary variable usage will increase Gas consumption of the code. Thus they should be removed from the code.

Path: ./contracts/Comptroller.sol : payer(L 423), repayAmount(L 424), liquidator(L 473), seizeTokens(L 536)

Recommendation: Remove unnecessary usage.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

L14. Use of Hard-Coded Values

Hard-coded values are used in computations and are not documented.

Path: ./contracts/RiskFund/ProtocolShareReserves.sol : releaseFunds() L 55 (70)

Recommendation: Convert these values into constants and explain their meaning.

Status: Fixed (Revised commit: 3ad3daa6ef5abf4bcc83ae5df7baa5cf6e186ea7)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.