HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Dorado Holding Limited
Date: January 26, 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 Dorado Holding Limited
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 tokens; Staking; Vesting; DEX (Margin Trading, Order Book)
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.vela.exchange/
Changelog	14.11.2022 - Review #1 30.11.2022 - Review #2 28.12.2022 - Review #3 06.01.2023 - Review #4.1 13.01.2023 - Review #4.2 26.01.2023 - Review #5

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 Dorado Holding Limited (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

Review #1 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	D3defe4d6b62b48fdb9d299d72c9f020761aa4a0 (19 Oct 2022)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/HbajZQp8nE2WzkXt1zhK/ (21 Oct 2022)
Contracts	File: contracts/**/*.sol

Review #2 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	4c64473e39bcdfd54aadb65cfe60d702a51cd07b (17 Nov 2022)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/Ga5jDzCyDWh4GHOFBuml/ (17 Nov 2022)
Contracts	File: contracts/**/*.sol

Review #3 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	86ed413f79701bdeccbdb193835492c98662bfa7 (12 Dec 2022)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/572iDPvehIVzwPTc531b/ (12 Dec 2022)
Contracts	File: contracts/**/*.sol

Review #4.1 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	90d7913b36ca9b4f5bb00afd9e66cc66fca3e979 (04 Jan 2023)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/AjrCo1QmEQ3fKoucIokD/ (04 Jan 2023)

Contracts	File: contracts/**/*.sol

Review #4.2 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	81c3213e9f3da9fceff6b78ae6a23fc776709ca9 (13 Jan 2023)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/qF45vljTvy9uJk4psY88/ (14 Jan 2023)
Contracts	File: contracts/**/*.sol

Review #5 scope

Repository	https://github.com/VelaExchange/vela-contracts
Commit	2fcde6212f48fdd51b5d7c2e68883d12b347d27f (22 Jan 2023)
Documentation	https://app.gitbook.com/s/LupLtausi1b6bKXJPlbN/~/revisions/gAQURW3OPuNVbuDu8g5W/ (25 Jan 2023)
Contracts	File: contracts/**/*.sol

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 **9** out of **10**.

- There is no NatSpec, but the main documentation book has a similar contract/functions description.
- The doc is written mostly in a conversational style, sometimes lacks formality and rigor, but the overall level of detail is satisfactory.

Code quality

The total Code Quality score is **4** out of **10**.

- Coding best practices are violated, there are cases of: naming convention violation, member order convention violation, missing visibility/mutability modifiers, Solidity type system is not properly utilized.
- Particular examples are listed in Low issues.

Test coverage

Test coverage of the project is **92.89%** (branch coverage).

- Some cases are not tested.
- Each test file is exactly one test, even though it looks like there are many small tests inside a file; the small tests share the same state within a file; therefore, they cannot be considered as separate tests.
- There are invocations of code whose effects are not checked (one way or another) in the tests.
- Only a narrow basic set of parameters is tested for a given functionality.

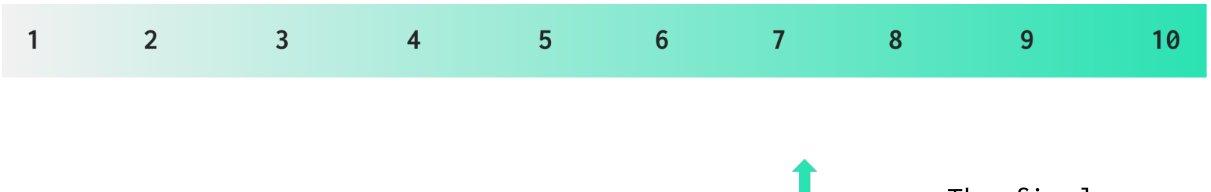
Security score

As a result of the audit, the code contains **0** critical, **0** high, **2** medium and **12** low severity issues. The security score is **8** 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: **7.1**.



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

Table. The distribution of issues during the audit

Review #	Low	Medium	High	Critical
1	15	3	14	3
2	9	0	8	0
3	9	0	5	1
4.1, 4.2	22	9	28	0
5	12	2	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more 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.	Failed
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.	Not Relevant
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.	Failed

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	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	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.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users. **Hacken Comment: users' funds can be locked or burned by the project privileged roles. Affected assets: vUSDC, esVELA (Escrowed VELA), VLP (Vela LP). VELA (Vela) token could not be burned or locked from a specific account.**	Failed
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.	Not Relevant
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.	Failed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
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 100%, 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

Vela - is a margin trading platform, with the core functionality implemented in smart contracts. Tokens are deposited and converted to an internal pseudo-USD token called vUSD. vUSD is the ubiquitous currency in the trading subsystem: all token trading pairs are against vUSD, user balances are in vUSD. Withdrawing funds from trading entails converting vUSD to any registered token of choice. Registered tokens - are those for which there is price feed in the oracle. The price oracle is managed exclusively by Vela.

In trading, there is the concept of “position”, which is a tuple of:
- Token to trade
- Whether to trade as long or as short
- The address of the trader

Each position has its unique ID, positions with the same parameters have different IDs.

The core trading functionality boils down to 2 functions:
- Add funds to a position at the current oracle price aka “increase position”. Each addition specifies the amount of collateral added, and the amount borrowed.
- Remove some or all funds from a position known as “decrease position”. The gain/loss is materialized against the current oracle price.

On top of the core functionality, there is the order book with common order features like “limit” and ”stop” prices, “take profit” and ”stop loss” prices with configurable position decrease percentages.

Vela has vesting and staking - it allows a user to temporarily lock up funds in exchange for gradually received rewards. Staking some tokens decreases the trading fees for a staker. Vela has a set of own tokens, which are distributed as rewards or given in exchange for locking other tokens in the platform.

More details are in the documentation (most recent publicly available version): https://vela-exchange.gitbook.io/vela-knowledge-base/

Contracts Summary

- **access/Governable.sol** - Common ‘gov’ role implementation. The role can be transferred.
- **access/Constants.sol** - All the constants in the project.
- **core/Multicall.sol** - A library code for atomically executing multiple contract calls, unused.
- **core/PriceManager.sol** - The highest-level price data facade, delegates to `VaultPriceFeed` internally.

- **core/SettingsManager.sol** - The host of most configuration parameters, used in different contracts.
- **core/TriggerOrderManager.sol** - The API/implementation for “take profit” and “stop loss” prices.
- **core/Vault.sol** - Core trading functionality (increase/decrease a position), position management, staking (a registered token in, VLP out).
- **core/VaultPriceFeed.sol** - Aggregate price oracle, composed of individual token price oracles.
- **core/VaultUtils.sol** - A collection of internal utilities used by *Vault*.
- **oracle/FastPriceFeed.sol** - A simple price get/set contract i.e. a price oracle contract.
- **staking/ComplexRewardPerSec.sol** - A utility contract that implements staking reward distribution for *TokenFarm*.
- **staking/TokenFarm.sol** - Vesting and staking.
- **tokens/BaseToken.sol** - Abstract ERC20 token contract with mint/burn functionality (descendants decide on whether they will use mint/burn).
- **tokens/MintableBaseToken.sol** - Abstract ERC20 token contract with mint/burn functionality. Introduces ‘minter’ role that can mint/burn.
- **tokens/eVela.sol** - Extends *MintableBaseToken*.
- **tokens/VLP.sol** - Extends *MintableBaseToken*.
- **tokens/vUSDC.sol** - Mintable and burnable token contract (not transferable).
- **tokens/VELA.sol** - ERC20 mintable token. Implements EIP-2612, EIP-2771. Supports pausing and unpausing (pausing blocks all transfers). Has a function to withdraw any ERC20 token accidentally sent to the contract itself.

Privileged Roles

- **core/PriceManager.sol**
  - owner: *setTokenConfig(..)*
- **core/SettingsManager.sol**
  - gov: *setFeeManager(..), setVaultSettings(..), setCloseDeltaTime(..), setCustomFeeForUser(..), setDelayDeltaTime(..), setDepositFee(..), setEnableDeposit(..), setFundingInterval(..), setFundingRateFactor(..), setLiquidateThreshold(..), setLiquidationFeeUsd(..), setMarginFeeBasisPoints(..), setMaxBorrowAmountPerAsset(..), setMaxBorrowAmountPerSide(..), setMaxBorrowAmountPerUser(..), setPositionManager(..), setReferEnabled(..), setReferFee(..), setStakingFee(..)*
- **core/Vault.sol**

    - owner: *setVaultSettings(..)*
    - manager known as “position manager”: *confirmDelayTransaction(..), triggerPosition(..), updateTrailingStop(..)*
- **core/VaultPriceFeed.sol**
    - gov: *setTokenConfig(..)*
- **oracle/FastPriceFeed.sol**
    - gov: *setAdmin(..)*
    - admin: *setDescription(..), setLatestAnswer(..)*
- **staking/ComplexRewardPerSec.sol**
    - owner: *add(..), addRewardInfo(..), emergencyRewardWithdraw(..), emergencyWithdraw(..)*
- **staking/TokenFarm.sol**
    - owner: *add(..), set(..), updateCooldownDuration(..), updateRewardTierInfo(..), updateVestingDuration(..)*
- **tokens/BaseToken.sol**
    - gov: *setInfo(..)*
- **tokens/MintableBaseToken.sol**
    - gov: *setInfo(..), setMinter(..)*
    - minter: *burn(..), mint(..)*
- **tokens/eVela.sol**
    - gov: *setInfo(..), setMinter(..)*
    - minter: *burn(..), mint(..)*
- **tokens/VLP.sol**
    - gov: *setInfo(..), setMinter(..)*
    - minter: *burn(..), mint(..)*
- **tokens/vUSDC.sol**
    - gov: *setInfo(..), addAdmin(..)*
    - admin: *burn(..), mint(..)*
- **tokens/VELA.sol**
    - DEFAULT_ADMIN_ROLE: *disableMetaTxns(..), enableMetaTxns(..)*
    - MINTER_ROLE: *mint(..)*
    - PAUSER_ROLE: *pause(..), unpause(..)*
    - RESCUER_ROLE: *rescueTokens(..)*

Risks

- Prices are completely controlled by the platform. Abuse or software glitches are possible.
- Most of the contracts have ‘gov’ (inherited from *contracts/access/Governable.sol*) and/or ‘owner’ (inherited from *@openzeppelin/contracts/access/Ownable.sol*) role, which has exclusive access to some functions, and is meant to be assigned to some platform-owned address(es). In each contract, ‘gov’ and ‘owner’ can

be assigned to only one address at a time. If the address is an externally-owned account, it means there is a single private key to which the role is assigned, and this key is potentially copied many times across computers in the platform's backend infrastructure (e.g. for horizontally scaling the blockchain clients). It may happen that the deployment model is such that the same private key is made 'owner' or 'gov' in all contracts.

'owner' or 'gov' could also be assigned to smart contracts. The deployment/configuration is out of scope of this audit; nevertheless, we note that it would be the most secure to assign 'owner' and 'gov' to some smart contracts which allow advanced key-management techniques like hierarchical roles/keys, multisig. If 'owner' or 'gov' are simply assigned to externally-owned accounts, there is an increased risk of the private key(s) loss or theft, with severe consequences. Both 'owner' and 'gov' can be transferred (by the bearer of the role); therefore, when the contracts are initially deployed with the naive EOA key-management approach, it could be upgraded to a more robust scheme later.

It is worth noting that in *contracts/oracle/FastPriceFeed.sol* there is no 'gov' transfer mechanism. Instead, every useful action can only be done by an 'admin', and 'gov' can only add/remove 'admin's. Therefore, it is possible to perform the upgrade to a smart contract described above by granting 'admin' role to the contract; after this kind of upgrade, it would be best to destroy the original 'gov' private key.

- An increase of an open position via *Vault.confirmDelayTransaction(..)* is executed at the current oracle price, no matter what it is. It is dangerous for users, because the price may significantly change between the moment the transaction is created locally by a user, and the moment it is actually executed on the chain. The usual remedy for this is to have some "slippage tolerance" setting for a user e.g. if the oracle price at the moment of execution differs more than N% (with respect to the locally observed market price) in the unfavorable direction - do not execute the order; this feature is implemented for "market" position opening order type - but not for increasing an already opened position.
- By design (check the documentation), platform's high-privilege roles may manipulate (as well as a block) users' funds, and supplies of tokens.
- The frontrunning-protecting delay is only applied to increases of already opened positions. It is possible to open a position, and thus call *Vault._increasePosition(..)* without a delay. This opens a front-running possibility, mitigated by the fact that there is a long delay to close a profitable position since the opening (no delay if the profit is 2%+), which makes it risky to chase a price increase despite the assumed time edge over the platform.

- The actual deployment/configuration of the contracts is out of the scope of audit. The contracts have limited mutability, but the initial state should be checked against one’s expectations.

Findings

■■■■ Critical

1. Access Control Violation

In the *VaultUtils* contract, the functions *takeVUSDIn(..)* and *takeVUSDOut(..)* allow to mint and burn vUSDC tokens. These functions are supposed to be accessed only by the contract *Vault* but are actually accessible by anybody. Therefore, anybody can mint vUSDC tokens for free.

Path:

- *contracts/core/VaultUtils.sol:takeVUSDIn(..),takeVUSDOut(..)*

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

Found At: Review #1

Status: Fixed (Review #2)

2. Access Control Violation

In *withdrawFees(..)*, anybody can withdraw (e.g. to their benefit) the value of vUSD from the contract in the equivalent amount of a given token (if it is registered in *PriceFeed*)

Path:

- *contracts/core/Vault.sol*

Recommendation: Access to functions with critical functionality should be limited. Contracts that are used only for test purposes should be excluded from the scope.

Found At: Review #1

Status: Fixed (Review #2)

3. Access Control Violation

In *Token*, anybody can mint tokens for free with no access restriction. However, it is not clear if this contract will be used in production.

Path:

- *contracts/tokens/Token.sol:mint(..),withdrawToken(..)*

Recommendation: Access to functions with critical functionality should be limited. Contracts that are used only for test purposes should be excluded from the scope.

Found At: Review #1

Status: Fixed (Review #2)

4. Invalid Calculations

Assume `Vault.totalVLP != 0`. When a user calls `Vault.stake(..)` for the first time, when the execution gets inside `Vault._updateReward(..)`, `rewardAmount` will be 0, because `user.amount` will be zero (it is increased after this function is called). Therefore, line 627 will not be executed, and `user.lastFeeReserves` will be left as 0. The second time the user calls `Vault.stake(..)`, `feeReserves.sub(user.lastFeeReserves)` at line 621 will be equal to `feeReserves` i.e. the user will claim the reward as if he were a staker from the very beginning (when feeReserves were 0).

Disclaimer: It is likely that this behavior was not intended. The correct behavior does not follow the provided documentation.

Path:

- `contracts/core/Vault.sol:_updateReward(..)`

Recommendation: Review the algorithm and test it thoroughly to be sure it is working as intended.

Found At: Review #3

Status: Fixed (Review #4.1)

■■■ High

1. Funds Lock

Native coins and tokens should have mechanisms of their withdrawal from the contract if they are accepted by the contract.

Path:

- `contracts/core/Router.sol`

Recommendation: Implement withdrawal mechanism of native coin and clarify the purpose of this receive function.

Found At: Review #1

Status: Fixed (Review #2)

2. Invalid Hardcoded Value

The `getTier(..)` function returns the fee percentage with discount depending on the amount staked on the contract. If the amount staked is between 100 000 and 250 000, the function returns 20 instead of 100 - 20.

Path:

- `contracts/staking/TokenFarm.sol:getTier(..)`

Recommendation: Hardcoded values should be correct.

Found At: Review #1

Status: Fixed (Review #2)

3. **Funds Lock Possibility**

   If the transfer at *contracts/staking/TokenFarm.sol:552* fails, the due redemption of the vested amount is blocked. It is not guaranteed that the contract will always have enough of *claimableToken* to support that transfer.

   Because the transfer of *claimableToken* is a bonus for vesting, and the primary amount that is expected to be returned in the vesting withdrawal is the unlocked amount of previously vested token, there are reasons to think that the failure to transfer *claimableToken* should not block the primary transfer.

   Path:

   - *contracts/staking/TokenFarm.sol:_claim(..)*

   Recommendation: Document this possibility or make the failure of the transfer of *claimableToken* non-critical to the success of the vesting claim transaction.

   Found At: Review #1

   Status: Fixed (Review #5)

4. **Highly Permissive Role Access**

   In *BetaTrading* *owner* can burn vUSDC tokens from any user without permission using the *upgradeBetaTrading(..)* function.

   Path:

   - *contracts/core/BetaTrading.sol*

   Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

   Found At: Review #1

   Status: Fixed (Review #2)

5. **Highly Permissive Role Access**

   An “admin” of vUSDC can burn tokens arbitrarily. It is possible to burn user’s tokens without user allowance.

   A “minter” of *MintableBaseToken* can burn tokens arbitrarily. It is possible to burn user’s tokens without user allowance.

   VLP and eVELA are implementing *MintableBaseToken*. Therefore, this issue applies for these two tokens.

   Paths:

   - *contracts/tokens/MintableBaseToken.sol*
   - *contracts/tokens/vUSDC.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #4.1)

6. **Highly Permissive Role Access**

   An “admin” of vUSDC can mint tokens arbitrarily. It is possible to mint vUSDC and use them in the *Vault* contract to retrieve other users' funds through the withdraw function.

   Path:

   - *contracts/tokens/vUSDC.sol*

   Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

   Found At: Review #1

   Status: Fixed (Review #4.1)

7. **Highly Permissive Role Access**

   The owner can create pools with arbitrary staking/reward tokens, which in turn may overlap with some tokens that the contract has funds in. By using rescueFunds the owner can withdraw the whole contract’s balance of any token (for example, everything that users staked) to their benefit.

   Path:

   - *contracts/staking/TokenFarm.sol*

   Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

   Found At: Review #1

   Status: Fixed (Review #3)

8. **Highly Permissive Role Access**

   The owner can set an arbitrarily high *unbondingPeriod*, which would block “unlocked” users’ withdrawals.

   Path:

   - *contracts/staking/TokenFarm.sol*

   Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

   Found At: Review #1

   Status: Fixed (Review #3)

9. Highly Permissive Role Access

vlpRate in *Vault* can be changed freely by the owner. It could be set to small/big values to the owner's benefit, as well as to 0 to block the *unstake(..)* function.

Path:

- *contracts/staking/Vault.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #3)

10. Highly Permissive Role Access

The owner can set *vaultUtils* arbitrarily. It is possible for the owner to set an implementation of *IVaultUtils* that significantly affects the logic concerning users' funds.

Path:

- *contracts/core/Vault.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #2)

11. Highly Permissive Role Access

An admin may use *recoverClaim(..)* to harvest rewards of any account in their favor.

Path:

- *contracts/tokens/YieldToken.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #2)

12. Highly Permissive Role Access

The owner can use *setTokenConfig(..)* to manipulate *tokenDecimals* or *minProfitBasisPoints* for its own benefit or to harm users financially.

Path:

- *contracts/core/Vault.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #3)

13. Highly Permissive Role Access

The owner can use *setRewardRate(..)* to set the fees arbitrarily high/low, for own benefit or at the expense of normal users.

Path:

- *contracts/core/Vault.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #4.1)

14. Highly Permissive Role Access

The owner can use *setFees(..)* to set *minProfitTime* so high that it would delay trading profits effectively forever. To ensure this, the owner can use *setTokenConfig(..)* to set *minProfitBasisPoints* to a high enough value.

Path:

- *contracts/core/Vault.sol*

Recommendation: Owners should not have access to funds that belong to users or provide specific documentation.

Found At: Review #1

Status: Fixed (Review #3)

15. Non-Finalized Code

The production code should not contain any functions or variables that are being used solely in the test environment. This will allow malicious parties to manipulate the code or users to trigger them accidentally.

Path:

- *contracts/staking/TokenFarm.sol:getTierTest(..)*

Recommendation: Remove all the code that is not used in production.

Found At: Review #2

Status: Fixed (Review #3)

16. Invalid Calculations

If *tierLevels.length > 2*, the code will never use *tierPercents[tierPercents.length - 2]* (second last tier percent).

However, any possible desired result of *getTier(..)* can be achieved by setting the right values with *updateRewardTierInfo(..)*.

The problem is that those values need to respect the unconventional layout dictated by the algorithm, which can lead to unintended and unpredictable behavior.

Note: *updateRewardTierInfo* ensures that *tierLevels.length == tierPercents.length*.

Path:

- *contracts/staking/TokenFarm.sol:getTier(..)*

Recommendation: Review the algorithm and test it thoroughly to be sure it is working as intended.

Found At: Review #2

Status: Fixed (Review #4.1)

17. Requirements Violation

The documentation says that "$eVELA is neither tradable nor transferable." but the eVela.sol implements a *transfer(..)* function and a *transferFrom(..)* function through BaseToken.sol.

Path:

- *contracts/tokens/eVela.sol*

Recommendation: The code should not violate requirements provided by the Customer.

Found At: Review #3

Status: Fixed (Review #4.1)

18. Non-Finalized Code

The production code should not contain any functions or variables that are being used solely in the test environment. This will allow malicious parties to manipulate the code or users to trigger them accidentally.

Path:

- *contracts/core/Vault.sol: Hardhat console import*

Recommendation: Remove all the code that is not used in production.

Found At: Review #3

Status: Fixed (Review #4.1)

19. Invalid Logic

The condition of the check at line 484 is equivalent to: `!pool.enableLock || user.status != Status.UNLOCKED || user.endTimestamp <= block.timestamp`

Consequently, if the lock is enabled, and a user is locked, they can withdraw.

Disclaimer: In the previously audited version of the code, this was not allowed, therefore we raise this as a likely issue. The locking logic is not described in the documentation attached to the audit.

Path:

- `contracts/staking/TokenFarm.sol:_withdraw(..)`

Recommendation: Review the code fragment and test it thoroughly to be sure it is working as intended.

Found At: Review #3

Status: Fixed (Review #4.1)

20. Funds Draining Possibility

Anybody can generate an endless number of new accounts for the sake of calling `BetaTrading.claim(..)`, and getting vUSD. If `claimAmount` is small enough, it may be not financially viable (due to gas fees) to perform the attack; there are no definitions in the code or documentation that specify what `claimAmount` is going to be. Even for moderately low values of `claimAmount`, the attack could be performed despite the gas costs to drain the beta wallet and deny the service for honest beta users.

Path:

- `contracts/core/BetaTrading.sol:claim(..)`

Recommendation: The giveaway amounts should be distributed over identities that have been verified (website registration, email verification, anti-bot verification, etc.).

Found At: Review #4.1

Status: Fixed (Review #5)

21. Incorrect Logic (Requirements Violation)

The check at `contracts/core/TriggerOrderManager.sol:202` should be:

`_isLong && (_tpTriggeredAmounts[i] != 0 || price < _tpPrices[i])`

The check at `contracts/core/TriggerOrderManager.sol:204` should be:

`!_isLong && (_tpTriggeredAmounts[i] != 0 || price > _tpPrices[i])`

The check at `contracts/core/TriggerOrderManager.sol:209` should be:

`_isLong && (_slTriggeredAmounts[i] != 0 || price > _slPrices[i])`

The check at `contracts/core/TriggerOrderManager.sol:211` should be:

`!_isLong && (_slTriggeredAmounts[i] != 0 || price < _slPrices[i])`

Path:

- `contracts/core/TriggerOrderManager.sol:validateTriggerOrdersData(..)`

Recommendation: Fix the logical expressions.

Found At: Review #4.1

Status: Fixed (Review #5)

22. Invalid Hardcoded Value (Requirements Violation)

The check at `contracts/core/VaultUtils.sol:361` should be:

`_triggerPrices[2] == 1`

or

`_triggerPrices[2] == TRAILING_STOP_TYPE_PERCENT`

Path:

- `contracts/core/VaultUtils.sol:361`

Recommendation: Fix the hardcoded value, use variables/constants instead of hardcoded literals.

Found At: Review #4.1

Status: Fixed (Review #5)

23. Incorrect Logic (Requirements Violation)

The check at `contracts/core/VaultUtils.sol:416` should be:

`!_isLong && order.stpPrice >= price`

Path:

- `contracts/core/VaultUtils.sol:validateTrigger(..)`

Recommendation: Fix the condition.

Found At: Review #4.1

Status: Fixed (Review #5)

24. Missing Access Control

The function `contracts/core/Vault.sol:deposit(..)` can be called by anybody in someone's name (by providing someone's address as the first parameter). The function will fail if the address from the

first parameter did not provide enough of ERC20 allowance to Vault contract.

Nevertheless, the ERC20-allowance-based authorization is not enough here to prevent exploitation.

For an externally-owned account (which is the case for many if not most of Vela users), calling *Vault:stake(..)* or *Vault:deposit(..)* requires 2 separate transactions: allowance-creating transaction, and the transaction that does the actual call to the contract; these two steps can be done atomically in single transaction only via a special-purpose intermediary smart contract.

There's another function *Vault:stake(..)* in the contract, which is similar to *Vault:deposit(..)* in the way it was described.

An attacker could monitor the allowance-creating transactions of Vela users, and issue calls to *Vault:deposit(..)* or *Vault:stake(..)* in their name with the goal of calling the function that a user did not intend to call.

In result, users' funds would be burned for something they did not want, and they would be denied the service they wanted.

This attack could be executed despite the Gas costs with the goal of discrediting Vela.

Paths:

- *contracts/core/Vault.sol:deposit(..)*
- *contracts/core/Vault.sol:stake(..)*

Recommendation: Implement a proper access control.

Found At: Review #4.1

Status: Fixed (Review #5)

25. Requirements Violation

The calculation at *contracts/core/SettingsManager.sol:266* does not correspond to the documentation, which states the formula:

*fundingRate = fundingRateFactor * ((LongOI - ShortOI) / (LongOI + ShortOI))*

Path:

- *contracts/core/SettingsManager.sol:getNextFundingRate(..)*

Recommendation: Either fix the documentation or change the code to conform to it.

Found At: Review #4.1

Status: Fixed (Review #5)

26. Incorrect Logic

The documentation says that to decrease or close a position one of the following must hold:

- The position is not profitable
- The delay has passed
- The price has changed 2% or more since the opening

The last point was corrected by the Vela team during the audit to state: the price has increased 2%+ in the profitable direction.

The checks in *contracts/core/VaultUtils.sol:validateDecreasePosition(..)*, namely the *if* statement at line 278 only checks that the price went up 2% regardless of whether the position is long or short.

It should be fixed to return *true* either if the position is long, and the price went up 2%+, or if the position is short, and the price went down 2%+.

Path:

- *contracts/core/VaultUtils.sol:validateDecreasePosition(..)*

Recommendation: Apply the mentioned fix.

Found At: Review #4.1

Status: Fixed (Review #5)

27. Unfinalized Code

Paths:

- *contracts/core/TriggerOrderManager.sol:8*: Hardhat console import
- *contracts/core/Vault.sol:20*: Hardhat console import
- *contracts/core/VaultUtils.sol:15*: Hardhat console import
- *contracts/staking/ComplexRewarderPerSec.sol:11*: Hardhat console import
- *contracts/staking/TokenFarm.sol:12*: Hardhat console import

Recommendation: Remove non-production code.

Found At: Review #4.1

Status: Fixed (Review #5)

28. Highly Permissive Role Access

Token prices can be manipulated by changing *tokenDecimals*. Moreover, *tokenDecimals* should not change for a token, similarly to *ERC20.decimals(..)*.

Path:

- *contracts/core/PriceManager.sol:setTokenConfig(..)*

Recommendation: Document this possibility or allow setting token config no more than once per token.

Found At: Review #4.1

Status: Fixed (Review #5)

29. Highly Permissive Role Access

closeDeltaTime can be changed at any time by the bearer of 'gov' role, and it does not have an upper or lower bound. It can be set in a way that does not allow users to decrease (exit from) their positions effectively forever.

Path:

- *contracts/core/SettingsManager.sol:setCloseDeltaTime(..)*

Recommendation: Add explicit bounds check for the value.

Found At: Review #4.1

Status: Fixed (Review #5)

30. Highly Permissive Role Access

depositFee can be changed at any time by the bearer of 'gov' role, and it does not have an upper or lower bound. It can be set to 100% to fully consume amounts being transferred as the deposits.

Path:

- *contracts/core/SettingsManager.sol:setDepositFee(..)*

Recommendation: Add explicit bounds check for the value.

Found At: Review #4.1

Status: Fixed (Review #5)

31. Highly Permissive Role Access

liquidateThreshold can be changed at any time by the bearer of 'gov' role, and it does not have an upper or lower bound. It can be set to a value that allows liquidating any position of any user.

Path:

- *contracts/core/SettingsManager.sol:setLiquidateThreshold(..)*

Recommendation: Add explicit bounds check for the value.

Found At: Review #4.1

Status: Fixed (Review #5)

32. Highly Permissive Role Access

stakingFee can be changed at any time by the bearer of 'gov' role, and it does not have an upper or lower bound. It can be set to 100% to fully consume the amounts that are being transferred to the staking.

Path:

- *contracts/core/SettingsManager.sol:setStakingFee(..)*

Recommendation: Add explicit bounds check for the value.

Found At: Review #4.1

Status: Fixed (Review #5)

33. Highly Permissive Role Access

The check at *contracts/core/Vault.sol:393* will block users' ability to get their assets back, if the *settingsManager.isStaking* flag is activated (which can be done only by the platform).

Path:

- *contracts/core/Vault.sol:unstake(..)*

Recommendation: Document the platform's ability to block user assets or remove the ability.

Found At: Review #4.1

Status: Fixed (Review #5)

34. Highly Permissive Role Access

The check at *contracts/core/Vault.sol:427* will block users' ability to get their assets back, if the *settingsManager.isStaking* flag is activated (which can be done only by the platform).

Path:

- *contracts/core/Vault.sol:withdraw(..)*

Recommendation: Document the platform's ability to block user assets or remove the ability.

Found At: Review #4.1

Status: Fixed (Review #5)

35. Highly Permissive Role Access

The function *contracts/core/Vault.sol:setVaultSettings(..)* allows the 'owner' to change the implementation of *priceManager*, *settingsManager*, *triggerOrderManager*, and *vaultUtils*. This could be abused to gain at the expense of users.

Path:

- *contracts/core/Vault.sol:setVaultSettings(..)*

Recommendation: Document this ability explicitly or allow calling the function only once.

Found At: Review #4.1

Status: Fixed (Review #5)

36. Highly Permissive Role Access

The *if* statement at *contracts/tokens/BaseToken.sol:59* allows a bearer of 'handler' role to use anybody's balance without the allowance.

Path:

- *contracts/tokens/BaseToken.sol:transferFrom(..)*

Recommendation: Document this ability explicitly or remove the functionality.

Found At: Review #4.1

Status: Fixed (Review #5)

37. Highly Permissive Role Access

The *if* statement at *contracts/tokens/BaseToken.sol:113* means that the bearer of 'gov' role can disable the ability to transfer (i.e. use the asset in a meaningful way) for everybody who does not have 'handler' role.

Path:

- *contracts/tokens/BaseToken.sol:_transfer(..)*

Recommendation: Document this ability explicitly or remove the functionality.

Found At: Review #4.1

Status: Fixed (Review #5)

38. Highly Permissive Role Access

The bearer of 'owner' role can set use *contracts/staking/TokenFarm.sol:updateCooldownDuration(..)* at any moment to set *cooldownDuration* arbitrarily high to block un-staking for any period of time.

Path:

- *contracts/staking/TokenFarm.sol:updateCooldownDuration(..)*

Recommendation: Check explicit bounds for the parameter in the setter.

Found At: Review #4.1

Status: Fixed (Review #5)

39. Highly Permissive Role Access

Token prices can be manipulated by changing *priceDecimals*. Moreover, *priceDecimals* should not change for a token, similarly to *ERC20.decimals(..)*.

Path:

- *contracts/core/VaultPriceFeed.sol:setTokenConfig(..)*

Recommendation: Document this possibility or allow setting token config no more than once per token.

Found At: Review #4.1

Status: Fixed (Review #5)

40. Highly Permissive Role Access

The bearer of 'owner' role can use *contracts/staking/TokenFarm.sol:updateVestingDuration(..)* at any moment to arbitrarily prolong the vesting lock period represented by *vestingDuration* (and hence reduce individual claim amounts).

Path:

- *contracts/staking/TokenFarm.sol:updateVestingDuration(..)*

Recommendation: Check explicit bounds for the parameter in the setter.

Found At: Review #4.1

Status: Fixed (Review #5)

41. Highly Permissive Role Access

The bearer of 'PAUSER' role can use *contracts/tokens/VELA.sol:pause(..)* to block transfers of the token.

Path:

- *contracts/tokens/VELA.sol:_beforeTokenTransfer(..)*

Recommendation: Document the possibility explicitly or remove the mentioned functionality.

Found At: Review #4.1

Status: Fixed (Review #5)

42. Missing Access Control

The functions *addManyToWhitelist(..)* and *removeManyFromWhitelist(..)* have no access control, and allow anybody to add/remove somebody's trusted addresses - those that are allowed to deposit and stake in Vault on behalf of the one who entrusted it to them.

Path:

- *contracts/core/SettingsManager.sol*

Recommendation: Implement a proper access control

Found At: Review #4.2

Status: Fixed (Review #5)

43. Incorrect Constant Value

The definition at *contracts/access/Constants.sol:23* reads:

uint256 public constant MAX_FEE_BASIS_POINTS = 500; // 5%

The comment next to the value of the constant says “5%”, but it is actually 0.5%, because the value is divided by 10^5 (the value of *BASIS_POINTS_DIVISOR*) in the code, when it is used in calculations. The fact that this is an issue was confirmed with the Vela team.

Path:

- *contracts/access/Constants.sol:23*

Recommendation: Correct the value.

Found At: Review #4.2

Status: Fixed (Review #5)

Medium

1. Usage of Built-in Transfer

The built-in *transfer(..)* and *send(..)* functions use a hard-coded amount of Gas. In case when the receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Path:

- *contracts/tokens/Token.sol:withdraw(..)*

Recommendation: Replace *transfer(..)* and *send(..)* functions with call or provide special mechanism for interacting with a smart contract.

Found At: Review #1

Status: Fixed (Review #2)

2. Ignored Error

The function ignores the return value of the transfer.

Path:

- *contracts/tokens/Token.sol:withdrawToken(..)*

Recommendation: Propagate the return value up, so that the caller of *withdrawToken(..)* could react to an error.

Found At: Review #1

Status: Fixed (Review #2)

3. **Missing Events**

It is recommended to emit events after changing values. This will make it easy for everyone to notice such changes.

Path:

- *contracts/access/Governable.sol:setGov(..)*
- *contracts/core/BetaTrading.sol:updateExpirationTime(..),updateClaimAmount(..),claim(..)*
- *contracts/core/Router.sol:setGov(..)*
- *contracts/core/Vault.sol:initialize(..),setFees(..), setFundingRate(..),setRewardRate(..)*
- *contracts/core/VaultPriceFeed.sol:setGov(..),setSpreadThresholdBasisPoints(..),setPriceSampleSpace(..),setMaxStrictPriceDeviation(..)*
- *contracts/tokens/BaseToken.sol:setGov(..)*
- *contracts/tokens/vUSDC.sol:setGov(..)*
- *contracts/tokens/YieldToken.sol:setGov(..)*
- *contracts/tokens/YieldTracker.sol:setGov(..)*

Recommendation: Implement event emits after changing contract values.

Found At: Review #1

Status: Fixed (Review #2)

4. **Incorrect Logic**

When a position is decreased via *contracts/core/Vault.sol:_decreasePosition(..)*, there is the case when the fee has to be deducted from the collateral: see the lines 606-609 in the function *_reduceCollateral(..)*.

The fact that this situation happened is not properly recorded in the returned values of *_reduceCollateral(..)*: *usdOut* is equal to *usdOutAfterFee*, which makes the fee equal to 0 in the eyes for the caller of this function.

The further execution of *_decreasePosition(..)* will proceed as if the fee is simply 0 i.e. it will not be transferred to the system, even though it has been deducted from the position collateral.

Path:

- *contracts/core/Vault.sol:_decreasePosition(..),_reduceCollateral(..)*

Recommendation: Make sure the fee is correctly processed under the described circumstances.

Found At: Review #4.1

Status: Fixed (Review #5)

5. **Missing Validation (Requirements Violation)**

   The documentation states that it is not valid if a stop order price is already triggered at the moment of order creation. This is not checked in the code. The function where it would be the most appropriate to do the validation is *`VaultUtils.validatePosData(..)`*.

   Path:

   - *`contracts/core/VaultUtils.sol:validatePosData(..)`*

   Recommendation: Do the validation or remove the requirement from the documentation.

   Found At: Review #4.1

   Status: Fixed (Review #5)

6. **Missing Validation**

   `feeRewardBasisPoints` can be changed at any time, and it does not have an upper bound. It can be set to 100% or more, which would lead to unintended consequences.

   Path:

   - *`contracts/core/SettingsManager.sol:setVaultSettings(..)`*

   Recommendation: Add an explicit upper bound check for the value.

   Found At: Review #4.1

   Status: Fixed (Review #5)

7. **Missing Validation**

   `delayDeltaTime` can be changed at any time, and it does not have an upper or lower bound. It can be set in a way that does not allow users to increase their positions effectively forever.

   Path:

   - *`contracts/core/SettingsManager.sol:setDelayDeltaTime(..)`*

   Recommendation: Add explicit bounds check for the value.

   Found At: Review #4.1

   Status: Fixed (Review #5)

8. **Missing Validation**

fundingInterval can be changed at any time, and it does not have an upper bound. It can be accidentally set to a value that is too large, leading to unintended consequences.

Path:

- *contracts/core/SettingsManager.sol:setFundingInterval(..)*

Recommendation: Add explicit bounds check for the value.

Found At: Review #4.1

Status: Fixed (Review #5)

9. Missing Events

State changes are not tracked with events.

Paths:

- *contracts/core/BetaTrading.sol:claim(..)*
- *contracts/core/BetaTrading.sol:updateClaimAmount(..)*
- *contracts/core/BetaTrading.sol:updateExpirationTime(..)*
- *contracts/core/SettingsManager.sol:setVaultSettings(..)*

Recommendation: Emit the respective events.

Found At: Review #4.1

Status: Fixed (Review #5)

10. Inconsistent Data

The function *contracts/core/Vault.sol:_removeOrderId(..)* uses the *delete* operator to remove an active position id for a user from the array in *userPositionIds*.

The layout of array *userPositionIds[account]* for some *account* is *[0, 1, 2, …, n]*, where *n* is the length of that array minus 1.

delete userPositionIds[account][i] just sets the *i*-th element to 0, without actually removing it from the array.

0 is a valid position id, and *_removeOrderId(account, 0)* will do nothing at all: it will not change the array, and there will be no sign that the position/order id 0 was removed.

Path:

- *contracts/core/Vault.sol:_removeOrderId(..)*

Recommendation: This field is only written (costs extra gas) and never read in the contracts. Its purpose is to track the set of active position/order ids and provide the view for the off-chain Vela's software. With this in mind, the recommendation is to remove

it completely, and use events to track the active position ids off-chain.

Found At: Review #4.1

Status: Fixed (Review #5)

11. Missing Validation

The function *contracts/staking/TokenFarm.sol:updateTierInfo(..)* Should verify the sortedness of *_levels* (ascending) and *_percents* (descending). It should verify that each item in *_percents* is less than or equal to 100%. The issue is not severe, since the function is callable only by *owner*, and any mistakes can be fixed by calling this function again.

Path:

- *contracts/staking/TokenFarm.sol:updateTierInfo(..)*

Recommendation: Add the mentioned validations.

Found At: Review #4.1

Status: Reported

12. Missing Validation

In the function *contracts/core/TriggerOrderManager.updateTriggerOrders(..)*, there is no limit to the number of trigger prices. One could set a large number of trigger prices for an order to make it Gas-expensive for the platform to trigger the order. The issue is not severe, since the platform's off-chain order triggering logic could exclude orders having too many trigger prices by checking the number of trigger prices using the view function *getTriggerOrderInfo(..)*.

Path:

- *contracts/core/TriggerOrderManager.updateTriggerOrders(..)*

Recommendation: Add the mentioned validations.

Found At: Review #4.1

Status: Reported

Low

1. Outdated Solidity Version

Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version. The project uses compiler version 0.6.12.

Path:

- *contracts/*

Recommendation: Use a contemporary compiler version.

Found At: Review #1

Status: Fixed (Review #2)

2. **Precision Loss Due To Division**

   It is considered good practice to keep any data as accurate as possible. The function has the code which performs division before multiplication during the calculation. This may lead to a loss of precision.

   Path:

   - *contracts/core/Vault.sol:getNextFundingRate(..),getDelta(..)*

   Recommendation: Keep data actual to the current system state, perform multiplication before division.

   Found At: Review #1

   Status: Fixed (Review #3)

3. **Use of Hard-Coded Values**

   The function *validatePosType* takes as argument _posType is using values 0, 1, 2, and 3 without any explanations.

   The function *validateLiquidation* returns 0, 1, or 2 without any explanations.

   Paths:

   - *contracts/core/VaultUtils.sol:validatePosType(..),validateLiquidation(..)*
   - *contracts/core/Vault.sol:626*

   Recommendation: Convert these variables into constants.

   Found At: Review #1

   Status: Fixed (Review #4.1)

4. **Single Responsibility Pattern Violation**

   The same contract should not be responsible for multiple separate functionalities. Contract complexity is increased, and maintainability is lowered.

   Paths:

   - *contracts/staking/TokenFarm.sol*: staking and vesting could be split
   - *contracts/core/Vault.sol*: staking, core trading, order management could be split

Recommendation: Separate functionality between different contracts.

Found At: Review #1

Status: Reported

5. **Missing Zero Address Validation**

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

   This can lead to unwanted external calls to 0x0.

   Path:

   - *contracts/access/Governable.sol:setGov(..)*
   - *contracts/core/BetaTrading.sol:constructor(..)*
   - *contracts/core/Router.sol:constructor(..),setGov(..)*
   - *contracts/core/Vault.sol:constructor(..),initialize(..), setPriceFeed(..),setFeeManager(..)*
   - *contracts/core/VaultPriceFeed.sol:setGov(..),setTokens(..), setChainlinkFlags(..),setPairs(..)*
   - *contracts/core/VaultUtils.sol:constructor(..)*
   - *contracts/tokens/BaseToken.sol:setGov(..)*
   - *contracts/tokens/vUSDC.sol:setGov(..)*
   - *contracts/tokens/YieldToken.sol:setGov(..)*
   - *contracts/tokens/YieldTracker.sol:constructor(..),setGov(..),setDistributor(..)*

   Recommendation: Implement zero address checks.

   Found At: Review #1

   Status: Fixed (Review #2)

6. **Functions that Can Be Declared External**

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

   Paths:

   - *contracts/core/Vault.sol:getPosition(..),getPositionLeverage(..),getPositionDelta(..),deposit(..), withdraw(..)*
   - *contracts/core/VaultPriceFeed.sol:getPrice(..),getLastPrice(..)*
   - *contracts/core/VaultReader.sol:getVaultTokenInfo(..),getValidateInfos(..),getPositions(..)*
   - *contracts/core/VaultUtils.sol:validateLiquidation(..),updateCumulativeFundingRate(..),validateTrigger(..),getFundingFee(..),setLiquidateThreshold(..)*
   - *contracts/oracle/FastPriceFeed.sol:setAdmin(..),latestAnswer(..),latestRound(..),setLatestAnswer(..),getRoundData(..),setDescription(..)*

- *contracts/oracle/PriceFeed.sol:setAdmin(..),latestAnswer(..),latestRound(..),setLatestAnswer(..),getRoundData(..)*
- *contracts/staking/TokenFarm.sol:getTotalVested(..)*
- *contracts/tokens/Token.sol:mint(..),withdrawToken(..),deposit(..),withdraw(..)*

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

Found At: Review #1

Status: Fixed (Review #2)

7. Duplicate Code

Best practices and optimization are not applied in these contracts. Duplicate and inefficient code includes (but is not limited to):

Paths:

- Gov functionalities are copy pasted instead of inherited from *Governable*.
- *struct Position* is duplicated at *contracts/core/Vault.sol* and *contracts/core/VaultUtils.sol*.
- *contracts/core/VaultUtils.sol:validateTrigger(..),takeVUSDIn(..),takeVUSDOut(..)*: repeating patterns that can be extracted.
- *contracts/core/Vault.sol:getLastPrice(..)*: is sometimes called several times on the same call stack, yet the value it returns does not change. The value it returns should be reused.
- *contracts/core/VaultUtils.sol.validatePositionTPSL(..):* can be significantly simplified/reduced.

Recommendation: Apply best practices.

Found At: Review #1

Status: Fixed (Review #4.1)

8. Unindexed Events

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

Paths:

- *contracts/staking/TokenFarm.sol:VestingClaim,VestingDeposit,VestingWithdraw*
- *contracts/core/Vault.sol:IncreasePosition,LiquidatePosition,DecreasePosition,UpdatePosition,ClosePosition,Stake,Unstake*
- *contracts/core/VaultUtils.sol:TakeVUSDIn,TakeVUSDOut*

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

Found At: Review #1

Status: Fixed (Review #2)

9. Variables that Should be Declared Constant

State variables that do not change their value should be declared constant to save Gas.

Path:

- *contracts/core/Vault.sol:liquidityFeeBasisPoints*

Recommendation: Declare the above-mentioned variables as constants.

Found At: Review #1

Status: Fixed (Review #2)

10. Commented Code Parts

There are commented parts of code. This reduces code quality.

Path:

- *contracts/core/Vault.sol:117,409*

Recommendation: Remove commented parts of code.

Found At: Review #1

Status: Fixed (Review #2)

11. State Variables Can Be Declared Immutable

Variables value is set in the constructor. This variable can be declared immutable.

This will lower the Gas taxes.

Paths:

- *contracts/core/Vault.sol:vlp,vUSDC*
- *contracts/core/BetaTrading.sol:vUSDC*
- *contracts/core/Router.sol:vault,weather*
- *contracts/core/VaultUtils.sol:vault,vUSDC*
- *contracts/staking/TokenFarm.sol:vestingDuration,esToken,claimableToken*
- *contracts/tokens/Token.sol:_name,_symbol,_decimals*
- *contracts/tokens/YieldTracker.sol:yieldToken*

Recommendation: Declare mentioned variables as immutable.

Found At: Review #1

Status: Fixed (Review #3)

12. Unused Code

Paths:

- *contracts/staking/libraries/Math.sol*

- *contracts/tokens/MockToken.sol*
- *contracts/core/Vault.sol:hasDynamicFees,taxBasisPoints,isCustomFees,collateralAmounts,customFeePoints,tokenBalances*: unused or write-only fields

Recommendation: Remove unused code.

Found At: Review #1

Status: Fixed (Review #3)

13. Inconsistent Naming Convention

Most of the internal functions in the project are prefixed by _. There are examples which do not follow this rule.

Path:

- *contracts/Vault.sol*

Recommendation: Apply consistent naming rules.

Found At: Review #1

Status: Fixed (Review #2)

14. Style Guide Violation

The provided projects should follow the official guidelines.

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

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Paths:

- *contracts/core/Vault.sol*
- *contracts/staking/TokenFarm.sol*
- *contracts/tokens/MintableBaseToken.sol*
- *contracts/tokens/vUSDC.sol*

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

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

Some contracts are not formatted correctly.

Paths:

- *contracts/core/Vault.sol*
- *contracts/core/VaultUtils.sol*
- *contracts/oracle/FastPriceFeed.sol*
- *contracts/oracle/PriceFeed.sol*
- *contracts/staking/TokenFarm.sol*
- *contracts/tokens/BaseToken.sol*
- *contracts/tokens/Token.sol*
- *contracts/tokens/vUSDC.sol*
- *contracts/tokens/YieldToken.sol*

Recommendation: Follow the official Solidity guidelines.

Found At: Review #1

Status: Fixed (Review #2)

15. Inappropriate Typing

TokenFarm calls the *burn(..)* function on the variable *_esToken*. This variable is of type *IBoringERC20*, but this interface does not have such a function. The call is done through casting to an incompatible interface.

Path:

- *contracts/staking/TokenFarm.sol*

Recommendation: Implement appropriate typing.

Found At: Review #1

Status: Reported

16. Boolean Tautology

There are multiple cases like :

pool.enableLock == false || (pool.enableLock == true && user.status == Status.LOCKED)

which simplifies to

!pool.enableLock || user.status == Status.LOCKED

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

Path:

- *contracts/staking/TokenFarm.sol*

Recommendation: Remove boolean equality, and boolean expressions whose values can be inferred from the context without executing the code.

Found At: Review #2

Status: Fixed (Review #4.1)

17. Use of Hard-Coded Values

The hardcoded “1” looks dangerous because it is the position of an element in an array inside a foreign contract. The assumption that the right pool will always be the second one created (having position “1”) is perhaps too strong.

Path:

- *contracts/core/Vault.sol:512*

Recommendation: Avoid hardcoding values that could be not hardcoded. If the hardcoded value is a known constant - define it as a constant or use enums.

Found At: Review #2

Status: Fixed (Review #3)

18. Unused Variable

The following variables are never used.

Paths:

- *contracts/staking/TokenFarm.sol:stakedAmounts*
- *contracts/oracle/PriceFeed.sol:decimals*
- *contracts/oracle/FastPriceFeed.sol:decimals*

Recommendation: Remove unused variables.

Found At: Review #3

Status: Reported

19. Commented Code Parts

There are commented parts of the code. This reduces code quality.

Paths:

- *contracts/staking/TokenFarm.sol:118-121*
- *contracts/core/VaultUtils.sol:154,159*

Recommendation: Remove commented parts of code.

Found At: Review #3

Status: Fixed (Review #4.1)

20. Misleading Comments

There are comments in the code that are factually incorrect.

Path:

- *contracts/core/VaultUtils.sol:153*

Recommendation: Remove or correct the comments.

Found At: Review #3

Status: Fixed (Review #4.1)

21. Use Of Hard-Coded Values

The value is hardcoded, and it is not clear why exactly “1” is chosen.

Path:

- *contracts/core/VaultUtils.sol:159*

Recommendation: Do not hardcode values.

Found At: Review #3

Status: Fixed (Review #4.1)

22. Incorrect Logic

The *if* condition should be:

user.amount >= tierLevels[i]

This off-by-one error is insignificant, because *user.amount* is expected to be huge.

Path:

- *contracts/staking/TokenFarm.sol:406*

Recommendation: Apply the mentioned fix.

Found At: Review #4.1

Status: Fixed (Review #5)

23. Redundant SafeMath

Since Solidity 0.8, the overflow checks are built in, therefore SafeMath only introduces double overflow checks, makes code less readable, and increases code size without benefit. In the cases when an overflow error needs to have a message, that can be added separately/explicitly.

Paths:

- *contracts/core/PriceManager.sol:10*
- *contracts/core/SettingsManager.sol:16*
- *contracts/core/Vault.sol:23*

- *contracts/core/VaultPriceFeed.sol:12*
- *contracts/core/VaultUtils.sol:18*

Recommendation: Do not use SafeMath in Solidity 0.8 and above.

Found At: Review #4.1

Status: Fixed (Review #5)

24. Inefficient Code

Paths:

- *contracts/core/Vault.sol:lastPosId*: the field should be turned into the single global position id counter, instead of per-account counters; this is cheaper in Gas and achieves the same result (generation of unique position ids).
- *contracts/core/TriggerOrderManager.sol:executeTriggerOrders(..)* : The code only needs to check *either* SL or TP prices, but it checks both. Moreover, the loops could exit early once the amount percent variable has reached 100%.
- *contracts/core/TriggerOrderManager.sol:validateTPSLTriggers(..)* : The code only needs to check *either* SL or TP prices, but it checks both. Moreover, the loops could exit early once the amount percent variable is more than 0.
- *contracts/core/TriggerOrderManager.sol:validateTriggerOrdersData(..)*: The code could exit early on the first encountered violation, but it proceeds until the end despite violations
- *contracts/core/TokenFarm.sol:getTier(..)*: Since *tierLevels* are assumed to be sorted, the code could exit early on the first match between *user.amount* and an interval in *tierLevels*

Recommendation: Apply the mentioned optimisations.

Found At: Review #4.1

Status: Fixed (Review #5)

25. Inefficient Code

Paths:

- *contracts/core/TriggerOrderManager.sol:executeTriggerOrders(..)* : if the trigger prices were sorted, it would have been possible to exit the loops early on the first failure to trigger a price.
- *contracts/core/TriggerOrderManager.sol:validateTPSLTriggers(..)* : if the trigger prices were sorted, it would have been possible to exit the loops early on the first failure to trigger a price
- *contracts/core/ComplexRewardPerSec.sol:onVelaReward(..)*: instead of calculating *user.rewardDebt* on the line 267, the code could just remember *pool.accTokenPerShare* in some field e.g. *user.lastAccTokenPerShare*, and then calculate *pending* on

the line 243 as `user.amount * (pool.accTokenPerShare - user.lastAccTokenPerShare) / ACC_TOKEN_PRECISION`
- `contracts/core/TokenFarm.sol:_claim(..)`: lines 316 and 317 could be replaced with a more simple/efficient calculation:

```
uint256 totalClaimed = cumulativeClaimAmounts[_account];

uint256 amount = totalClaimed - claimedAmounts[_account];

claimedAmounts[_account] = totalClaimed;
```

Recommendation: Apply the mentioned optimizations.

Found At: Review #4.1

Status: Reported

26. Precision Loss Due To Division

Division is performed too early or too many times (e.g. a/b/c instead of a/(b*c)). Division leads to loss of precision; therefore, it should be avoided or deferred as much as possible.

Paths:

- `contracts/core/Vault.sol:342`
- `contracts/core/Vault.sol:454`

Recommendation: Eliminate or defer division as much as possible.

Found At: Review #4.1

Status: Fixed (Review #5)

27. Default Visibility

Paths:

- `contracts/core/BetaTrading.sol:betaExpiration`
- `contracts/core/PriceManager.sol:priceFeed`
- `contracts/core/TriggerOrderManager.sol:vault`
- `contracts/core/Vault.sol:priceFeed`
- `contracts/core/Vault.sol:vlp`
- `contracts/core/Vault.sol:vUSDC`

Recommendation: Add explicit visibility modifier.

Found At: Review #4.1

Status: Fixed (Review #5)

28. Redundant Code

Paths:

- *contracts/staking/interfaces/IFarmDistributor.sol:totalAllocPoint(..)*: unused function, not implemented anywhere in the codebase.
- *contracts/core/SettingsManager.sol:validatePosition(..)*: the return value is unused.
- *contracts/core/Vault.sol:UserInfo:lastFeeReserves*: the field is unused.
- *contracts/core/VaultUtils.sol:317,319-323*: dead code.
- *contracts/core/SettingsManager.sol:18-32*: the constants are duplicated in multiple contracts.
- *contracts/oracle/PriceFeed.sol*: the file is almost completely identical to *contracts/oracle/FastPriceFeed.sol.*
- *_getPositionKey(..)*: function is duplicated in multiple contracts.
- *contracts/core/Vault.sol*: unused imports.
- *contracts/core/VaultUtils.sol*: unused imports.
- *contracts/staking/TokenFarm.sol*: unused imports.
- *contracts/core/TriggerOrderManager.sol:executeTriggerOrders(..)*: the code that processes TP prices is almost identical to the code that processes SL prices, yet it is duplicated for both TP and SL prices.
- *contracts/core/TriggerOrderManager.sol:validateTPSLTriggers(..)*: the code that processes TP prices is almost identical to the code that processes SL prices, yet it is duplicated for both TP and SL prices.
- *contracts/core/TriggerOrderManager.sol:validateTriggerOrdersData(..)*: the code that processes TP prices is almost identical to the code that processes SL prices, yet it is duplicated for both TP and SL prices.
- *contracts/core/VaultUtils.sol:distributeFee(..),takeVUSDIn(..),takeVUSDOut(..)*: the methods have the same code pattern that is duplicated.
- *contracts/staking/ComplexRewardPerSec.sol*: the transfer pattern that decides on *isNative* is duplicated many times.

Recommendation: Eliminate the redundancies.

Found At: Review #4.1

Status: Fixed (Review #5)

29. Dangling Comment

Comments refer to something that does not exist in the code.

Paths:

- *contracts/core/PriceManager.sol:26*. Fixed (Review #5)
- *contracts/core/Vault.sol:275*
- *contracts/staking/TokenFarm.sol:101*

Recommendation: Remove dangling comments.

Found At: Review #4.1

Status: Fixed (Review #5)

30. Commented Code Parts

Commented-out code may mislead a reader.

Paths:

- *contracts/core/Vault.sol:49*
- *contracts/core/Vault.sol:435*

Recommendation: Remove commented-out code.

Found At: Review #4.1

Status: Fixed (Review #5)

31. Inappropriate Naming

Code members are named in a way that does not well represent their meaning/purpose.

Path:

- *contracts/core/Vault.sol:UserInfo:amount*: a better name would be ‘vlpAmount’
- *contracts/core/Vault.sol:lastAddedAt*: a better name would be ‘lastStakedAt’
- *contracts/core/Vault.sol:_increaseTotalUSD(..)*: a better name would be ‘_accountDeltaAndFeeIntoTotalUSDC’
- *contracts/staking/TokenFarm.sol:totalVestingSupply*: a better name would be ‘totalLockedVestingAmount’
- *contracts/staking/TokenFarm.sol:cumulativeClaimAmounts*: a better name would be ‘unlockedVestingAmounts’
- *contracts/staking/TokenFarm.sol:lastVestingTimes*: a better name would be ‘lastVestingUpdateTimes’
- *contracts/staking/TokenFarm.sol:vestingBalances*: a better name would be ‘lockedVestingAmounts’
- *contracts/staking/TokenFarm.sol:_burn(..)*: a better name would be ‘_decreaseLockedVestingAmount’

Recommendation: Give the members names that correspond to their purpose.

Found At: Review #4.1

Status: Fixed (Review #5)

32. Inconsistent Code Formatting

Code formatting is not applied or differs across the codebase.

Path:

- *contracts/*

Recommendation: Format the code using a tool.

Found At: Review #4.1

Status: Fixed (Review #5)

33. Redundant Code

Paths:

- *contracts/core/Multicall.sol*: a copy-pasted library code that is not used in the other contracts.
- *contracts/core/SettingsManager.sol:customFeePoints*: the field is never read in the contracts, only written.
- *contracts/tokens/BaseToken.sol:admins*: the field is unused.
- *contracts/staking/ComplexRewarderPerSec.sol:RewardInfo:startTimestamp*: the field is never read in the contracts, only written; it can be computed as either pool.startTimestamp or the previous reward endTimestamp.
- *contracts/staking/TokenFarm.sol:totalLockedUpRewards*: the field is unused.
- *contracts/staking/TokenFarm.sol:ACC_TOKEN_PRECISION*: the field is unused.
- *contracts/oracle/FastPriceFeed.sol:aggregator*: the field is unused.
- *contracts/oracle/FastPriceFeed.sol:description*: the field is never read in the contracts, only written.
- *contracts/oracle/FastPriceFeed.sol:answers*: the field is never read in the contracts, only written.
- *contracts/oracle/FastPriceFeed.sol:latestAts*: the field is never read in the contracts, only written.
- *contracts/staking/ComplexRewardPerSec.sol:emergencyRewardWithdraw(..)*: *nonReentrant* modifier is excessive, because the access is already controlled with *onlyOwner* modifier.
- *contracts/staking/ComplexRewardPerSec.sol:emergencyWithdraw(..)*: *nonReentrant* modifier is excessive, because the access is already controlled with *onlyOwner* modifier.
- *contracts/staking/ComplexRewardPerSec.sol:onVelaReward(..)*: *nonReentrant* modifier is excessive, because the access is already controlled with *onlyDistributor* modifier.
- *contracts/staking/ComplexRewardPerSec.sol:onVelaReward(..)*: *nonReentrant* modifier is excessive, because the access is already controlled with *onlyDistributor* modifier.
- *contracts/staking/TokenFarm.sol:336*: the code makes an ERC-20 transfer to a beneficiary B, and then measures the difference between the balance of B after the transfer and before it, instead of just using the very amount that's been transferred.
- *contracts/core/SettingsManager.sol*: unused imports.
- *contracts/core/SettingsManager.sol:updateCumulativeFundingRate(..)*: duplicate map lookups.
- *contracts/core/SettingsManager.sol:validatePosition(..)*: duplicate mapping lookups.
- *contracts/core/SettingsManager.sol:getNextFundingRate(..)*: duplicate mapping lookups.

- *contracts/core/Vault.sol*: every time *_increasePosition(..)* is called, *_collectMarginFees(..)* is called right before it, and then one more time inside it with the same parameters.
- *contracts/core/VaultUtils.sol:validatePosData(..)*: the same code pattern is repeated for _isLong, and !_isLong.
- *contracts/core/VaultUtils.sol:validateTrailingStopInputData(..)*: the same code pattern is repeated for _isLong, and !_isLong.
- *contracts/core/VaultUtils.sol:validateTrailingStopPrice(..)*: the same code pattern is repeated for _isLong, and !_isLong.
- *contracts/core/VaultUtils.sol:validateTrigger(..)*: the same code pattern is repeated for _isLong, and !_isLong.
- *contracts/staking/ComplexRewardPerSec.sol*: the reward formula calculation is duplicated.
- *contracts/staking/TokenFarm.sol:withdraw(..),emergencyWithdraw(..),_deposit(..)*: the methods have the same code pattern that is duplicated.
- *contracts/core/Vault.sol*: inheriting *Ownable* is redundant.
- *contracts/core/VaultUtils.sol*: inheriting *Governable* is redundant.

Recommendation: Eliminate the redundancies.

Found At: Review #4.1

Status: Reported

34. Default Visibility

Path:

- *contracts/core/VaultUtils.sol:settingsManager*

Recommendation: Add explicit visibility modifier.

Found At: Review #4.1

Status: Reported

35. Field Not Marked Constant

Paths:

- *contracts/tokens/BaseToken.sol:name*: should be constant due to ERC-20
- *contracts/tokens/BaseToken.sol:symbol*: should be constant due to ERC-20
- *contracts/tokens/VELA.sol:_maxSupply*: field value is hardcoded, never modified

Recommendation: Add *constant* modifier to the fields.

Found At: Review #4.1

Status: Reported

36. Field Not Marked Immutable

Paths:

- *contracts/core/VaultUtils.sol:priceManager*
- *contracts/core/VaultUtils.sol:settingsManager*
- *contracts/tokens/VELA.sol:_trustedForwarder*

Recommendation: Add *immutable* modifier to the fields.

Found At: Review #4.1

Status: Reported

37. Redundant Logical Expressions

```
bool foo;
if (someBool) {
      foo = true;
} else {
      foo = false;
}
```

is the same as

```
bool foo = someBool;
```

Another example:

```
bool foo;
if (someBool) {
      foo = true;
} else {
      foo = false;
}
return foo;
```

is the same as

```
if (someBool) {
      return true;
}
return false;
```

Or even more simply:

return someBool;

This category includes boolean expressions, which contain terms whose value can be inferred from the context. The following code has some of the described kinds of redundancy.

Paths:

- *contracts/core/VaultUtils.sol:validateConfirmDelay(..)*
- *contracts/core/VaultUtils.sol:validateDecreasePosition(..)*
- *contracts/core/VaultUtils.sol:validatePosData(..)*
- *contracts/core/VaultUtils.sol:validateTrailingStopPrice(..)*
- *contracts/core/VaultUtils.sol:validateTrigger(..)*: note that *statusFlag* is effectively boolean.
- *contracts/core/VaultUtils.sol:validateSizeCollateralAmount(..)*. Fixed (Review #5)

Recommendation: Eliminate the redundancies.

Found At: Review #4.1

Status: Reported

38. Double Inheritance

The same contract is inherited more than once when it is easy to avoid.

Path:

- *contracts/token/VELA.sol*: ERC20 does not need to be an explicit superclass, because it already comes from ERC20Permit.

Recommendation: Remove double inheritance.

Found At: Review #4.1

Status: Reported

39. Inappropriate Typing

Solidity type system is not used correctly, which reduces the compiler's ability to rule out mistakes in the code early.

Paths:

- *contracts/oracle/FastPriceFeed.sol:answer*: the field's type admits negative values, whereas the concept the field represents does not allow negative values. Fixed (Review #5)
- *contracts/staking/TokenFarm.sol*: TokenFarm is used as IFarmDistributor in other contracts, yet it does not implement the interface explicitly.

Recommendation: Apply the mentioned per-item suggestions.

Found At: Review #4.1

Status: Reported

40. Solidity Style Guidelines Violation

The guidelines are described at:
https://docsiditylang.org/en/v0.8.17/style-guide.html

Paths:

- *contracts/staking/ComplexRewarderPerSec.sol*: structs should go before fields.
- *contracts/staking/ComplexRewarderPerSec.sol:ACC_TOKEN_PRECISION* : the field is named like a constant, but it is not one.

Recommendation: Fix the inconsistencies with the guidelines.

Found At: Review #4.1

Status: Reported

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.